AI Based Crack Recording and Repair using Autonomous Robot and Bio-Concrete Agent

Dissertation
Zur Erlangung des akademischen Grades Doktor-Ingenieur (Dr.-Ing.)
Genehmigte Dissertation von Zhongxin Xia
Tag der Einreichung: 2. Oktober 2024, Tag der Prüfung: 26. November 2024

1. Gutachten: Prof. Dr.-Ing. Uwe Rüppel
2. Gutachten: Prof. Dr.ir. Eddie Koenders
Darmstadt, Technische Universität Darmstadt

TECHNISCHE UNIVERSITÄT DARMSTADT

Fachbereich Bau- und Umweltingenieurwissen-schaften

Institut für Numerische Methoden und Informatik im Bauwesen

Acknowledgements

I am deeply grateful to the Prof. Dr.-Ing. Uwe Rüppel and Prof. Dr.ir. Eddie Koenders for providing me with this fascinating doctoral research topic and for their guidance. In 2013, as a civil engineering undergraduate exchange student, I came to Germany to continue my studies in civil engineering. My undergraduate thesis focused on improving soil foundations using the technology of microbially induced calcium carbonate precipitation (MICP). I was deeply impressed by the incredible capabilities of these small bacteria. For thousands of years, concrete has been the most commonly used binding material in civil engineering. However, producing concrete consumes a lot of energy and generates a significant amount of carbon dioxide. Microbial-induced calcium carbonate can partially replace concrete to reinforce foundations and repair cracks in an environmentally friendly process. After graduating from college, I chose to work, but during my work, I never forgot the technology and felt that it had great application prospects. Therefore, after working for half a year, I decided to continue my postgraduate studies at the Karlsruhe Institute of Technology.

I intended to continue researching this technology during my graduate studies, but the professors there told me that MICP technology had no future and that they had stopped researching it. This response poured cold water on my enthusiasm for the research with MICP. During my graduate studies, artificial intelligence suddenly surged in popularity, becoming a highly sought-after technology. The impressive capabilities of artificial intelligence models ranging from robust image recognition and data analysis to strategic decision-making showcased in the game of Go have greatly impressed me. I am convinced that this technology will significantly impact the field of civil engineering. In the final year of my master's studies and the first two years of my PhD, I developed some applications that integrated machine learning models. These applications could analyze environmental data collected by edge computing devices like Raspberry Pi, Jetson Nano, and Arduino. I discovered that advanced machine learning models could function like intelligent assistants, effectively aiding users in efficiently addressing various engineering tasks, including predicting river flows, estimating housing prices, and detecting structural damage. I was particularly excited when Prof. Dr.-Ing. Uwe Rüppel and Prof. Dr.ir. Eddie Koenders proposed this PhD topic, which involved developing a robotic system that integrates MICP technology with artificial intelligence to achieve automated crack repair.

In the process of developing this robot, I learned a lot of new knowledge and gained much enjoyment. Furthermore, I realized that robots could integrate various automation and visualization technologies, and with the rapid advancement of artificial intelligence, robots will become increasingly powerful. It is entirely possible for them to replace workers and independently complete engineering tasks automatically. No matter how boring, dangerous, or hard the task is, as long as the algorithm is well-developed and the robot is fully charged, it will work continuously, regardless of wind, rain, or day and night, greatly improving construction efficiency. If these intelligent robots can achieve sufficient performance and stability, while remaining affordably priced, they are sure to gain significant market acceptance. It can be imagined that deploying a large number of robots on construction sites will be a revolution in civil engineering. If one day humans explore space and build habitats on other planets, intelligent robots will undoubtedly be the best labor force to build engineering facilities in harsh space environments.

Once again, I am deeply grateful to the two professors for providing this doctoral topic, which has opened up my research ideas and determined my future research direction. I will strive for this direction for the rest of my life.

Zhongxin Xia

Zusammenfassung

Beton ist eines der weltweit am häufigsten verwendeten Baumaterialien. Betonausrisse, ein häufig auftretendes Phänomen, können durch zahlreiche Faktoren in den verschiedenen Lebenszyklusphasen des Betons verursacht werden. Im Laufe der Zeit können unbehandelte, schmale Risse größer und tiefer werden, was die Struktur gefährdet. Die manuelle Erkennung und Reparatur von Rissen an großen Betonbauwerken ist jedoch sowohl zeitaufwendig als auch arbeitsintensiv. Besonders bei schmalen Rissen, die leicht übersehen werden und deren Reparatur mit herkömmlichen Methoden mühsam ist. Zudem sind die in traditionellen Techniken verwendeten Reparaturmittel nicht umweltfreundlich. Eine relativ neue Methode zur Reparatur schmaler Risse ist die Verwendung des umweltfreundlichen Biobetonmittels Basilisk ER7, das einfach anzuwenden ist und Betonausrisse bis zu 0,6 mm verschließen kann. Basilisk ER7 wird jedoch normalerweise durch Sprühen des Mittels auf die gesamte Betonoberfläche, um die Abdeckung schmaler Risse sicherzustellen, was zu einem hohen Verbrauch des Reparaturmittels führen kann. Wenn Arbeiter Risse manuell inspizieren und dann Basilisk ER7 auf jeden einzelnen Riss spritzen, könnte dies tatsächlich den Verbrauch des Reparaturmittels sparen. Dieser Prozess erfordert jedoch erheblichen manuellen Aufwand und Aufmerksamkeit, um jeden Riss genau zu identifizieren und zu behandeln.

Diese Doktorarbeit zeigt die Entwicklung eines kompakten und kostengünstigen Roboters namens „CrackRepairBot“, der speziell dafür entwickelt wurde, schmale Risse, Trocknungs- und Schrumpfrisse sowie Verschleiß auf flachen Betonoberflächen mit dem Reparaturmittel Basilisk ER7 zu reparieren. Unterstützt durch Computer-Vision-Techniken wie Bildsegmentierung und Bildverarbeitung mittels des OpenCV-Tools kann der Roboter Risse schnell erkennen, zwischen schmalen und breiten Rissen unterscheiden und das Reparaturmittel gezielt nur auf die schmalen Risse spritzen. Im Vergleich zur üblichen Anwendungsmethode des Sprühens auf die gesamte Fläche hilft diese Funktion, das Reparaturmittel zu sparen. Da schmale und breite Risse oft zusammen auftreten und das Reparaturmittel Basilisk ER7 nur für schmale Risse geeignet ist, wurde in dieser Arbeit auch ein automatisches Aufzeichnungs- und Visualisierungssystem für breite Risse entwickelt, um Ingenieuren zu ermöglichen, in Zukunft andere Methoden zur Reparatur dieser breiten Risse zu ergreifen.

Das wichtigste Merkmal des in dieser Arbeit entwickelten robotischen Systems ist die Verteilung verschiedener Funktionsmodule auf verschiedene Edge-Computing Module, die über kabelgebundene Verbindungen kommunizieren. Der leistungsstarke Einplatinencomputer, der als „Gehirn“ des gesamten Systems fungiert, wird hauptsächlich zur Bereitstellung des maschinellen Lernmodells und zur Echtzeit-Inferenz dieses Modells verwendet. Die Inferenz-Ergebnisse des maschinellen Lernmodells werden an den vorhandenen Einplatinencomputer und Mikrocontroller des Roboters gesendet, die dann die Bewegungen des Roboters, die Rissaufzeichnung und das Sprühsystem steuern. Die in diesem Robotersystem verwendeten Algorithmen basieren auf Open-Source-Frameworks. Ein solches Systemdesign stellt sicher, dass das gesamte System nicht übermäßig von Hardware einer bestimmten Marke abhängig ist, was den Austausch der Hardware in Zukunft je nach Bedarf neuer Anwendungsszenarien erleichtert. Mit anderen Worten, dieses Systemdesign erhöht die Flexibilität und Anpassungsfähigkeit des Systems, sodass neue Edge-Computing Geräte nach Bedarf leichter integriert und aufgerüstet werden können, ohne andere Geräte im Netzwerk wesentlich zu beeinträchtigen. Da der leistungsstarke Einplatinencomputer den Großteil der Rechenanforderungen des Systems übernimmt, können der Roboter und der Mikrocontroller in kostengünstigen, rechenarmen Versionen ausgewählt werden, was die

Gesamtkosten für die Roboterentwicklung senkt. Der in dieser Arbeit verwendete Roboter ist gemäß dem Systemdesign der relativ kostengünstige und Open-Source JetAuto Pro. Jetson Orin Nano verwendet die Ergebnisse des Bildsegmentierungsalgorithmus YOLOv8, um Arduino und Jetson Nano zu steuern. Wenn Jetson Orin Nano über eine 10-fach zoombare Kamera einen Betonausriss von bis zu 0,6 mm Breite erkennt, sendet es einen Befehl an Arduino, das normalerweise geschlossene Magnetventil zu öffnen und das Reparaturmittel aus einer druckbeaufschlagbaren Wasserflasche auf den Riss zu sprühen. Überschreitet die Breite des Risses 0,6 mm, erteilt Jetson Orin Nano über ein Ethernet-Kabel Befehle an Jetson Nano, das sofort die Kamera verwendet, um ein Bild des breiten Risses aufzunehmen und über das Robot Operating System (ROS) Koordinaten aus der Karte zu erhalten, um diese Koordinaten und das Bild in MongoDB Atlas zu speichern. Die Positionen dieser Risse und ihre entsprechenden Bilder werden in einer integrierten Webanwendung für Building Information Modeling (BIM) und Geographic Information Systems (GIS) auf der Basis der 3D-Geoplattform Cesium visualisiert.

Um eine autonome Navigation zu erreichen, wurden die vom Roboter verwendeten Karten aus dem IFC (Industry Foundation Classes) Modell und mit einer Drohne aufgenommenen Fotos konvertiert. Anschließend nutzt der Roboter das LiDAR und die durch Python-Skripte gesetzten Mehrfachzielpunkte, um den gesamten Boden umfassend zu überprüfen. Die durch Python-Skripte gesetzten Zielpunkte können auch für die Routenplanung während der Aushärtungsphase des Reparaturmittels verwendet werden, sodass der Roboter seinen vorherigen Weg erneut verfolgt, um Wasser zur Feuchtigkeitswartung der Risse zu sprühen.

Der Roboter führte zunächst spezifische Funktionstests an rissigen Betonen und auf dem Boden in Innenräumen durch. Anschließend wurde ein umfassender Test auf der Betonoberfläche eines Skateparks durchgeführt, bei dem alle Funktionen des Roboters effektiv arbeiteten. Das am besten trainierte YOLOv8-Modell klassifiziert jedoch gelegentlich bestimmte Objekte fälschlicherweise als Risse, und seine Leistung muss noch weiter verbessert werden. Diese Arbeit zeigt, dass intelligente Roboter Arbeiter bei der Durchführung von zeitaufwendigen und arbeitsintensiven Aufgaben wie der Reparatur schmaler Risse und der Erfassung breiterer Risse unterstützen können, wodurch die Arbeitskosten bei Bauprojekten gesenkt werden. Es zeigt sich, dass die Entwicklung von Robotern im Bereich des Bauingenieurwesens großes Anwendungspotenzial hat. In Zukunft zielt diese Arbeit darauf ab, das entwickelte System auf Drohnen für die Risserkennung und -reparatur in Hochhäusern einzusetzen. Um die Einschränkungen der Verallgemeinerungsfähigkeit des YOLOv8-Modells zu überwinden, ist geplant, fortschrittlichere Algorithmen zu verwenden, um eine genaue Risserkennung zu erreichen.

Abstract

Concrete is one of the most widely used building materials globally. Concrete cracks, a common occurrence, can result from numerous factors across the life cycle stages of concrete. With time, unrepaired narrow cracks may enlarge and get deeper, undermining the structure. But manual detection and repair of cracks by large concrete structures are both time-consuming and labor-intensive. Especially for narrow cracks, which are easily overlooked and cumbersome to repair using traditional methods. Moreover, the repair agents employed in traditional techniques are not environmentally friendly. A relatively new method for repairing narrow cracks is the use of the environmentally friendly bio-concrete agent Basilisk ER7, which is easy to use and can seal concrete cracks up to 0.6 mm. However, Basilisk ER7 is generally applied by spraying the agent onto the entire concrete surface to ensures the coverage of narrow cracks, it can be quite wasteful of the repair agent. If workers manually inspects for cracks first and then applies the Basilisk ER7 to each crack individually, it could indeed save on the amount of repair agent. But this process requires significant manual effort and attention to accurately identify and treat each crack.

This doctoral dissertation demonstrates the development of a compact and low-cost robot named "CrackRepairBot", specifically designed for repairing narrow cracks drying- and shrinkage cracking and wear on flat concrete surfaces using repair agent Basilisk ER7. Assisted by computer vision techniques such as image segmentation and image processing via OpenCV tool, the robot can swiftly identify cracks, distinguish between narrow and wide ones, and then apply the repair agent exclusively to the narrow cracks. Compared to the common application method using entire area spraying, this feature helps to save the repair agent. Since narrow cracks and wide cracks often occur together, and the repair agent Basilisk ER7 is only suitable for narrow cracks, this work has also developed a automatic recording and visualization system for wide cracks to facilitate engineers to take other methods to repair these wide cracks in the future.

The most significant feature of the robotic system developed in this work is the distribution of various functional modules across different edge computing modules, which achieve communication through wired connections. The high-performance single-board computer, acting as the "brain" of the entire system, is primarily used for deploying machine learning model and achieving real-time inference of this model. The inference results from the machine learning model are sent to the robot's existing single-board computer and microcontroller, which then control the robot's movements, crack recording and spraying system. The algorithms used in this robot system are based on open-source frameworks. Such a system design ensures that the entire system is not overly dependent on hardware from a specific brand, facilitating the replacement of hardware in the future based on the needs of new application scenarios. In other words, this system design enhances the system's flexibility and adaptability, allowing for easier upgrades and integration of new edge computing devices as needed without significantly affecting other devices within the network. Moreover, since the high-performance single-board computer takes on the vast majority of the system's computational demands, the robot and microcontroller can be selected for cost-effective, low-computational versions, thereby reducing the overall development cost of the robot.

According to the system design, the robot used in this work is the relatively inexpensive and open-source JetAuto Pro. Jetson Orin Nano uses the results deduced from the image segmentation algorithm YOLOv8 to control the Arduino and Jetson Nano. When the Jetson Orin Nano detects a concrete crack up 0.6 mm wide via 10x zoomable camera, it will send commend to Arduino to open the normally closed solenoid water valve and allow the repair

agent to spray onto the crack from a pressurizable water bottle. If the crack's width exceeds 0.6 mm, the Jetson Orin Nano will issue commands to Jetson Nano via an Ethernet cable, and Jetson Nano will immediately use the camera to take a picture of the wide crack and obtain coordinates from the map through the Robot Operating System (ROS), and then store the coordinates and picture in MongoDB Atlas. The locations of these cracks and their corresponding pictures will be visualized in a Building Information Modeling (BIM) and Geographic Information Systems (GIS) integrated web application based on 3D geospatial platform Cesium.

To achieve autonomous navigation, the maps used by robot were converted from IFC (Industry Foundation Classes) model and photo taken by drone. Then, the robot will utilize the LiDAR and the multiple target points set by Python script to comprehensively cover and check the entire ground. The target points set by Python scripts can also be used for route planning during the curing phase of repair agent, allowing the robot to retrace its previous path to spray water for humidity maintenance of cracks.

The robot initially performed specific functional tests on cracked concrete samples and indoor floor. Subsequently, comprehensive test was conducted on the concrete surface of a skate park, where all functionalities of the robot operated effectively. However, the best-trained YOLOv8 model occasionally misclassifies certain objects as cracks, and its performance still requires further improvement. This work demonstrates that intelligent robot can assist workers in completing time-consuming and labor-intensive task such as narrow crack repair and wider crack recording, thereby reducing labor costs in construction project. It can be seen that the development of robots in the field of civil engineering has significant application potential. In the future, this work aims to deploy the developed system on drones for crack detection and repair in high-rise buildings. To overcome the limitations in the generalization ability of the YOLOv8 model, more advanced algorithms are planned for use to achieve accurate crack detection.

Table of Contents

Acknowledgements

Zusammenfassung

Abstract

Table of Contents

1. ..... Introduction 1
   1.1. Motivation and Problem Definition 1
   1.2. Objective 2
   1.3. Structure and Methodology of the Work 3

2. ..... Repair of Narrow Cracks in Concrete 5
   2.1. Types of Concrete Crack 5
   2.2. Traditional Methods for Narrow Crack Repair 6
   2.3. Narrow Cracks Repair using Bio-Concrete Agent 9
      2.3.1. Principle of MICP 9
      2.3.2. Experimental Exploration 10
      2.3.3. Instructions of Basilisk ER7 15
   2.4. The Challenges of Narrow Crack Repair 17

3. ..... Technical Basis of the System 19
   3.1. Machine Learning for Crack Recognition 19
      3.1.1. Convolutional Neural Network 20
      3.1.2. Training Model with Transfer Learning 23
      3.1.3. Underfitting and Overfitting 23
      3.1.4. Evaluating Performance of Model 25
   3.2. Edge Computing Devices for Robot 26
      3.2.1. Jetson Platform 27
      3.2.2. Arduino 31
      3.2.3. Other Edge Computing Devices 32
   3.3. Robotic Technology 34
      3.3.1. Robot Operating System 34
      3.3.2. Robot Mapping 38
      3.3.3. Robot Localization 39
      3.3.4. Related Robots 41
   3.4. BIM-GIS Integration using Cesium 45

4. ..... Concept of AI-Based Robot CrackRepairBot 48
   4.1. The Specific Application Method of Basilisk ER7 48

4.2. Overall Concept of the Robot 50
4.3. Independence and Flexibility of Robot 51
4.3.1. Open-Source Robotic Platform 51
4.3.2. Open-Source Algorithms and Software Frameworks 52
4.3.3. Distributed Control System Architecture 53
4.4. Image Segmentation for Cracks Filtering 54
4.4.1. YOLO Network 54
4.4.2. Live Image Quality of the Camera 56
4.4.3. Calculation of Maximum Crack Width 57
4.4.4. Distinguishing between Narrow and Wide Cracks 59
4.5. Solutions for Rapidly Spraying Adequate Amounts of Agent 59
4.5.1. Water Pump 62
4.5.2. Water Valve and Pressurizable Water Bottle 63
4.5.3. Checking Flow Rate 64
4.6. Navigation Method for Cracks Inspection 65
4.6.1. Creating Map from 3D Model and Image 65
4.6.2. Path Planning 66
4.7. Recording and Visualizing Wide Cracks 69
4.7.1. System Overview 69
4.7.2. Crack Localization and Visualization 70
5. ..... Implementation of AI-Based Robot CrackRepairBot 72
5.1. Building the Robot 72
5.1.1. Hardware Selection 72
5.1.2. Prototype 77
5.1.3. System Connection 78
5.2. Building the Map for Robot 81
5.3. Path Planning using Python Script 85
5.4. Crack Segmentation using YOLO Model 86
5.4.1. Training YOLO Model using Transfer Learning 86
5.4.2. Deploying the Trained Model in Jetson Orin Nano 89
5.5. Determining Crack Width 93
5.5.1. Crack Segmentation Extraction 93
5.5.2. Real-Time Calculation of the Maximum Width 94
5.5.3. Using Suitable Camera 96
5.6. Control of the Spraying System 99

5.6.1. Controlling the Water Valve via Arduino 99
5.6.2. Testing Spraying Capacity 100
5.7. Recording Wide Crack by Onboard Computer 102
5.7.1. Frequency Control of the Command from Jetson Orin Nano 102
5.7.2. Reading and Converting Coordinates 103
5.7.3. Camera Warm-Up 105
5.7.4. Storing Data in MongoDB Atlas 106
5.8. Visualizing Wide Cracks in BIM-GIS Web-Application 107
5.8.1. IFC to 3D Tiles 107
5.8.2. Color and Visibility Settings of Surface 109
5.8.3. Cracks Visualization 110
5.8.4. Additional Features 112
6. ..... Evaluation and Application Example 116
6.1. Practical Specific Tests for Certain Functions 116
6.1.1. Test on Concrete Samples with Crack 116
6.1.2. Test Indoors 119
6.2. Application Example 122
7. ..... Summary and Outlook 127
7.1. Conclusion 127
7.2. Future Work 128
7.2.1. Deployment on Drone 128
7.2.2. Integrating Multimodal AI Model 130
List of Abbreviations 135
List of Tables 136
List of Figures 136
Bibliography 143

1. Introduction

On one hand, this work presents the development of a compact, low-cost robot, designed specifically for repairing narrow cracks due to drying- and shrinkage cracking and wear on flat concrete surfaces using environmentally friendly bio-concrete agent and crack segmentation algorithm. On the other hand, system and web application have also been developed in this work to record and visualize wide cracks, facilitating engineers to subsequently take measures to repair these wide cracks.

1.1. Motivation and Problem Definition

Concrete is one of the most widely used building materials worldwide, valued for its durability, strength, and versatility. It is employed in a wide range of construction projects, including buildings, bridges, roads, and dams. The history of concrete spans thousands of years (Moropoulou et al., 2005). The material that closely resembles modern concrete was first developed by Joseph Aspdin in 1824, who invented Portland cement by burning finely ground chalk and clay until carbon dioxide was removed. This invention marked a significant milestone in the history of concrete and led to its widespread use in construction. François Coignet's use of steel rods within concrete was also a pivotal step in the evolution of construction materials, allowing for more robust and resilient structures. Cracks in concrete are common; they can be caused by a variety of factors, each related to different stages of the concrete life cycle, from mixing and pouring to hardening and usage. For instance, over time, concrete naturally ages and undergoes wear and tear. Certain chemicals in the environment, such as sulfates, can react with concrete components.

If concrete cracks are not repaired in time, several consequences can arise. Over time, narrow cracks can grow larger, compromising construction. These cracks provide pathways for the penetration of chlorides, water, and oxygen, leading to rust and corrosion of steel. As corrosion continues, the expansion of rust can lead to significant cracking and spalling of the concrete cover, exposing more rebar to the corrosive environment and eventually compromising the structural integrity of the concrete, as shown in Figure 1-1.

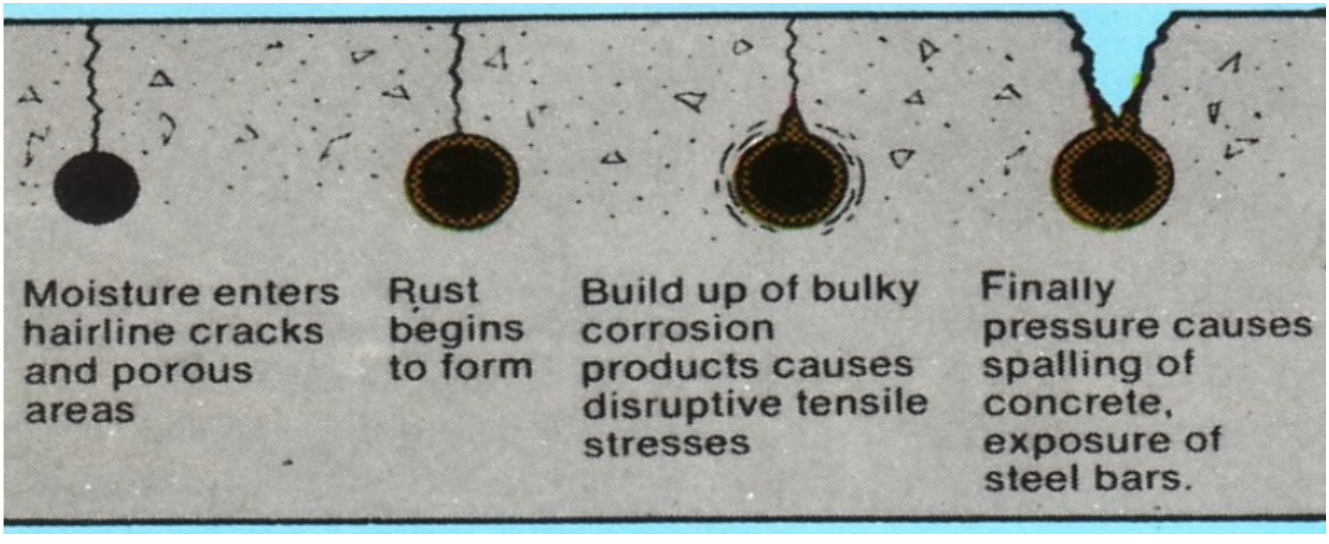


Figure 1-1: Corrosion of steel reinforcement cracks (Happho, 2024)

A lot of construction projects around the world are constructed using concrete, timely and effective crack repair is crucial to maintaining structural integrity, safety, and longevity, which is why there is a consistent demand for reliable crack repair solutions. There are several methods for repairing narrow concrete cracks, each suited to different types of cracks and their severities. The materials commonly used in narrow concrete crack repair methods, such as epoxy resins, polyurethane sealants, and concrete itself, can have environmental drawbacks. There is growing interest in developing more sustainable and eco-friendly alternatives that reduce the reliance on such materials. Bio-concrete represents an innovative and environmentally friendly approach to addressing the problems associated with concrete cracking and the environmental impact of traditional repair materials. In early 2010, microbiologist Henk Jonkers from Delft University came up with the solution of a self-healing concrete product (Jonkers et al., 2010). The Basilisk ER7 is a bio-concrete repair agent based on study of Henk Jonkers particularly suitable for sealing narrow cracks due to drying- and shrinkage cracking and wear in horizontal concrete structures. This repair agent, due to its low viscosity, simply needs to be sprayed onto the surface of the crack. It will penetrate into the interior of the crack and seal it by inducing the formation of calcium carbonate within the crack. According to the latest product data sheet of Basilisk ER7, the agent can seal (static) concrete cracks in horizontal surfaces up to 0.3 mm in one treatment. The widest (static) concrete cracks in horizontal surfaces that this repair agent can repair is 0.6 mm, which requires a minimum of 3 treatments (Basilisk, 2022). Relative to the traditional method of extruding and injecting epoxy resin directly into the cracks, this repair agent is very convenient to use. However, narrow cracks on the ground are often so fine that they are not easily noticeable to the naked eye and can be easily overlooked. If one wants to manually spray each crack with a nozzle, it would be very time-consuming and labor-intensive. Therefore, in practice, especially in large buildings, it is common to use a method of entire area spraying to cover the entire concrete surface that requires repair. This approach is wasteful of the repair agent, and in reality, cracks vary in size and even with area spraying, the agent is ineffective for wide cracks. Hence, this work will involve the development of an intelligent robot that can automatically detect narrow cracks on the ground and then apply the bio-concrete agent onto those narrow cracks. This would not only save time and effort but also conserve the repair agent.

1.2. Objective

This doctoral dissertation presents the development of a compact and low-cost robot named “CrackRepairBot”, specifically designed to repair narrow cracks due to drying- and shrinkage cracking and wear on flat concrete surfaces using an environmentally friendly bio-concrete agent. Since narrow cracks and wide cracks often occur together, and the bio-concrete agent is only suitable for narrow cracks. Therefore, computer vision techniques such as image segmentation and image processing via OpenCV tool are applied in the robot to screen out narrow and wide cracks. The robot only sprays repair agent onto the narrow cracks to repair them, which saves the use of repair agent compared to manual labor and entire area spraying method. For wide cracks, the robot will record the cracks and visualize them on the web application that integrates BIM (Building Information Modeling) and GIS (Geographic Information System) technologies for subsequent repair using other methods. Such application leverages the strengths of BIM for detailed structural information and GIS for geographical context, offering a comprehensive platform for the management and visualization of structural health data. This integration allows engineers to not only identify and measure cracks but also to analyze their impact on the structural integrity over time, facilitating more informed maintenance and repair decisions.

A notable feature of the CrackRepairBot is the distribution of various functional modules across different edge computing units. The high-performance single-board computer, serving as the "brain" of the system, is used primarily for deploying machine learning model and achieving real-time inference. The results from the machine learning inference are transmitted to the robot's existing single-board computer and microcontroller, which then manage the robot's movements, wide crack recording and spraying system operations. And the algorithms employed in this robotic system are based on open-source frameworks. This system design enhances the system's flexibility and adaptability, allowing for easier upgrades and integration of new edge computing devices as needed without significantly affecting other devices within the network. Additionally, since the high-performance single-board computer handles the majority of computational tasks, cost-effective, low-computation microcontroller and robot platform can be utilized, thereby reducing the overall development costs of the robot. Figure 1-2 shows the overview of the objective in this work.

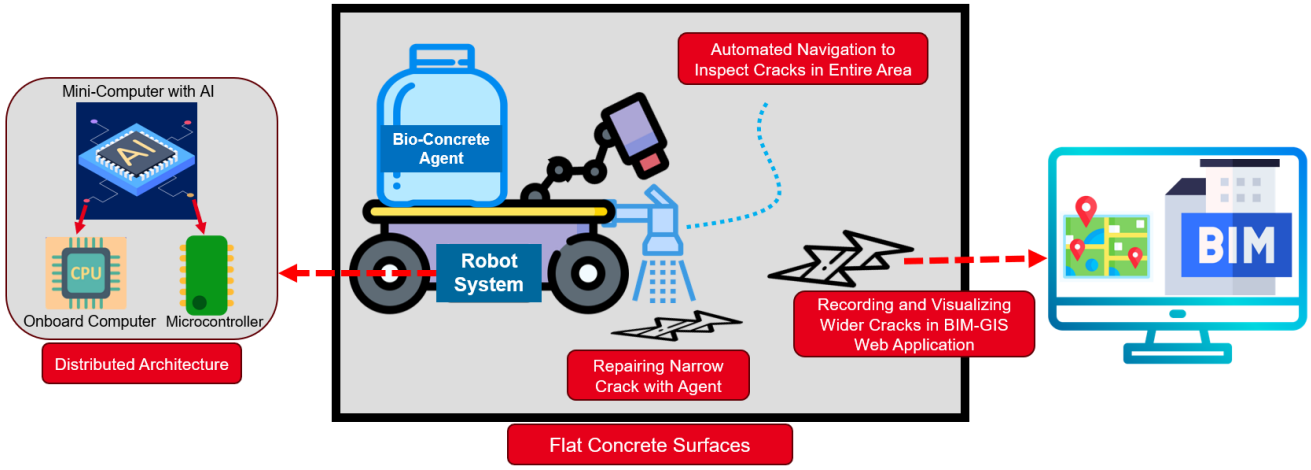


Figure 1-2: System overview of the robot and the web application for crack repair and recording

1.3. Structure and Methodology of the Work

The research approach used in the work is based on the Design Science Research Methodology (DSRM). The DSRM is a research approach that aims to create innovative solutions for practical problems through the integration of design and scientific methods. It is particularly relevant in contexts where existing solutions are insufficient or non-existent, and there is a need for new, more effective approaches to solving complex problems. The DSRM process typically involves an iterative cycle of design, development, and evaluation, with the goal of creating new artifacts that solve real-world problems (Peffers et al., 2007).

Figure 1-3 shows the DSRM process and the corresponding chapters in this thesis. This thesis mainly includes 7 parts. In chapter 1, the motivation and problem are described, as well as the objective determined therefrom. As a first step in the DSRM process, the basics of narrow concrete crack repair will be analyzed through the common types of concrete cracks, traditional and bio-conrete agent based methods for narrow cracks repair in concrete. And thus, the challenges of repairing narrow cracks are identified. The end of this chapter also introduces a preliminary concept for repairing narrow concrete cracks. In the second step of the DSRM process, the specific principles and requirements for the respective problem are presented in chapter 3. The principles of machine learning algorithms used for crack recognition will be explained in detail. The development of robot is a key focus of this work, and therefore, the edge computing devices and the robotic technologies will be introduced. Several

comparable robots designed for crack recognition will also be reviewed as a reference for the robot developed in this work. Finally, a visualization platform integrating BIM and GIS will be presented. These technologies are introduced to address the challenges in repairing narrow cracks, as outlined in Chapter 2. The aim is to combine these technologies to develop a specialized robot tailored for the challenges. Chapter 4 builds on the objectives set in Chapter 1 and, integrating the technologies outlined in Chapter 3, develops the conceptual framework for the robot's hardware and software systems, along with the technical roadmap for crack visualization. As this work aims to develop a robot that utilizes the Basilisk ER7 repair agent, this chapter begins by analyzing the specific characteristics of the agent to derive the essential functionalities the robot must incorporate. Next, a comprehensive schematic to define the overall technical strategy is proposed, followed by a detailed breakdown of the implementation of each aspect of the design. Chapter 5 focuses on turning the design concepts from Chapter 4 into practical applications. It begins with selecting the appropriate hardware for the robot, based on project requirements, and establishing the control and communication systems. The chapter then details the methods for map generation and path planning. For crack inspection, the YOLO neural network is employed to segment cracks. The width of the cracks is measured through image segmentation and OpenCV, allowing the differentiation between narrow and wide cracks. For narrow cracks, an Arduino-controlled water valve with pressurizable water bootle is used to spray the repair agent, while for wide cracks, an automatic system is developed to record crack information. The chapter concludes by demonstrating how a web application, integrating BIM and GIS, is used to visualize the wide cracks. Chapter 6 focuses on testing the functionalities of the robot. Initially, each subsystem is evaluated in a controlled environment to determine whether it operates according to the design specifications. Subsequently, the robot's complete functionality is tested on a concrete surface with cracks, serving as a validation of the system's overall reliability. The final chapter provides a summary of the research and offers prospects for future work. Beyond its capabilities of repairing and recording cracks, the robot can be further enhanced to perform a variety of additional functions, making it applicable to other engineering tasks.

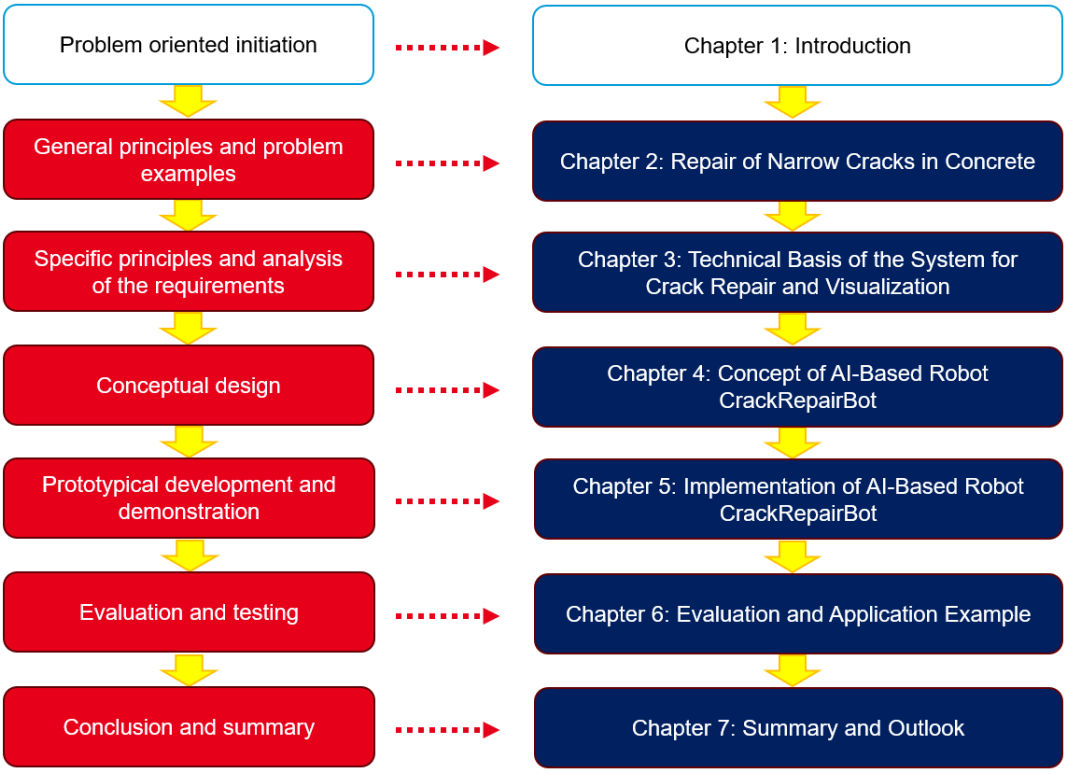


Figure 1-3: Steps of the DSRM process model (left) in the structure of the work (right)

2. Repair of Narrow Cracks in Concrete

This chapter categorizes different types of cracks and lists conventional techniques for repairing narrow cracks. It also details the working mechanisms and practical applications of bio-concrete repair agent. The chapter concludes with a comparison of both traditional and bio-concrete agent based repair methods, emphasizing their limitations and thus outlining the preliminary objectives of this work.

2.1. Types of Concrete Crack

Crack formation in concrete is inevitable over time due to a variety of factors, including environmental influences, design imperfections, and construction-related problems. According to Airports Council International (ACI) committee 224, cracking can also be classified into two main categories based on the time of the crack occurrence, i.e., cracks occurring before concrete hardening (plastic stage) or after concrete hardening (ACI Committee 224, 2007).

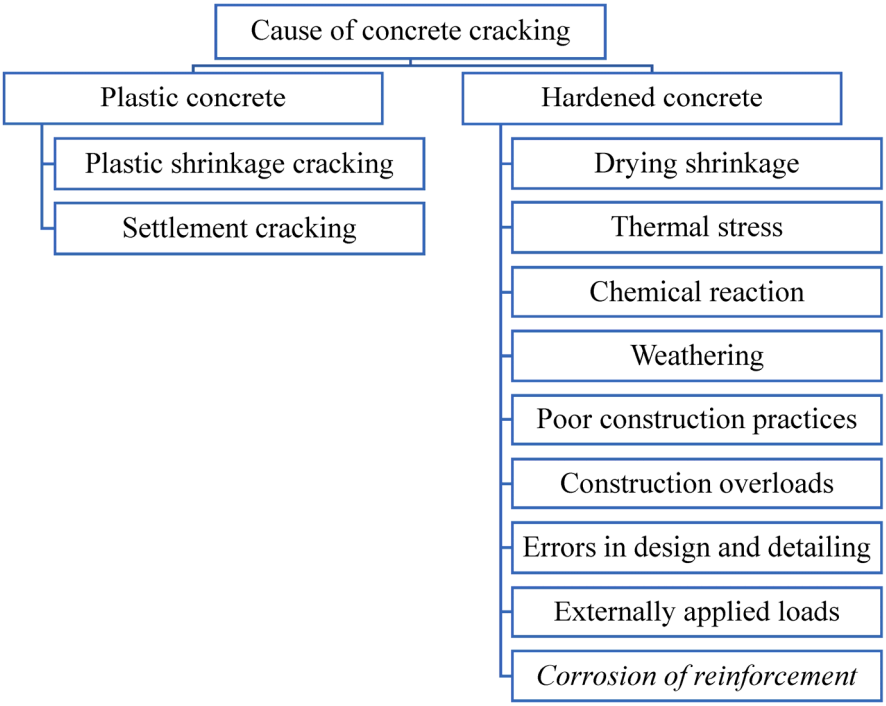


Figure 2-1: Causes of concrete cracking according to ACI committee 224 (Poursaee and Ross, 2022)

The cracking phenomena can be further subdivided. For example, cracking due to applied loads can be characterized as flexural, shear, or torsional cracks. Cracking can also be characterized by size, location, and severity. Internal microcracking, or non-structural cracks, result from the intrinsic properties of concrete and its ingredients. Structural cracks in concrete impact the stability and integrity of a structure. They usually result from excessive loads, poor construction practices, or foundation settlement. These cracks are more significant and can pose serious risks if left unaddressed. They often appear wide, and deeper and are more irregular in shape compared to non-structural cracks. Structural cracks demand immediate attention and professional assessment to determine their root cause and necessary remedial actions. Non-structural cracks in concrete generally occur only

on the surface due to factors like minor shrinkage during curing, temperature changes, or surface drying. These cracks are narrow, affect the appearance and may widen or get deeper over time.

Narrow cracks in concrete are a common occurrence and can result from various factors. Here are some common types of narrow cracks that can appear in concrete structures:

1. **Shrinkage Cracks**: They are caused by the reduction in volume of the concrete as it dries and cures. They are typically random in pattern and can appear on the surface of the concrete.
2. **Drying Shrinkage Cracks**: They are caused by the loss of moisture during the drying process, leading to volume reduction and the potential for cracking.
3. **Hairline Cracks**: They are extremely thin cracks that are often hair-width or finer. They can be caused by a variety of factors including shrinkage, settlement, or minor movement in the concrete structure.
4. **Temperature Cracks**: They occur due to fluctuations in temperature, causing the concrete to expand and contract. These cracks often form as the concrete cools after being placed on a hot day or during seasonal temperature changes.
5. **Thermal Stress Cracks**: These cracks are a result of thermal stress within the concrete due to sudden temperature changes or exposure to extreme heat or cold.
6. **Settlement Cracks**: They occur when concrete settles unevenly due to improper compaction or variations in the supporting soil.
7. **Corrosion-Induced Cracks**: These cracks are a result from the expansion of corroded reinforcing steel within the concrete, creating internal pressures that crack the concrete.
8. **Microcracks from Overloading**: They develop when concrete is subjected to loads beyond its capacity.
9. **Freeze-Thaw Cracks**: They are caused by the expansion of water within the concrete pores during freezing conditions, leading to internal stresses.
10. **Structural Cracks**: Although narrow, these cracks can indicate a significant structural issue. They are caused by excessive stress or load on the concrete structure.

2.2. Traditional Methods for Narrow Crack Repair

If narrow concrete cracks are not promptly repaired, it can lead to several issues. Narrow Crack repair maintains the structural integrity and aesthetics of concrete structures. There are several common narrow crack repair methods such as epoxy injection, polyurethane injection, routing and sealing, patching, dry pick and overlays treatment.

Epoxy injection is an economical method of repairing narrow, non-moving cracks in concrete walls, slabs, columns and piers and is capable of restoring the concrete to its pre-cracked strength as it has a bond strength even higher than that of concrete. Epoxy flows easily and can penetrate into narrow cracks and fill the deep ends of the crack under high pressure. The injection process is minimally invasive, requiring only small holes to be drilled for injection. Epoxy has a relatively quick cure time, allowing for rapid crack repairs. It works exceptionally well on concrete, wood, and metal surfaces and can be applied on most interior and exterior surfaces. Figure 2-2 shows the steps to repair cracks by epoxy injection: (1) Attaching the injection nipples along the crack. (2) Sealing the crack to prevent the epoxy compound from leaking out before it has gelled. (3) (4) Injecting the epoxy into the crack.

Figure 2-2: Steps to repair cracks by epoxy injection (Strong-Tie Company, 2024)

Polyurethane injection works well in repairing non-structural cracks in wet and dump areas. Polyurethane expands about 2-40 times its original volume. It is ideal for filling and sealing concrete cracks completely. Its fast-setting property makes it suitable for repairing cracks in wet areas. Its elastomeric nature makes it suitable for moving cracks, as it can expand to fill new voids. Polyurethane injection is recommended for actively leaking cracks due to its watertight sealing properties. Crack repair using polyurethane injection follows the same procedure as epoxy injection. As shown in Figure 2-3 (a), 0.5 inch injection packers were placed roughly 4 inch from the crack in 0.5 inch holes that were drilled at a 45-degree angle back into the crack. Polyurethane was injected until positive refusal of the hydro-active resin was visible from the face of the crack as well as on the exterior side of the wall when a through crack was injected, as shown in Figure 2-3 (b).

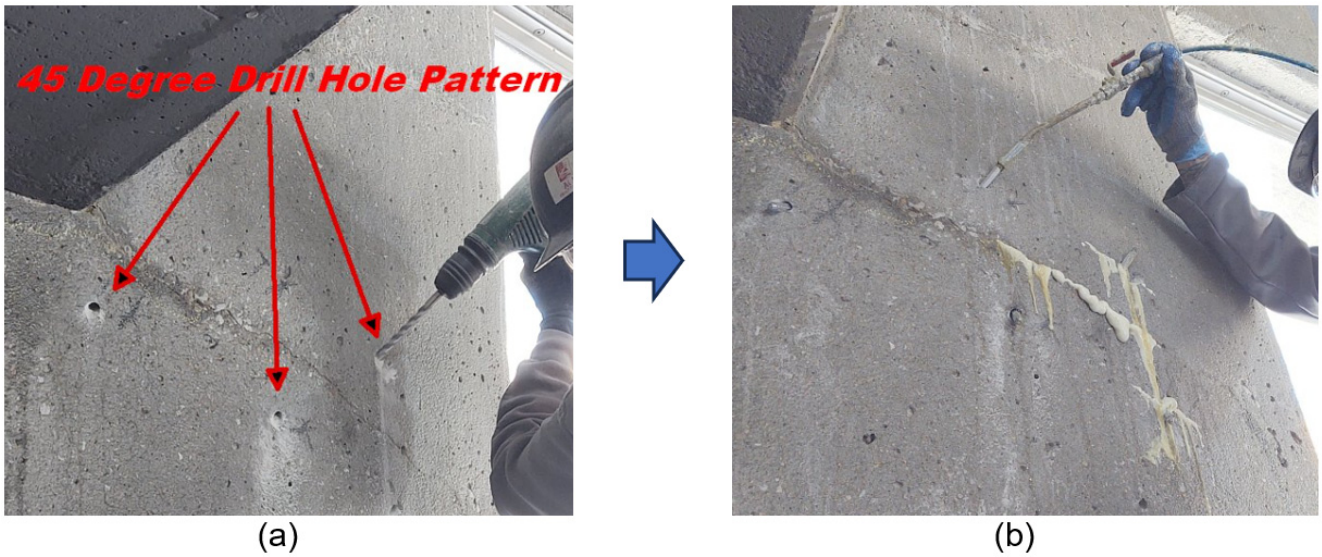


Figure 2-3: Steps to repair cracks by polyurethane injection (SealBoss, 2024)

Routing and Sealing can be used in conditions requiring remedial repair and where structural repair is not necessary. This method involves enlarging the crack along its exposed face and filling and sealing it with a suitable joint sealant. This is a common technique for crack treatment and is relatively simple. The procedure is most applicable to approximately flat horizontal surfaces such as floors and pavements. Concrete crack repair sealants are made of silicones, epoxies, polysulfides, and asphaltic materials. The procedure of routing and sealing involves these steps: (a) Prepare a groove along the crack using tools, (b) Clean the groove, allowing the surface of the groove to dry, (c) Apply sealant generously to the groove and allowing it to cure.

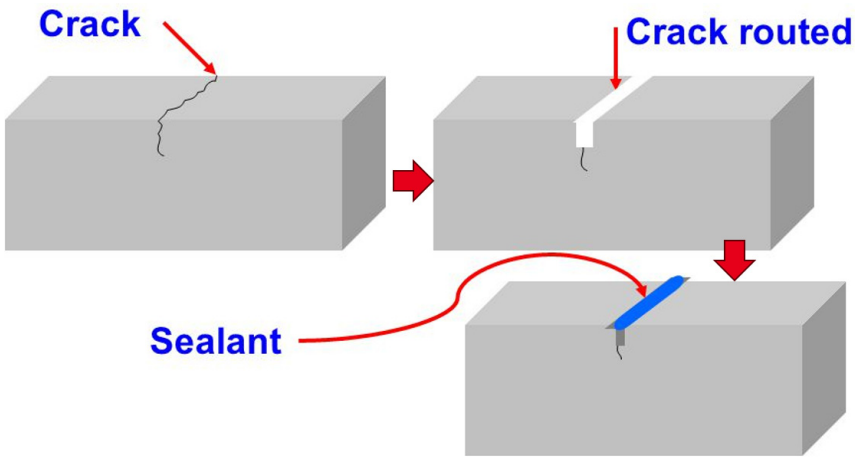


Figure 2-4: Steps to repair cracks by routing and sealing (Goodwin, 2009)

Patching is one of the simplest methods of hairline crack repair. Man can use patching compounds like vinyl concrete patches or hydraulic cement to cover hairline cracks in concrete. Using non-shrink concrete grout whose volume does not change when dried gives the best results. The procedure for patching hairline cracks involves these steps: (a) Dislodge loose concrete and remove debris, (b) Hose down the stairs to help the patching compound bond, (c) Place a wood plank against the cracked step to create a frame, (d) Mix a dry concrete vinyl patch with water, if necessary, (e) Use a trowel to apply the vinyl concrete patcher, (f) Remove the wood plank once the patch has set.

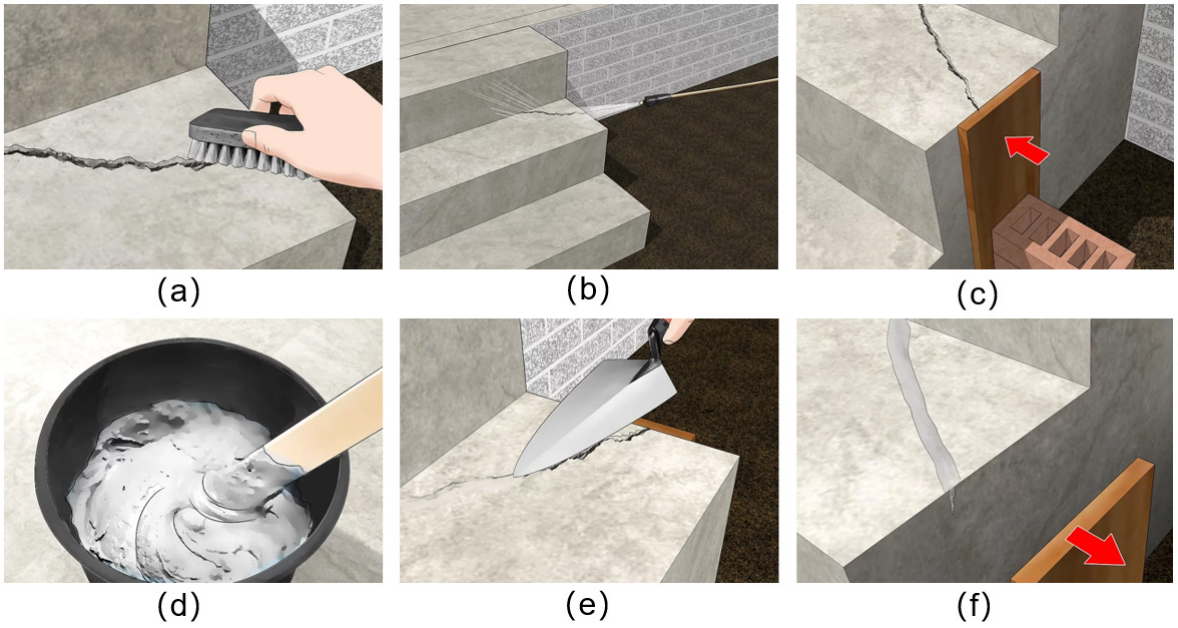


Figure 2-5: Steps to repair a crack wide than 0.64 cm in stair with a vinyl concrete patcher (wikiHow, 2023)

Dry Pack is a method used for repairing concrete surfaces, particularly for filling narrow slots cut for the repair of dormant cracks. It involves the hand placement of a low water content mortar, which is then compacted to produce a tight bond with the existing concrete. This method is known for its effectiveness in producing durable, strong, and water-tight repairs, especially when the cracks are no longer active and have stabilized. As shown in Figure 2-6, repairing cracks by dry pack include these steps: (a) Widen cracks to a slot and clean the cracks thoroughly, (b) Pouring in dry mix concrete, (c) Spray a small amount of water for curing to ensure the mortar gains strength and durability.

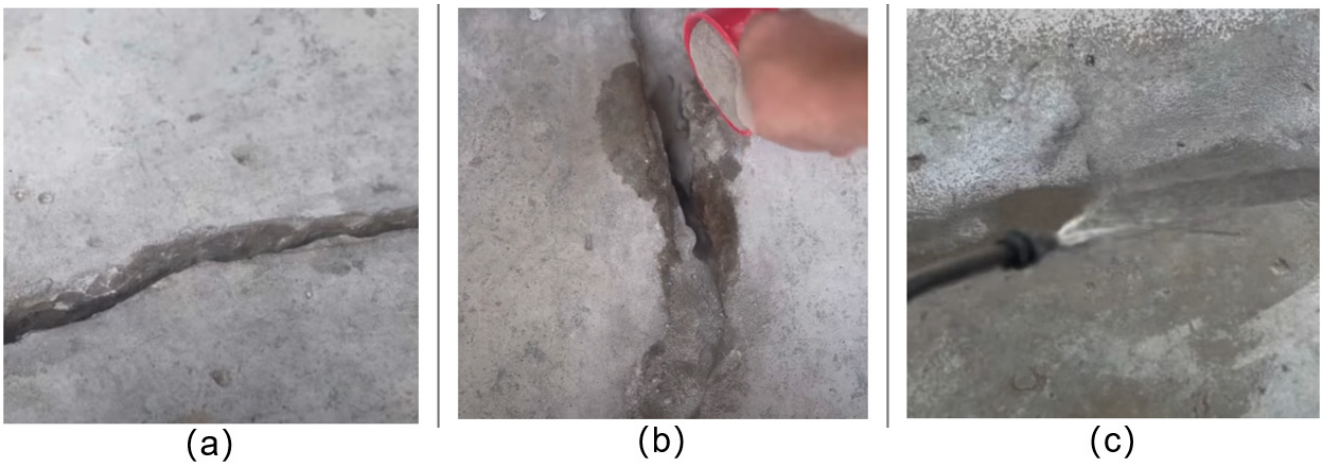


Figure 2-6: Steps to repair cracks by dry pack (Jovi, 2023)

Overlays Treatment can be used to repair fine surface cracks in structural slabs and pavements if there will not be further significant movement across the cracks. Overlays involve applying a new layer of material over the existing concrete surface. This can be done with various materials such as polymer-modified Portland cement mortar or concrete, silica fume concrete, or even asphalt and asphalt concrete. Overlays can be bonded, partially bonded, or unbonded to the existing concrete. They are used to seal areas with a large number of cracks where individual treatment would be too expensive. Overlays can also provide a new wearing surface, protection against water and chloride ion intrusion, and can be decorative.

2.3. Narrow Cracks Repair using Bio-Concrete Agent

Narrow cracks repair using bio-concrete agent is based on Microbiologically Induced Calcium Carbonate Precipitation (MICP) technology.

2.3.1. Principle of MICP

MICP is a natural biogeochemical process that involves the precipitation of calcium carbonate ($CaCO_3$) as a result of microbial activities. This process is facilitated by various microorganisms, which can catalyze the precipitation of calcium carbonate through their metabolic processes. The mechanism behind bio-concrete is MICP. Calcium carbonate can be precipitated in three polymorphic forms, which in the order of their usual stabilities are calcite, aragonite and vaterite (Antony et al., 2011). The main groups of microorganisms that can induce the carbonate precipitation are photosynthetic microorganisms such as cyanobacteria and microalgae; sulfate-reducing bacteria; and some species of microorganisms involved in nitrogen cycle. Several mechanisms have been identified by which bacteria can induce the calcium carbonate precipitation, including urea hydrolysis, denitrification, sulfate production, and iron reduction. Several applications of this process have been proposed, such as remediation of cracks and corrosion prevention in concrete, biogrout, sequestration of radionuclides and heavy metals.

Figure 2-7 clearly illustrates the detailed biochemical reaction process occurring at the level of a single cell by urea hydrolysis. The Bacillus microorganism produces urease, which serves as an enzyme that hydrolyzes urea into ammonia and carbon dioxide. This production of ammonia elevates the pH (Potential of hydrogen) in the environment surrounding the bacteria, fostering the formation of $CaCO_3$ under calcium-rich conditions. The

accumulation of precipitation on the bacterial surface can be attributed to the nucleation effect, as the negatively charged cell surface is likely to attract Ca^{2+} ions.

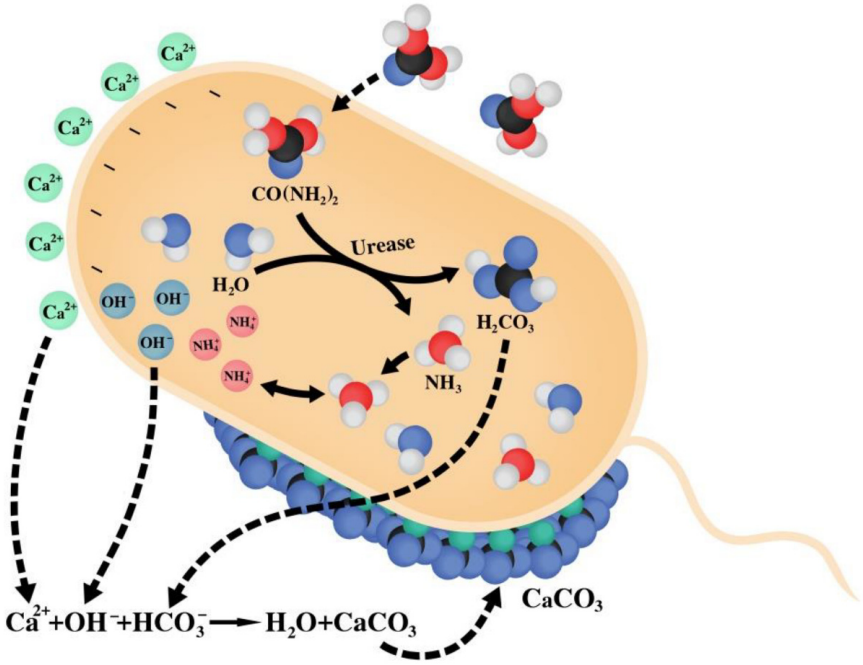


Figure 2-7: MICP mechanism diagram at a single-cell level (Sun et al., 2021)

Figure 2-8 displays the accumulation of calcium carbonate at the multicellular level. Calcium carbonate acts like an adhesive, binding individual cells together.

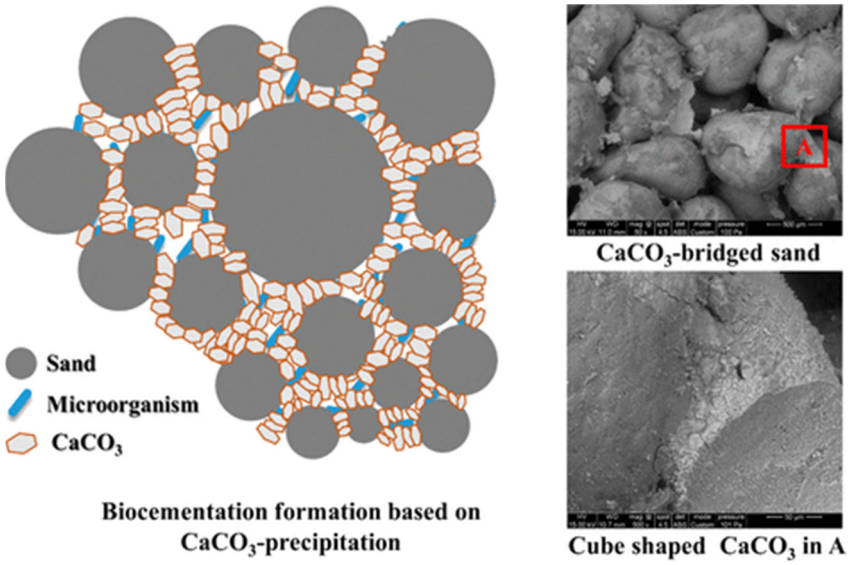


Figure 2-8: Accumulation of calcium carbonate at the multicellular level (Choi et al., 2017)

2.3.2. Experimental Exploration

The cracks repair in concrete via MICP seems a promising approach since traditional sealants may degrade over time or are environmentally toxic, whereas calcium carbonate is a more durable and benign crack sealant (Siddique and Chahal, 2011). Bang and his research team applied S. pasteurii cells for crack remediation, and they found, the overall performance of the concrete was significantly enhanced by treatment with microbial calcite

in simulated concrete cracks and cement mortar beams (Bang et al., 2010). Van Tittelboom et al., (2010) compared the crack healing potential of bacteria and traditional repair techniques, they found that cement grout only covered the surface of the samples and did not fill the cracks because of the big grain size of the grout. Epoxy and bacteria treatment, by contrast, resulted in complete filling of cracks.

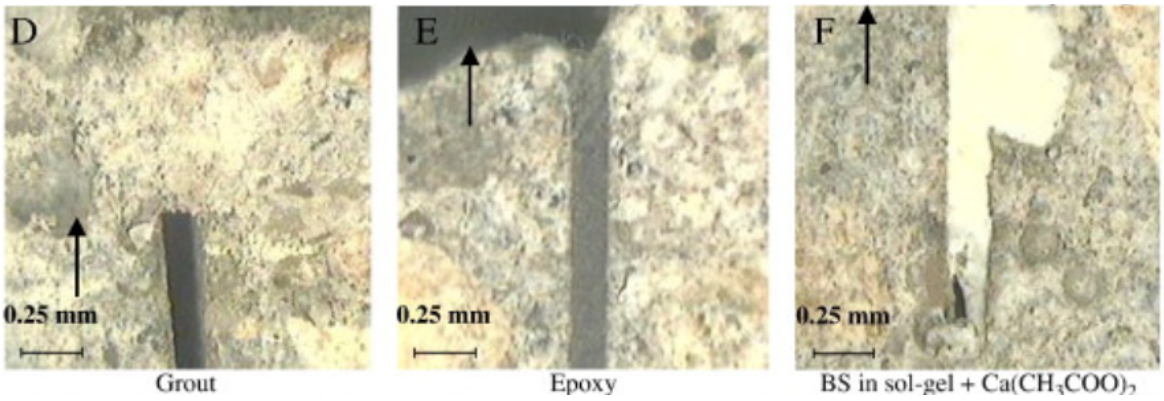


Figure 2-9: Visual evaluation of crack repair (Van Tittelboom et al., 2010)

There are different repair methods for cracks of different widths. For narrow cracks, such as those with a width of less than or equal to 1 millimeter, the cracks can be sealed by the calcium carbonate precipitated from bacteria and nutrient solution. For wide cracks, the addition of fine sand is necessary to achieve effective sealing.

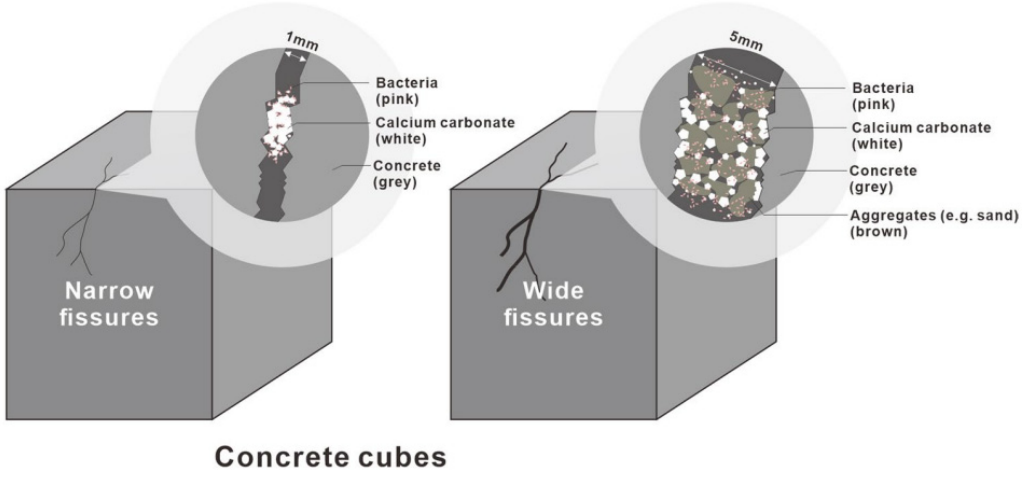


Figure 2-10: Schematic drawing of healing process of concrete cubes with narrow and wide cracks through the MICP treatment (Zhang et al., 2023)

The method to repair narrow cracks using MICP is to allow bacteria solution and substrate penetrate into the cracks, and the induced carbonate would be deposited at the opposite surfaces of the crack till the gap is filled. The potential use of bacteria specifically capable of mineral production for filling pores and cracks in concrete has been recently explored (Bang et al., 2001; De Muynck et al., 2008b, 2008a; Ramachandran et al., 2001). In these studies, the immersion process needed to cure the specimens. However, with the structures in service, it is impractical to use the immersion systems. In other words, it is impractical to cure the cement mortar in media. Jongvivatsakul et al., (2019) dropped Bacillus sphaericus and urea solutions daily on the crack with a width of 0.4 mm. After 20 days of treatment, the MICP-treated sample showed 43 % higher compressive strength than that of cracked sample. In addition, it is comparable in terms of water tightness to control mortar made without artificial crack.

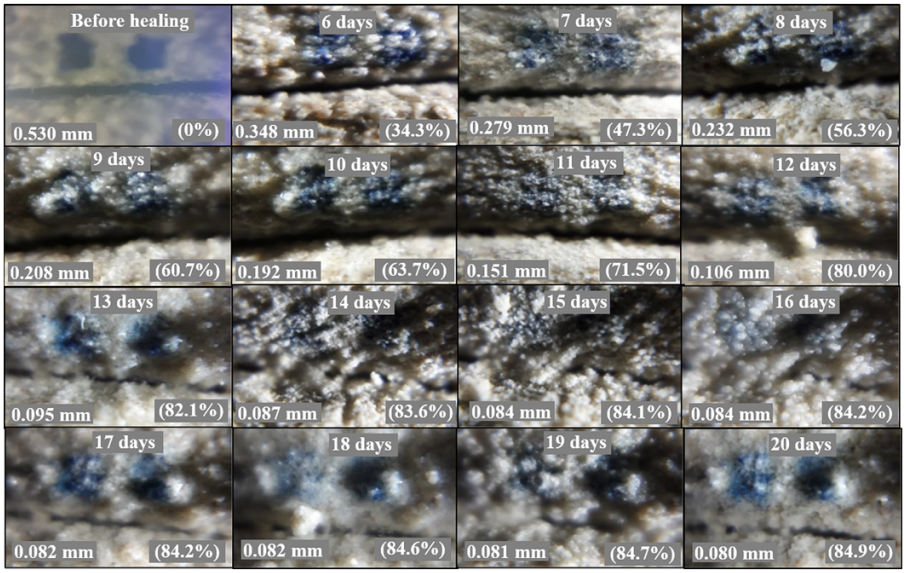


Figure 2-11: Remediation of mortar crack with a width of 0.4 mm (Jongvivatsakul et al., 2019)

For the majority of these studies, Bacillus ureolytic bacteria were employed as the biological agents for synthesizing calcium carbonate-derived minerals. The process of calcium carbonate generation by these bacteria relies on the enzymatic decomposition of urea into ammonia and carbon dioxide. A potential drawback of this reaction mechanism is the significant environmental nitrogen loading. Table 1 shows the intermediate and final byproducts generated in different metabolic pathways involved in the MICP process. These metabolic pathways have some undesirable intermediate or end byproduct generation, which is harmful to the ecosystem or environment such as hydrogen sulfide in the sulfur cycle and methane oxidation and formaldehyde during photosynthesis.

Table 1: Byproducts and its effect of different metabolic pathways involved in MICP (Jain et al., 2021)

Metabolic pathways	Byproducts	Consequences
Photosynthesis	Oxygen (O_2) and formaldehyde (CH_2O)	Hazardous to health
Methane oxidation	Hydrogen sulfide (H_2S)	Toxic, odorous gas
Sulfur cycle	Carbon dioxide and hydrogen sulfide (H_2S)	Toxic, odorous gas
Ammonification	NH_3 (ammonia)	Toxic gas
Urea hydrolysis/ureolysis	NH_3 (ammonia) and NH_4^+ (ammonium)	Toxic gas Forms toxic salts
Denitrification	Nitric oxide (NO) and nitrogen dioxide (NO_2) (intermediate) N_2 and carbon dioxide (complete)	Intermediate products are detrimental for aquatic systems, agriculture and atmosphere

In order to avoid the drawback, the combination of non-ureolytic bacteria with organic calcium source calcium lactate as a two-component self-healing system in concrete was investigated (Jonkers et al., 2010). The calcium lactate is converted to calcium carbonate due to bacterial metabolism. All the components added to the concrete mixture become an integral part of the concrete. This metabolic conversion of calcium lactate does not lead to the generation of significant quantities of ammonia. To enhance the frequently observed self-healing ability of concrete, specific healing agents can be integrated into the concrete matrix (Wiktor and Jonkers, 2011). During the experiment, the cracked mortar specimens (one control and one containing bacteria) with numerous individual

cracks of varying widths were placed horizontally in tap water (with a 3.5 cm water column covering the specimens) in a plastic bucket. Experimental results showed crack-healing of up to 0.46 mm-wide cracks in bacterial concrete, as shown below.

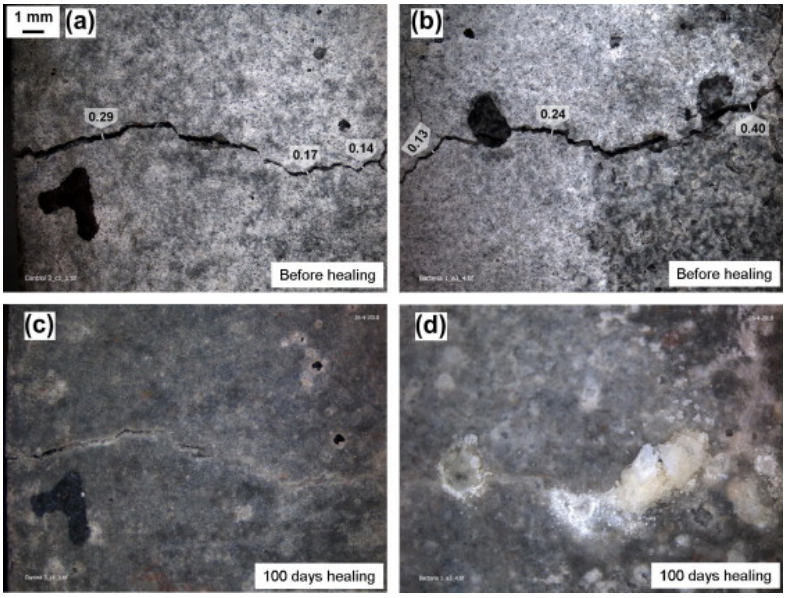


Figure 2-12: Stereomicroscopic images of crack-healing process in control mortar specimen (Wiktor and Jonkers, 2011)

For wide crack, Ramachandran et al. (2001) plugged artificially cracked cement mortar using bacteria combined with sand as a filling material, and the cement mortar was cured in urea and Calcium Chloride ($CaCl_2$) medium. It was found that use of bacteria improves the stiffness and compressive strength of concrete. Achal et al., (2013) also filled the crack within the cement mortar with Bacillus ureolytic bacterial cells and natural sieved sand. Both sets of specimens were cured in media containing urea and $CaCl_2$ for 7 and 28 days. The crack of depth 27.2 mm and width 3.0 mm was successfully repaired with a porosity reducing over 50% and a compressive strength improving about 40%.

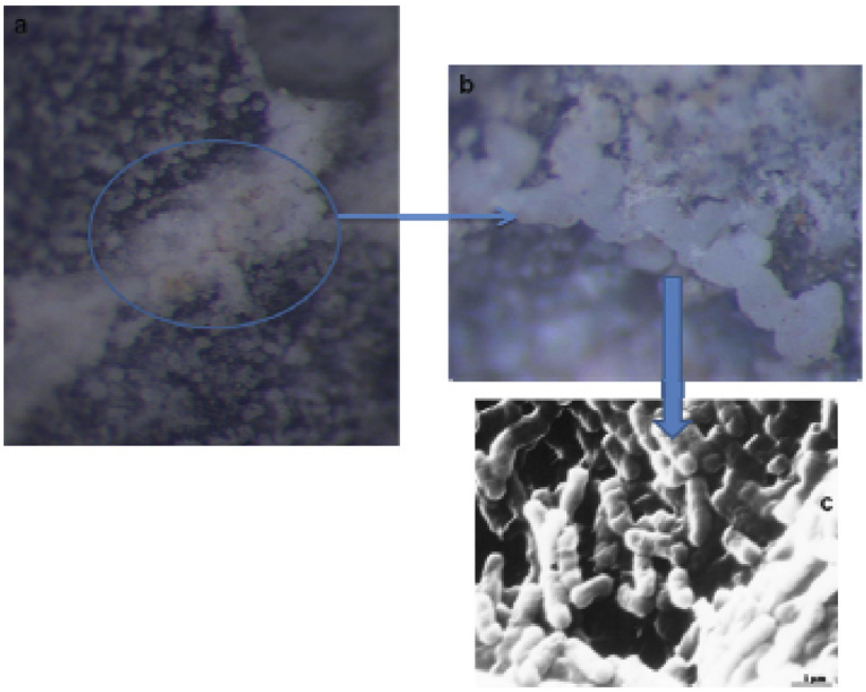


Figure 2-13: Microscopic image of: (a) crack remediated area, (b) enlarged portion of crack remediated area showing calcite precipitation, and (c) Electron micrograph showing rod shaped bacteria embedded in crack remediated area (Achal et al., 2013)

While most research on microbial crack self-healing concrete utilizes a single type of microorganism and is conducted under conditions favorable for the microorganism's survival, the actual working environment of concrete mortar is often complex and variable (Khaliq and Ehsan, 2016). Thus, a microbial consortia composed of various microorganisms can perform more complex tasks, and has better performance in resisting environmental fluctuation of self-healing materials compared with single microorganism (Zhang et al., 2019). Zhang et al. (2019) explored a systematic investigation of the microbial product and healing efficiency during crack healing in concrete induced by mixed cultures and pure cultures. The morphology of the cracks in the prismatic-type specimens at different healing times is shown in Figure 2-14. Concrete samples containing MC-Ao consortia displayed the greatest crack healing width of 1.22 mm after 28 days, whereas samples with pure cultures and MC-Aa consortia showed narrower healing widths of 0.79 mm and 0.73 mm, respectively.

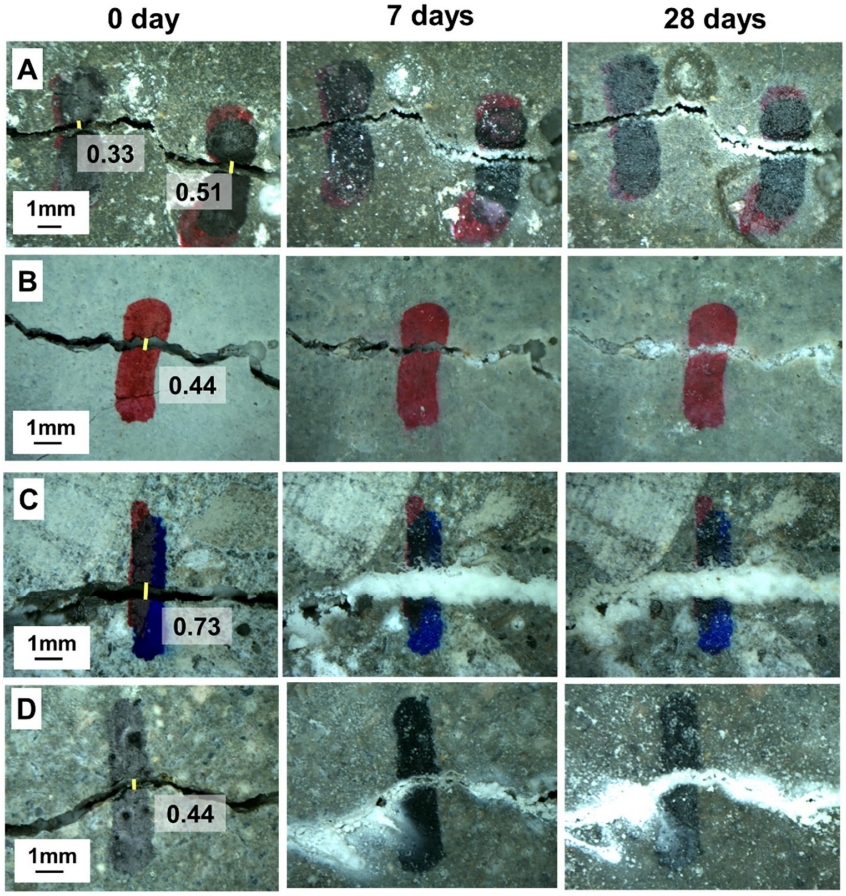


Figure 2-14: Microscopic images of control (A), PC (B), MC-Ao (C) and MC-Aa (D) crack healing processes (Zhang et al., 2019)

Spencer et al. (2020) used ureolytic Sporosarcina pasteurii and jute fibres to produce biocemented soil columns via MICP in the laboratory. The results indicated that columns containing 0.75% untreated jute fibers (by weight of sand) exhibited unconfined compressive strengths approximately six times greater on average compared to biocemented sand columns without jute fibers. Columns with jute fibers had $CaCO_3$ contents measured by calcimeter that were at least three times higher than those containing only sand. The inclusion of fibers within the biocemented sand sustains the long-term activity of Sporosarcina pasteurii bacteria, enabling the continuation of the MICP process without requiring multiple injections of bacteria. This can reduce the production costs of biocemented sand material and is particularly beneficial in scenarios where multiple treatments of cementation medium are needed to achieve low permeability and high strength. This research provides a good inspiration for repairing large cracks.

2.3.3. Instructions of Basilisk ER7

Basilisk ER7 is a 2-component low viscosity solution which contains natural enzymes and nutrients. Both components must be dissolved separately in warm water (ca.40-degree) before application. The ratio of component A to B is 2:1. The product is functional in temperature range of 10 - 40˚C.

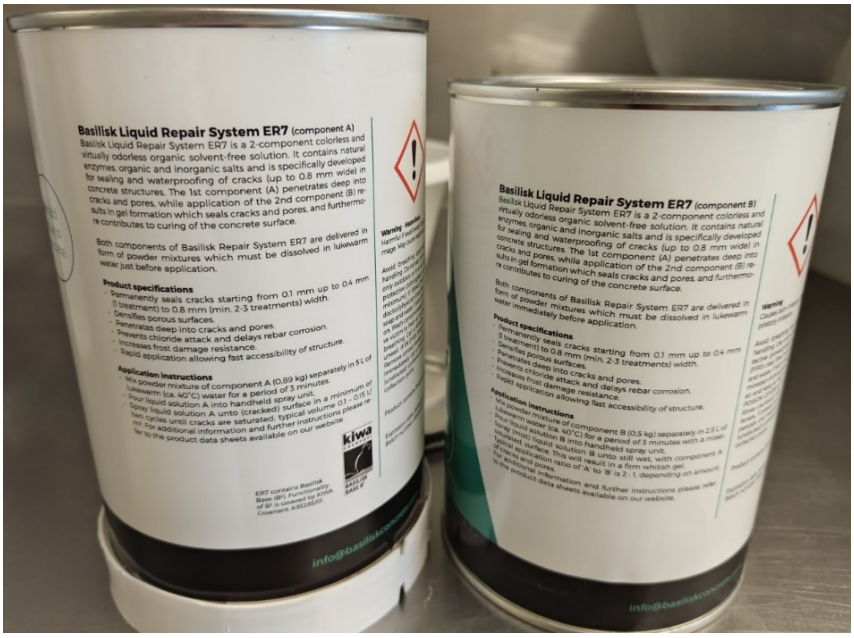


Figure 2-15: Component A and B of Basilisk ER7

Cracks must be clean and dry prior to treatment to allow effective penetration of the liquid repair system. The repair agent, as a low-viscosity liquid, can penetrate into the interior of cracks, induce calcium carbonate precipitation through the metabolism of bacteria, and thus seal the cracks from within. Sealing of cracks occurs internally and is usually not visible at the crack surface, easily leading people to mistakenly believe that the crack has not been repaired.

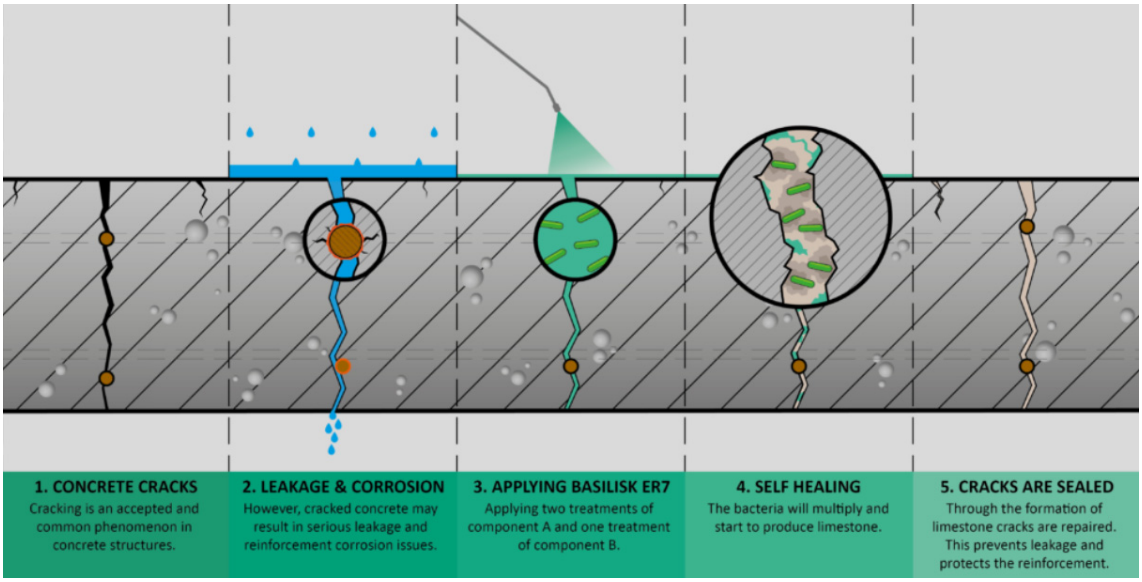


Figure 2-16: Schematic diagram of Basilisk ER7 for crack sealing (Basilisk, 2021a)

As shown in Figure 2-17, the repair agent Basilisk ER7 has already been applied in various scenarios, such as bus lanes, parking lots, tunnel floors, precast concrete elements, and balconies (Basilisk, 2021b).

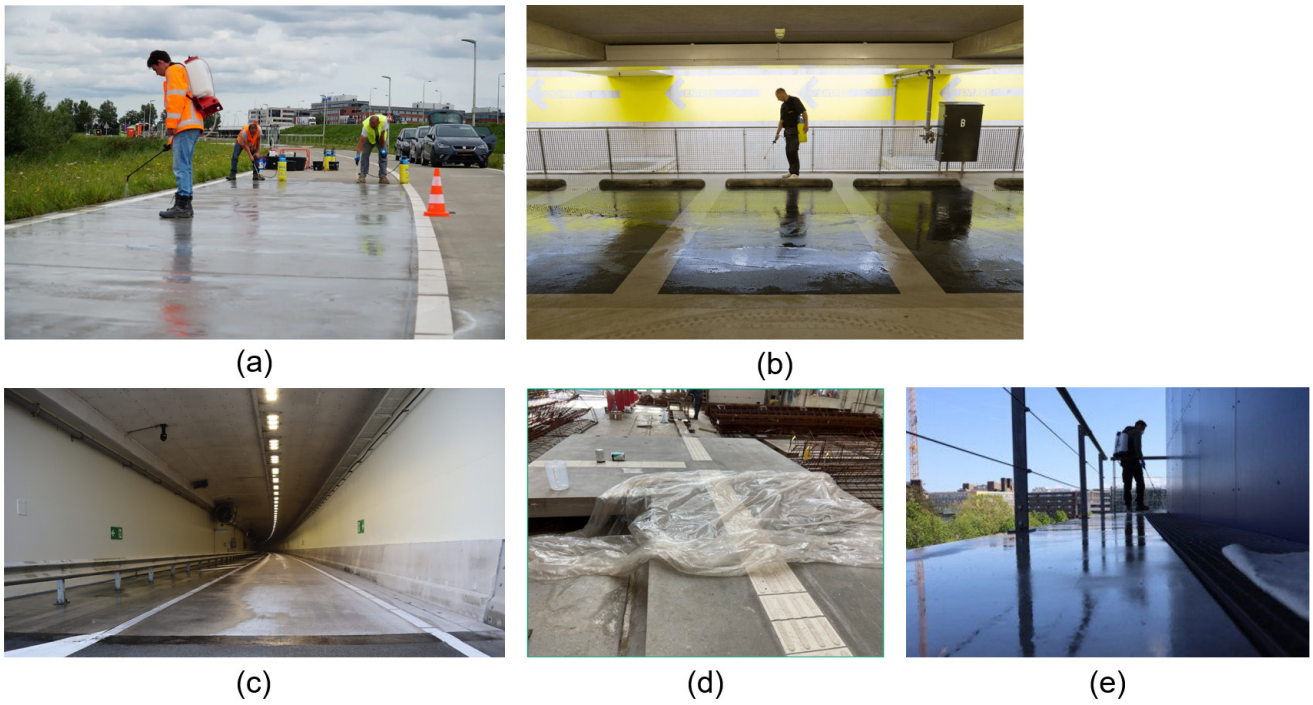


Figure 2-17: Application examples of Basilisk ER7 (Basilisk, 2021b)

The road surface of a bus lane Hoogwaardig Openbaar Vervoer Schiphol has cracks in the concrete along the entire route. The Province of Noord-Holland chose Basilisk ER7 to solve this sustainability problem. As shown in Figure 2-17 (a), the workers applied a surface treatment method by using pressure sprayer to spray the agent on the entire road surface. After treatment, the water permeability of the concrete under 1 meter of water pressure decreased by an average of 93%. A second treatment at 20 centimeters of water pressure yielded even better results, with water permeability being (nearly) fully restored post-treatment. Another benefit of Basilisk ER7 is the enhanced freeze/thaw resistance of the surface. After two treatments, this resistance increases by an average of 57%, with exceptional cases showing up to 79%. As a result, the lifespan is demonstrably extended by at least 15 years (Basilisk, 2021c).

The large-scale application of the Basilisk ER7 has been conducted on an intermediate floor of a parking garage in Apeldoorn, as shown in Figure 2-17 (b). Due to the large size of the surface area to be treated, two different application methods were selected for this pilot. The first 6000 m² was treated using a liquid recovery system with scrubbing machines, while the second 6000 m² was handled with handheld high-pressure spraying devices. The entire floor has been sprayed with the repair agent. After six weeks, the liquid repair system showed noticeable effectiveness, significantly reducing water penetration. Within a few months to six months following the treatment, wet spots on the ceiling and the floor below were no longer visible during rainy periods (Basilisk, 2021a).

Figure 2-17 (c) shows the repair lots of cracks in tunnel using Basilisk ER7. Van with a 1.000 L tank was used to achieve surface treatment, reaching a productivity of 1.000 m² per hour for spraying agent on tunnel surface. All the shrinkage cracks in 2 tunnel tubes were dealt with within a day without causing significant downtime (Basilisk, 2021d).

Figure 2-17 (d) shows the crack repair in precast elements with Basilisk ER7. After production, some precast elements exhibited shrinkage cracks ranging from 0.1 to 0.25 mm. Since these elements will be used on platforms, exposure to winter salts could lead to significant durability concerns. Basilisk ER7 can seal the existing cracks completely, protecting the reinforcement and preventing future durability issues (Basilisk, 2021e).

One of the LUMC hospital buildings has hairline cracks in the concrete of the balconies. The cracks were too small for concrete repair with injections. Applying a coating involves a higher initial investment and needs to be reapplied every 10 to 15 years. Moreover, it alters the appearance of the concrete surface. As shown in Figure 2-

17 (e), the workers applied a surface treatment method by using pressure sprayer to spray the agent on the entire balconies surface. After treatment, the concrete appearance looks the same, but the moisture no longer penetrates and the reinforcement is well protected again (Basilisk, 2021f).

2.4. The Challenges of Narrow Crack Repair

As noted in Section 2.2, in traditional methods for repairing narrow cracks on concrete surfaces, the injection method is relatively convenient. As shown in Figure 2-18, crews are using polyurethane to repair cracks in taxiways to keep the historic Milan-Malpensa MXP International Airport running at top efficiency (Glewwe, 2017). The tough high penetration polyurethane can quickly soak into small cracks and repairs them before they become big problems. The MXP Airport saves time and money by repairing cracks early when they are small.

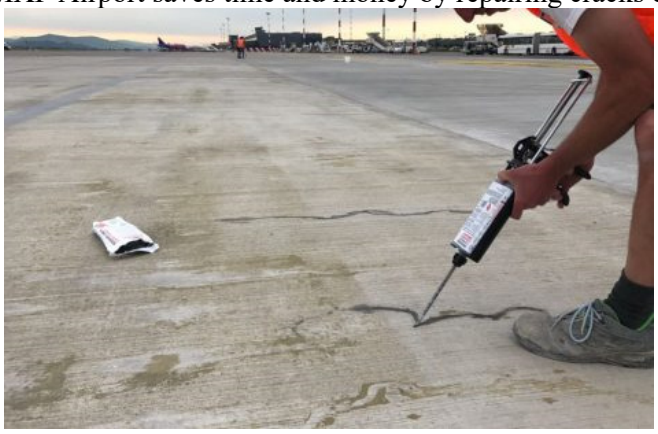


Figure 2-18: Crack repairs at Milan-Malpensa MXP International Airport using polyurethane injection (Glewwe, 2017)

However, airport runways and aprons cover vast areas, and cracks may be widespread throughout. The large surface area makes it challenging to detect and monitor narrow cracks, especially when they are difficult to identify in the early stages. Narrow cracks are often so small that they can be easily overlooked during inspections. This makes it challenging to detect and repair them promptly, potentially leading to further deterioration if they are not addressed in time. And workers need to carefully clean the cracks and apply the repair materials with precision, which is time-consuming and labor-intensive, especially on a large scale. From Section 2.3, it can be seen that although the repair agent Basilisk ER7 is more environmentally friendly compared to traditional repair agents and its usage is relatively simple, it is only suitable for narrow cracks with a maximum width of 0.6 mm. Therefore, surface treatment method is often used by spraying, which leads to significant agent waste. Moreover, in reality, wider cracks may be found next to narrow ones, but Basilisk ER7 is ineffective on wider cracks. If these wider cracks are not noticed and repaired in time, they could pose a more serious problem than the narrow cracks. In practice, many locations have the similar challenges in maintaining large concrete surfaces, including the detection and repair of narrow cracks:

- **Highways and Interstates:** Large expanses of highways and interstates are constructed from concrete. Routine maintenance is essential for these surfaces to repair cracks and prevent further degradation.
- **Ports and Container Terminals:** Ports and container terminals feature extensive concrete areas designed to facilitate the handling and storage of heavy cargo containers. These areas are subject to considerable stress and abrasion, resulting in the development of cracks that necessitate consistent repair efforts.
- **Parking Lots and Garages:** Large commercial or industrial parking lots and multi-level parking garages typically have substantial concrete surfaces that require ongoing maintenance, with a particular focus on high-traffic zones.
- **Industrial Facilities and Warehouses:** Industrial complexes and warehouses often feature extensive concrete floors designed to bear the weight of heavy machinery, equipment, and vehicles. These floors are susceptible to cracking and necessitate regular maintenance to address such issues.

- **Dams and Spillways:** Massive concrete surfaces of large dams and spillways are under constant pressure from water, making cracks in these structures potentially critical and necessitating immediate attention.
- **Military Bases:** Military airfields, training grounds, and storage facilities often have vast concrete surfaces similar to those at civilian airports. Cracks may also appear on these concrete surfaces and need to be repaired in a timely manner.
- **Sports Arenas and Stadiums:** Large sports arenas and stadiums with concrete floors, particularly those used for events requiring heavy equipment setups, can experience crack issues over time.
- **Runways at Other Transportation Hubs:** Transportation hubs such as large bus terminals or railway stations may have extensive concrete platforms and driveways that need similar maintenance.

In many Western countries, labor costs are very high, and there's often a shortage of workers willing to perform repetitive and monotonous tasks like crack inspection and repair. Developing an autonomous robot to perform these tasks using Basilisk ER7 could significantly reduce labor costs and dependency on human workers. Through automating the crack detection and repair process, robot would increase the efficiency of maintenance work. It would continuous operate without the need for breaks, ultimately speeding up the repair process and reducing downtime for critical infrastructure. Integrating machine learing algorithms ensures that even the very narrow and most easily overlooked cracks would be identified. By integrating additional computer vision techniques, the robot can distinguish between narrow and wide cracks, allowing the it to spray repair agent only onto the narrow cracks, thus saving the repair agent. At the same time, the robot should be able to record information about the wide cracks and visualize them on a webpage. This could facilitates the timely detection of significant cracks, enabling prompt corrective actions to be undertaken, thereby mitigating the risk of further deterioration. Using an eco-friendly repair material like Basilisk ER7 aligns with the growing emphasis on sustainable practices in the construction industry. It could also make the robot more appealing to companies and governments looking to reduce their environmental footprint. The robot's size should be minimized to allow it to navigate through various spaces like a floor-cleaning robot. Smaller robots also have relatively lower production costs. To further reduce costs, it is preferable that the software and hardware used for developing the robot are open-source, which not only eliminates additional fees but also facilitates the robot's expandability. Figure 2-19 shows the features of the planned robot. With the rapid advancements in artificial intelligence (AI), new materials, and edge computing and chips technology in modern society, the development of this robot tailored to these requirements is feasible.

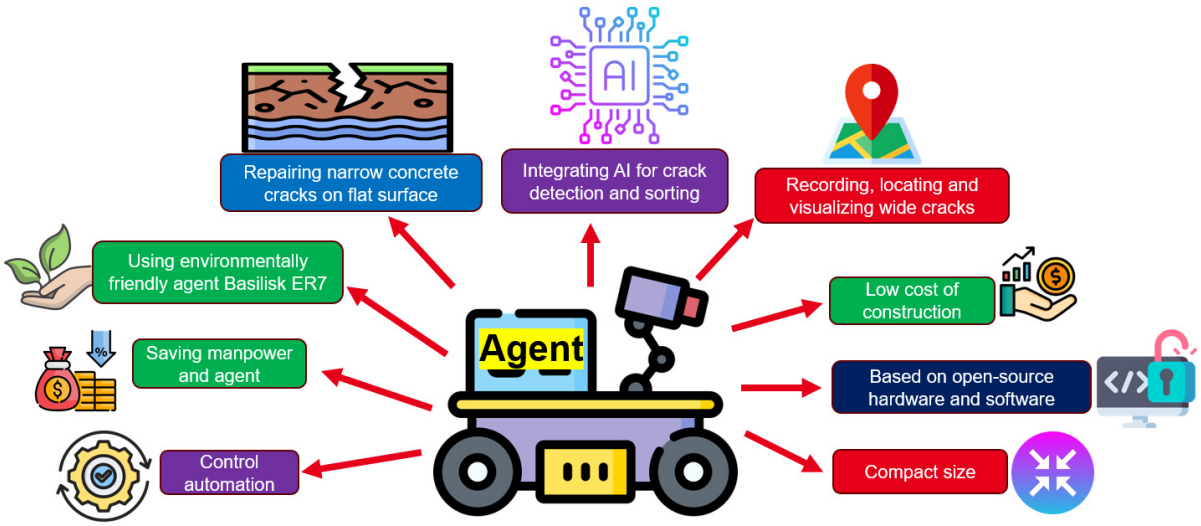


Figure 2-19: Overall features of the planned robot

3. Technical Basis of the System

This work aims to develop an intelligent robot and a corresponding web application for visualizing wide cracks, involving a variety of software and hardware knowledge, such as machine learning, edge computing, robotic systems, and web-based 3D visualization. Based on the results of this chapter, the concept will be developed and implemented as a prototype.

3.1. Machine Learning for Crack Recognition

Traditionally, crack evaluation is conducted manually through human field surveys. For example, crack width ruler is a simple gauge that has been designed to provide inspectors with a low-cost alternative to a graduated microscope for determining the width of a crack in concrete. However, these manual survey methods have poor repeatability and reproducibility, need excessive time and labor. Also, the data collected may vary with different raters due to subjectivity.

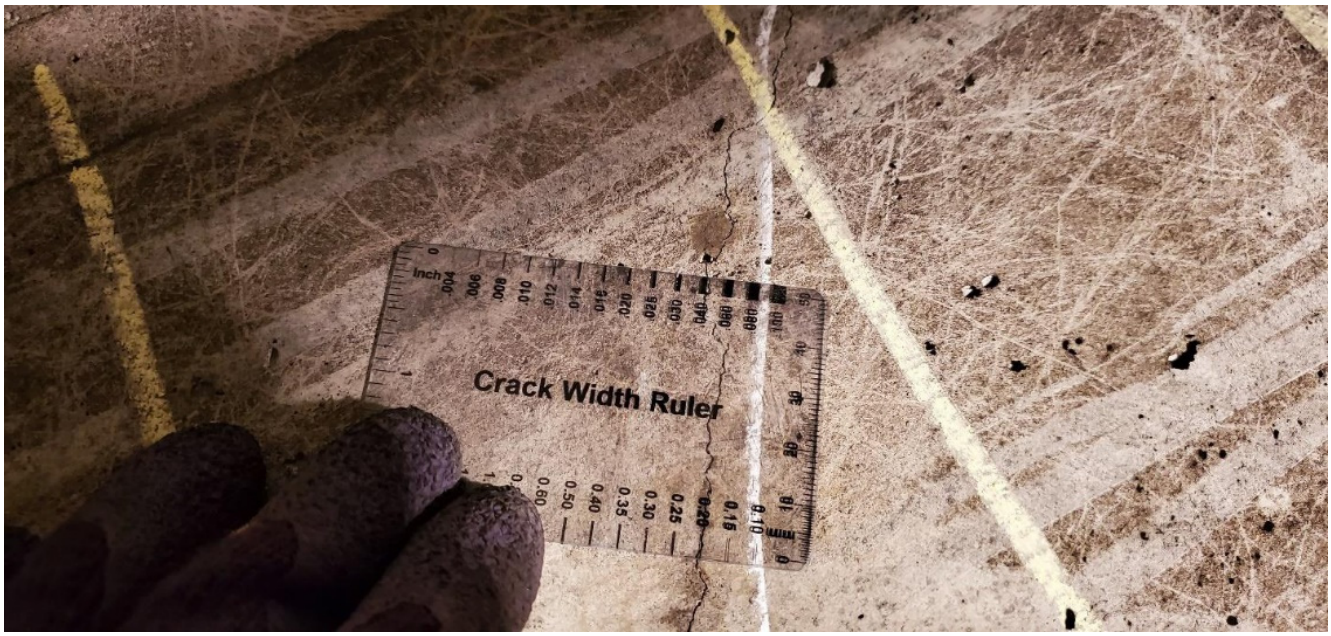


Figure 3-1: Using a crack width ruler for approximating crack sizes in visual inspections (OSNDT, 2024)

To overcome the shortcomings of manual surveys, there have been substantial research efforts to develop automated crack survey methods (Gupta and Dixit, 2022; Nguyen et al., 2023). Having the ability to perform various tasks with outstanding performance, machine learning has become a popular technique in almost every field. It is a subset of artificial intelligence that focuses on the development of algorithms capable of learning and making predictions based on data. In other words, the machine learning algorithms use data to improve their performance over time. They learn from the data provided to them, rather than being explicitly programmed to perform a specific task. Data-driven machine learning models can underperform if the training dataset is insufficient in quantity or lacks diversity in the features they represent. This can lead to models that are not robust enough to generalize well to new data. Types of machine learning includes Supervised Learning, Unsupervised Learning, Semi-Supervised Learning and Reinforcement Learning.

3.1.1. Convolutional Neural Network

Computer vision plays a crucial role in modern technology, allowing machines to interpret and understand the visual world. Three essential tasks in computer vision are segmentation, detection, and classification.

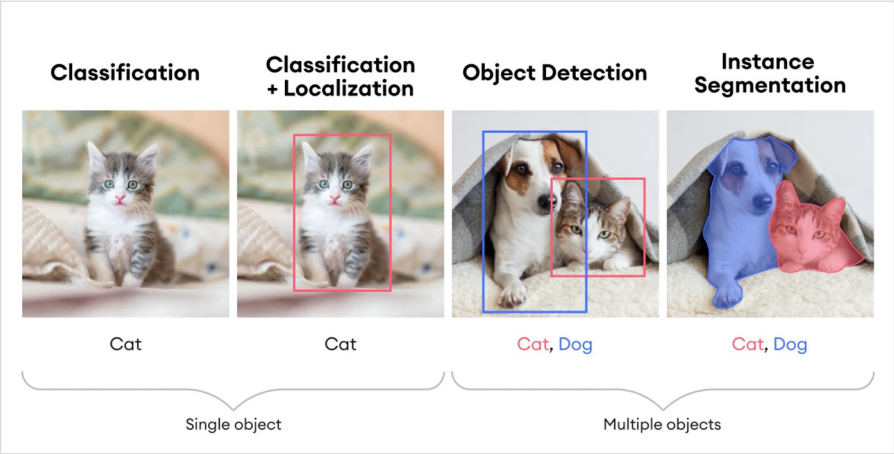


Figure 3-2: Comparison among computer vision tasks (SuperAnnotate, 2023)

Artificial neural network is a model inspired by the biological neural network. In biological neural network, neurons perform the processing, dendrites receive signals from other neurons, the soma sums all the incoming signals, and the axon transmits these signals to other cells. An artificial neural network consists of connected artificial neurons, which loosely model the neurons in a brain. Through layers of nodes and connections with adjustable weights, the network learns to transform input data into meaningful outputs. The training process involves forward propagation of data and backpropagation of errors to optimize the weights, allowing the network to learn and improve.

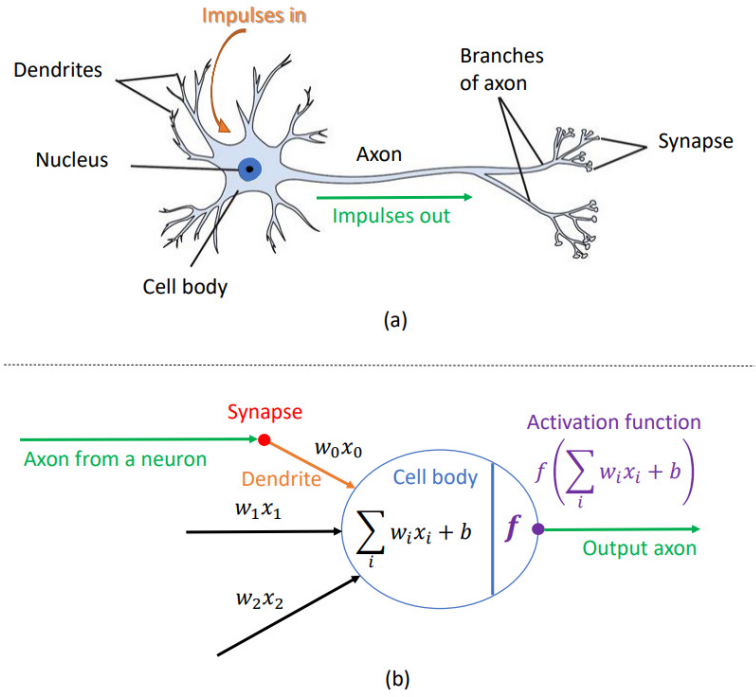


Figure 3-3: A comparison between a human neuron and a neural network neuron (Roffo, 2017)

A Convolutional Neural Network (CNN) is a deep learning algorithm specifically designed for object recognition. One of the key characteristics of CNNs is their ability to automatically and effectively extract features from images,

which is crucial for object recognition. A variety of pre-trained CNN architectures, including SegNet, VGG-16, ResNet, Inceptionv3, YOLO and EfficientNet, have demonstrated very good performance. Beyond object recognition tasks, CNNs are versatile and can be applied to a range of other domains, such as natural language processing, time series analysis, and speech recognition. The CNN network is also inspired by the structure of the human brain. As shown in Figure 3-4 (a), four Brodmann areas are related to the ventral visual stream. The figure also includes a block diagram highlighting some of the numerous forward and backward projections between these areas. Figure 3-4 (b) is a sketch of the AlexNet convolutional neural network, where pairs of convolution operators followed by a max pooling layer are roughly analogous to the hierarchical structure of the biological visual system

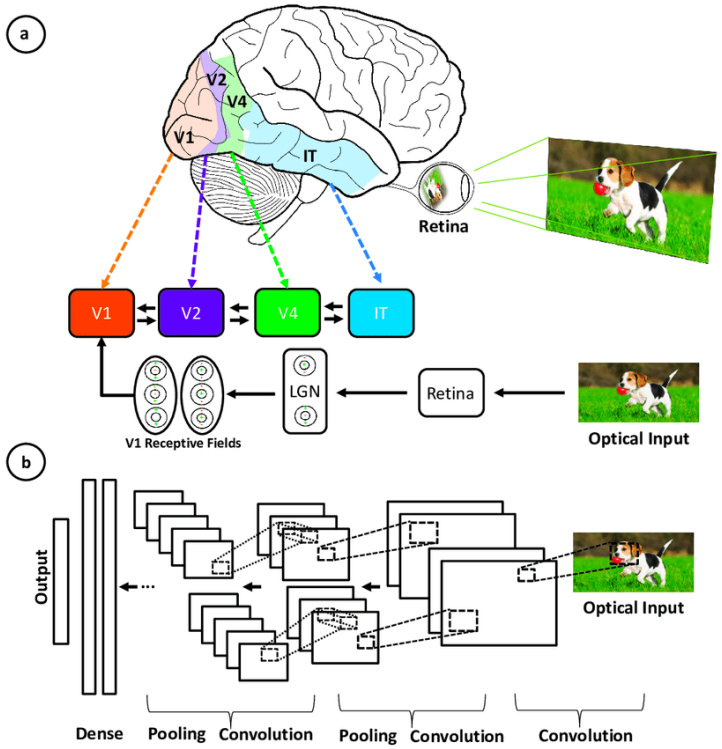


Figure 3-4: Illustration of the correspondence between the regions associated with the primary visual cortex and the layers in CNN (Roffo, 2017)

As shown in the figure below, there are three main parts of a CNN architecture.

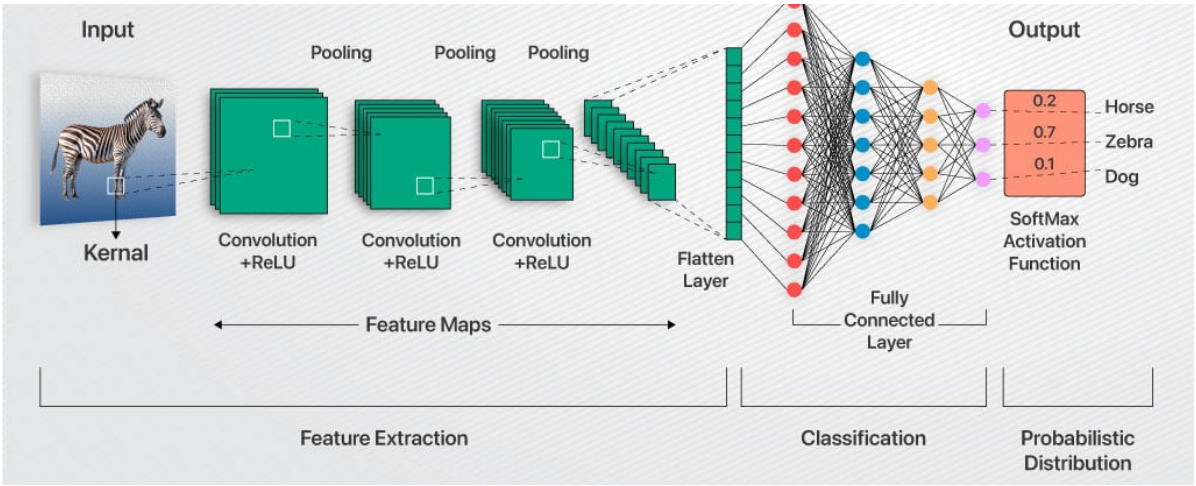


Figure 3-5: A typical CNN architecture (Sahai, 2024)

The network of feature extraction consists of many pairs of convolutional or pooling layers. Feature extraction aims to reduce the number of features present in a dataset. It creates new features which summarizes the existing features contained in an original set of features. The convolution layer is the core building block of the CNN, it is used to extract the feature from the input dataset. It applies a set of learnable filters known as the kernels to the input images. The filters/kernels are smaller matrices usually 2×2, 3×3, or 5×5 shape. it slides over the input image data and computes the dot product between kernel weight and the corresponding input image patch. The output of this layer is referred as feature maps. The pooling layer replaces the output of the network at certain locations by deriving a summary statistic of the nearby outputs. This helps in reducing the spatial size of the representation, which decreases the required amount of computation and weights. The pooling operation is processed on every slice of the representation individually. Fully connected layer takes the input from the previous layer and computes the final classification or regression task. As shown in the figure below, crack recognition using CNN can be divided into three task categories: classification, object detection and segmentation.

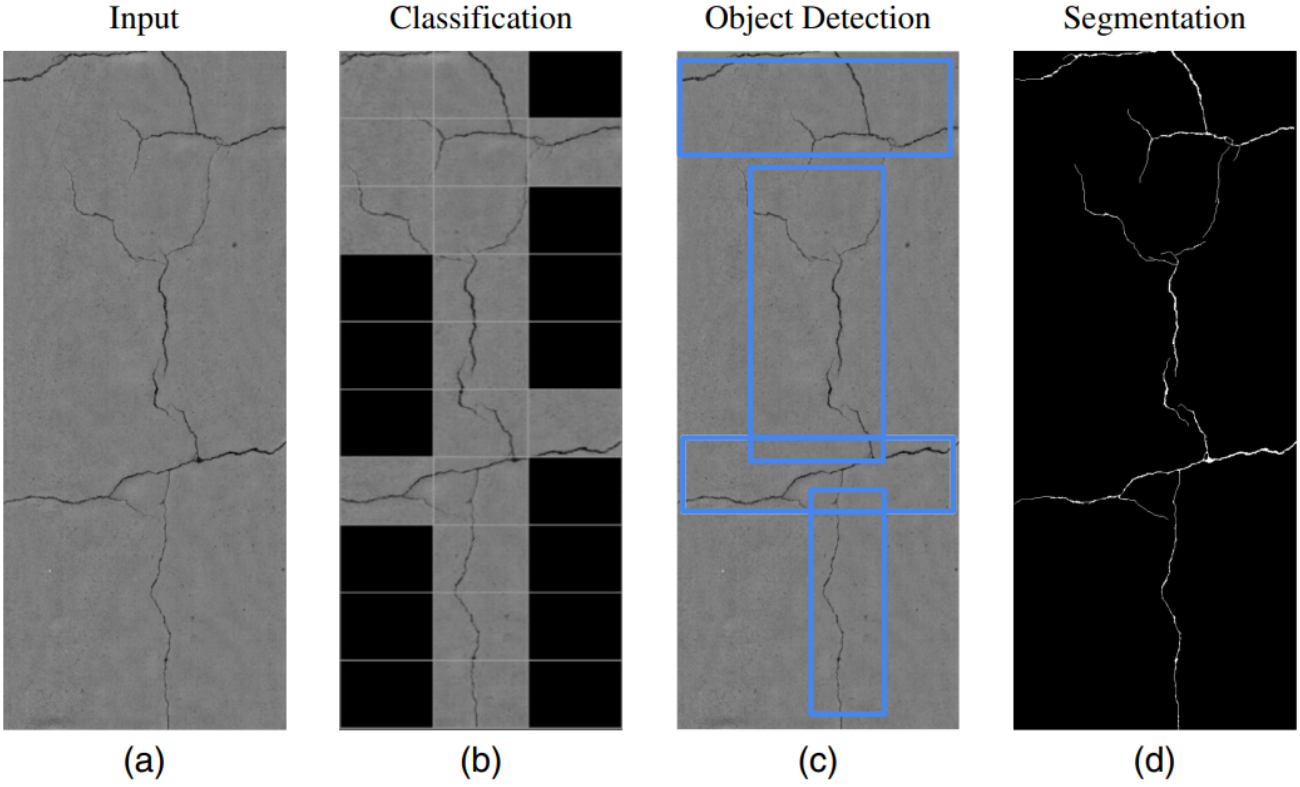


Figure 3-6: Crack recognition using CNN: (a) Input image; (b) classification: image patches classified as crack or noncrack; (c) object detection: bounding boxes generated around areas that contain cracks; and (d) segmentation: pixels classified as crack or noncrack (Hsieh and Tsai, 2020)

Classification is valuable for making binary decisions, such as determining the presence or absence of a particular object or anomaly within an image. Image classification can be deployed to determine the presence or absence of crack on the surface of concrete. In 2016 deep learning has been applied to the classification of cracks (Schmugge et al., 2016; Zhang et al., 2016). Object detection can be used to locate cracks within an image, identifying their presence and general position (Cha et al., 2017). Segmentation generates a pixel-wise prediction of cracks in the images; In other words, segmentation enables the determination of cracks on a concrete surface, identification of their locations, measurement of crack sizes, classification of crack types, and extraction of important crack features. Crack segmentation has become the current trend in machine learning crack recognition. In 2017, crack

segmentation has been employed to assist remote visual examinations of nuclear power plant components (Schmugge et al., 2017). These algorithms provide a comprehensive analysis of cracks for maintenance and safety evaluations.

3.1.2. Training Model with Transfer Learning

Transfer learning is a machine learning technique where a model pre-trained on one task is repurposed as the starting point for another task. This approach is computationally efficient and yields better results, especially with small datasets. By leveraging the knowledge (features, weights, etc.) from previously trained models, these models already have an understanding of essential features. Instead of starting from scratch, researchers and data scientists often begin with a pre-trained model proficient in object classification and capable of recognizing general features such as edges and shapes in images.

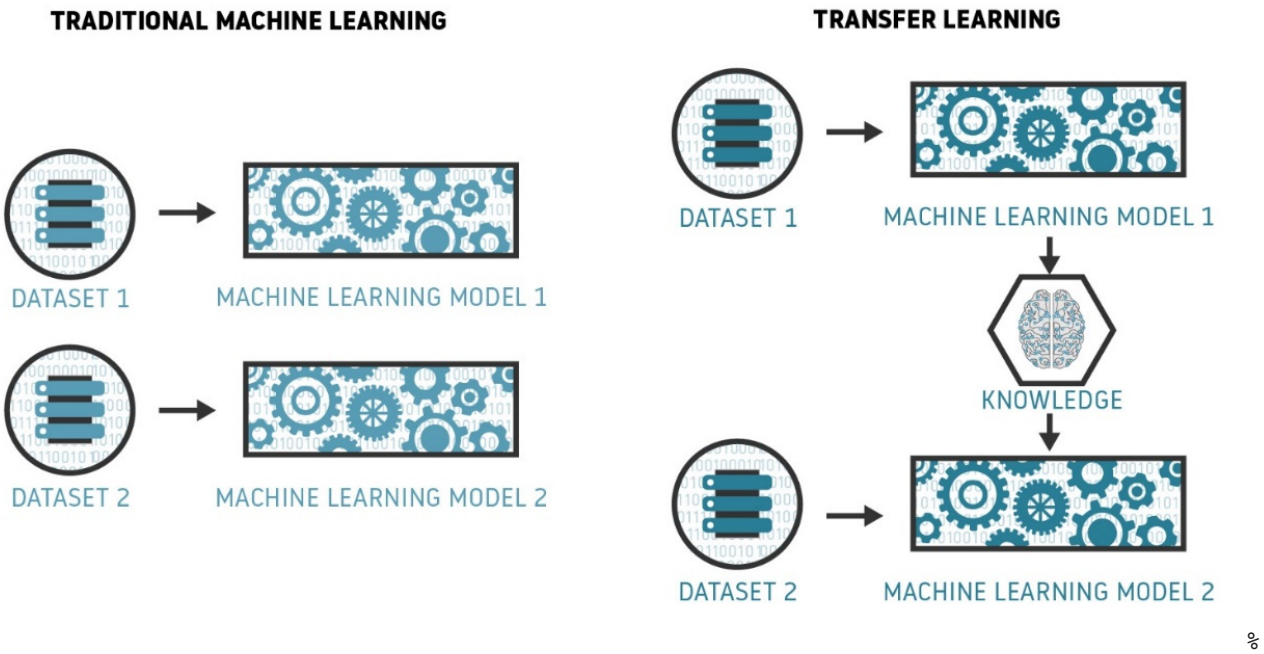


Figure 3-7: Traditional machine learning vs Transfer learning (Martinez, 2020)

Transfer learning presents certain challenges, one of the primary being negative transfer. This occurs when the accuracy of a deep learning model decreases after further training. Biologically, this can be compared to existing knowledge hindering new learning. Negative transfer can result from significant differences between problem domains or the model's inability to adapt to the new domain's data set and its nature. The task of crack segmentation is similar to that of animals segmentation in terms of object detection and object segmentation. Thus, transfer learning can be effectively applied to train a network such as YOLO for this purpose.

3.1.3. Underfitting and Overfitting

Overfitting and underfitting are two common issues encountered in the training of neural networks and other machine learning models. Overfitting occurs when the model is very complex and fits the training data very closely. This will result in poor generalization of the model. This means the model performs well on training data, but it won't be able to predict accurate outcomes for new, unseen data. Underfitting occurs when a model is too

simple and is unable to properly capture the patterns and relationships in the data. This means the model will perform poorly on both the training and the test data.

The Figure 3-8 shows the examples of model underfitting, model fitting good, and a model overfitting of classification, regression, and deep learning models. In Deep Learning, it's crucial to monitor the learning process to ensure the model is fitting well to the training data without overfitting or underfitting. The training and validation error over each iteration during the model training process are essential tools for this purpose.

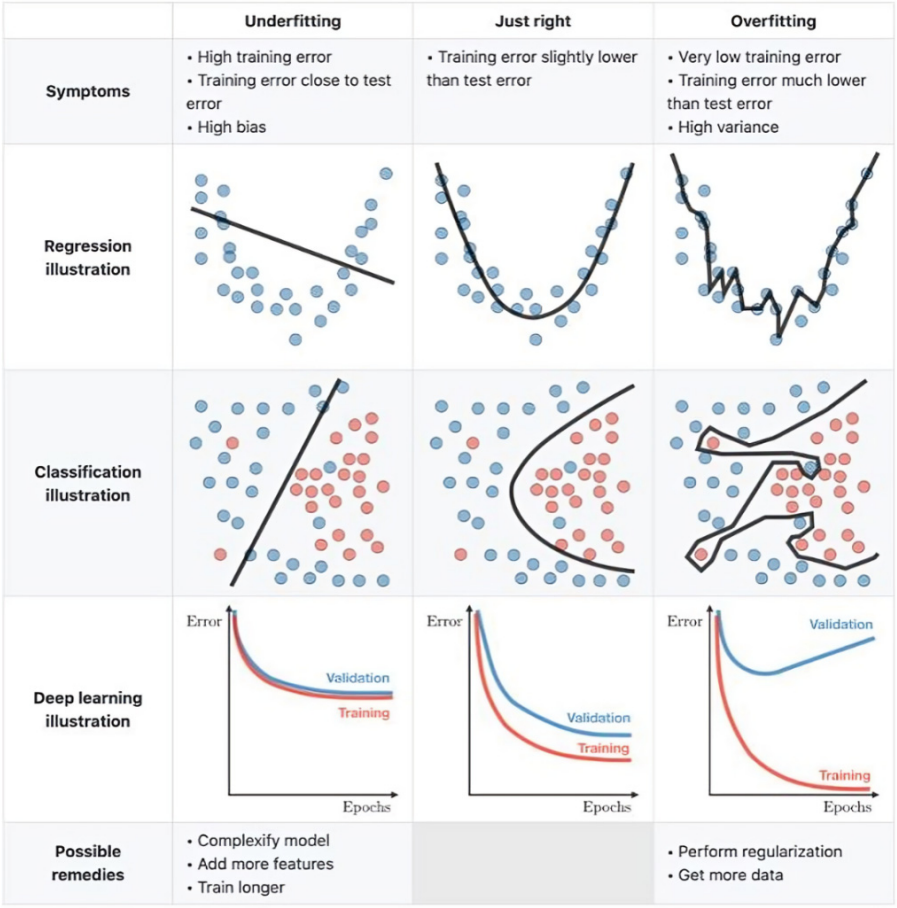


Figure 3-8: Examples of model underfitting, model fitting good, and model overfitting (SHRIVASTAVA, 2020)

There are some strategies to handle overfitting of computer vision tasks:

- **Data Augmentation**: Boost the diversity and number of training examples by performing random transformations on existing images. Techniques like rotation, flipping, and scaling can help create new variants of the original data, making the model more robust.
- **Increasing Training Data**: Gather more labeled data to train the model. This helps in reducing the bias of the model.
- **Early Stopping**: Monitor the model's performance on a validation set during training and stop the training process when the validation loss starts to increase. Early stopping prevents the model from overfitting by finding the optimal point where it achieves good performance without overfitting.
- **Model Simplification**: Simplify the model architecture by reducing the number of layers, nodes, or parameters. A simpler model is less likely to overfit and can generalize better to new data. Or consider using a different network with simple architecture.

There are some strategies to handle underfitting:

- **Model Complexity:** Increase model complexity, such as using more layers in a neural network or higher-degree polynomial regression, to address underfitting. A more complex model can capture more intricate patterns and improve performance.
- **Model Selection:** Experiment with different algorithms or model architectures to find the one that best fits the data and problem at hand. Different algorithms have varying levels of complexity and flexibility, and choosing the right model can help overcome underfitting.
- **Train Longer:** Increase the number of training epochs or extend the duration of training. Sometimes models need more time to learn complex relationships in the data.

3.1.4. Evaluating Performance of Model

Segmentation network such as YOLOv8 uses a combined loss function, consisting of four components. This enables the model to simultaneously perform object detection and semantic segmentation. **box_loss** measures the difference between the predicted bounding boxes and the actual bounding boxes of objects in the training data. **seg_loss** refers to the objective function used during the training of neural networks for semantic segmentation. It quantifies the discrepancy between the predicted segmentation masks (output by the model) and the ground truth masks (annotated in the training data). **cls_loss** measures the correctness of the classification of each predicted bounding box. In other words, it evaluates how well the model labels a predicted box with the correct class. **dfl_loss** is designed to address the class imbalance issue in classification and semantic segmentation tasks.
Confusion Matrix is a performance measurement for machine learning classification problem where output can be two or more classes. It is a table with 4 different combinations of predicted and actual values. It is extremely useful for performance metrics Recall, Precision and Accuracy.

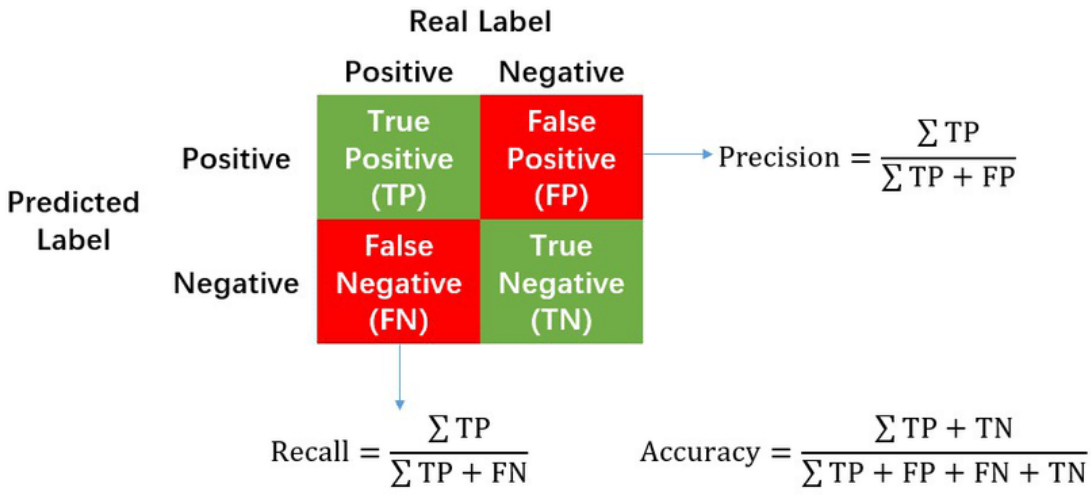


Figure 3-9: Calculation of Precision, Recall and Accuracy in the confusion matrix (MA et al., 2019)

Intersection over Union (IOU) is a performance metric used to evaluate the accuracy of annotation, segmentation, and object detection algorithms. It quantifies the overlap between the predicted bounding box or segmented region and the ground truth bounding box or annotated region from a dataset. By semantic segmentation, IOU is used to assess the quality of the segmented regions. It allows for the measurement of how well the model identifies the boundaries of objects.

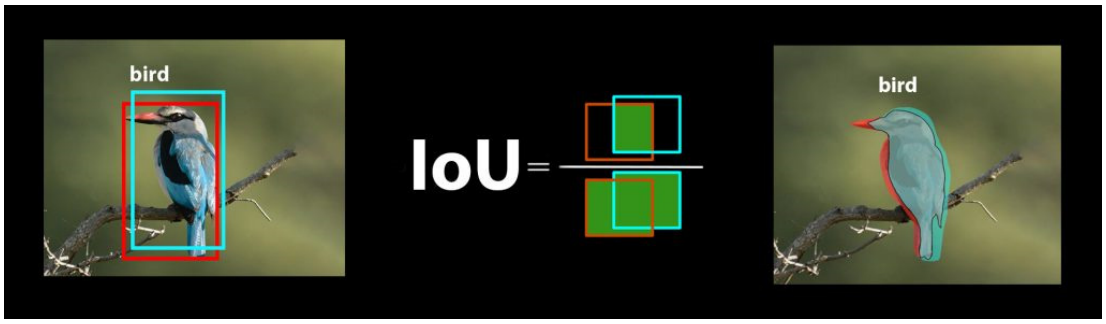


Figure 3-10: IoU: Measuring Overlap Between Ground Truth and Prediction Segments (Kukil, 2022)

Mean Average Precision (mAP) is commonly used to analyze the performance of object detection and segmentation systems. Many object detection algorithms, such as MobileNet-SSD and YOLO use mAP to evaluate their models. In segmentation tasks, mAP50 assesses how well the model can predict the shape of objects within an image, with at least 50% overlap between the predicted mask and the ground truth mask. A higher mAP50 score indicates better model performance in detecting or segmenting objects at an IoU threshold of 0.5. mAP50-95(B) and mAP50-95(M) are calculated over the standard range of IoU thresholds from 0.5 to 0.95. The "B" often stands for "bounding box" and "M" typically stands for "mask".

F1 confidence curve is a graphical representation that illustrates the relationship between the F1 score and different confidence thresholds in object detection or segmentation tasks. It visualizes how changes in the threshold affect the balance between precision and recall. This curve can be used to determine the optimal confidence threshold where the model achieves the best balance between precision and recall for segmentation tasks.

3.2. Edge Computing Devices for Robot

In the field of robotics, edge computing devices often serve as critical components that bring computing power closer to the source of data. This allows for faster processing and decision-making, which is essential for real-time operations such as navigation, object detection, and manipulation. By integrating edge computing into a robot's architecture, developers can design systems that are more responsive, efficient, and capable of operating autonomously in environments where network connectivity may be limited or unreliable.

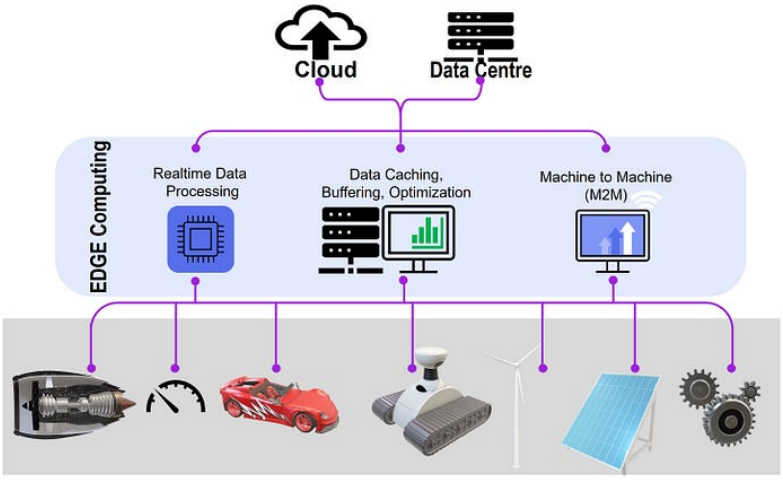
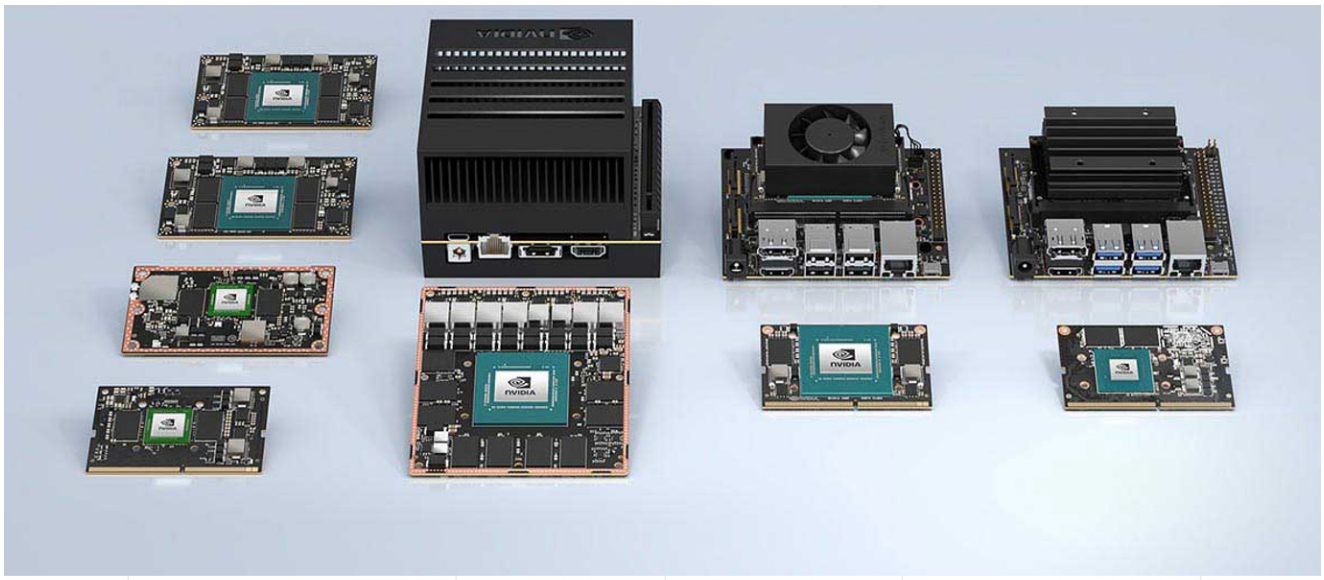


Figure 3-11: Edge Computing hierarchy (Bhandari, 2023)

3.2.1. Jetson Platform

NVIDIA Jetson is a series of embedded computing boards from NVIDIA, equipped with NVIDIA's GPU (Graphics Processing Unit), which are powerful for accelerating AI networks. The Jetson Nano is an entry-level AI platform for education and makers. It provides 472 GFLOPS (Floating Point Operations Per Second) of compute performance. This compute performance is relatively weak. Jetson Orin Nano is considered the new base version within the Jetson Orin family, offering a more affordable entry point for edge AI and robotics applications while still providing significant computational capabilities. Depending on the configuration, the Jetson Orin Nano can achieve AI performance of up to 20 TOPS (Tera Operations Per Second) or even 40 TOPS, making it capable of handling multiple concurrent AI inference pipelines. This device is ideal for small, low-power autonomous machines and is well-suited for robotics and edge AI applications.



	Jetson AGX Orin Series			Jetson Orin NX Series		Jetson Orin Nano Series		Jetson TX2 Series			
	Jetson AGX Orin 64GB	Jetson AGX Orin Industrial	Jetson AGX Orin 32GB	Jetson Orin NX 16GB	Jetson Orin NX 8GB	Jetson Orin Nano 8GB	Jetson Orin Nano 4GB	TX2	TX2 4GB	TX2 NX	Jetson Nano
AI Performance	275 TOPS	248 TOPS	200 TOPS	100 TOPS	70 TOPS	40 TOPs	20 TOPs	1.33 TFLOPS			472 GFLOPS

Figure 3-12: Jetson modules and AI performance (NVIDIA, 2024a)

NVIDIA JetPack SDK (Software Development Kit) includes Linux kernel, Ubuntu desktop environment, and a complete set of libraries for acceleration of GPU computing, multimedia, graphics, and computer vision. The Jetson models all carry a Tegra processor from Nvidia that integrates an ARM (Advanced RISC Machines) architecture CPU (central processing unit).

CUDA (Compute Unified Device Architecture)

There are 3 famous types of Processing Units: CPU, GPU and TPU (Tensor Processing Unit). A CPU is a hardware component that is the core computational unit in a server. It handles all types of computing tasks required for the operating system and applications. A GPU can break complex problems into many separate tasks and execute the same operation in parallel on many independent data elements. Designed for parallel processing, the GPU is used in a wide range of applications, including high-performance computing and rendering for graphics and video, speeding up simulations, calculations and modeling for scientific research, and artificial intelligence

training and inference. As shown in Figure 3-13, the components that make up CPUs and GPUs are analogous; they both comprise control units, ALU (arithmetic logic units), caches, and DRAM (Dynamic Random Access Memory). The main difference is that GPUs have a lot of smaller, simpler control units, ALUs, and caches.

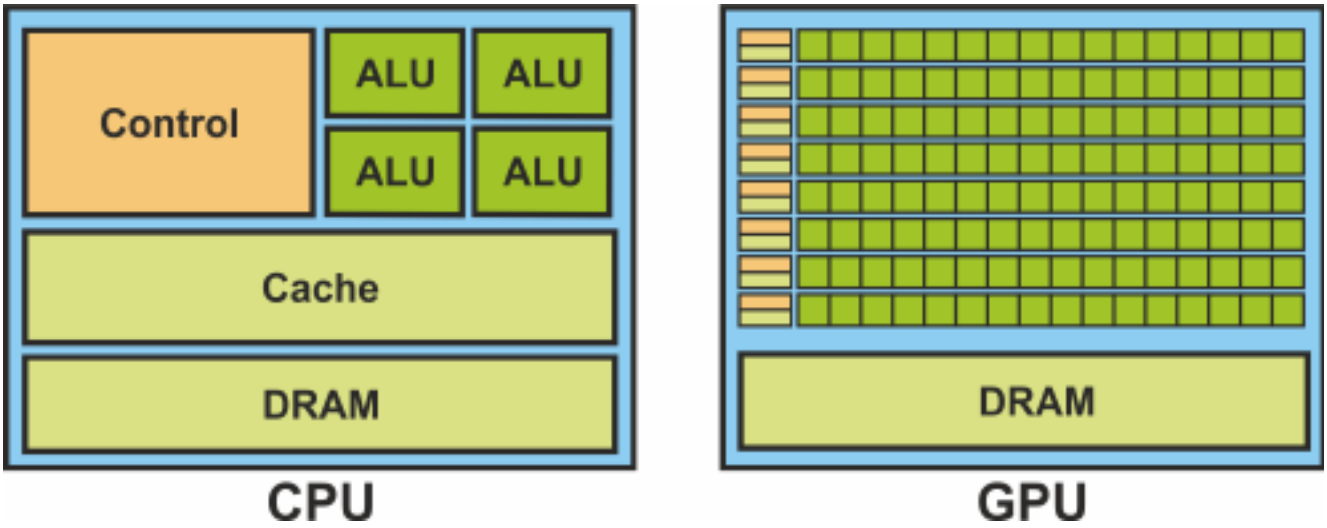


Figure 3-13: Comparison of CPU and GPU architectures (Paz-Gallardo and Plaza, 2011)

CUDA was first introduced by NVIDIA in 2006. It is a parallel computing platform and programming model that allows developers to speed up their applications by harnessing the power of GPU accelerators (Oh, 2012). CUDA provides a simple C/C++ based interface. The CUDA compiler leverages parallelism built into the CUDA programming model as it compiles program into code. While CUDA offers significant advantages, it's essential to consider its limitations as well:

- **Platform Dependency:** One of the main disadvantages of CUDA is that it is exclusive to NVIDIA GPUs, meaning applications developed with CUDA cannot run on GPUs from other manufacturers like AMD. This limitation affects the portability and scalability of CUDA-based applications. Additionally, in some regions or countries, acquiring high-end GPUs can be challenging due to supply chain issues, import tariffs, governmental regulations, or international trade disputes. These factors can make it difficult and expensive to obtain the necessary hardware. Trade sanctions or disruptions in the supply chain might lead to a shortage of GPUs, which could hinder the initiation of new projects and disrupt ongoing critical applications and research, causing significant inconvenience and losses for users.
- **Energy Consumption:** GPUs are known for high power consumption, especially under full load. Applications that heavily rely on CUDA for extended periods can lead to increased energy consumption, which might be a consideration for energy-sensitive environments.
- **High Cost:** High-end GPUs come with a steep price tag, which can be a substantial financial burden for individual developers, startups, or educational institutions. For large-scale projects requiring extensive parallel computing resources, the cost issue becomes even more critical.

TPUs are a type of application-specific integrated circuit developed by Google specifically for accelerating machine learning workloads. Unlike GPUs, which were originally designed for graphics processing and later adapted for AI. TPUs are integrated into Google's cloud computing platform, Google Cloud, providing AI researchers and developers with easy access to their capabilities. Cloud TPUs are also available on Kaggle, 20 hours per week for free. Cloud TPUs provide the versatility to accelerate workloads on leading AI frameworks, including PyTorch, JAX, and TensorFlow. Google's research shows that in AI inference tasks using neural

networks, the TPU is 15x to 30x faster than contemporary GPUs and CPUs. The TPU also achieves much better energy efficiency than conventional chips, achieving 30x to 80x improvement in TOPS/Watt measure (Jouppi, 2017).

ONNX (Open Neural Network Exchange)

ONNX is a community project that Facebook and Microsoft initially developed. The project aims to create an open file format designed to represent machine learning models in a way that allows them to be used across different AI frameworks and hardware.

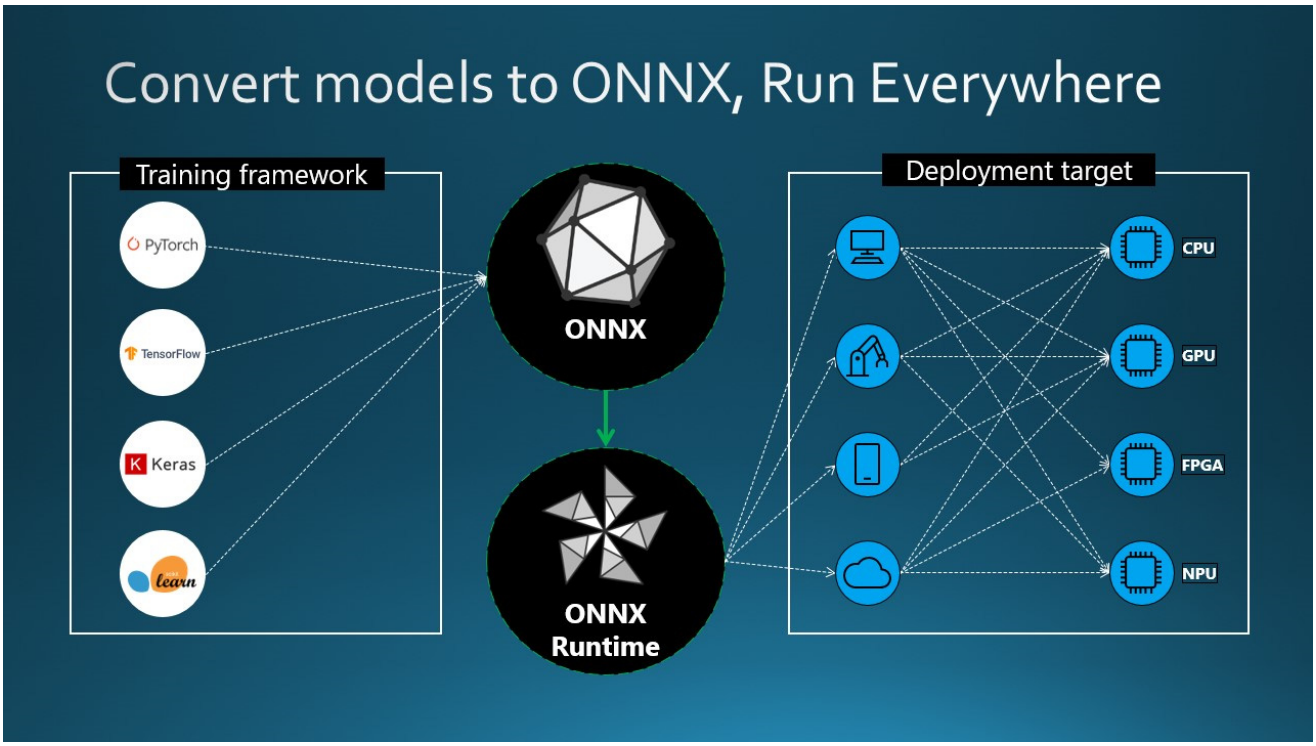


Figure 3-14: ONNX: Flexible framework for model deployment and compatibility in machine learning (Microsoft, 2024)

ONNX models can be used to transition between different frameworks seamlessly. For instance, a deep learning model trained in PyTorch can be exported to ONNX format and then easily imported into TensorFlow or TensorRT (Turing Tensor R-Engine). Alternatively, ONNX models can be used with ONNX Runtime. ONNX Runtime is a versatile cross-platform accelerator for machine learning models that is compatible with frameworks like PyTorch, TensorFlow, TFLite, scikit-learn, etc. ONNX Runtime optimizes the execution of ONNX models by leveraging hardware-specific capabilities. This optimization allows the models to run efficiently and with high performance on various hardware platforms, including CPUs, GPUs, and specialized accelerators (ONNX Runtime developers, 2018; Xu, 2021).

TensorRT (Turing Tensor R-Engine)

Real-world applications have significantly higher computing demands, require additional processing, and impose stricter latency constraints, especially with larger deep learning models. TensorRT provides superior optimizations for compute-intensive deep learning applications, making it an invaluable tool for inference. TensorRT, built on the CUDA parallel programming model, is a machine learning framework that is published by NVIDIA to run machine learning inference on their hardware. This toolkit optimizes deep learning models for

Nvidia GPUs and results in faster and more efficient operations. It's well-suited for real-time applications like object detection. ONNX enables smoother conversion from various model formats. Converting deep learning models into the TensorRT format allows developers to realize the potential of NVIDIA GPUs fully. TensorRT is known for its compatibility with various model formats, including TensorFlow, PyTorch, and ONNX, providing developers with a flexible solution for integrating and optimizing models from different frameworks. This versatility enables efficient model deployment across diverse hardware and software environments. In actual deployment, it is necessary to convert the model from PyTorch format to ONNX format, and then convert the ONNX format model to TensorRT engine format.

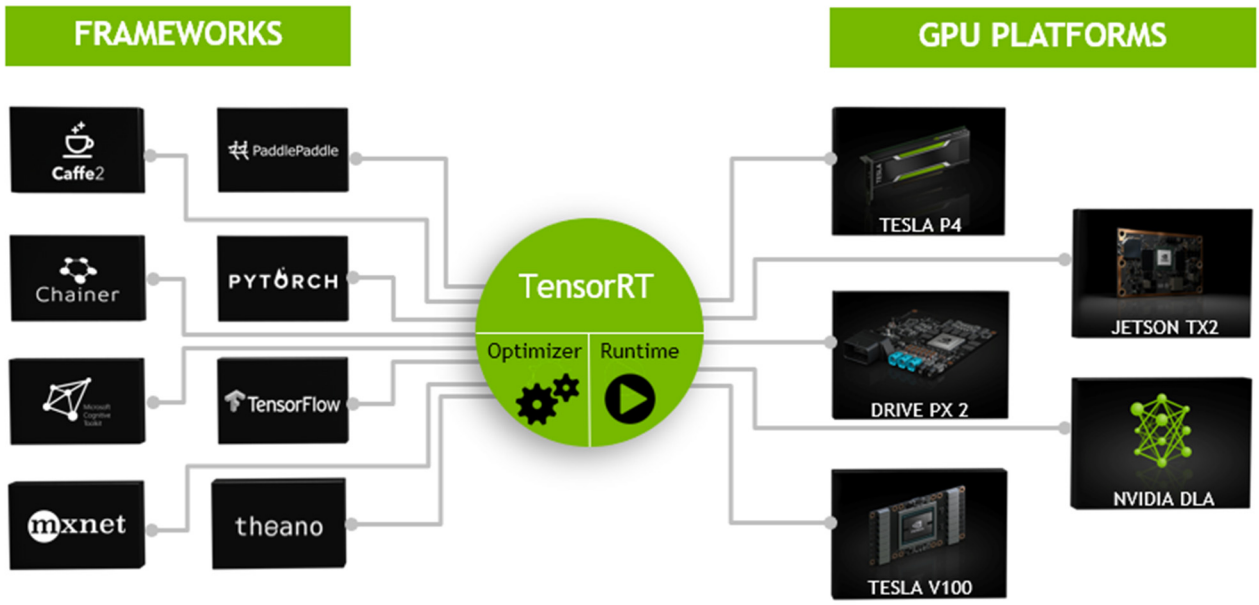


Figure 3-15: TensorRT: A programmable inference accelerator (NVIDIA, 2018)

TensorRT models offer a range of key features that contribute to their efficiency and effectiveness in high-speed deep learning inference (Ultralytics, 2024a):

- **Precision Calibration**: TensorRT allows models to be fine-tuned for specific accuracy requirements. This includes support for reduced precision formats like INT8 and FP16, which can further boost inference speed while maintaining acceptable accuracy levels.
- **Layer Fusion**: The TensorRT optimization process involves layer fusion, which combines multiple layers of a neural network into a single operation. This reduces computational overhead and enhances inference speed by minimizing memory access and computation.
- **Dynamic Tensor Memory Management**: TensorRT efficiently manages tensor memory usage during inference, reducing memory overhead and optimizing memory allocation. This leads to more effective GPU memory utilization.
- **Automatic Kernel Tuning**: TensorRT employs automatic kernel tuning to select the most optimized GPU kernel for each layer of the model. This adaptive approach ensures the model fully leverages the GPU's computational power.

From the figure below, it can be seen that with the support of TensorRT, the model's inference speed becomes faster.

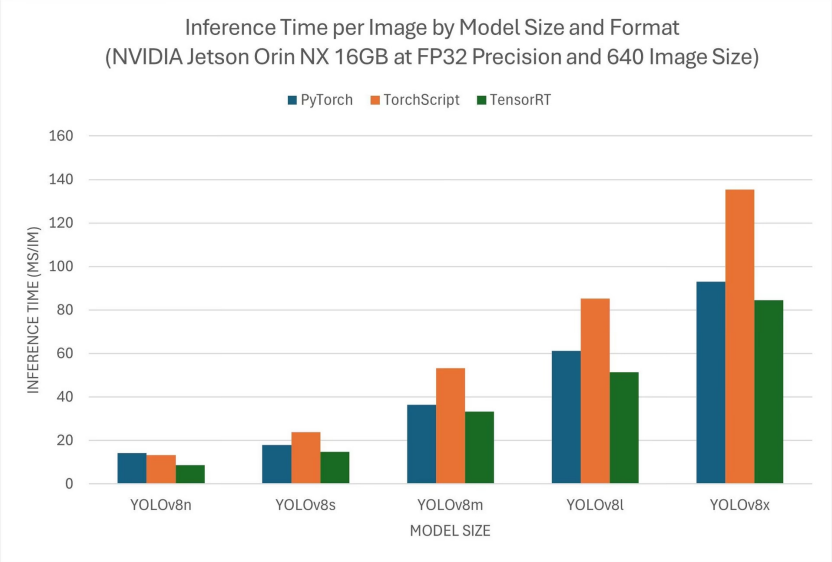


Figure 3-16: Comparison chart of the inference time per image by model size and format (Ultralytics, 2024b)

3.2.2. Arduino

An Arduino is an open-source electronics platform and is popular among hobbyists, designers, and anyone interested in creating interactive projects. From left to right, the devices shown in Figure 3-17 are the Arduino UNO R3, Arduino R4 WiFi, Arduino Nano, and Arduino Nano ESP32. The Arduino UNO R4 WiFi is one of the latest and most powerful versions released by Arduino. It integrates an ESP32-S3 chip, providing WiFi and Bluetooth capabilities, which are ideal for Internet of Things (IoT) applications. And it uses a more powerful microcontroller (ARM Cortex-M4), offering improved processing power.

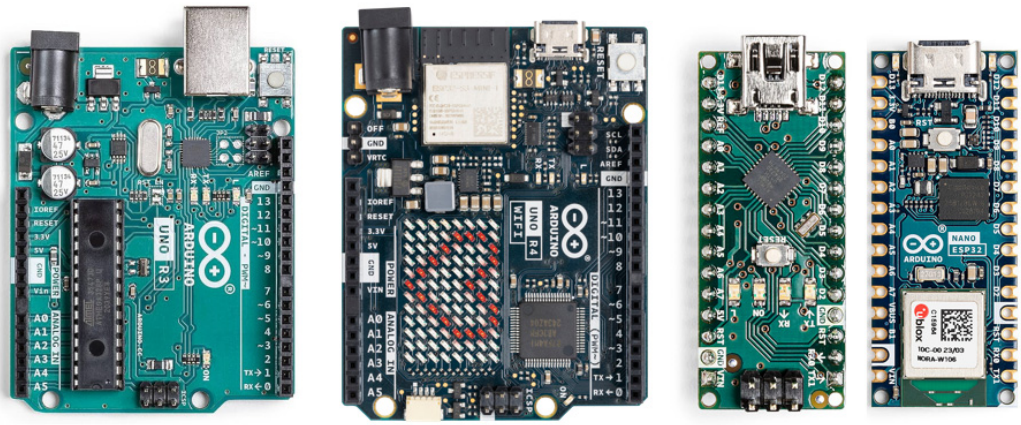


Figure 3-17: Arduino family

Arduino boards are relatively inexpensive compared to other microcontroller platforms. The Arduino UNO R4 WiFi costs only 25 Euro. The Arduino IDE (Integrated Development Environment) can run on Windows, Macintosh OSX, and Linux operating systems. Arduino UNO has 14 digital input/output pins, 6 analog inputs, a USB (Universal Serial Bus) connection, a power jack, an ICSP (In-Circuit Serial Programming) header and a reset button. Arduino has a vast number of libraries to support a wide variety of sensors. These libraries make it easier to interface with sensors and other hardware components. Therefore, this work uses Arduino to control the relay, and solenoid water valve.

3.2.3. Other Edge Computing Devices

When selecting a single-board computer for a robotics project, one must first assess the project's specific requirements. The single-board computer's capabilities should be robust enough to manage the tasks required and remain cost-effective. Equally critical is ensuring compatibility between the single-board computer and other hardware components and software tools employed in the project. The physical dimensions, or form factor, of the single-board computer also merit consideration as they will dictate the overall size of the robot. Furthermore, the availability of community support and comprehensive documentation for the single-board computer is essential, as these resources facilitate troubleshooting and further development. Ultimately, the selection of a single-board computer for a robotics project hinges on aligning with the project's defined needs and objectives.

The Raspberry Pi is a series of small single-board computers developed in the United Kingdom by the Raspberry Pi Foundation to promote the teaching of basic computer science in schools and in developing countries. The original model became operational in 2012, and the latest models have continued to be developed and released. Raspberry Pi is known for its low cost, making it accessible to a wide range of users and hobbyists. It can be used for a wide variety of projects, from simple programming tasks to more complex applications like media centers, game consoles, and even as a server. Much of the software associated with Raspberry Pi is open source, encouraging a collaborative and innovative approach to learning and development. Raspberry Pi 5 features the Broadcom 2.4GHz quad-core Arm processor, making it up to three times faster than the previous generation. With RAM (Random Access Memory) variants up to 8GB, this is the fastest, smoothest Raspberry Pi experience yet. Compared with Nvidia Jetson Nano 4G, Raspberry Pi 5 8G has 4GB more RAM memory, 1.68x faster CPU speed and 2667 MHz higher RAM speed.

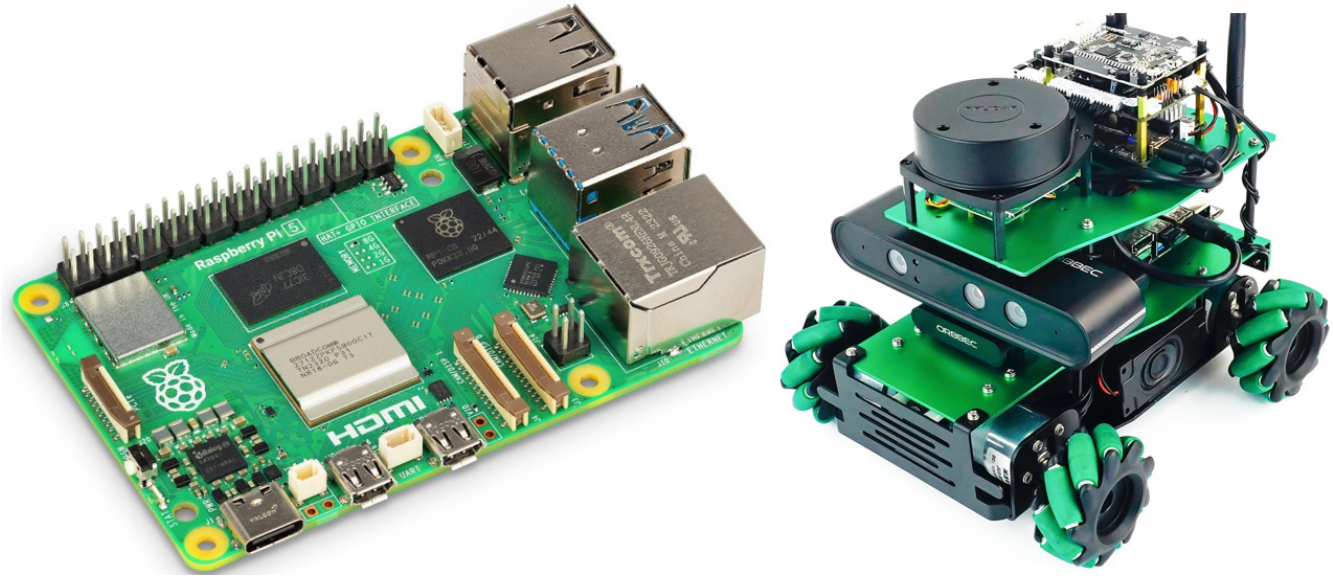


Figure 3-18: Raspberry Pi 5 (left) and Yahboom Raspberry Pi 5 robot (right)

As shown in Figure 3-19, YOLOv8 benchmarks were run by the Ultralytics team on nine different model formats measuring speed and accuracy. Benchmarks were run on both Raspberry Pi 5 and Raspberry Pi 4 at FP32 precision with default input image size of 640. The model running in NCNN format achieves the fastest inference speed. The export to NCNN format feature allows user to optimize Ultralytics YOLOv8 models for lightweight device-based applications. If running in the most common PyTorch format on a Raspberry Pi 5, the inference speed is approximately 500 milliseconds per image, which is not sufficient for real-time inference.

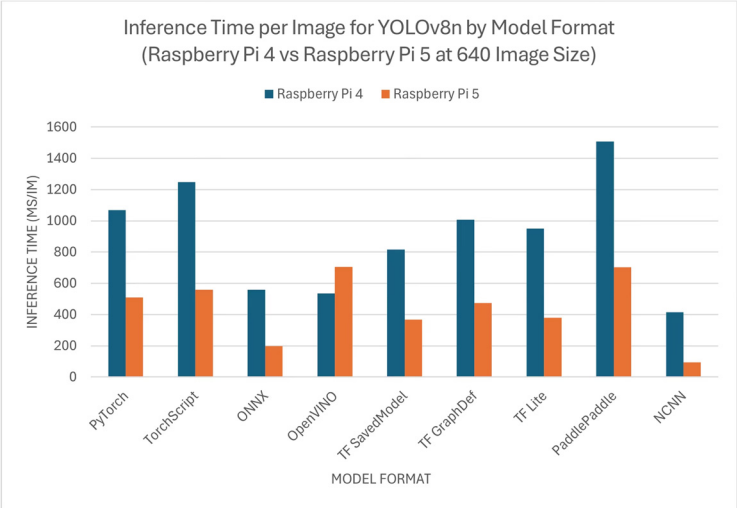


Figure 3-19: Comparison chart of inference time per image for YOLOv8n (Ultralytics, 2024c)

The Coral Edge TPU is a compact device that adds an Edge TPU coprocessor to Raspberry Pi. However, the current Coral Edge TPU runtime builds do not work with the current TensorFlow Lite runtime versions anymore. In addition to that, Google seems to have completely abandoned the Coral project, and there have not been any updates between 2021 and 2024.

The Horizon X3 is a development board featuring the Sunrise 3 AI Edge Arm processor. It includes a processor operating at 1.2 GHz and a 5 TOPS NPU (Neural-network Processing Unit) with dual "Bernoulli" BPUs (Branch Processing Units). The RDK Ultra, a high-performance development kit, boasts a 96 TOPS edge inference capability and an 8-core ARM processor. Designed to accelerate AI tasks more efficiently than GPUs and CPUs, a NPU alleviates the load on these components, enhancing overall computer performance. NPUs expedite AI machine learning tasks like speech recognition, background blurring in video calls, and object detection in photo or video editing. The BPU, developed by Horizon, is a proprietary high-performance AI processor architecture that integrates AI algorithms with hardware design. The Horizon X3 serves as the primary control development board for the RDK X3 robot. It fully supports ROS 2, offers 5 TOPS of computing power, and consumes a maximum of 10 watts. Compared to the Raspberry Pi 5, STEM32, and Arduino UNO, it provides higher computing power and lower power consumption. The machine learning model format supported by Horizon X3 is the ONNX format. To leverage this platform's BPU for accelerating inference speed, it is necessary to convert models from the PyTorch format to the format required by this platform.

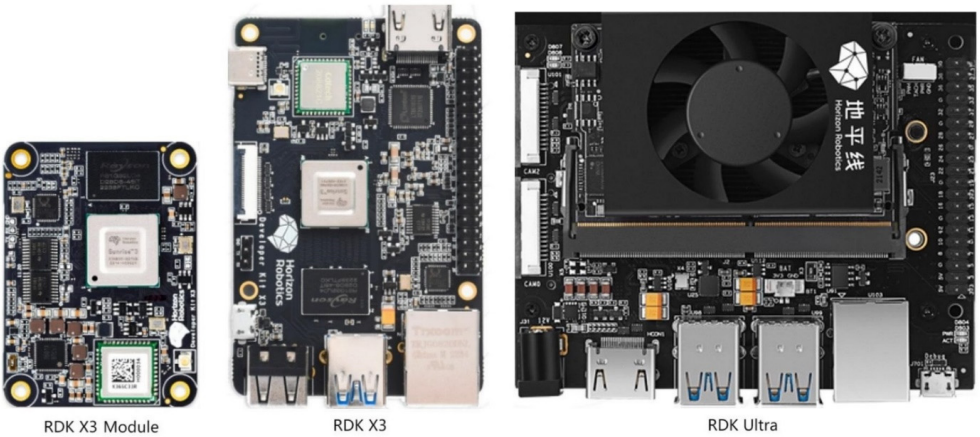


Figure 3-20: Horizon Sunrise X3 and Ultra Board (Yahboom, 2024)

hobot_llm is a Large Language Model (LLM) Node integrated into the Horizon RDK platform, enabling users to engage with LLM at the edge (Horizon, 2024). Currently, it supports two types of user interactions: direct text chat via terminal and subscribing to text messages, which are then published as text outputs.

Orange Pi AIPro is a new single board computer for AI applications that features Huawei Ascend AI quad-core 64-bit processor delivering up to 20 TOPS (INT8) or 8 TOPS (FP16) of AI inference performance. Orange Pi 5 Plus adopts Rockchip RK3588 new-generation 64-bit ARM processor, large-core main frequency up to 2.4GHz, integrated ARM GPU, embedded with high-performance 3D and 2D image acceleration modules, built-in AI accelerator NPU with a computing power of up to 6 TOPS, optional 4GB, 8GB, 16GB or 32GB memory, with up to 8K display processing capabilities. Orange Pi 5 Plus supports Orange Pi OS, the official operating system developed by Orange Pi. At the same time, it supports Android 12.1, OpenWRT, Debian 11, Debian 12, Ubuntu 20.04 and Ubuntu 22.04 and other operating systems. However, this series of single-board computers is deficient in relevant robotics tutorials and associated robots.

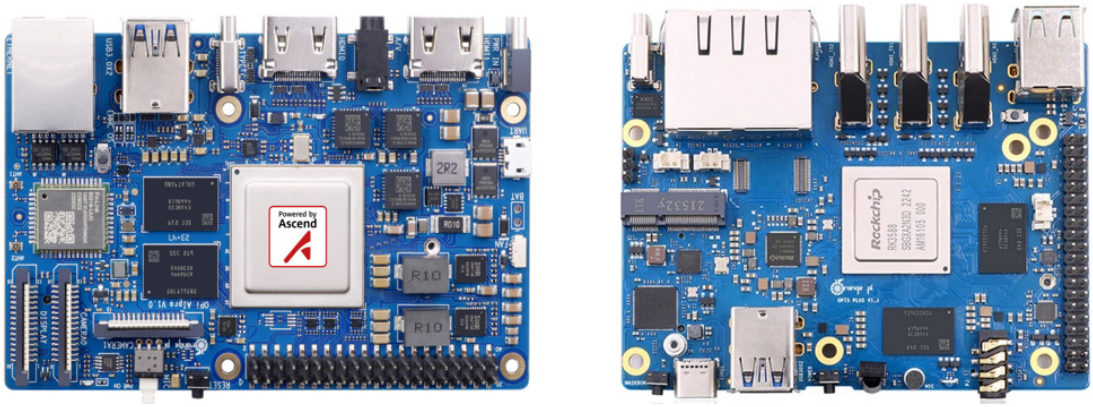


Figure 3-21: Orange Pi AIPro (left) and Orange Pi 5 Plus 8GB (right)

3.3. Robotic Technology

This section covers several fundamental yet essential technologies relevant to robotics.

3.3.1. Robot Operating System

ROS is a fully open-source operating system for robots and was developed in 2006 to provide developers with a set of open-source software frameworks, libraries, and tools to create applications for robots. It is in fact a meta-operating system, something between an operating system and a middleware. A new version, ROS 2, was released in 2017. ROS 2 brings quality of service, support for embedded systems and real-time scenarios. ROS 2 is designed to be cross-platform, the officially supported platforms for ROS 2 include Ubuntu, Debian, macOS, Windows and Raspberry Pi, etc. A convenient alternative method for configuring ROS 2 is through the utilization of Docker. By employing Docker, individuals can operate ROS 2 within a containerized ecosystem, streamlining the configuration process and fostering uniformity across various platforms.

Isaac ROS is a suite of hardware-accelerated, high-performance, low-latency ROS 2 packages designed for developing autonomous robots using the power of Jetson and other NVIDIA platforms (NVIDIA, 2024b). It includes traditional ROS 2 nodes and Isaac-specific GPU-accelerated packages to boost hardware performance. NVIDIA Omniverse Isaac Sim is a robotics simulation toolkit for the NVIDIA Omniverse platform. It offers

essential features for building virtual robotic environments and conducting experiments. This toolkit provides researchers and practitioners with the necessary tools and workflows to create accurate simulations and synthetic datasets. Isaac Sim supports navigation and manipulation applications through ROS 1 or ROS 2 and simulates sensor data from devices like cameras, LiDAR, and IMU (Inertial Measurement Unit) (NVIDIA, 2024c). It aids in various computer vision tasks, including domain randomization, ground-truth labeling, segmentation, and generating bounding boxes.

ROS can be implemented in any modern programming language. The primary languages used for ROS development are C++ and Python, but there are also experimental libraries available for other languages such as Java, Lua, and Lisp (ROS Wiki, 2018). ROS's communication mechanism is essential for developing complex robotic systems.

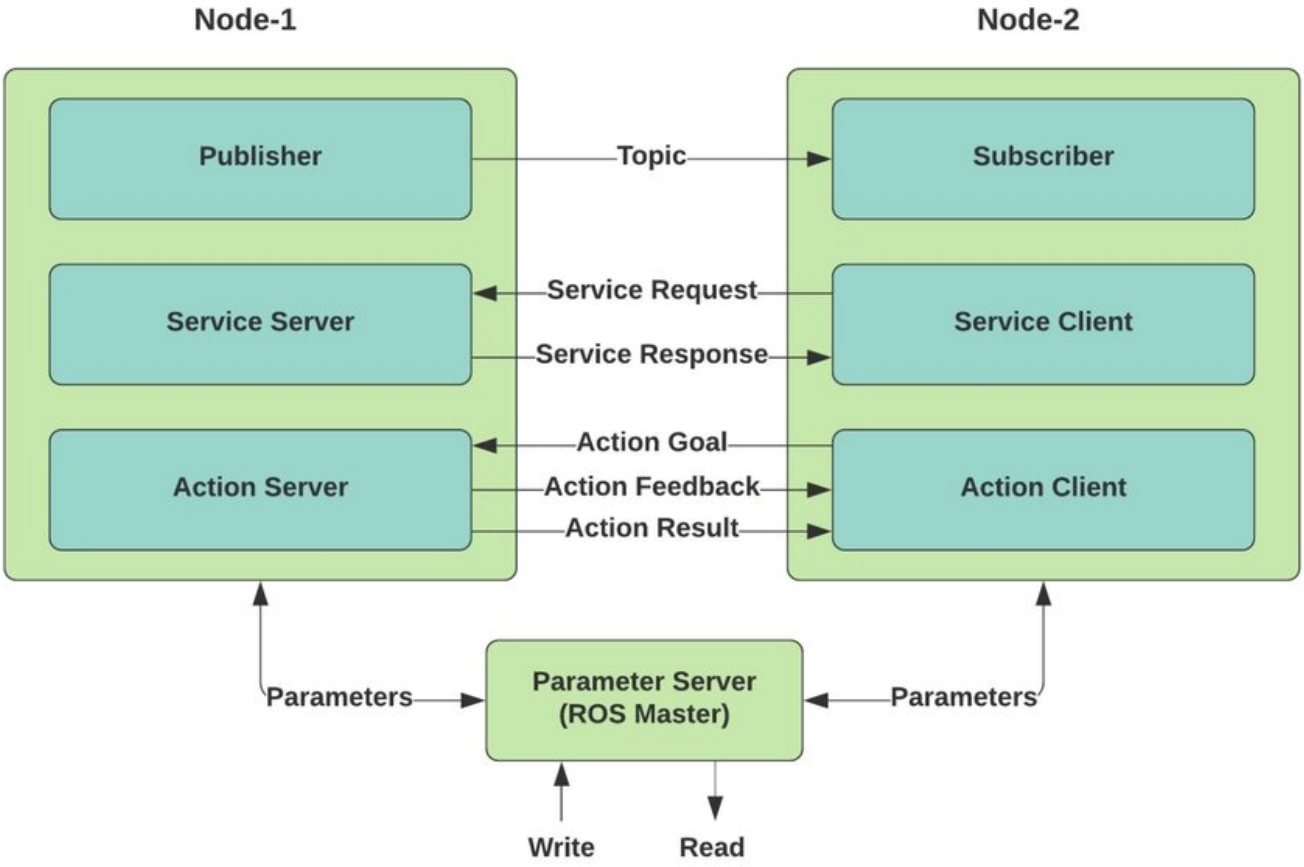


Figure 3-22: Schematic of ROS communication mechanism (RS et al., 2023)

As shown in Figure 3-22, there are some key components of ROS's communication mechanism:

- **Nodes:** Nodes are independent executable programs that perform certain tasks within the ROS system.
- **Topics and Services:** Topics are the channels through which nodes can publish and subscribe to data. The sending of messages over a topic is called publishing, and receiving the data through a topic is called subscribing. A publisher can send messages to multiple subscribers simultaneously. Service communication is one-to-one. A client node sends a request to a service, and the service node processes the request and returns a response.
- **Actions:** Actions are an asynchronous communication mechanism, allowing nodes to communicate with each other. A client node sends a goal to an action server, and the action server executes the goal and

returns the result. Action communication is one-to-one and can provide feedback on the progress and result of the goal.

- **ROS Master:** ROS Master is responsible for managing and coordinating all nodes in the system. Without starting ROS master, the nodes will not find each other and send messages.
- **Messages:** A message can consist of primitive data types, such as integers, floating points, and Booleans. The ROS messages flow through the ROS topic, which can only send/receive one type of message at a time.

ROS includes a variety of useful tools that facilitate the development, simulation, visualization, and operation of robots. Launch files are very common in ROS, they provide a convenient way to start up multiple ROS nodes and a master. Gazebo is a powerful simulation environment for ROS that provides a realistic platform to test robots and algorithms before deploying them in the real world. MoveIt consists of various packages used for manipulating robotic arms. It encompasses functionalities such as motion planning, manipulation, control, inverse kinematics, 3D perception, and collision detection. Qt Toolbox provides graphical interfaces for debugging and monitoring ROS nodes, topics, services, and parameters. TF (Transforms) is a powerful tool for keeping track of multiple coordinate frames over time and for transforming points, vectors, and other data between these frames. RViz (Reconstruct Visualization Interface) is a highly extensible visualization tool that allows users to create custom interfaces for their robots through its plugin mechanism. The built-in displays for laser data and image data visualization are examples of the official plugins provided by ROS, and these can be extended or replaced with custom ones to suit specific needs. RViz has several key features that make it a powerful tool for robot developers and researchers. Here are some of the main features:

- **Navigation Map Generation:** Utilizing data from Simultaneous Localization and Mapping (SLAM) or other mapping algorithms, RViz can create and exhibit detailed two-dimensional or three-dimensional environmental maps.
- **Plugin System:** RViz is built with a plugin system that allows developers to create and add new functionality through custom displays, tools, and panels.
- **3D Viewing:** RViz provides a 3D view of the robot's environment, allowing users to visualize data such as maps, sensor data, and robot models in a realistic and interactive way.
- **Navigation Stack Integration**: RViz includes tools for working with the navigation stack, such as 2D Nav Goal and 2D Pose Estimate, which are used for setting goals and initial poses for robots.

As shown in Figure 3-23, the left panel is the Displays panel. It has a list of plugins. These plugins enable user to view sensor data and robot state information. The color red designates the X-axis, with the area directly in front of the robot representing the positive direction of the X-axis. Green denotes the Y-axis, with the positive direction extending from the robot's right to its left side. Blue signifies the Z-axis, where the positive direction is upward from bottom to top. The Laser Scan display shows data from the *sensor_msgs/LaserScan* message. In essence, this functionality enables the display of real-time LiDAR scanning outcomes, allowing for comparison with an existing map.

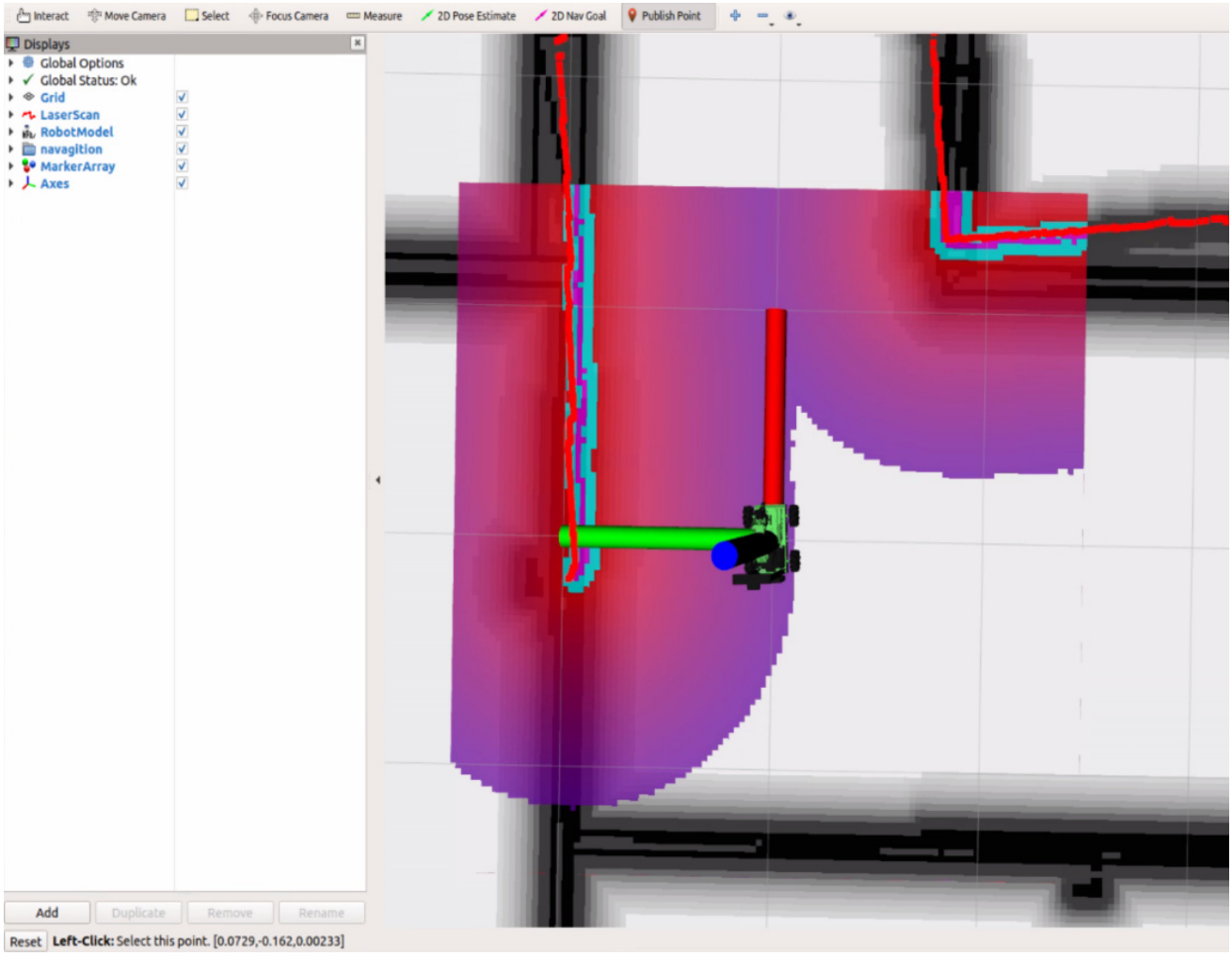


Figure 3-23: Visualizing the robot model and laser data in RViz

The Publish Point tool lets user select an object in the visualizer and the tool will publish the coordinates of that point based on the frame at the bottom. The 2D Nav Goal is a tool in RViz that allows users to set navigation goals for robots on a two-dimensional map. Users can set a goal by clicking on a location on the map and dragging to select the orientation. This goal is then published to a ROS topic with a message type of *geometry_msgs/PoseStamped*. The robot's navigation system can then receive this goal and begin to plan a path to reach that location.

In certain applications like automated patrolling, there's a demand for a robot to receive multiple navigation goals simultaneously and to sequentially visit these goals. This capability is not available in RViz. The Navi_waypoints plugin, which is an extension of the RViz Display class, introduces a user interface for interacting with waypoints (Zou, 2024). This plugin enables users to easily add multiple navigation targets, and to adjust their positions and heading angles interactively using Interactive Markers. For automated patrolling, the plugin offers a Loop mode that connects the first and last targets, facilitating cyclic navigation. Users can also specify whether the navigation between targets should be continuous or paused. For ease of reuse, the plugin allows users to save their navigation targets in YAML files, which can be quickly loaded for future use. Additionally, users have the option to set whether each navigation target should be approached continuously or with a pause, and these settings can be stored in a YAML file for direct loading during subsequent uses.

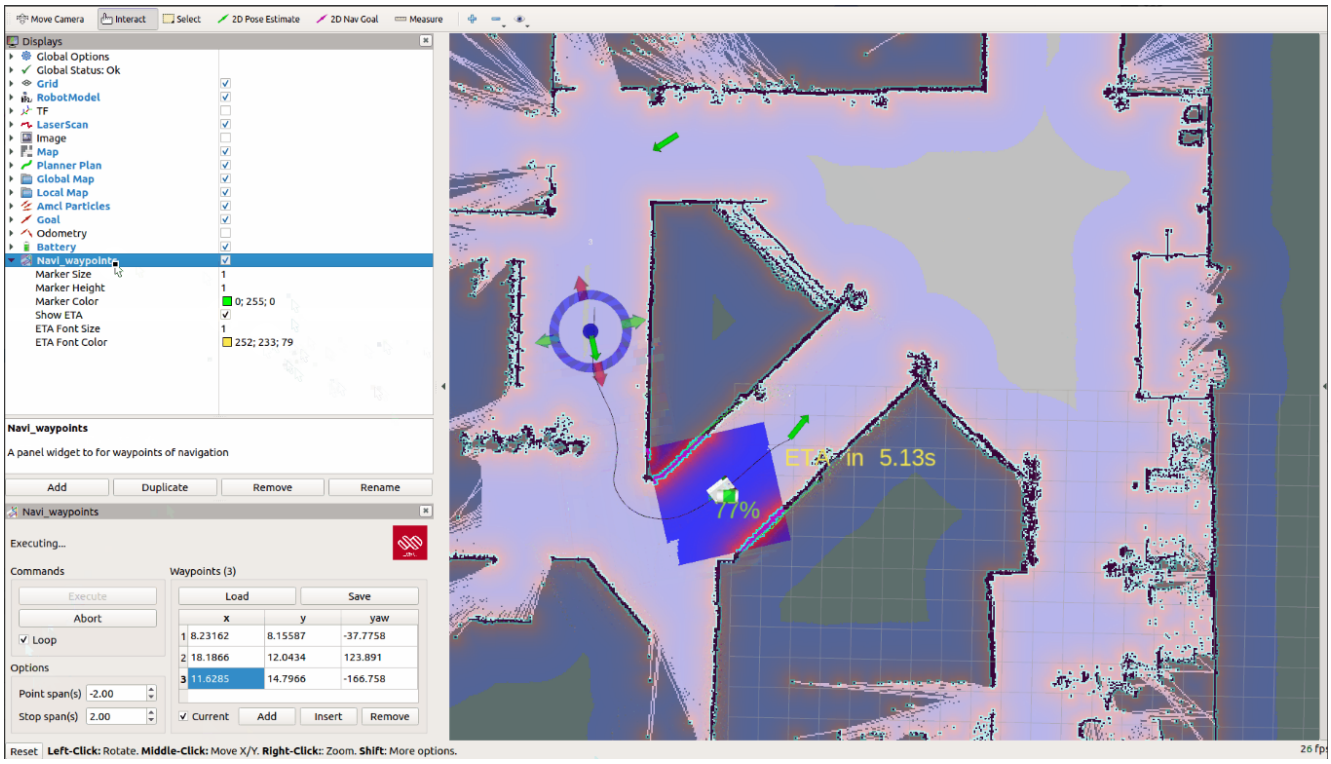


Figure 3-24: GUI of RViz plugin Navi_waypoints (Zou, 2024)

3.3.2. Robot Mapping

In environments where pre-existing maps are unavailable, SLAM (Simultaneous Localization and Mapping) enables robots to understand and navigate their surroundings by simultaneously creating a map and determining their position within it in real-time. This capability is crucial for applications in unknown or dynamic environments, such as autonomous consumer robotics like drones and vacuum cleaners, as well as self-driving cars. With significant advancements in computer processing speeds and the availability of low-cost sensors such as cameras and laser range finders, SLAM algorithms have become practical for a growing number of fields. SLAM involves two primary types of technology components. The first is sensor signal processing, or front-end processing, which relies heavily on the types of sensors used. The second is pose-graph optimization, or back-end processing, which is independent of the specific sensors involved.

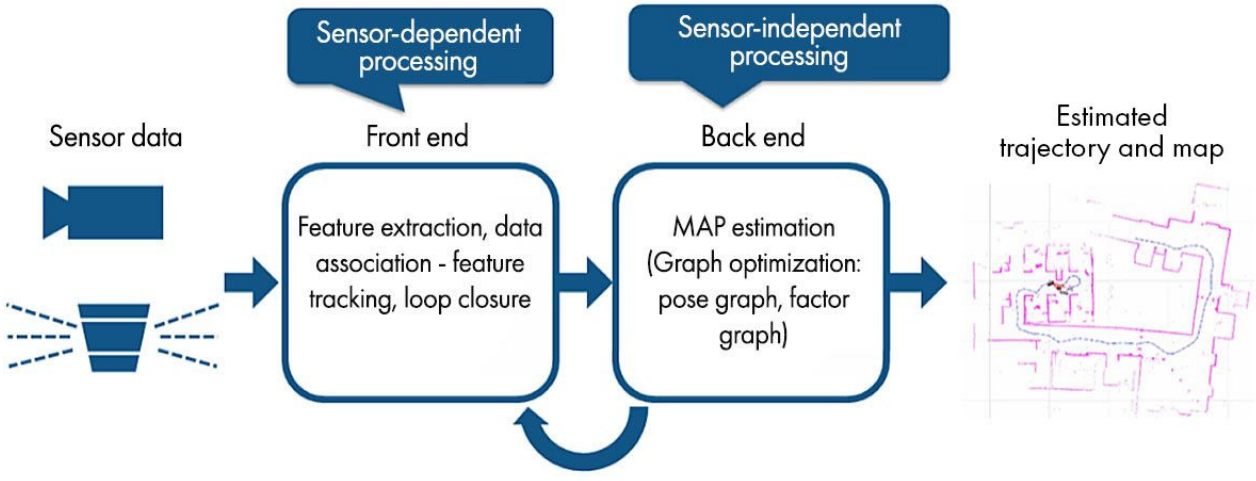


Figure 3-25: SLAM processing flow (MathWorks, 2024)

Camera-based SLAM uses high-resolution images to detect features through computer vision algorithms, while LiDAR-based SLAM relies on depth and geometry information to identify objects and create realistic 3D maps. Visual SLAM determines a device's position and orientation relative to its surroundings while simultaneously mapping the environment using only visual inputs from a camera. Feature-based visual SLAM typically tracks points of interest across consecutive camera frames to triangulate the camera's 3D position, creating a 3D map in the process. By analyzing pixel color variations, the robot can identify various objects in the scene. Computer vision algorithms like SIFT (Scale-Invariant Feature Transform) and ORB (Oriented FAST and Rotated BRIEF) are commonly employed to detect these features. In contrast, LiDAR-based SLAM employs a laser sensor to produce a 3D map of the environment. LiDAR measures the distance to objects by emitting light pulses that bounce back to the sensor, recording the Time of Flight (ToF). Known for its speed and accuracy, LiDAR is ideal for diverse environments and conditions. The highly accurate point cloud generated by LiDAR is beneficial across various industries.

Multi-sensor SLAM leverages a combination of sensors, such as cameras, IMUs (Inertial Measurement Units), GNSS, LiDAR, radar, and others, to enhance the precision of SLAM algorithms. By combining the strengths of different sensors and compensating for their individual limitations, multi-sensor SLAM achieves superior performance. For instance, while cameras provide detailed visual data, they may struggle in low-light or high-speed scenarios; LiDAR maintains performance in various lighting conditions but may have difficulty with certain surface materials. Integrating data from multiple sources, multi-sensor SLAM offers a more reliable solution than single-sensor methods.

The development of the robot for this work is similar to that of an automatic floor vacuum robot, as both need to go through the entire floor to detect ground conditions. Robot vacuums use built-in navigation systems to cover the floor area, ranging from simple random paths to sophisticated algorithm-based intelligent path planning. To clean effectively, vacuum robots first need to map the home. Vacuum robots use various technologies to create these maps, including camera-based mapping, LiDAR-based mapping, and gyroscope or accelerometer mapping. SLAM is often integral to both local and global path planning processes. Local and global path planning are crucial concepts in the fields of robotics and navigation systems. They are used to define strategies for moving from a start point to an end point within an environment. Global path planning, also known as global route planning, is concerned with finding a path from the start position to the target position given a comprehensive map of the environment. The map includes information about all the static obstacles and the layout of the environment. Local path planning with LiDAR focuses on real-time navigation and obstacle avoidance. LiDAR provides up-to-the-moment data about the robot or vehicle's immediate surroundings, enabling it to make quick adjustments to its path to avoid collisions. In practice, LiDAR-based local and global path planning are integrated for effective navigation. The global path provides a long-term route towards the destination, leveraging the detailed environmental map generated or updated by LiDAR. As the robot follows this route, the local path planning system uses real-time LiDAR data to make immediate adjustments for obstacle avoidance and to adapt to dynamic changes in the environment. This dual approach ensures that the system is both efficient, by following an optimized global path, and safe, by dynamically adjusting to real-time conditions.

3.3.3. Robot Localization

Robot localization techniques refer to methods for determining the exact location and orientation of a robot in its environment. Here are some common robot positioning technologies:

- **Odometry:** Estimates the robot's movement distance and direction by tracking the rotation of its wheels or mobile components.
- **IMU:** Estimates the robot's motion and orientation by measuring acceleration and rotational velocity.

- **Global Navigation Satellite Systems (GNSS):** Uses satellite signals to determine the robot's global location in outdoor environments. GNSS signals can be weak or unreliable in indoor environments and certain outdoor locations, which can lead to significant positioning errors. This can be particularly problematic when trying to accurately locate small objects such as cracks. Instead of relying solely on satellite data like traditional GNSS systems, RTK (Real-Time Kinematic) GNSS systems incorporate supplemental data from a nearby base station, enhancing the precision of the GNSS output. This results in positioning data with centimeter-level accuracy, which is highly suitable for a wide range of applications. However, RTK can be relatively expensive due to the need for specialized hardware and the establishment of reference stations or the use of commercial networks for correction data.
- **LiDAR:** Creates a map of the environment by emitting laser beams and measuring the time it takes for the reflections to return and performs SLAM. It is particularly useful for robotics and autonomous systems because it provides high-resolution, accurate distance measurements that are essential for navigation and obstacle avoidance. It is also used in conjunction with other sensors, like GNSS and IMUs, to improve the overall positioning accuracy of a robot or autonomous vehicle. It's important to note that LiDAR systems can be affected by environmental factors such as fog, smoke, and dust, which can reduce their effectiveness. However, advancements in LiDAR technology are continually addressing these challenges, making it a robust choice for many applications. The process of LiDAR localization includes these steps: (1) The LiDAR sensor continuously scans the environment, emitting laser beams and measuring the distance to nearby objects. This distance data is represented as a 2D point cloud or a laser scan, showing the robot's surroundings. (2) A pre-built map of the environment is required, typically created using a technique like SLAM. The robot's LiDAR data is compared to this map to find a matching section of the environment. This comparison helps determine where the robot is within the map. (3) The most common algorithm for this purpose is Adaptive Monte Carlo Localization (AMCL). AMCL uses a particle filter to represent possible positions of the robot in the map. Each particle represents a hypothesis of where the robot might be. The LiDAR data helps weigh these hypotheses based on how well the observed data matches the expected data from the map. (4) Based on the highest-weighted particles, the robot's most likely position in the map is estimated. RViz visualizes this estimated position along with the map, LiDAR data, and the robot's pose. The robot's position is updated in real-time as it moves and scans its environment.
- **Visual SLAM:** Combines visual odometry and machine vision techniques for localization and map building in unknown environments.
- **Optical Tracking Systems:** Uses optical markers or visual markers to track the robot's position, often used in research and high-precision applications.
- **Tactile and Proximity Sensors:** Provides positioning information through physical contact or proximity to obstacles.

Selecting the appropriate positioning technology usually depends on the robot's application scenario, required precision, cost, and environmental conditions. In practical applications, a combination of multiple technologies is often used to achieve more reliable positioning.

3.3.4. Related Robots

Some researchers have developed robots specifically for detecting and assessing cracks. These robots often utilize advanced technologies such as computer vision and machine learning to identify cracks with high precision. These robots can be used as a reference for the robot developed in this work.

Genova et al., (2024) introduced an adaptive, autonomous system designed for surface crack detection and repair, integrating advanced robotic and sensing technologies. The system employs the Intel RealSense D435 camera for identifying cracks, a laser scanner for precise measurements, and an extruder with a pump for material deposition. A novel validation process, using 3D-printed crack specimens, simulates real-world cracks and ensures repeatability in testing. This research demonstrates that adaptive crack filling systems are more efficient and effective compared to fixed-speed methods, with experimental data confirming superior precision and consistency. The system features the DeepLabV3+ network for crack segmentation, a UR10e robotic arm for operation, and a Raspberry Pi as its processor.

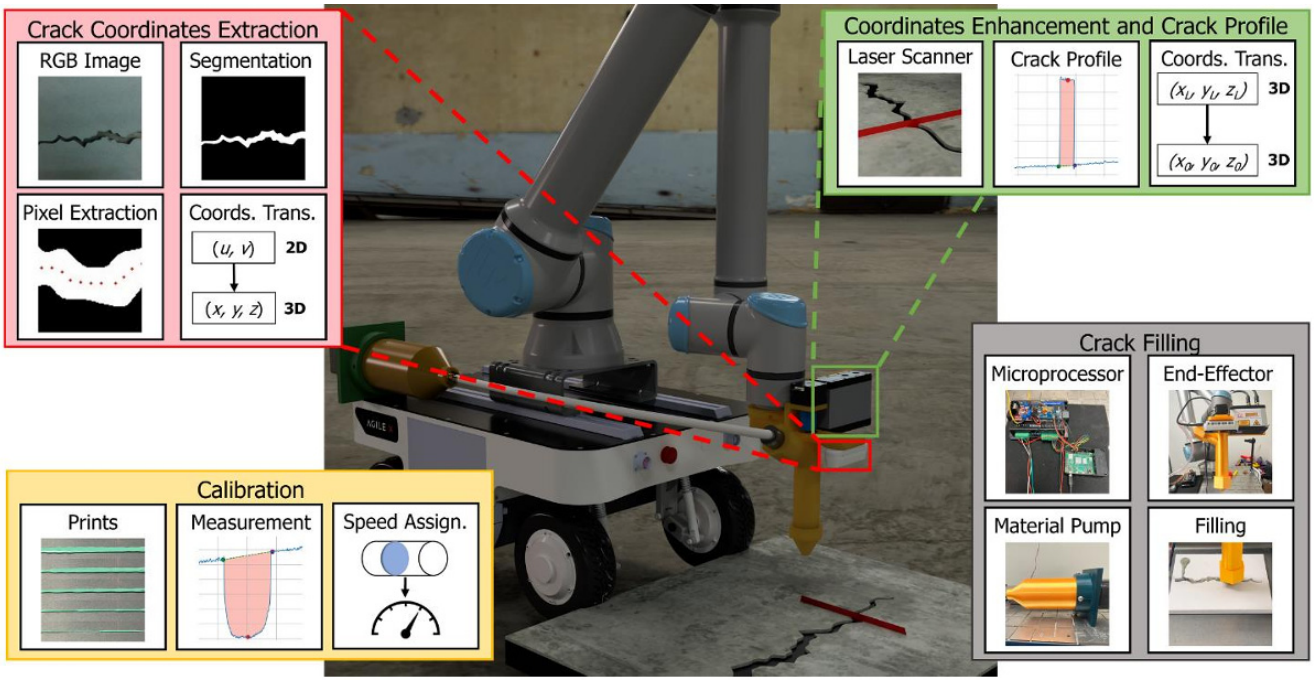


Figure 3-26: System overview of the vision-based adaptive robotics for autonomous surface crack repair (Genova et al., 2024)

Khan et al., (2023) developed a robotic system for real-time automated data collection and analysis. The system autonomously navigates pavements, capturing surface images. A deep-learning-based semantic segmentation framework, named RCDNet, was introduced and deployed on the robot's onboard computer to detect cracks in the captured images. The robot completed an inspection of a 3 m × 2 m area in 10 minutes and a 2.5 m × 1 m area in 6 minutes, demonstrating a significant reduction in the time required for manual inspections. The vision system of the robot utilizes a Logitech c922 Pro HD Stream Webcam. The Graphical User Interface (GUI) communicates with the primary hardware controller via socket communication, where the robotic platform acts as the server and the host PC functions as the client. The primary controller is an Intel Nuc mini PC with six cores, 8GB of RAM, and runs on Windows 10.

(**a**) Image collection in indoor (**b**) Image collection in outdoor

Figure 3-27: Automated navigation and data collection in the indoor and outdoor environment (Khan et al., 2023)

Ramalingam et al., (2021) introduced a deep learning-based framework for pavement inspection using a self-reconfigurable robot called Panthera. The framework employs SegNet to distinguish the pavement from other objects. A Deep Convolutional Neural Network (DCNN) is utilized to detect and localize pavement defects and garbage. Additionally, a Mobile Mapping System (MMS) is used for geotagging these defects. The system was implemented on the Panthera robot, equipped with NVIDIA GPU cards. Experimental results demonstrated that the technique achieves high accuracy in detecting pavement defects and litter. The results highlight the suitability of the system for real-time garbage detection and future deployment in sweeping or cleaning tasks. The self-reconfigurable pavement sweeping robot, Panthera, can adjust its shape between a contracted and extended state, as shown in Figure 3-28 (a) and Figure 3-28 (b), respectively. This capability allows the robot to navigate varying pavement widths, avoid static obstacles, and adapt to changes in pedestrian density.

a) Panthera on pavement in contracted state b) Panthera on pavement in extended state

Figure 3-28: Self-reconfigurable pavement inspection and cleaning robot Panthera (Ramalingam et al., 2021)

As shown in Figure 3-29, the primary system of robot Panthera uses GPU for running the SegNet framework and NVIDIA Jetson nano GPU embedded board for run the inspection CNN module. Other control modules, such as those for localization, reconfiguration, and cleaning device control, are managed by an Arduino-Mega microcontroller. These control modules are configured as ROS slaves to communicate with the primary control system. The slave units manage the various sensor interfaces and generate the necessary control signals, including Pulse Width Modulation (PWM) signals, to drive the motors. Panthera sensory system consists of various sensing

devices include mechanical limit switch, Intel Realsense depth sensor, absolute encoder and GNSS (Global Navigation Satellite System) tracking module.

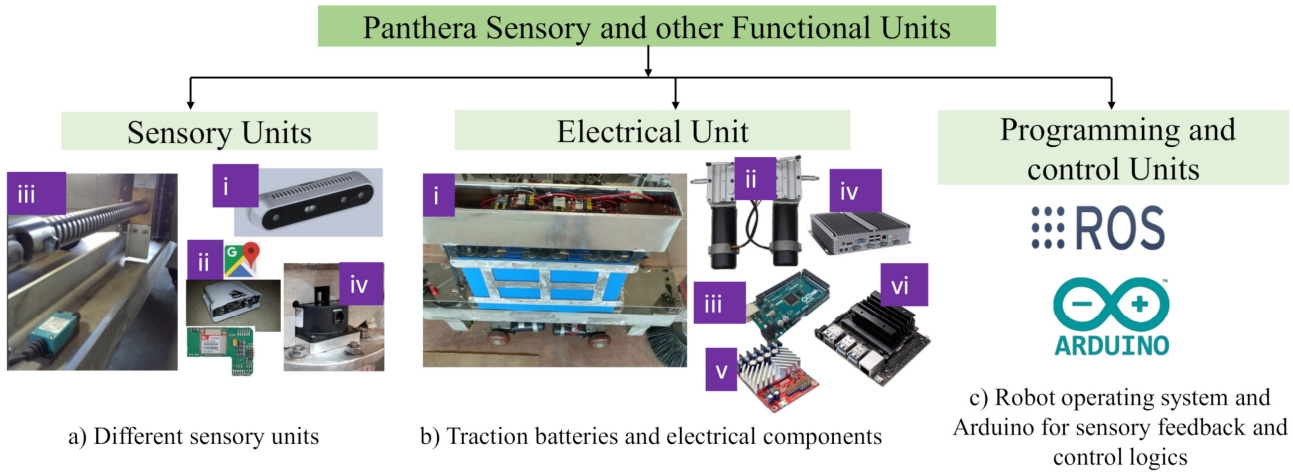


Figure 3-29: Sensory, electrical, programming and control units in Panthera (Ramalingam et al., 2021)

As shown in Figure 3-30, Mobile Mapping System is used for tracking the location of the defects. It is executed in the primary control unit and utilizes three data sources for defect localization: the physical distance data estimated via the Intel Realsense camera, and data from two hardware modules such as odometric distances by wheel encoder and GNSS data that supply latitude and longitude information. The data is transmitted to a remote monitoring unit using a 4G (Fourth Generation) wireless communication module.

Figure 3-30: Independent trial for the mobile mapping system and the defect identification: (a) Defect location on global map and (b,c) detected defects and patches (Ramalingam et al., 2021)

The detection of surface and subsurface defects is important for ensuring the structural reliability of airport runways. Gui and Li (2020) developed an automated system for data collection and analysis on airport runways using a robotic platform. This system integrates a camera for surface inspection and Ground Penetrating Radar (GPR) for subsurface assessment. To provide a comprehensive view of the entire runway, the camera and GPR data are precisely aligned to generate a continuous mosaic for visualization across large runway spans.

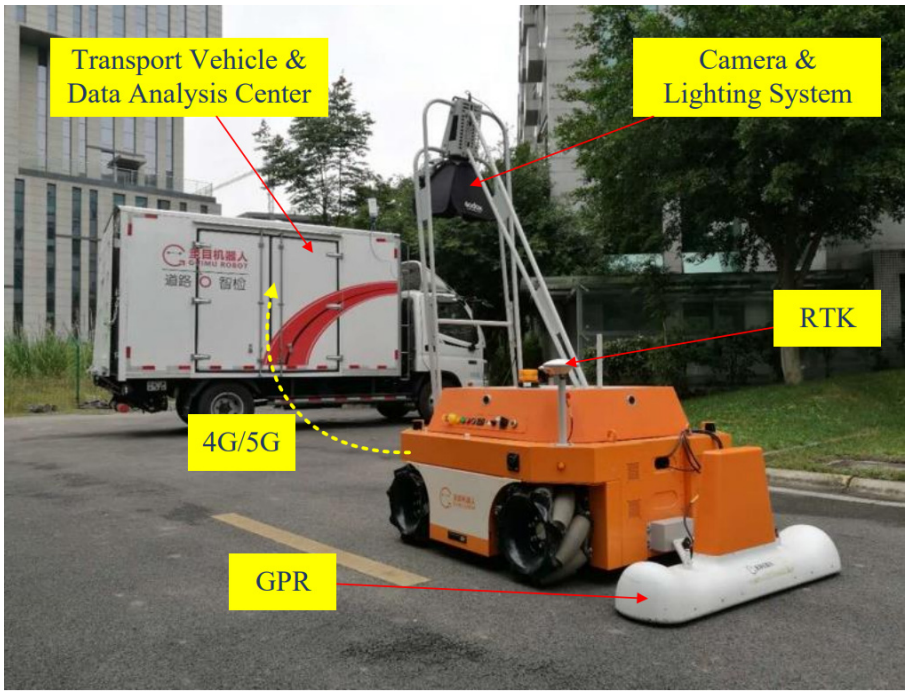


Figure 3-31: Sensor configuration of robot system (Gui and Li, 2020)

The motion control of all these robots mentioned above needs to be controlled either by humans or through scripts. However, in recent years, large language models (LLMs) are advancing rapidly, they have demonstrated an extensive capacity for actionable knowledge, which can be leveraged for robotic manipulation through reasoning and planning. There has been a shift towards utilizing LLMs to control robot movements. These models can interpret natural language instructions and translate them into actions, opening up new possibilities for more intuitive and flexible robot control. Stanford Vision and Learning Lab proposed VoxPoser, a method to generate robot trajectories for manipulation tasks from natural language instructions (Huang et al., 2023). It begins with recognizing that LLMs excel at inferring affordances and constraints from free-form language instructions. More importantly, their code-generation capabilities facilitate interaction with vision-language models, enabling the generation of 3D value maps that anchor inferred knowledge within the agent's observation space. It should be noted that in this experiment, the image recognition algorithm used by the robot is not based on traditional CNNs but instead utilizes a vision-language model.

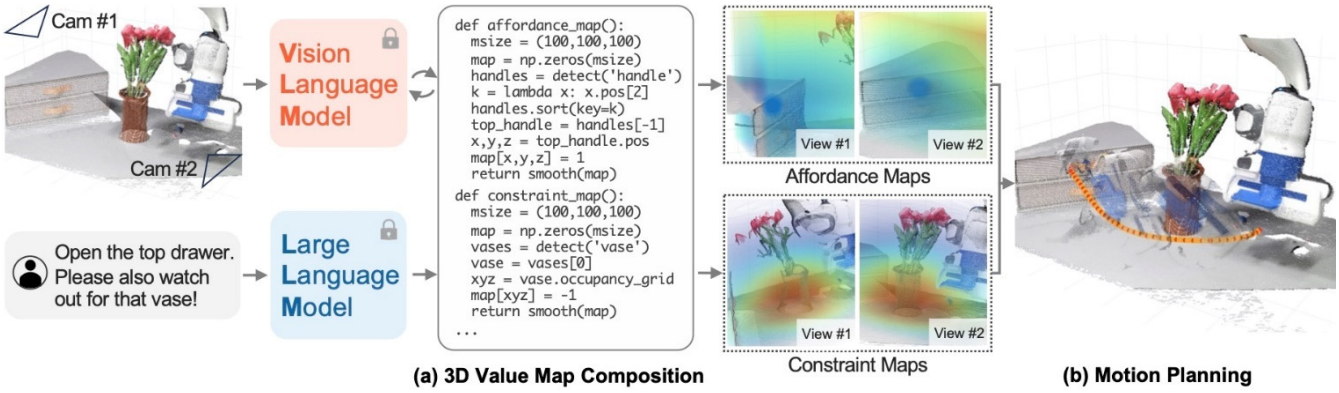


Figure 3-32: (a) LLMs generate code, which interacts with VLMs, to produce a sequence of 3D affordance maps and constraint maps grounded in the observation space of the robot. (b) The composed value maps then serve as objective functions for motion planners to synthesize trajectories for robot manipulation (Huang et al., 2023)

Robotic Transformer 2 (RT-2) is also a significant advancement in the field of robotics and artificial intelligence, developed by Google DeepMind. It is a novel vision-language-action (VLA) model that learns from both web and robotics data, and translates this knowledge into generalised instructions for robotic control, while retaining web-scale capabilities (Chebotar and Yu, 2023).

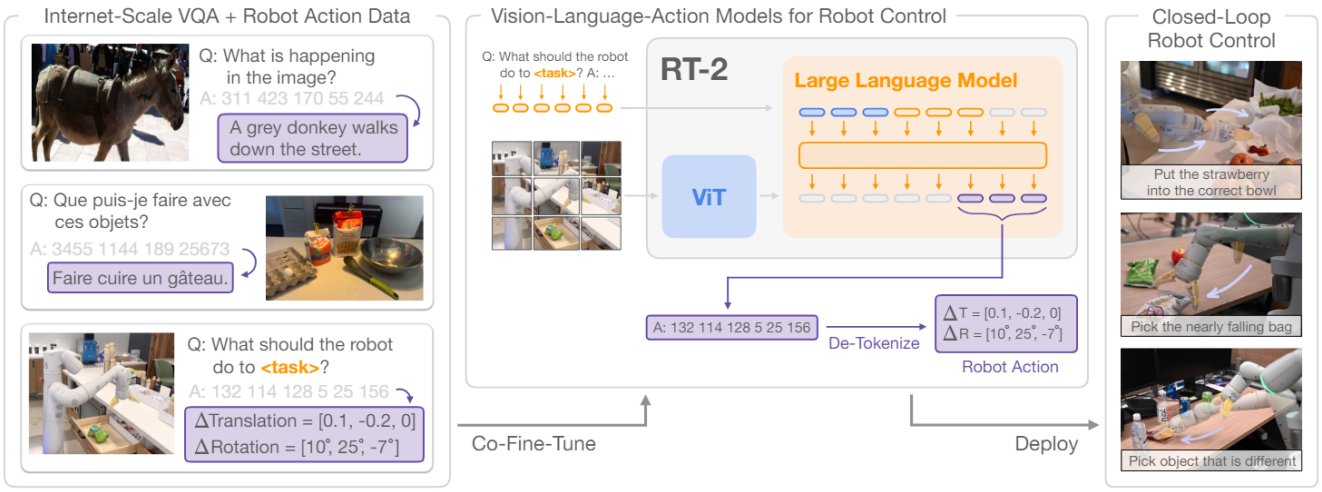


Figure 3-33: Vision-language model is trained with Internet-scale vision-language datasets. During inference, the text tokens are de-tokenized into robot actions, enabling closed loop control (Chebotar and Yu, 2023)

With the continuous advancement of multimodal intelligent models, such as LLMs and VLMs, robots are expected to become increasingly intelligent. Deploying multimodal models on robots will be a significant trend in robotic development.

3.4. BIM-GIS Integration using Cesium

Due to global warming, the occurrence of natural disasters and extreme weather events is becoming more frequent, leading to increased environmental damage to concrete structures. Therefore, the building health management such as concrete crack inspection and repair should not be limited to the building itself but should also consider the surrounding environment for comprehensive management. In this regard, the integration of BIM and GIS is particularly important. BIM provides detailed information about a building's structure and systems, which can be used to monitor and assess its condition over time. GIS, on the other hand, offers spatial context and can integrate data about the surrounding environment, including weather patterns, natural disasters, and other geographical factors that can impact a building's integrity. By integrating BIM and GIS, it can create a dynamic model that not only tracks the health of the building itself but also considers external factors that could lead to damage or deterioration. This approach allows for more proactive maintenance and risk assessment, helping to ensure the safety and longevity of structures in a changing world. Especially for large-scale infrastructure projects, an information-rich platform that aggregates geographic and infrastructural design information to provide geometric 3D visualization combined with semantic information can simplify the project management.

Cesium is an open web platform that is primarily designed for large-scale 3D mapping and geospatial applications. It is a powerful tool that, much like Google Earth, presents 3D scenes in the form of a global Earth model and can incorporate various engineering projects from different locations. It offers a rich set of features that allow developers to overlay a variety of data on the Earth's surface, such as terrain, satellite imagery, and 3D model,

enabling interactive browsing, querying, and analysis. It comprises several key components that together provide a robust toolkit for building and hosting 3D geospatial data:

- **Cesium ion:** A platform service that offers streaming and hosting for 3D content. It allows users to upload their content, which is then optimized as 3D Tiles, hosted in the cloud, and streamed to CesiumJS viewer.
- **Cesium for Unity and Cesium for Unreal:** Extensions that bring Cesium's 3D geospatial capabilities to Unity and Unreal Engine, respectively. These are particularly useful for game development and real-time 3D visualization projects.
- **CesiumJS:** An open-source JavaScript library for creating world-class 3D globes and maps. It offers a high-precision WGS84 (World Geodetic System 1984) globe and integrates various map imagery, such as Google Maps, Bing Maps, ArcGIS and OpenStreetMap. Cesium first introduced 3D Tiles in 2015 and shepherded its acceptance as an Open Geospatial Consortium (OGC) Community Standard in 2019. Built on glTF and other 3D data types, 3D Tiles is an open standard and support a variety of 3D geospatial data types such as point clouds, buildings, and photogrammetry. It is designed to allow CesiumJS viewer to handle and display a vast amount of 3D geospatial data.

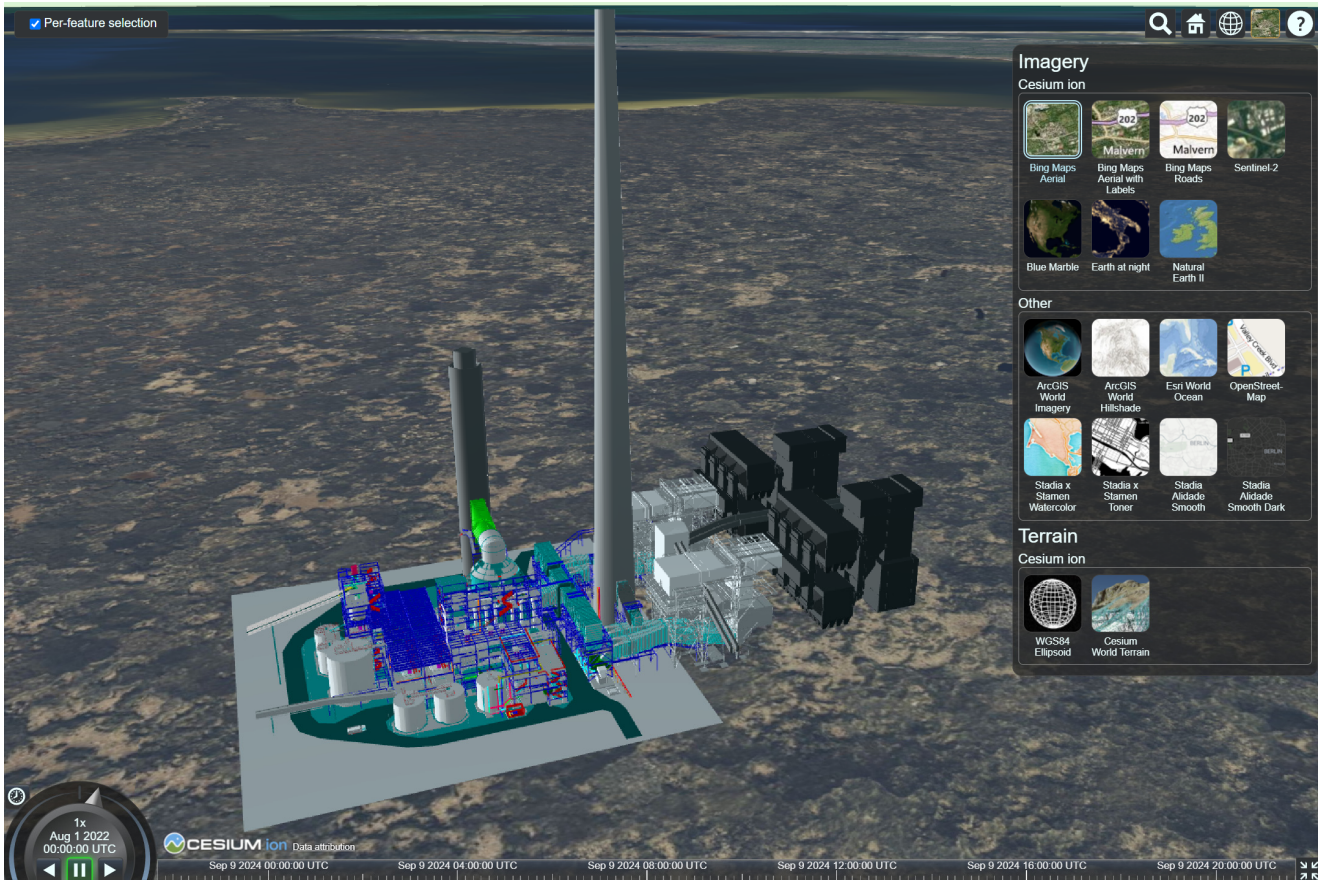


Figure 3-34: An example about BIM-GIS Integration with 3D Tiles converted from BIM model

Cesium has developed a new Design Tiler and Revit Add-In to transform IFC and Revit files into 3D Tiles, now only available for early access and feedback via architecture, engineering, construction, and operations (AECO) Tech Preview Program (Braig, 2024). The IFC is an open standard that has been broadly adopted across the AECO industry. Its comprehensive data structure encapsulates not only geometric details but also metadata, such as material characteristics, schedules, and the relationships among building components, making it highly suitable for complex projects that demand precise and interoperable data.

Figure 3-35: The new Design Tiler and Revit Add-In improve workflows and capabilities that place AECO content in a 3D geospatial context (Braig, 2024)

By supporting IFC, 3D Tiles can be created from various AECO disciplines, including architecture, structural engineering, and MEP (mechanical, electrical, and plumbing) systems. IFC is widely used in the AECO industry, and by converting IFC data to 3D Tiles, it becomes easier to share and view this data across different platforms and applications. Utilizing IFC data in the form of 3D Tiles will reduce the need for manual optimization of this detailed content for visualization and distribution.

Figure 3-36: Sample IFC data including structure and MEP as 3D Tiles (Braig, 2024)

4. Concept of AI-Based Robot CrackRepairBot

In this chapter, the main concepts of this work will be presented based on prior problem analysis and definition as well as relevant technical foundations. Since the repair agent used by the robot is Basilisk ER7, its specific usage method provides valuable guidance for the development of the robot. Therefore, the beginning of this chapter first introduces the usage method of Basilisk ER7. Then, the overall concept will be discussed, followed by a description of the various system according to the requirements. The robot, named CrackRepairBot, is primarily tasked with repairing narrow cracks and is also equipped with the functionality to record and visualize wide cracks. The following sections will provide a detailed introduction to each area of the concept.

4.1. The Specific Application Method of Basilisk ER7

A fundamental function of the robot developed in this work is to ensure the correct application of Basilisk ER7. According to the latest product data sheet of Basilisk ER7, the following points are important for the development of robot (Basilisk, 2022):

Application Volume: Application of the agent is straightforward, user can simply spray the liquid directly onto the cracks. Typical application volume of component is between 0.25 – 0.3 L/m for crack repair with a direct jet spray and 0.45 – 0.75 L/m² for complete surface treatment, but both may vary depending on present crack- and pore number and volume. This means that if the robot sprays repair agent along the cracks with a jet, the flow rate from the nozzle must reach 0.25 – 0.3 L/m.

(a) (b)

Figure 4-1: Two application methods of Basilisk ER7: (a) jet spray and (b) complete surface treatment (Basilisk, 2021g)

Narrow and Wide Cracks: It is important to note that the ER7 repair agent was initially advertised on its packaging and website as capable of repairing cracks up to 0.8 mm (Basilisk, 2021g). However, the latest product data sheet has revised this value to a more conservative maximum of 0.6 mm. Permanent sealing of pores and cracks up to 0.3 mm only requires a single treatment while wider cracks (up to 0.6 mm wide) require a minimum of 3 treatments with an interval of minimally 6 weeks. This means that the robot must be able to distinguish between narrow cracks and wide cracks. As the repair agent is ineffective on cracks exceeding 0.6 mm in width, applying it to such cracks would result in unnecessary waste.

Order of Use and Gel Formation: Both liquid solutions must be transferred separately in pressurizable spray units. Component A contains bacteria and calcium lactate, and component B contains reaction promoters. The solution of component A must be applied in amounts sufficient to penetrate deep into occurring cracks and pores and saturate them. The solution of component A, which contains suspended particles, does not substantially change the appearance of the cracks when applied, as shown in Figure 4-2. This indicates that the robot's crack recognition after the application of componet A remains unaffected.

Figure 4-2: The appearance of the crack after spaying the solution of component A

However, spaying the solution of component B results in formation of a firm gel, covering and sealing cracks and pores, as soon as brought into contact with the solution of component A, see Figure 4-3 (a). Figure 4-3 (b) shows the condition of the crack surface one day after the application of component B. Wide cracks require multiple applications of agent. However, after applying the solution of component B, the white gel will affect the robot's accurate identification of cracks. Consequently, adjustments are needed in the robot's algorithm. During the second and third rounds of spraying, the robot should no longer distinguish between narrow and wide cracks. It should spray the solution as long as a crack is detected. When modifying the algorithm, it is important to avoid affecting other system functions. As shown in Figure 4-3 (b), because the crack is wider, the white gel and the induced calcium carbonate do not completely seal the crack opening, allowing the crack to be roughly identified. Although the robot will apply agent indiscriminately to both narrow and wide cracks at this stage, it still conserves agent compared to complete surface treatment.

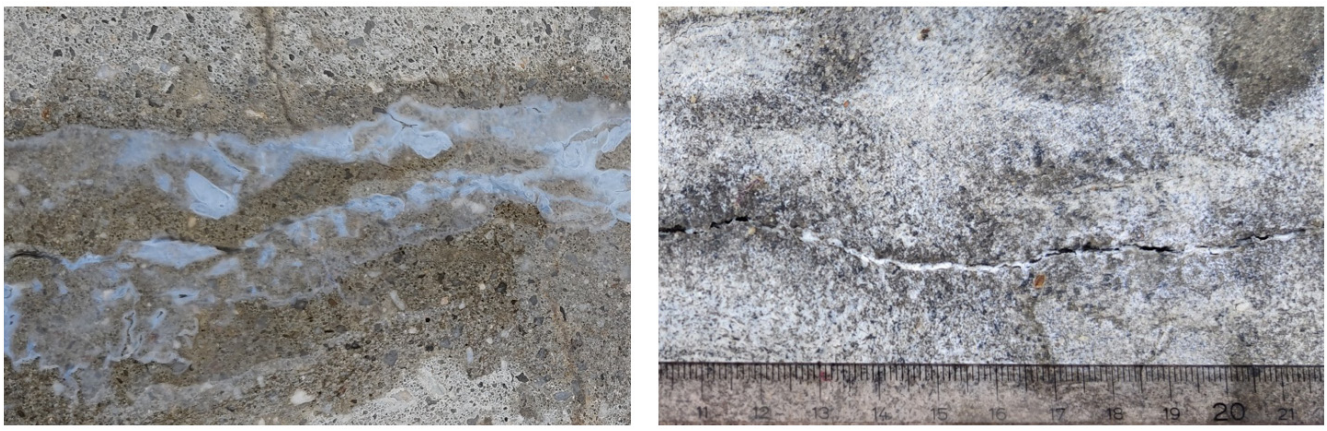


Figure 4-3: Gel formation due to the application of component B: (a) the immediate effect after application and (b) the condition observed one day after the application

Maintaining Moisture: The Figure 4-4 shows the timeline for the application of the repair agent. Reaction time is 6 weeks, depending on in situ temperature. In other words, the conditioning takes 6 weeks. During this entire period moisture must be available to prevent the cracks from drying out. If moisture is not naturally available, it must be provided actively. This means that during the maintenance phase, multiple watering applications by the robot are required. To optimize efficiency, the robot's path planning algorithm should be reusable to avoid spending time re-planning the route each time. For wide cracks up to 0.6 mm, multiple applications of the repair agent are also needed, making a reusable path planning scheme very practical.

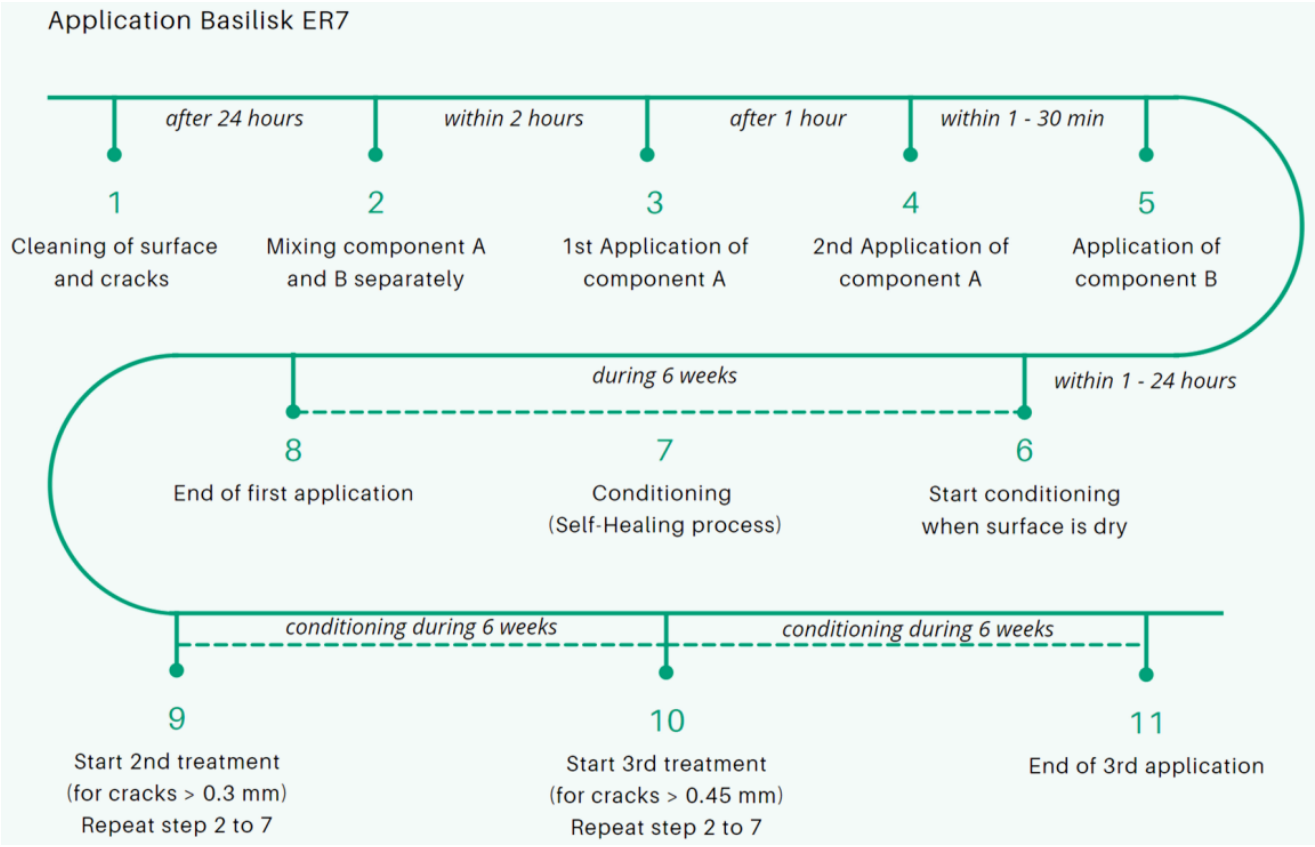


Figure 4-4: Timeline of application Basilisk ER7 (Basilisk, 2022)

4.2. Overall Concept of the Robot

From the analysis of the repair agent's application methods in the previous section, several critical functionalities for the robot have been identified. These include the ability to instantly dispense the appropriate amount of repair agent via the nozzle, the use of crack segmentation technology to distinguish between narrow and wide cracks, the flexibility to modify the algorithm with minimal impact on other system functions, and a reusable path planning algorithm for repeated operations. The figure below shows the simplified presentation of overall concept of the robot. The robot's control system includes two compact computers and one microcontroller. The primary computer, functioning as the "brain", is responsible for the deployment of machine learning model. The secondary computer and microcontroller, acting as the "cerebellum", control the robot's movements, recording function and spraying system. Additionally, the robot is equipped with two USB cameras, one dedicated to capturing images and the other for crack detection via real-time video feed. A comprehensive inspection of the target area is

conducted using multi-point navigation. Detected wide cracks will be visualized on a web application built using the Cesium.

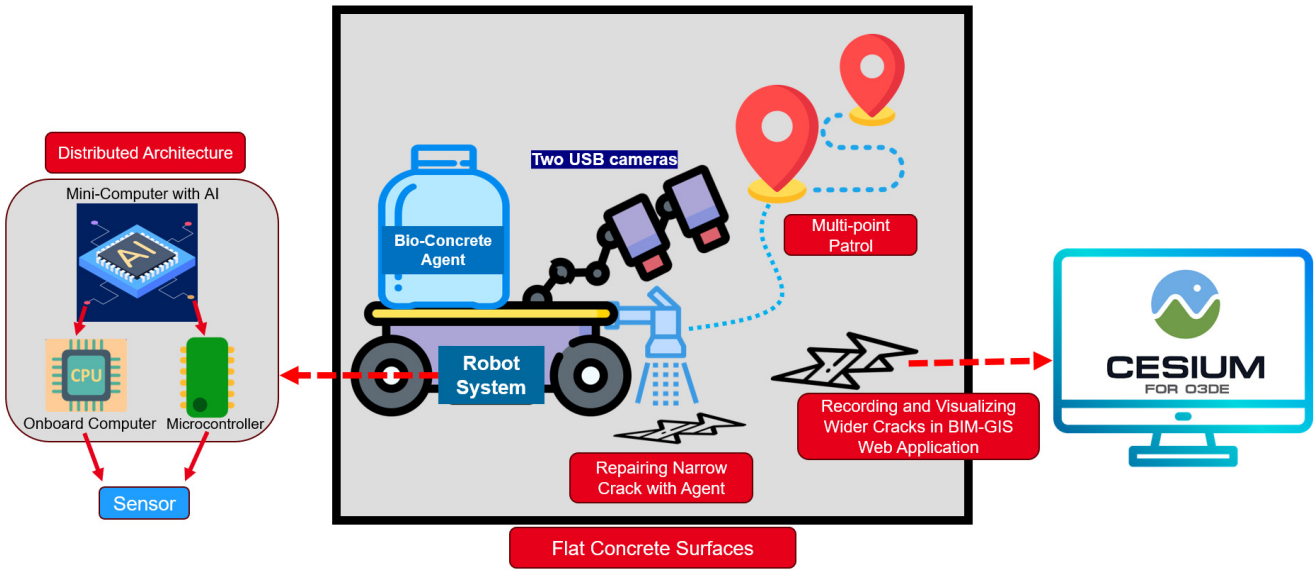


Figure 4-5: Simplified presentation of overall concept of the robot

4.3. Independence and Flexibility of Robot

To mitigate the reliance on a single manufacturer's products, this work proposes three strategies for the development of robot: utilizing open-source robotic platform, adopting open-source algorithms and software frameworks, and implementing a distributed control system architecture.

4.3.1. Open-Source Robotic Platform

Currently, the market offers a variety of open-source and cost-effective robots. For instance, companies such as Hiwonder and Yahboom have developed many open-source robots geared towards education, which are readily available on platforms like Amazon, eBay, and AliExpress. These sellers also provide comprehensive user manuals, allowing developers to quickly grasp the working mechanisms of both the hardware and software. Since developing a robot entirely from scratch is time-consuming and labor-intensive, the robot used in this work was purchased directly from the market and then customized to meet specific task requirements. The robot used in this work should be small in size and cost-effective. A compact design allows for efficient ground movement, low energy consumption, and longer battery life. However, the robot must not be too small, as it needs to carry repair materials while moving around to repair cracks. The onboard computers of open-source robots available on the market are generally Raspberry Pi, NVIDIA's Jetson series, and Horizon Robotics' RDK series. These systems operate on ROS and support programming in Python. The deployment of machine learning algorithms also primarily relies on Python. Therefore, Python scripts can be used to translate the inference structures of machine learning into commands that control the robot. *rospy* is a pure Python client library for ROS. The *rospy* client API (Application Programming Interface) enables Python programmers to quickly interface with ROS Topics, Services, and Parameters, as shown in the figure below.

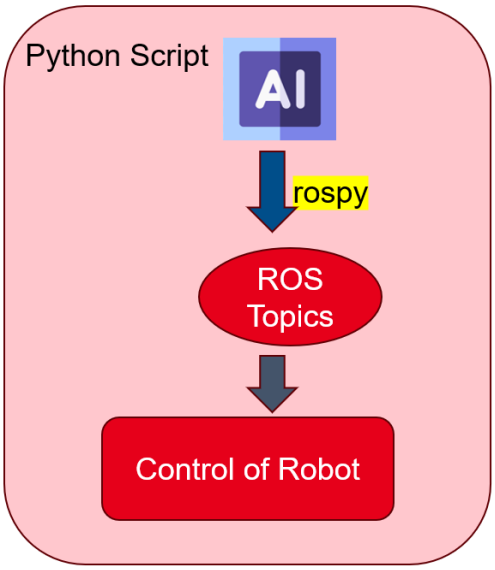


Figure 4-6: The principle of converting machine learning results into robot control commands

4.3.2. Open-Source Algorithms and Software Frameworks

Choosing open-source algorithms and software frameworks for developing a robot has several significant advantages:

- **Cost Reduction:** Open-source algorithms and software frameworks is free, which significantly reduces development costs compared to commercial software that often requires expensive licenses.
- **Transparency and Security:** The code in open-source algorithms and software frameworks is available for anyone to view and edit. This transparency allows for easier identification and resolution of security vulnerabilities.
- **Avoiding Vendor Lock-in:** Commercial software often leads to vendor lock-in, where developers become dependent on a particular vendor's technology and services, making it costly to switch to other solutions. Open-source software is generally platform-independent, giving developers more flexibility.
- **Customization:** Open-source algorithms and software frameworks allows developers to modify the code to meet specific needs. This is crucial for complex and diverse projects.
- **Community Support:** Open-source projects usually have active communities where developers can find help, share knowledge, and exchange experiences. This support is often broader, providing insights and solutions from a global network of contributors.

Therefore, in the process of developing the robot, this work uses free databases, free remote desktop control software, open-source model format conversion algorithms, and so on. MongoDB as a data storage solution provides a powerful document-based database system without the costs associated with commercial databases. Cesium combines BIM and GIS capabilities, offering free tools for 3D mapping and building information modeling, unlike commercial software like ArcGIS and Autodesk that require expensive license. YOLO networks are efficient open-source tools for image segmentation and object detection, saving time and cost in developing high-performance image processing algorithms. NoMachine provides free remote desktop control, allowing development teams to manage and debug robots remotely without purchasing expensive remote-control software.

Py3DTilers offers an open-source tool for converting IFC models to 3D Tiles, providing greater flexibility and cost savings in BIM and 3D data processing.

In summary, basing development on open-source technologies offers tremendous benefits in terms of technical, economic, and social aspects, providing a more free, flexible, and efficient path for developing complex systems like robots.

4.3.3. Distributed Control System Architecture

Even inference generally requires less computation than training, the demands for real-time processing, concurrency, and model complexity mean that significant computational power is still necessary. And there are future plans to deploy LLMs and VLMs in robot to achieve embodied intelligence. Therefore, when selecting such a small single-board computer as the brain for the robot, the computing power of the computer must be sufficiently strong. Designing the robot's control system as a distributed architecture is a very logical and effective approach, particularly when considering cost, software compatibility, and future scalability. The advantages of this method are as follows:

- **Cost Reduction by Selecting Robotic Platform:** By running the computationally intensive machine learning models on a dedicated small computer rather than the robot's onboard computer, it can significantly reduce the overall cost of the robot. The onboard computer and the robot's body only need to handle basic control tasks, which allows it to be less powerful and therefore less expensive. This approach helps in lowering the manufacturing costs and, consequently, a relatively inexpensive robotic platform can be used.
- **Software Compatibility:** Running complex machine learning models on the onboard computer might lead to compatibility issues with the existing software. By separating the machine learning tasks to a dedicated small computer, it can avoid these potential conflicts, making the onboard system more stable and focused on its core functionalities.
- **Flexibility in Hardware Upgrades:** To enhance the robot's intelligence, there are plans to implement more sophisticated algorithms in the future, which suggests that the current hardware may not be adequate for these advancements. Therefore, upgrading just the dedicated single-board computer rather than the entire robot control system is more flexible. This distributed design makes the robot more scalable, and future hardware and software upgrades become simpler and more cost-effective.
- **Effective Use of Microcontrollers:** Using microcontrollers like Arduino to control simple subsystems within the robot, such as the spraying system, is a smart choice. Microcontrollers are inexpensive, reliable, and supported by a vast amount of open-source resources.

In summary, the approach of modularizing different tasks and using appropriate hardware for each task not only enhances the system's flexibility and scalability but also significantly reduces cost and complexity. This distributed control system design allows the robot to be powerful and cost-effective while being well-prepared for future technological advancements.

In distributed systems, it is essential to address communication challenges between different modules. Most compact computers, such as single-board computers (e.g., Raspberry Pi, NVIDIA Jetson), as well as many industrial and embedded personal computers, typically have Ethernet interfaces. These Ethernet ports allow the computers to connect to a network or directly to each other, enabling fast and reliable communication, which is

especially useful for real-time applications. Communication between a computer and a microcontroller like Arduino can be efficiently handled via USB data cables or through pin connections (e.g., serial communication using TX/RX pins). USB provides a straightforward and widely supported method for communication, while pin connections allow for direct, low-level communication. Both methods are reliable and commonly used in embedded systems for controlling peripherals and exchanging data. In addition to wired connections, computers and microcontrollers can also communicate wirelessly through Wi-Fi, Bluetooth, Zigbee, etc. In this work, the interconnections between the modules are illustrated as follows.

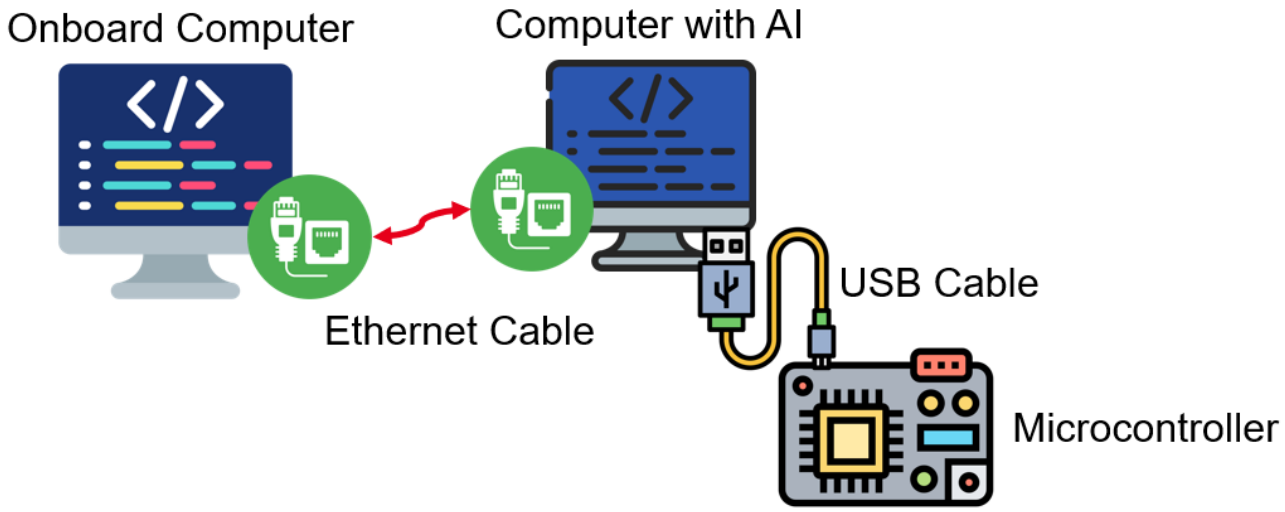


Figure 4-7: The connection schematic of the distributed system

4.4. Image Segmentation for Cracks Filtering

The bio-concrete repair agent utilized in this work is the Basilisk ER7, which is effective solely for cracks with a width of 0.6 millimeters or less. Consequently, the robot must be capable of detecting and filtering cracks, activating the spraying mechanism only when the crack width is within this threshold. To achieve this, image segmentation technology is essential for identifying and classifying the cracks. Image segmentation is typically used to locate objects and boundaries (lines, curves, etc.) in images. More specifically, image segmentation involves assigning a label to each pixel in an image, ensuring that pixels sharing the same label exhibit similar characteristics

4.4.1. YOLO Network

In recent years, YOLO networks have become the leading approach in real-time object recognition due to their optimal balance between computational efficiency and detection accuracy. Since its inception in 2015, the YOLO model family has evolved through several iterations, with each new version enhancing the capabilities of its predecessors. As a one-stage object detection model, YOLO processes an entire image in a single forward pass through a convolutional neural network. Its popularity stems largely from its speed; YOLO can process images incredibly fast, making it suitable for real-time object detection. The YOLO network has now been updated to YOLOv10 (A. Wang et al., 2024). YOLOv9 sets a new standard for real-time object recognition (C. Y. Wang et al., 2024). The new models offer greater accuracy with less computation compared to previous versions.

Traditional deep neural networks have faced issues such as vanishing and exploding gradients. However, techniques like batch normalization and advanced activation functions have largely mitigated these problems. YOLOv9 provides a deeper analysis of the information bottleneck issue. Enhancing YOLO with open-vocabulary detection, YOLO-World integrates vision-language modeling and is pre-trained on large-scale datasets. It introduces the Re-parameterizable Vision-Language Path Aggregation Network (RepVL-PAN) and region-text contrastive loss to improve interaction between visual and linguistic data. Compared to YOLOv9-C, YOLOv10-B achieves the same performance with 46% less latency and 25% fewer parameters.

The Average Precision (AP) metric of various YOLO networks is shown in Figure 4-8. COCO AP(%) refers to the Average Precision (AP) metric used in the context of the COCO (Common Objects in Context) dataset for object detection tasks. The COCO dataset is a large-scale dataset that is commonly used to benchmark computer vision algorithms, particularly for object detection and instance segmentation.

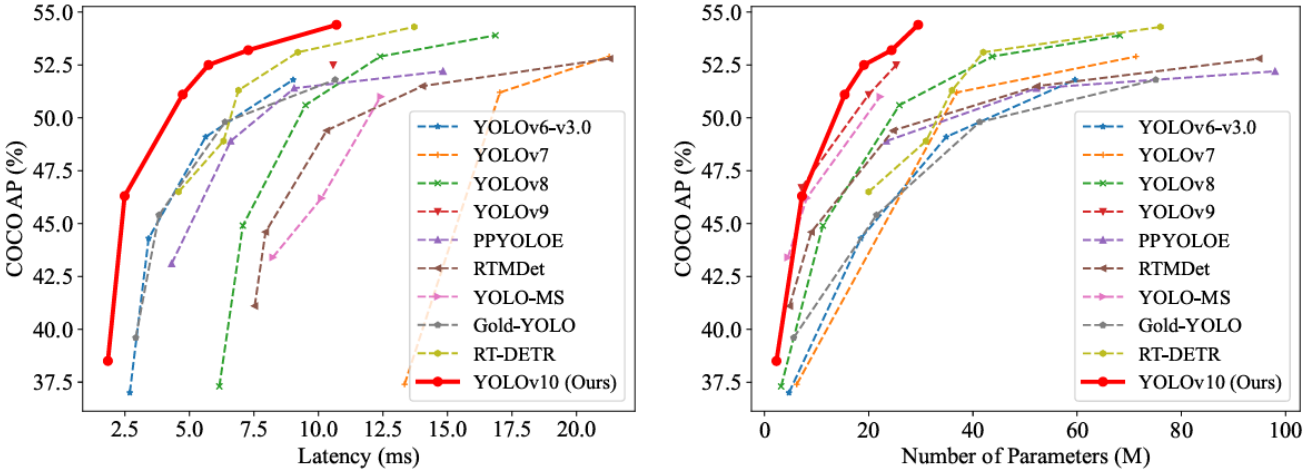


Figure 4-8: Comparisons of YOLO networks in terms of latency-accuracy (left) and size-accuracy (right) trade-of (A. Wang et al., 2024)

The pre-trained model for image segmentation of YOLOv8 and YOLOv9 can be found on GitHub, but the image segmentation model of YOLOv10 has not been released yet (Ultralytics, 2024d). YOLOv8 and YOLOv9 have a robust community, which has contributed to its stability and reliability. And both of them support a full range of vision AI tasks, including detection, segmentation, pose estimation, tracking, and classification. This versatility allows users to leverage capabilities of YOLOv8 and YOLOv9 across diverse applications and domains.

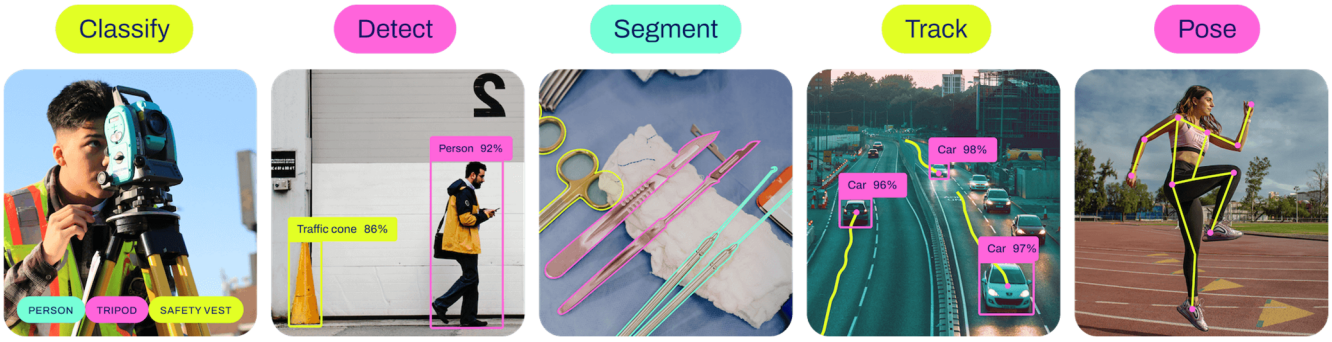


Figure 4-9: YOLO supports a full range of vision AI tasks (Jocher et al., 2023)

YOLOv8 Detect, Segment and Pose models are pre-trained on the COCO dataset, as well as YOLOv8 Classify models are pretrained on the ImageNet dataset. Track mode is available for all Detect, Segment and Pose models. The Table 2 provides an overview of the performance metrics of pre-trained YOLOv8 segmentation models.

YOLOv8n-seg is the smallest segmentation model in YOLOv8, as it has the fewest parameters, resulting in the smallest model size of 6.73 MB and the fastest inference speed. However, the inference accuracy is the lowest.

Table 2: Performance Metrics of YOLOv8 models trained on COCO (Ultralytics, 2024e)

Model	size (pixels)	mAP^{box} 50-95	mAP^{mask} 50-95	Speed CPU ONNX (ms)	Speed A100 TensorRT (ms)	params (M)	FLOPs (B)
YOLOv8n-seg	640	36.7	30.5	96.1	1.21	3.4	12.6
YOLOv8s-seg	640	44.6	36.8	155.7	1.47	11.8	42.6
YOLOv8m-seg	640	49.9	40.8	317.0	2.18	27.3	110.2
YOLOv8l-seg	640	52.3	42.6	572.4	2.79	46.0	220.5
YOLOv8x-seg	640	53.4	43.4	712.1	4.02	71.8	344.1

YOLOv8 needs Python >= 3.8 environment with PyTorch >= 1.8. This implies that YOLOv8 can only be deployed on versions of Jetpack that are 4.4 or higher. The Jetpack contains the custom version of Ubuntu Linux for NVIDIA's Jetson hardware. YOLOv9 models are also pretrained on the COCO dataset, they require same environment as YOLOv8. The YOLOv8 and YOLOv9 Segment models will be tested for the crack segmentation.

4.4.2. Live Image Quality of the Camera

In real-time image segmentation processes, the frame rate and image quality of the camera are crucial. Since the robot is moving, a camera with a high frame rate that can provide clear images without motion blur. Most USB webcams on the market offer a frame rate of 30 FPS (Frames Per Second). Frames per second (FPS) is a metric used to evaluate the performance of display devices in contexts such as video capture, playback, and gaming. FPS denotes the frame rate, which is the number of consecutive images shown each second. At 30 FPS, it is equivalent to approximately 33.33 milliseconds per frame. This means that at a frame rate of 30 FPS, the duration of each frame is roughly 33.33 milliseconds. This measurement is crucial for discussing video quality in both capturing and playback. The human brain can process about 10 to 12 FPS, with higher frame rates perceived as smooth, continuous motion. Standard full-motion video for human generally operates at 24 FPS or more (Brunner, 2023). However, this standard is only for the human brain. For robot with detection task, the camera's frame rate should be faster than the inference speed of the neural network to ensure smooth and real-time processing of the video stream. If the camera's frame rate is too slow compared to the neural network's inference speed, it can lead to dropped frames, stuttering video, and delayed reactions in applications that require real-time analysis. Moreover, the speed at which a robot moves and the frame rate of the camera influence each other by real-time analysis. To determine whether the frame rate of a camera is suitable for a specific application, testing is essential. By assessing the clarity and quality of the images captured during robot movement, user can evaluate if the camera's frame rate meets the requirements for the task. If the images are consistently sharp and free of motion blur, the frame rate is likely appropriate for the speed at which the robot operates. Conversely, if the images are blurry, it may indicate that the frame rate is too low for the robot's movement speed, and adjustments may be needed. Potential solutions for these adjustments include reducing the robot's movement speed or upgrading to a camera with a higher frame rate.

Capturing narrow cracks on the ground clearly is not an easy task. They are so narrow, standard cameras may not have the resolution necessary to capture fine details. Using a zoomable camera or getting closer to the surface

with a fixed focal length camera are both effective strategies for capturing narrow cracks in concrete surface. The figure below illustrates the installation of cameras on the robot for the two strategies mentioned.

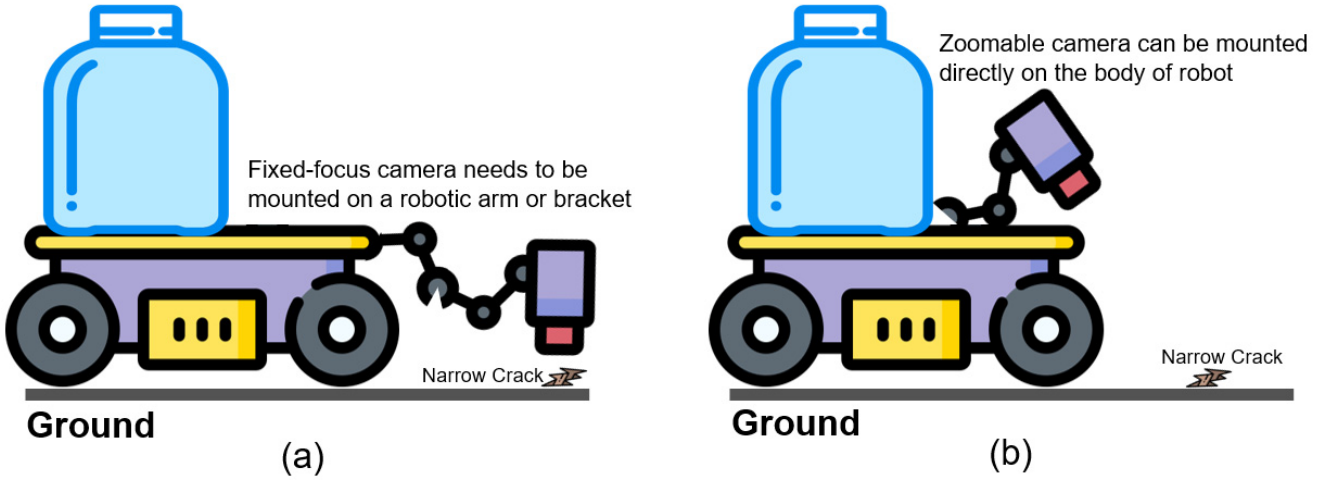


Figure 4-10: Two strategies for capturing narrow cracks on the ground with fixed-focus and zoomable camera

If a fixed-focus camera is used, the advantage is that the image will not be distorted. However, the narrow cracks on the ground are often very small, and the camera needs to be as close to the ground as possible to capture the details of the cracks. For this, a robotic arm or a bracket is required to mount the camera, and it is necessary to ensure that the robotic arm or bracket does not shake when the robot is operating to avoid changes in focus, as shown in Figure 4-10 (a).

If a zoom-capable camera is used, the camera can be directly mounted on the robot, which makes it very stable, and by adjusting the focus, it can zoom in on the narrow cracks on the ground to capture more details, as shown in Figure 4-10 (b). However, it should be noted that zoom-capable cameras have image distortion issues. If the distortion is too severe, such a zoom-capable camera cannot be used. It is also important to note that, prior to the robot's operation, the zoom camera's focal length must be manually calibrated and kept constant, while its angle and height are securely fixed. These preparations are essential to ensure the camera's focal length remains unchanged during operation, thereby guaranteeing consistent and reliable crack segmentation performance.

On a moving robot, running real-time image segmentation can be challenging with auto-focus or auto-zoom cameras because these features typically involve mechanical adjustments that take time to complete. The delay introduced by the auto-focus or auto-zoom mechanism may not align well with the rapid inference speed of the vision model, potentially leading to latency issues in the overall system. As a result, fixed-focus cameras or zoom-capable cameras are often preferred for real-time applications on moving robots to ensure consistent and immediate image capture without the delay.

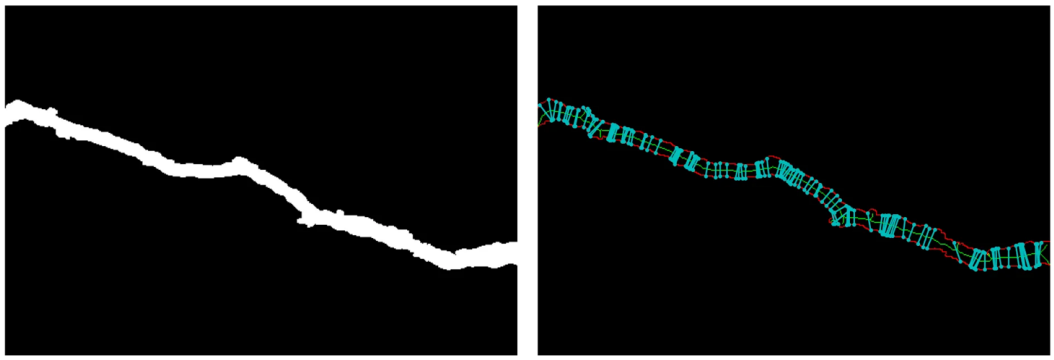
The specific choice of strategies depends on the selected robotic platform. Different platforms may have different requirements and capabilities, which will influence the decision on whether to use a fixed-focus camera or a zoom-capable camera for crack detection and analysis.

4.4.3. Calculation of Maximum Crack Width

Calculating the width of cracks involves image segmentation and image width measurement techniques. After applying image segmentation techniques to differentiate the crack from the surrounding area, width calculation method like the orthogonal skeleton line method can be used to calculate the width of the cracks by measuring

the perpendicular distance from the medial axis to the edges of the crack. This involves estimating the normal vector at various points along the medial axis and calculating the distance to the nearest edge points in the direction of these normal. This approach involves the following steps:

1. Employing edge detection algorithms to identify the edges of the crack.
2. Performing a medial axis transformation to obtain the skeleton of the crack.
3. Estimating the normal vector and calculating the shortest distance from the crack edge to the skeleton in the direction of the normal vector, which gives the crack width.



(a) (b)

Figure 4-11: Image processing: (a) crack segmentation result and (b) applying orthogonal skeleton line method (SubChange, 2022)

Additionally, the maximum inscribed circle algorithm can also be used to calculate the crack width. This method determines the maximum width of the crack by computing the largest inscribed circle within the crack's contour. The specific steps of the maximum inscribed circle algorithm include:

1. Identifying the crack contour.
2. Calculating the bounding box of the contour and determining the search range for the inscribed circle.
3. Randomly selecting points within the contour and calculating the inscribed circles corresponding to these points.
4. Through iterative search, the largest inscribed circle is found, and its diameter is the maximum width of the crack.

As shown in Figure 4-12, each red mask of crack is marked with an inscribed circle at its widest point. The dark blue inscribed circle is the largest among all the cracks, meaning the location of the dark blue inscribed circle represents the widest point of these cracks.

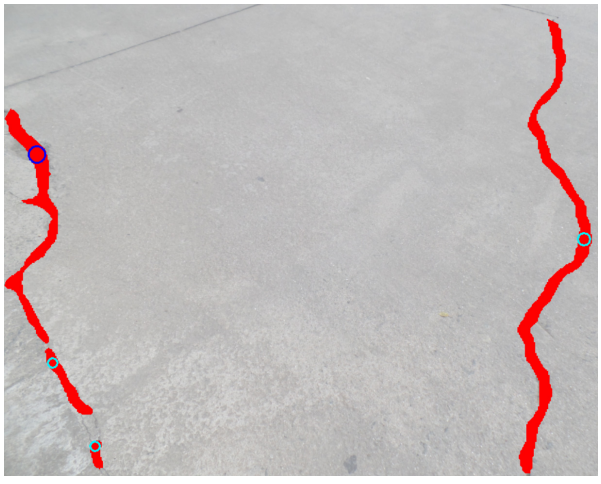


Figure 4-12: The calculation results of the inscribed circle algorithm (Jiang, 2022)

From the perspective of code implementation, this approach is relatively simpler compared to orthogonal skeleton line method. The Chapter 5 will attempt to deploy this algorithm for calculating the maximum width of a crack. When deploying algorithm for calculating crack widths, one condition that must be met is real-time performance; the algorithm must be able to instantly derive the width of the crack without excessive delay. Further verification of the inference speed of the maximum inscribed circle algorithm will be conducted in the chapter 5.

4.4.4. Distinguishing between Narrow and Wide Cracks

Due to the absence of a physical reference, image processing algorithms will only provide measurements in units of pixels. Therefore, the actual width of the crack can be determined via the scale factor. Here are the specific steps for implementation:

1. **Maintain Constant Camera Height and Focus**: Keeping the camera at a consistent height above the ground and adjust the focus to ensure a clear image. Once the focus is set, keep it unchanged.
2. **Determing the Scale Factor**: Placing a measuring ruler on the ground and using a pixel measurement tool to determine how many pixels correspond to 1 centimeter on the measuring ruler in the image. This gives back a scale factor for the camera's field of view under the given height and focus conditions.
3. **Image Segmentation**: Applying an image segmentation algorithm to isolate the crack from the surrounding area.
4. **Calculate Crack Width**: Using the maximum inscribed circle algorithm to calculate the widest part of the segmented cracks.
5. **Convert Pixels to Actual Distance**: Converting the pixel measurement of the crack to an actual distance using the scale factor determined earlier. This allows user to determine whether the crack is a narrow crack (up to 0.6 mm) or a wide crack (wider than 0.6 mm).

Determing a threshold is another effective method to differentiate between narrow and wider cracks. A 1-mm wide crack can be selected using a ruler, and under fixed camera height and focal length conditions, image segmentation combined with the maximum inscribed circle algorithm can be applied to determine the pixel width of the 1-mm crack. Sixty percent of this width is used as the threshold for distinguishing between narrow and wider cracks.

4.5. Solutions for Rapidly Spraying Adequate Amounts of Agent

The flow rate from the nozzle must reach 0.25 – 0.3 L/m by crack treatment. Insufficient spray will negatively impact the effectiveness of crack repair. Two possible solutions can be considered to address this problem. The first approach uses a high-power pump that, upon activation, quickly draws a portion of the liquid from the container and sprays it onto the crack. The second approach employs a pressurizable water bottle combined with a solenoid water valve. When the normally closed solenoid water valve is open, the pressurized liquid will be sprayed out immediately onto the crack. If the robot does not detect any cracks or detect wide cracks, it will turn off the power supply to the water pump or valve, and the spraying system will stop spraying. Below is a simple logic of the system's operation.

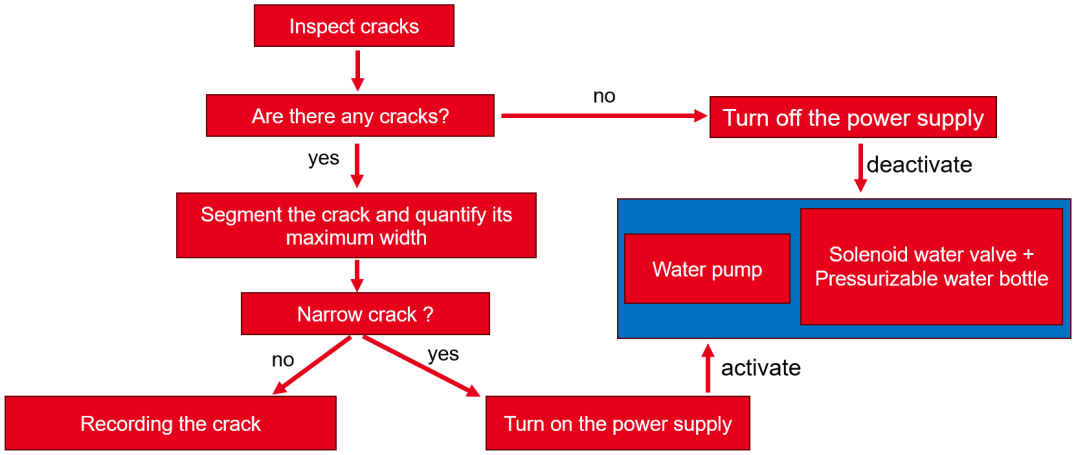


Figure 4-13: The logic of the system operation for cracks

Relay

High-performance water pumps and solenoid water valves generally require a voltage of 12V or higher. However, the voltage provided by single-board computers (e.g., Raspberry Pi, NVIDIA Jetson) is usually at most 5V. Therefore, in a robotic system, it is necessary to use a battery pack or power bank to supply 12V or higher power to the water pump and valve. A relay is an electrically operated switch. It consists of a coil, which receives an electric signal and converts it to a mechanical action and contacts that open and close the electric circuit. The single channel 5V relay module works on 5V, but the input signal can come directly from microcontroller output working at 3V or 5V to control relays. The single-board computer can indirectly control the spraying system through relay by switching the power supply on and off.

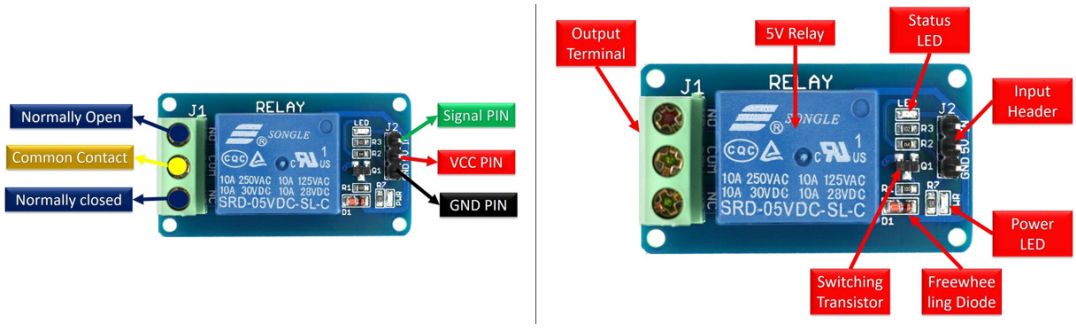


Figure 4-14: A single channel 5V relay module (Microcontrollers Lab, 2021)

The signal pin is mainly used for receiving the HIGH signal or LOW signal output of the microcontroller, sensor, or logic device. The relay has two different types of electrical contacts inside – normally open (NO) and normally closed (NC). In the normally open configuration, when the relay receives a HIGH signal via the signal input pin, the 5V switch closes and allows current to flow from the Common Contact terminal to the NO terminal. In the normally closed configuration, a HIGH signal will open the switch and interrupts the 5V current, a LOW signal will close the switch and allows current to flow from the Common Contact terminal to the NC terminal. The VCC pin requires a 5V DC (Direct Current) power supply to function, while the GND (Ground) pin connects to the power supply's GND terminal. When the relay is activated and the coil is energized via the signal input pin, the status LED lights up. It is quite easy to control a relay using an Arduino. Typically, this involves connecting the relay's control pin to a digital pin on the Arduino and connecting the relay's power input to the power source. The following shows the connection schematic of the entire sprinkler system using the NC configuration of relay.

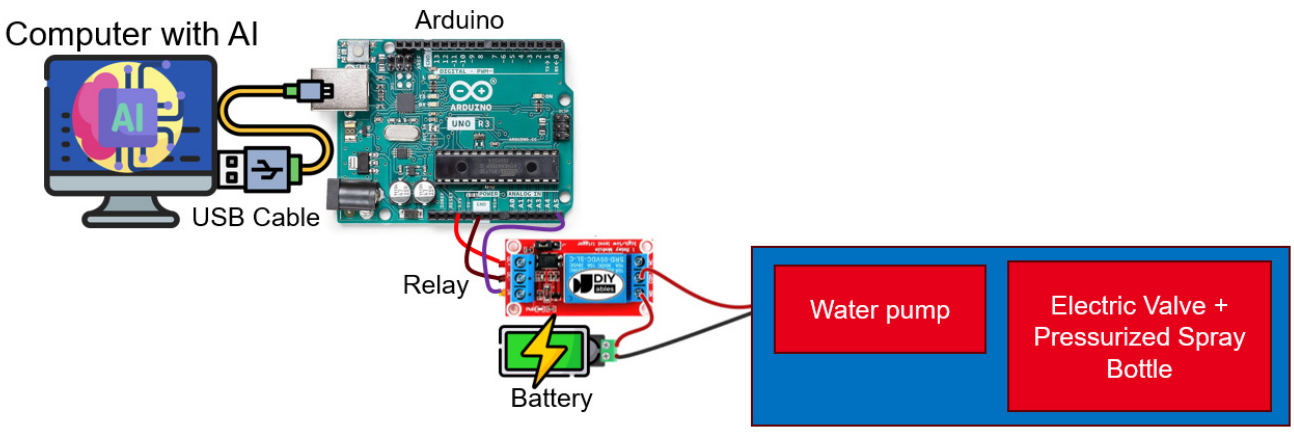


Figure 4-15: The connection schematic of the entire sprinkler system.

Adjustable Nozzle

The robot's motion during the crack repair spraying process makes precise alignment challenging. To enhance the effectiveness of the spraying operation, it is recommended that the spraying system be equipped with an adjustable nozzle. This nozzle should have a configurable spraying angle to maximize the coverage area over the cracks. There are different types of spray nozzles available for different use. It's important to choose the right spray nozzle to get the best result because spray nozzles determine the volume, drop size and spray pattern, as shown in Figure 4-16 (a) and Figure 4-16 (b). Some nozzles on the long sprayer pole are manually adjustable, as shown in Figure 4-16 (c). The spray angle is a measurement of the fluid's angle as it leaves the nozzle. The spraying system employed in this work uses a nozzle with a 40-degree spray angle. Due to the difficulty of accurately targeting the cracks while the robot is in motion, the 40-degree spray angle allows for a broader distribution of the repair agent, facilitating more extensive coverage. A 60-degree spraying angle is excessively broad, as it leads to a greater dispersion area of the repair agent, consequently resulting in an inadequate volume of the agent being applied to the crack's surface. If it is necessary to use a 60-degree spraying angle, the flow rate must be adjusted upwards accordingly.

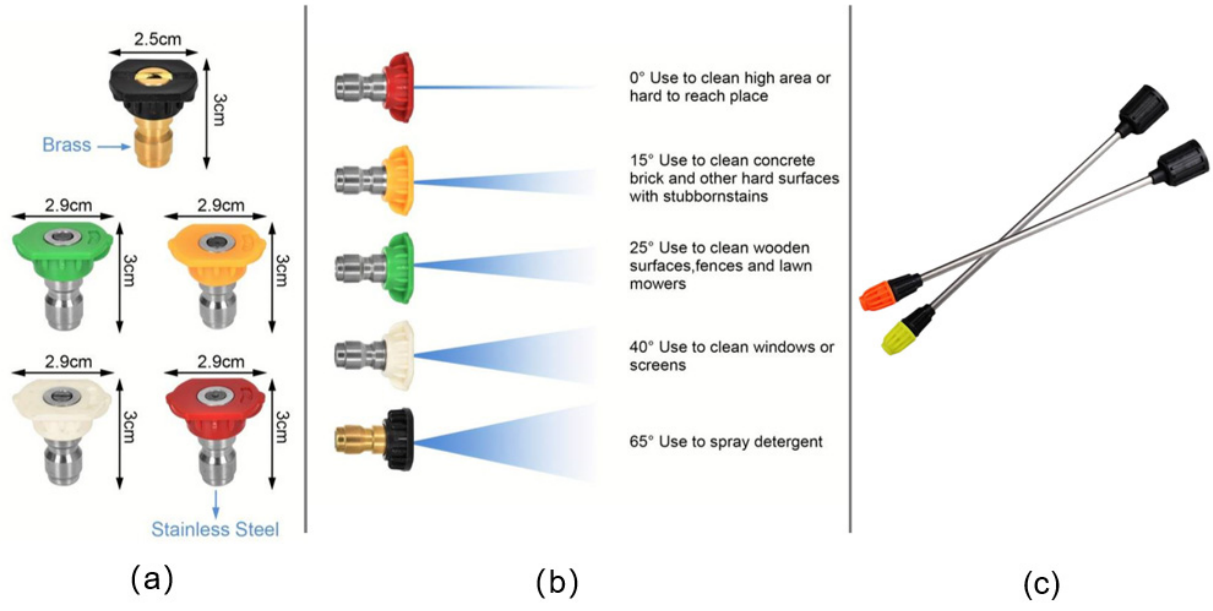


Figure 4-16: Adjustable nozzle and sprayer pole (Cozi Life, 2024)

4.5.1. Water Pump

The diagram below illustrates a spraying system constructed with a water pump on the robot, specifically employing a submersible pump.

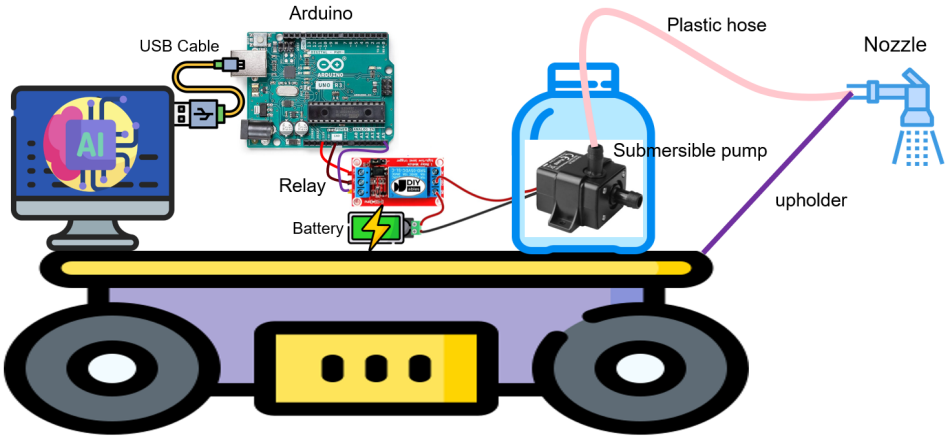


Figure 4-17: Schematic diagram about spraying system constructed with a water pump

The spraying capacity of this system depends on the power of the water pump. The use of submersible pumps is favored due to their numerous advantages. A key benefit is that submersible pumps do not require priming, as they are already immersed in the fluid they are meant to pump. This immersion makes them highly efficient, reducing the energy needed to move water into the pump since water pressure naturally forces the fluid into the pump. Submersible pumps are designed with hermetically sealed motors, allowing them to be fully submerged without the risk of electrical damage. It drives an impeller, a rotating component with curved blades, which spins to create centrifugal force, pushing the fluid away from the center and generating pressure. This pressurizable fluid is then directed through a diffuser, which regulates and guides the flow, converting high-velocity, low-pressure flow into high-pressure flow. The high-pressure fluid is discharged through an outlet or into a connected pipe system, ensuring efficient transportation of the fluid.

However, if the liquid level in the container exceeds the height of the nozzle, siphoning can occur when the pump begins to draw water. According to the laws of physics, the siphon effect occurs when a tube is filled with liquid and one end is submerged in a container of liquid at a higher level than the other end. If the higher end is open to the atmosphere and the lower end is lower than the surface of the liquid in the container, the liquid will flow from the higher container through the tube to the lower container. This happens because the atmospheric pressure on the liquid in the higher container pushes the liquid up the tube and into the lower container. Once the flow starts, it continues until the liquid levels equalize, or the flow is interrupted. This siphoning effect will cause the repair agent to continue flowing through the pipeline, even after the pump has stopped. To prevent this, it is recommended to raise the nozzle using a support or to install the container at a lower position. However, the nozzle should not be positioned too high, as it might lead to liquid splashing onto the robot during movement. Whether the container can be installed lower depends on the robot's structure. In some cases, modifications to the robot's structure would be needed, which can be quite troublesome.

4.5.2. Water Valve and Pressurizable Water Bottle

The following describes a spraying system built by combining a solenoid water valve with a pressurizable water bottle. This system eliminates siphoning problems and does not require high-power electronic components. The system's strong spraying capability primarily depends on the pressure manually applied to the pressurizable water bottle in advance. Once the normally closed solenoid water valve is energized, the liquid in the bottle will be instantly sprayed out under pressure. However, as the number of sprays increases, the pressure in the bottle decreases, requiring timely manual re-pressurization.

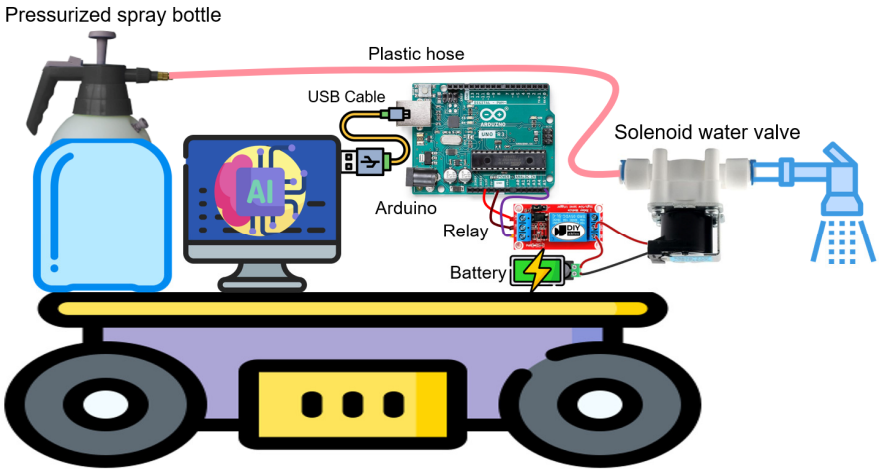


Figure 4-18: Schematic diagram about spraying system built by combining a water valve with a pressurizable spray bottle

Solenoid valves are the most commonly utilized control components in fluid systems. Their functions include shutting off, releasing, dosing, distributing, or mixing fluids. Solenoid valves are primarily classified into two categories: normally open (NO) and normally closed (NC). In a normally closed valve, when power is supplied, the magnetic field pulls the plunger upward, opening the valve to allow the medium to flow. The magnetic force lifts the plunger against a spring, which returns the plunger to its original position when power is interrupted, causing the valve to close. This type is commonly used for safety reasons, ensuring that the valve closes during a power failure. Conversely, a normally open valve permits the medium to flow without power, as the plunger remains retracted. When power is applied, the magnetic field pushes the plunger downward, closing the valve.

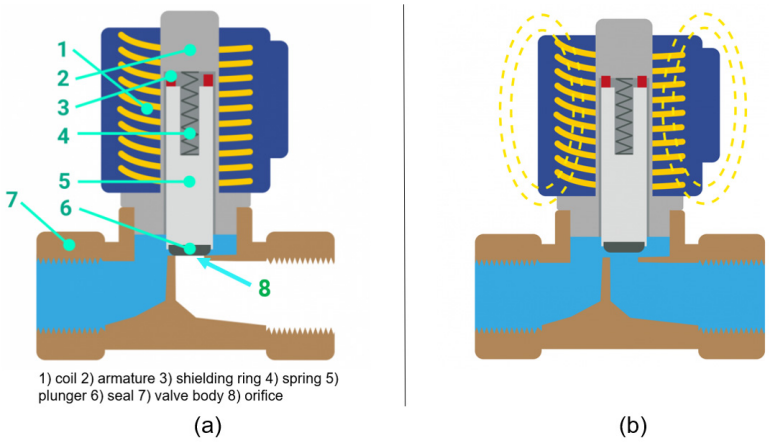


Figure 4-19: Normally closed valve: (a) unpowered and (b) powered states (BOLA, 2024)

Several factors can influence the water flow rate of solenoid valves, the primary factors include:

1. **Upstream and Downstream Pressure**: The difference in pressure between the upstream and downstream sides of the valve affects the flow rate.
2. **Valve Orifice Size**: The size of the valve's orifice plays a significant role in determining the flow rate. Larger orifices generally allow for greater flow.
3. **Valve Design**: The internal design of the solenoid valve, including the shape of the orifice and the flow path, can influence the flow characteristics.
4. **Fluid Type and Viscosity**: Different fluids have different viscosities, which can affect the flow rate through the valve.
5. **Solenoid Valve Power**: The power supplied to the solenoid valve's coil can influence its operation speed, which in turn can affect the rate at which the valve opens or closes, impacting the flow rate.

Whether to choose the water pump or the solenoid water valve with the pressurizable water bottle to build the spraying system needs to be considered in the chapter 5 in conjunction with the selected robot as well as the water pump and solenoid valve available on the market.

4.5.3. Checking Flow Rate

According to the manual of the repair agent Basilisk ER7, the flow rate from the nozzle must reach 0.25 – 0.3 L/m by crack treatment. It means that 0.25 to 0.3 liters of the Basilisk ER7 are required for 1 meter of crack length. The spraying capacity of the sprinkler system must therefore be quantitatively checked. As shown in Figure 4-20 (a), the liquid flow meter can be installed between the water valve or pump and the sprinkler nozzle to measure the flow rate. When the sprinkler system is activated and the robot is moved forward 1 meter, the flow rate of the sprinkler system can be determined by the flow meter. However, this solution is quite cumbersome to operate, as it requires not only the purchase of a flow meter of the appropriate size but also the installation of the flow meter. As shown in Figure 4-20 (b), another simple method is to use a stopwatch and a measuring cup to determine the time T it takes for the spraying system to spray 100 milliliters of liquid. Given the robot's speed V, the spraying capacity of the system can be calculated using simple mathematics as $0.1 / (T \times V)$, with the corresponding unit being liters per meter.

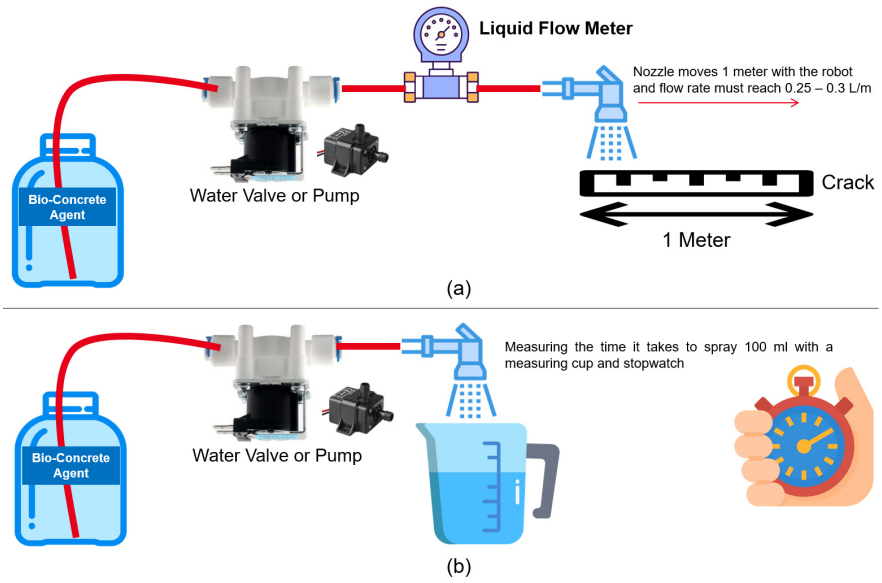


Figure 4-20: Solutions for checking the flow rate

4.6. Navigation Method for Cracks Inspection

Open-source robots, such as those from Hinwonder and Yahboom, typically provide three manual methods for controlling the robot's movement. One method is to remotely control the robot using a joystick, another is to use the keyboard of a remotely connected laptop, and the third is to use RViz to set target points for the robot to patrol sequentially. This work plans to develop an automated solution that allows the robot to follow a predefined route using a simple Python script. This solution has a clear advantage: the pre-planned route is reusable.

4.6.1. Creating Map from 3D Model and Image

Maps are crucial for robots, much like eyes are for humans. They provide the necessary spatial awareness and navigational information that allows the robot to understand its environment, plan its movements, and avoid obstacles effectively. The maps needed for the robot can be scanned and reconstructed using a LiDAR. To achieve an ideal map, a high-quality LiDAR is required, which implies a higher cost. In this work, simpler approaches are used for constructing the robot's map. The inexpensive LiDAR that comes with the open-source robot is intended only for obstacle avoidance. There are cost-effective LiDAR options available that can be integrated into robots for mapping and navigation purposes, such as the RPLIDAR A1, which can offer 360° full-scan detection and the ability to build maps of the area (Slamtec, 2024).

RViz supports a variety of map formats, with the most commonly used being the Occupancy Grid Map, which utilizes the PGM (Portable Gray Map) format. In the map file, each pixel represents an area in the real world, where white indicates free space, black signifies obstacles, and gray denotes unknown areas. For indoor environments, the corresponding 3D models of buildings, such as IFC models, or architectural drawings can be used to create maps for robots. For outdoor environments, aerial views from Google Earth or drone-captured images can be converted into PGM format maps supported by RViz with the help of some tools. These methods allow for the creation of maps that are suitable for both indoor and outdoor navigation without relying on high-cost LiDAR systems for mapping purposes. The figure below presents a comparison of the advantages and disadvantages of different methods for generating PGM map for robot.

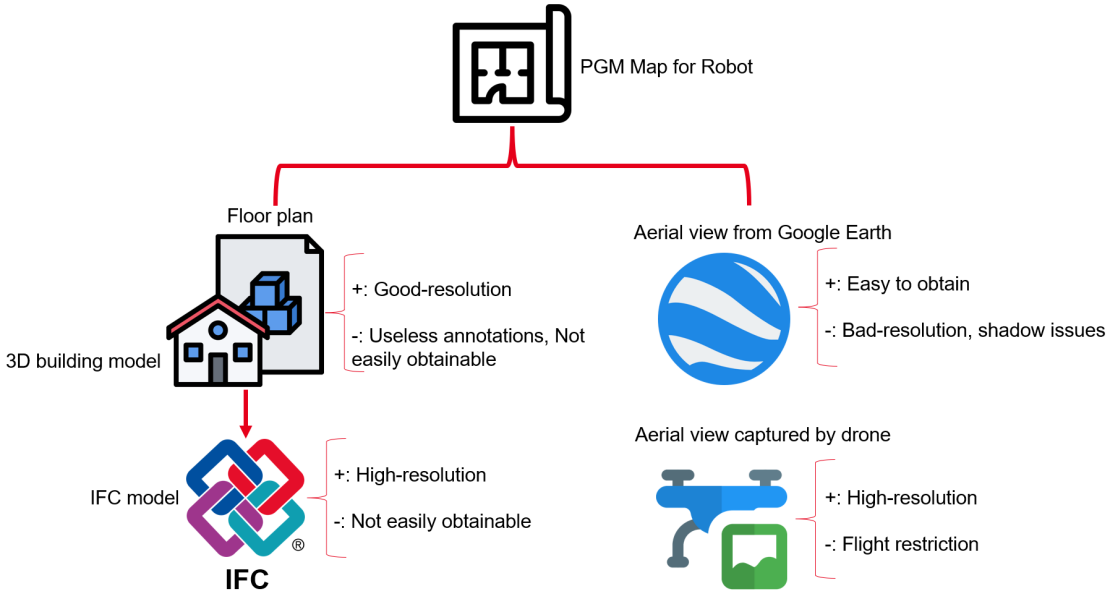


Figure 4-21: Methods to create PGM map for robot

However, not all buildings may have an IFC model available. Additionally, architectural drawings often contain many annotations that may need to be removed later using image editing software to create a clean map for robotic navigation. Screenshots of aerial view taken from Google Earth may not always have high image quality, and they can sometimes include artifacts such as reflections from trees or buildings. But they are often usable for general purposes. Post-processing with image editing software can help improve the quality. One of the advantages of Google Earth is that it provides aerial view of almost any location. The advantages of using aerial imagery captured by drones include clear and high-resolution images. However, there are many restrictions on drone operations in various places. For instance, drone flights are often restricted in or near airports, which are considered no-fly zones for safety reasons. Additionally, some areas may have privacy concerns or are temporarily restricted due to events or other circumstances. It's important to check local regulations and no-fly zone maps provided by drone manufacturers or local aviation authorities before planning any drone flight operations.

4.6.2. Path Planning

The automatic floor robot vacuum uses coverage path planning algorithms to ensure every part of each segment is cleaned. Common path planning algorithms include spiral algorithm, wall follow, 'S' shaped pathway and random walk.

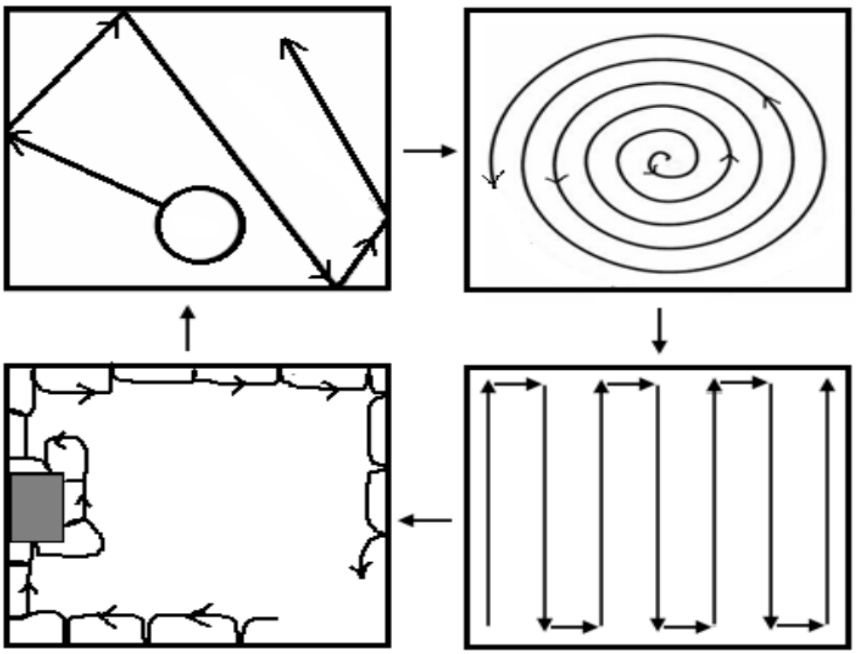


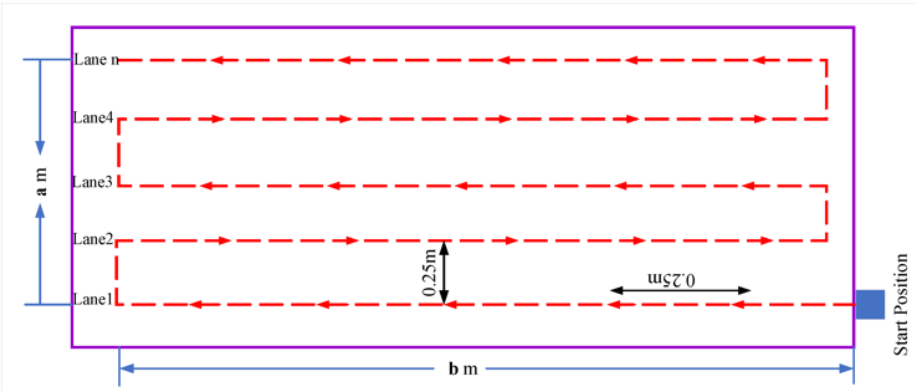
Figure 4-22: Different algorithms executing in cyclic order in the robot's cleaning cycle (Hasan et al., 2014)

Spiral algorithm allows the robot to create an increased circle from centre point till an obstacle is sensed. By 'S' shaped pathway algorithm, the robot starts by moving straight until it encounters an obstacle (such as a wall or furniture). Upon detecting an obstacle, the robot executes two 90-degree turn in the opposite direction and resumes moving straight again. It continues this process, repeating the S-shaped pattern as it navigates the room. The wall follow algorithm allows the robot to move along wall. A random walk does not require any precise realization of the route plan. The robot moves in forward direction until an obstacle is sensed and then it stops if there is any barrier. Next, it turns by comparing sensor readings from the left or right direction in which to turn and finally by generating a random number it decides how much to turn. The robot continues those four algorithms one after another (combined mode) until the whole surface area is fully cleaned.

Khan et al., (2023) developed an automated navigation algorithm that enables the robot called AMSEL to autonomously navigate a predefined rectangular area for pavement inspection, with dimensions of a meters in width and b meters in length. The robot receives these dimensions (a and b) as input through a user interface on a host computer. Upon receiving the input and command to move, the robot initiates its navigation, proceeding lane by lane to collect data. The number of lanes is determined by the width of the inspection area. As depicted in Figure 4-23 (b), the robot employs a stop-and-go method, advancing a fixed distance of 25 cm before stopping to gather data. Algorithm in Figure 4-23 (a) outlines the robot's automated navigation and inspection procedure. According to this algorithm, once the robot gets the navigation command, it calculates the number of lanes (n_l) and the number of steps (n_s) it needs to take in each lane. The number of lanes is calculated by dividing the survey area's width (a) by 0.25 meters, considering the robot's inspection interval is 25 cm. Similarly, the number of steps is calculated by dividing the area's length by 0.25 meters, as the robot moves 25 cm with each step. The robot then begins its task: it captures an image, detects cracks, and once the cracks are identified, the image, along with the robot's location on the grid and the crack density, is sent to the host computer. The robot then moves to the next scanning spot to repeat the process. But the algorithm is unsuitable for irregular maps.

```
1  n_l=a/0.25
2  n_s=b/0.25
3  for i ←(1, n_l) do
4      for i ←(1, n_s) do
5          Capture image
6          Detect Cracks
7          Send image
8          Calculate crack density d
9          Move 25cm forward
10     end
11     if i <n_l then
12         if (i %2)!=0 then
13             Turn right
14             Move 25cm forward
15             Turn right
16         else if (i %2)=0 then
17             Turn left
18             Move 25cm forward
19             turn left
20     else
21         Scan is completed
22     end
23 end
24 if (a %1)!=0 then
25     Turn right
26     Move "a" meter forward
27     turn left
28 else if (a %1)=0 then
29     Turn left
30     Move "a" meter forward
31     Turn left
32     Move "b" meter forward
33     Turn left
```

(a)



(b)

Figure 4-23: Automated navigation of robot AMSEL: (a) Algorithm for automated navigation and inspection and (b) the diagram of the survey area for the robot (Khan et al., 2023)

To conduct defect inspection of airport runways, Gui and Li (2020) used multi-point navigation method to allow the robot to navigate within a predefined surveyed region on the airport runway to collect images and GPR data, as shown in Figure 4-24. The robot begins by moving from the starting point to point A, then follows a straight path for each scan. The sensor suite covers a 1-meter width per scan. After completing the current scan, the robot moves to the next one, continuing until the entire area is fully covered. At the end of each scan, the robot navigates to the turn point using an omni-directional path, then shifts to the adjacent scan to ensure no areas are missed.

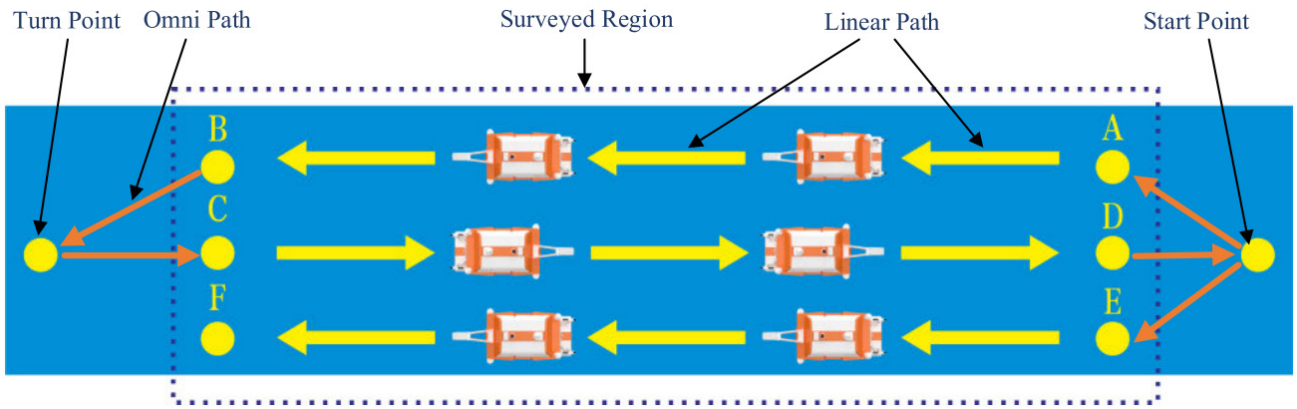


Figure 4-24: Schematic of the robot path planning on the airport runway (Gui and Li, 2020)

In view of the advantages and disadvantages of the above three options, this work will utilize a multi-point navigation method for the robot. Users are enabled to choose specific areas for inspection and define several target points for the robot via a Python script. As the robot navigates through each target point, it concurrently conducts an inspection of the ground for cracks. In regions where inspection is unnecessary, the robot will bypass those areas and avoid performing crack detection. This method helps to save time.

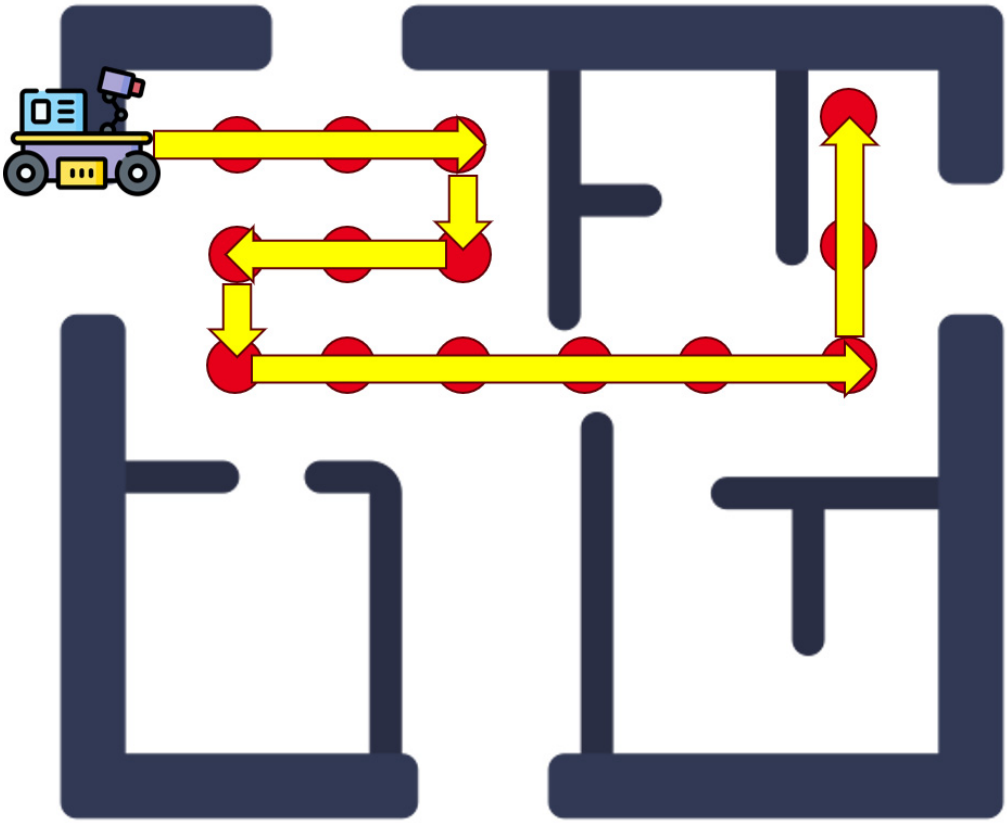


Figure 4-25: Schematic of the robot path planning for cracks inspection via multi-point navigation method

It should be noted that the multi-point navigation approach by Gui and Li (2020) was feasible because their robot's sensing suite Ground Penetrating Radar can cover a 1-meter width per scan. In contrast, the robot under development in this work does not aim to incorporate a spraying system with multiple nozzles and cameras at this stage due to its complexity. To ensure comprehensively inspection of the ground for cracks, the number of target points will be increased to avoid missing any region.

4.7. Recording and Visualizing Wide Cracks

The bio-concrete repair agent used in this work is only suitable for narrow cracks. However, in reality, concrete surfaces cannot be expected to have only narrow cracks. If wide cracks are not promptly detected and repaired, the problems they cause can be more severe than those caused by narrow cracks.

4.7.1. System Overview

Figure 4-26 provides a comprehensive summary of the concepts for recording and visualizing the wide cracks.

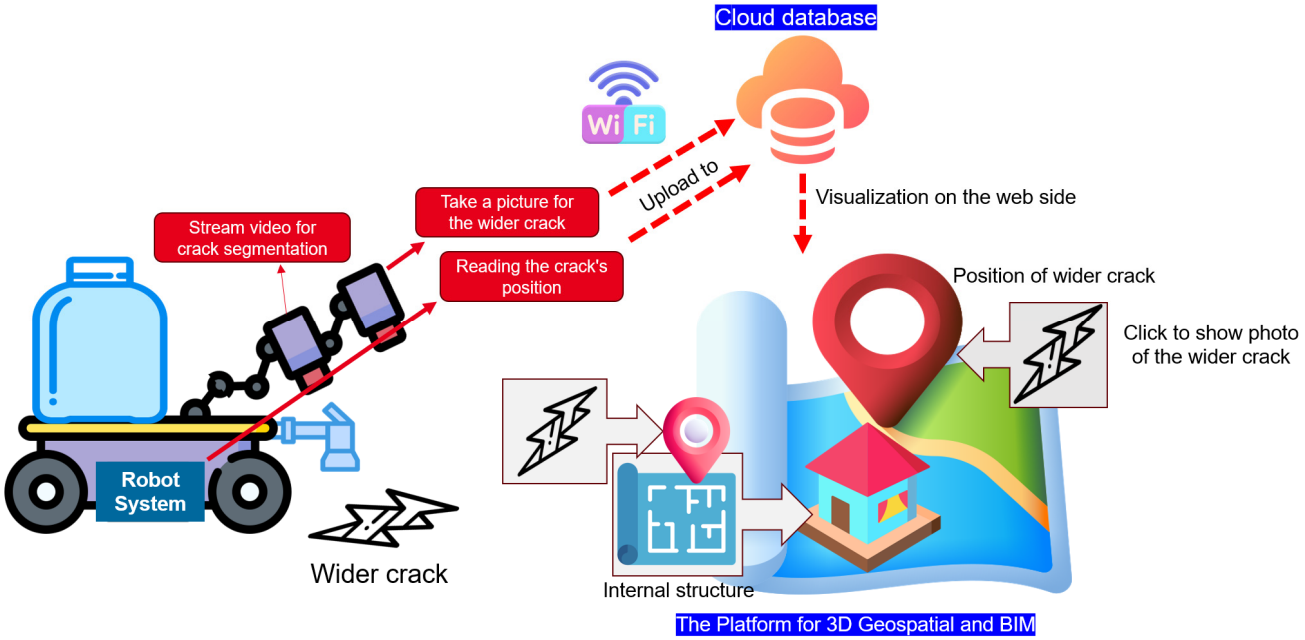


Figure 4-26: System overview of wide cracks recording and visualizing

When recording wide cracks, it's important not only to determine their location relative to the map but also to have photographs of the cracks. This visual information assists engineers in assessing the cracks and selecting the most appropriate repair methods. Having both the location data and images of the cracks provides a comprehensive understanding of the issue, which is crucial for effective maintenance and repair planning. Taking photographs of wide cracks has an additional benefit: machine learning models may not always be 100% accurate in identifying wide cracks. Dirt on the ground, leaves, and other debris might be mistakenly recognized as wide cracks. To avoid being misled by errors in the machine learning model, engineers can use the photos to double-check whether the recorded areas indeed have cracks that require repair. This visual verification step ensures that the maintenance and repair actions are targeted and accurate, reducing the risk of unnecessary interventions and improving the efficiency of the inspection process.

Typically, a USB camera can both stream video and capture images simultaneously. However, the system developed in this work will employ two separate cameras—one dedicated to providing real-time video feeds to the machine learning model and the other dedicated to capturing images. This configuration offers the benefit of using a wide-angle or panoramic camera for image capture, allowing for a broader field of view when necessary. In contrast, the camera providing the real-time video feed cannot use a wide-angle or panoramic lens, as such lenses would cause significant distortion of image that could compromise the accuracy of image segmentation.

The captured images of cracks and their location information will be uploaded to a cloud database via a wireless network, rather than being stored locally. The advantage of doing so is that engineers can quickly view the results through a web application on their mobile phones or tablet devices. They can collaborate to give a comprehensive assessment of the cracks and adopt appropriate repair measures after consultation. This approach facilitates efficient and informed decision-making in maintenance and repair processes.

In the web application designed in this work for visualizing and assessing structural cracks, especially in large buildings, it is beneficial to integrate BIM and GIS technologies. This integration provides a comprehensive platform that includes both geographical and architectural information, which is essential for engineers when evaluating cracks. Here's how such a system would work:

1. **Red Point Indicators:** Wide cracks are marked with red points on the map within the web application, making them easy to locate.
2. **Photo Visualization:** Clicking on these red points brings up the corresponding photos of the cracks, allowing for a visual assessment.
3. **BIM-GIS Integration:** The application combines BIM for detailed architectural information with GIS for geographical context. This is particularly useful as it allows engineers to understand not only where the crack is located within the structure but also how it relates to the surrounding environment.
4. **Building Information:** Since cracks can affect the structural integrity of a building, having access to detailed building information is crucial for engineers to assess the severity and potential impact of the cracks.
5. **Internal and External Views:** For large buildings, being able to visualize both the external structure and internal components is important. The ability to 'see through' the building allows users to identify the location of internal cracks.
6. **Various Locations:** The detection locations may be far apart, and users can easily switch between multiple locations.

This integrated approach using BIM and GIS within a web application ensures that all relevant information is readily available, facilitating more informed and efficient decision-making in maintenance and repair operations.

4.7.2. Crack Localization and Visualization

Crack localization is essentially equivalent to the robot's real-time positioning, which can be achieved through various technologies, depending on the environment and the required accuracy. Yahboom GNSS module is a positioning and navigation module. It supports BeiDou Navigation Satellite System in China, GPS, GLONASS (Global Navigation Satellite System) in Russia, and QZSS (Quasi-Zenith Satellite System) in Japan. It can be connected to common development boards such as computer, Raspberry Pi, Jetson, etc. The seller Yahboom supplied a Python script specifically designed for the Jetson Nano. By executing this script, the Jetson Nano is enabled to incept the GNSS signals. The geographic coordinate system used in this GNSS module is also WGS84.

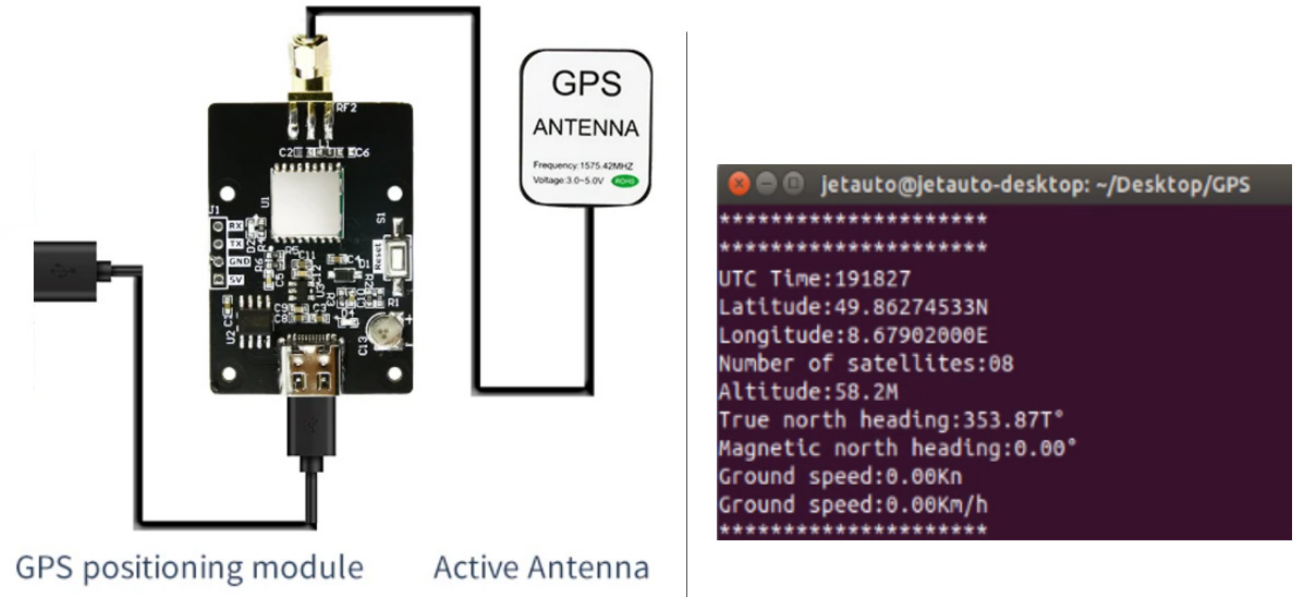


Figure 4-27: Yahboom GNSS module (left) and the coordinate data acquired by this GNSS module on the Jetson Nano terminal (right)

However, the module has an error of up to 2.5 meters. The precision required for locating small objects like cracks may not be sufficiently met by GNSS alone, especially in indoor environments or certain outdoor locations where signal strength is weak. For such precise positioning needs, alternative technologies may be more appropriate. Many open-source robots like Hiwonder and Yahboom are now equipped with LiDAR sensors. With the aid of a map, these robots can perform real-time positioning using LiDAR, and the data can be visualized in real-time using RViz. When robot detects a wide crack, it only needs to read its real-time position of robot relative to the map. This position corresponds to the location of the wide crack on the map. As shown in Figure 4-28, to acquire the robot's real-time position, the single-board computer running the machine learning algorithm must send command to the onboard computer. The onboard computer then uses the ROS tool to retrieve the robot's real-time position. This real-time position data represents the three-dimensional distance relative to the robot's initial position, with the Z-axis remaining constant. For visualization on a unified 3D map, such as on a platform like Cesium, the robot's real-time coordinates must be converted to the WGS84 coordinate system. This conversion is necessary because WGS84 is the global standard for geographic coordinates and is widely used for integrating data from various sources into a consistent geographic framework.

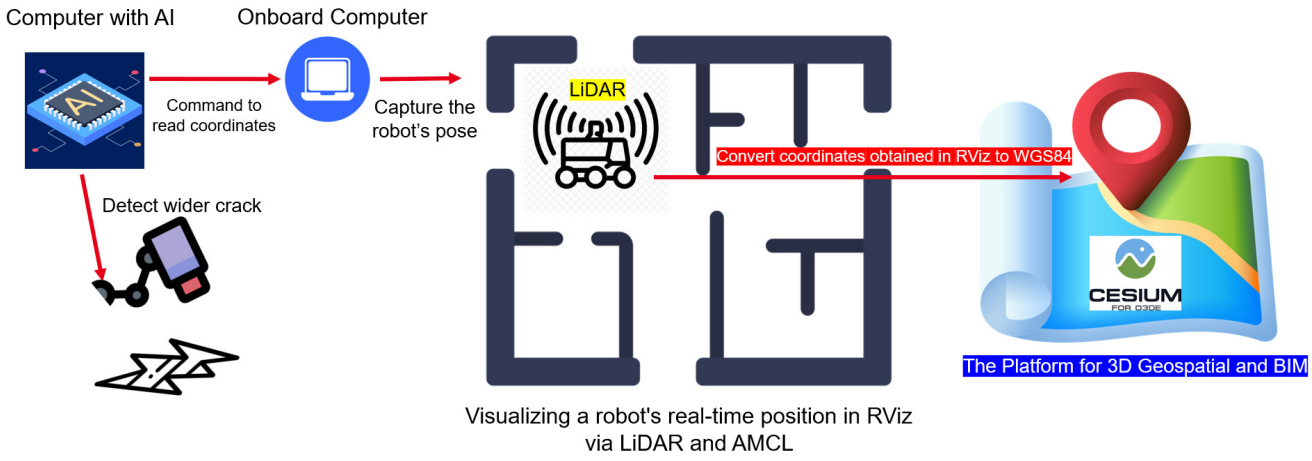


Figure 4-28: Schematic diagram of the crack localization principle by the robot

5. Implementation of AI-Based Robot CrackRepairBot

This chapter presents the implementation of the AI-based robot CrackRepairBot. The goal is to develop a comprehensive system for concrete crack repair, where the robot integrates the functionalities of cracks detecting, recording, and repairing. Building on the concepts outlined in Chapter 4, specific software and hardware components will be selected accordingly. Open-source software will be prioritized for software selection, while the hardware components will be chosen from affordable, cost-effective, mature, freely available options on the market. These selection criteria aim to minimize reliance on a single supplier and make it easier to replace components if necessary. Notably, after selecting the appropriate robotic development platform, the corresponding spraying system will be constructed on this platform.

5.1. Building the Robot

In the process of constructing the robot, the interrelationship between hardware, software, and functionality must be considered. Hardware selection will directly affect the development of the corresponding software and the implementation of functionalities.

5.1.1. Hardware Selection

Robotic Platform

The open-source robotic platform selected for this work is Hiwonder's JetAuto Pro, which is powered by NVIDIA Jetson Nano B01. It contains an affordable 2D LiDAR, the SLAMTEC A1, and features 2D SLAM mapping algorithms, as well as existing capabilities for dynamic obstacle avoidance and route planning. With the LiDAR, users can manually set up multi-point navigation through RViz. The system of JetAuto Pro is Jetpack 4.6 including ROS 1. With 4 omnidirectional mecanum wheels, JetAuto Pro can move 360°. The default movement speed of the robot is about 0.2 m per second. The movement speed of the robot can be adjusted by modifying the code.

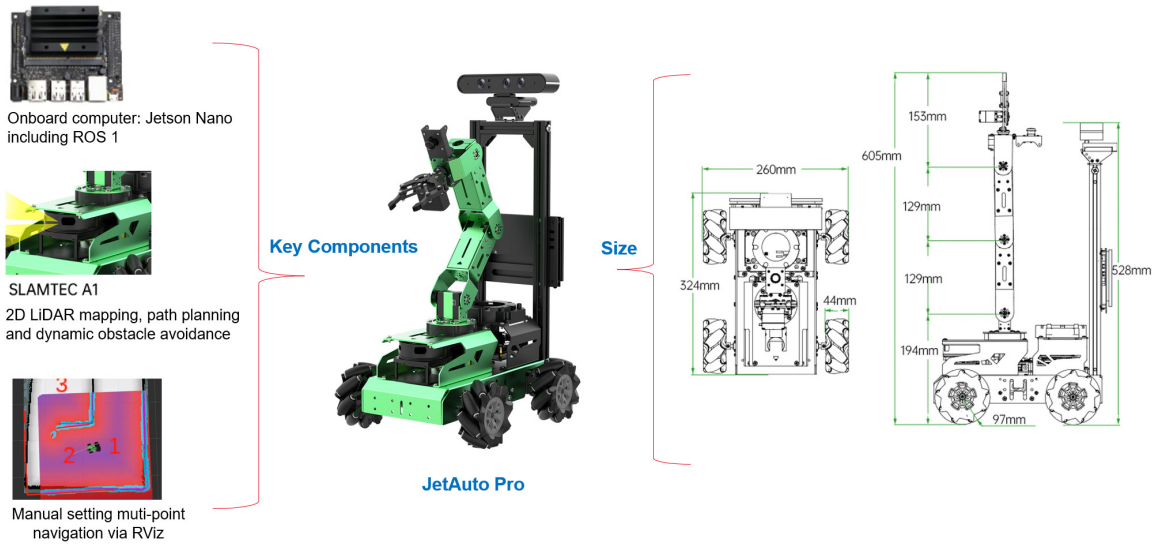


Figure 5-1: Key Components and size of JetAuto Pro (Hiwonder, 2024)

Pressurizable Water Bottle and Solenoid Water Valve

The robot measures 324 millimeters in length and 260 millimeters in width, with a height of 528 millimeters at the rear of the body. Based on the structure of the robot's body, there is a perfect fit for a 1-liter pressurizable water bottle with a base width of 115 millimeters and a height of 285 millimeters at the back, which is used to hold the repair agent or water. Users can manually pressurize the water bottle by using a pressure rod.

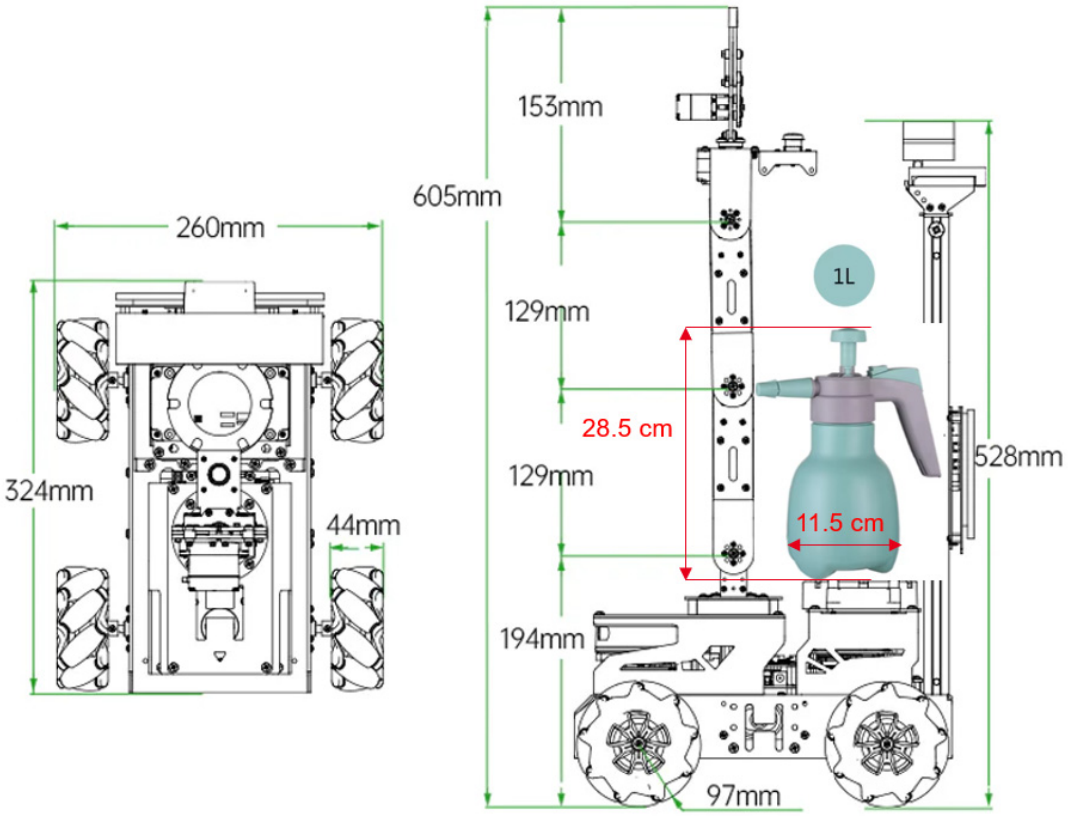


Figure 5-2: Schematic of mounting the pressurizable water bottle on robot

When selecting solenoid water valve, it's crucial to avoid those with excessively small orifices, as they are prone to clogging due to suspended particles in the repair agent. Most solenoid water valves on the market have an input voltage of 12V or higher, which can be powered by dry cell batteries.

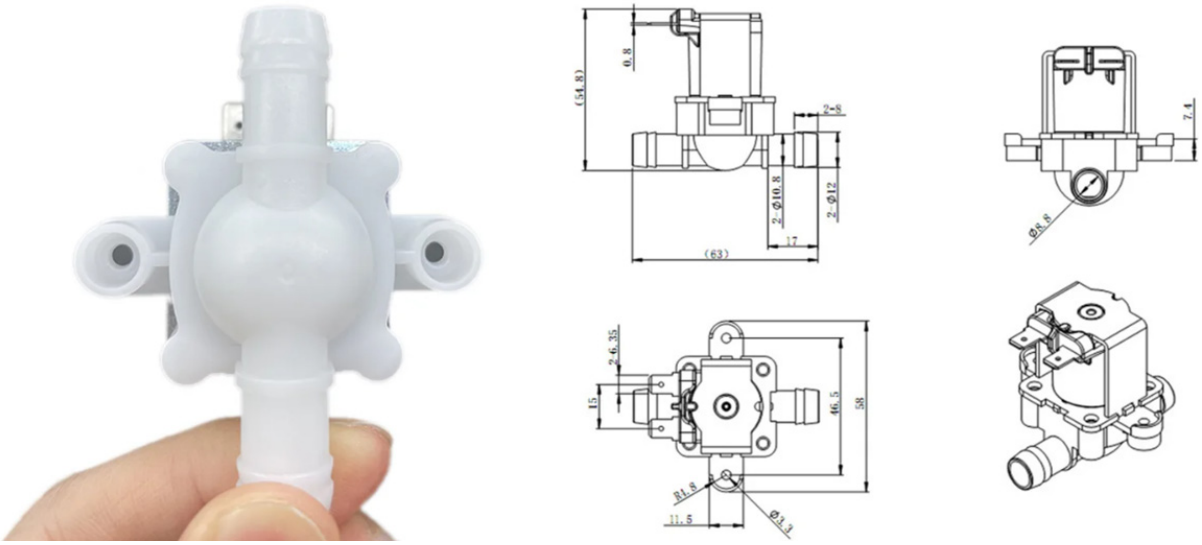


Figure 5-3: The size of solenoid water valve used in this work (AliExpress, 2024)

The 12V solenoid valve is powered by 8 x 1.5V dry cells placed in a plastic battery storage box with an ON/OFF switch and cables for connection.

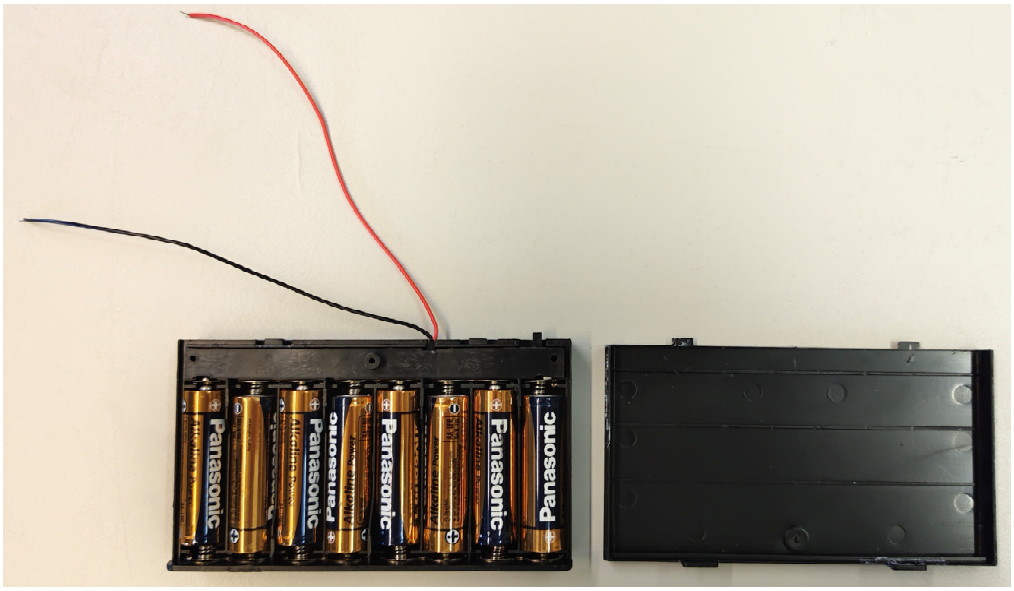


Figure 5-4: Plastic battery storage box and 8 x 1.5V dry cell

The robot is equipped with the ELEGOO Arduino Uno R3, sourced from AliExpress, which offers a considerably more affordable alternative to the official Arduino version. Additionally, the 5v relay component, procured from AliExpress, is priced at only about 1 Euro.

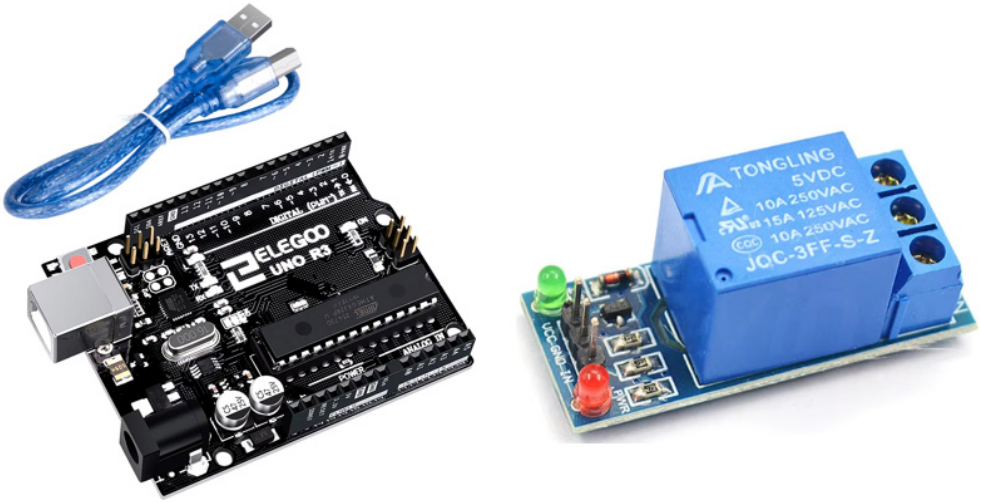


Figure 5-5: ELEGOO Arduino Uno R3 (left) and 5V relay module (right)

USB Camera

The NVIDIA Jetson Nano and Jetson Orin Nano are AI development boards designed for edge computing and support various types of cameras, including CSI (Camera Serial Interface) and USB interfaces. If a camera is not compatible with the devices, user may encounter issues such as the camera suddenly not being detected. This can happen due to various reasons, such as driver issues. The built-in camera on the robotic arm of JetAuto Pro has this compatibility issue during testing. As shown in Figure 5-6, sometimes, although the Jetson Nano terminal shows that the camera is detected, an error occurs when calling the camera, indicating that the camera cannot be opened.

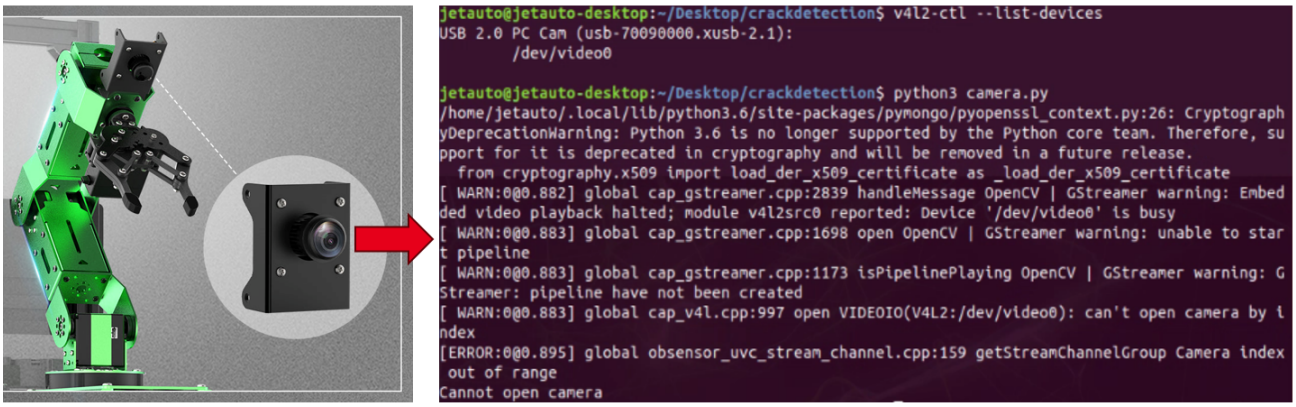


Figure 5-6: Compatibility issues with the built-in camera of JetAuto Pro

To ensure stable image capture during the movement of a robot, it's essential to have a secure and steady mounting for the camera. One effective approach is to utilize the robot's own mechanical arm to fix the camera in place. This setup can significantly reduce the impact of vibrations and maintain a consistent angle and distance from the object being observed. The clarity of the image captured by a camera can affect the performance of crack segmentation. If a camera is mounted at the robot's own mechanical arm and it observes objects at a distance, the details in the captured image may be lost. Using a USB camera with optical zoom capabilities, such as one that offers 10x zoom, can help mitigate this issue. By zooming in on the scene, computer can capture a closer view of the objects or areas of interest. And the overall image quality also depends on other factors such as the camera's sensor size, lens quality, and lighting conditions. Additionally, if the robot is moving quickly, a camera with a high frame rate can help capture clear images without motion blur.

In summary, the USB camera used for photographing wide cracks must at least be compatible and have a sufficient frame rate. The USB camera used for providing real-time video feed to machine learning models must support zoom functionality and deliver a sufficiently high frame rate. Figure 5-7 shows some suitable USB cameras for the robot that can run smoothly on Jetson Nano and Jetson Orin Nano. (a) is the Logitech C270 camera, which is inexpensive, costing about 10 euros, with a frame rate of 30 FPS, but the image quality is only 720p. It can be used as an entry-level USB camera for capturing wide cracks. (b) refers to the Intel Realsense 3D camera, which is a depth camera commonly equipped on robots, priced at approximately 300 euros with a frame rate of 30 FPS and image quality of 1920 × 1080. It is not zoomable and can be used for capturing wide cracks and 3D mapping. (c) is a 10x optical zoom USB industrial camera, priced at approximately 60 euros, it has a resolution of 3840x2880 with an aperture of f/2.8, supports 30 FPS, operates at a working voltage of 5V, features manual focus, and is compatible with Windows, Linux, Android, and macOS systems (tmall, 2024).

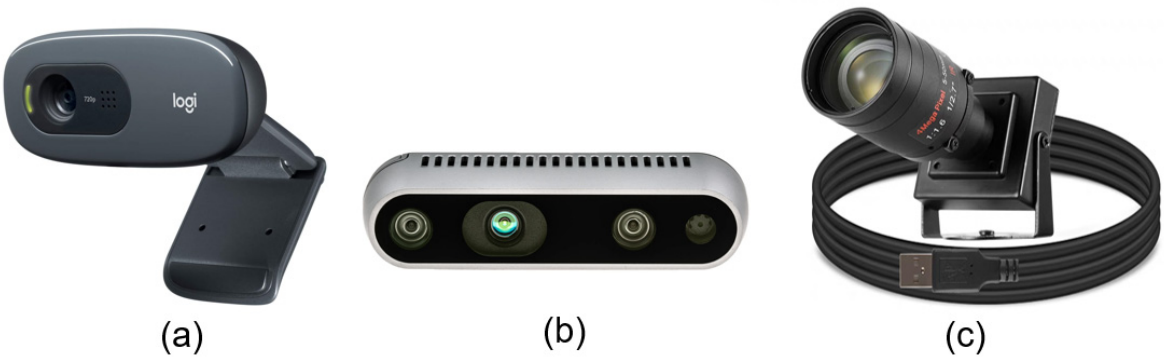


Figure 5-7: Recommended compatible USB cameras for the robot

It's important to note that all zoom-capable cameras have some degree of image distortion, which is a natural consequence of the optical design required to achieve variable focal lengths. Figure 5-8 (a) shows the result of a 1-millimeter-wide crack being magnified by the camera (c) shown in Figure 5-7, there is a slight distortion at the bottom of the image and some slight blurring around the edges, but it is acceptable. If a zoomable camera produces serious distortion as shown in Figure 5-8 (b) when magnifying an image, then such a camera is not suitable for crack segmentation. To minimize image distortion, common approaches include using high-quality lenses specifically designed to reduce distortion or applying software-based post-processing techniques for distortion correction.

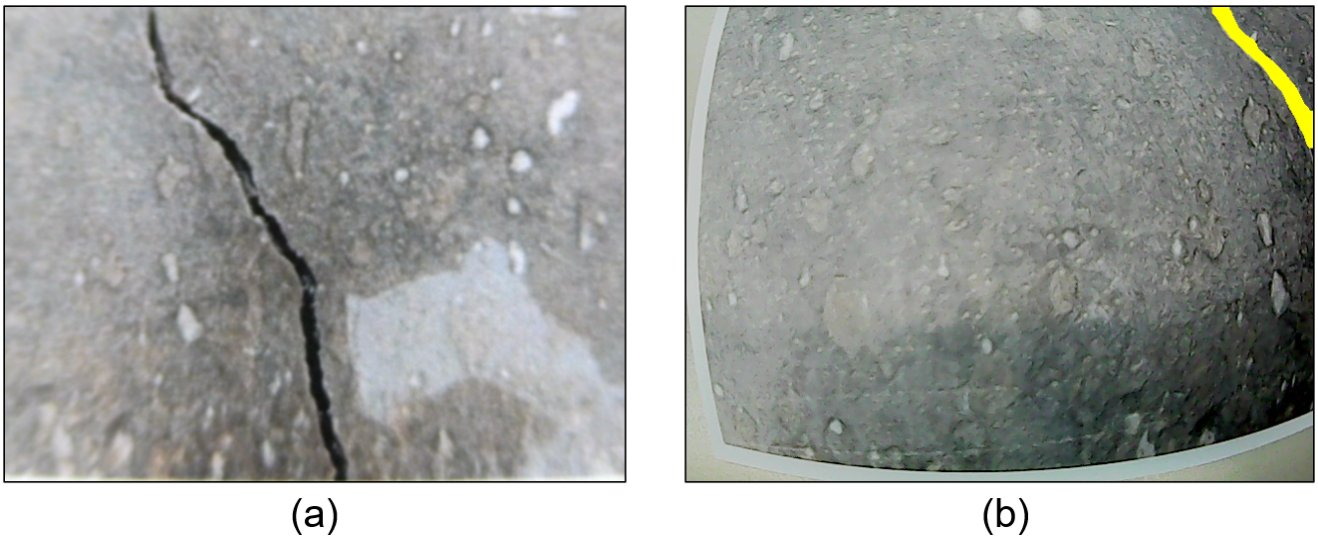


Figure 5-8: Different effects of zooming in on images with different types of zoomable camera

It is also important to note that the larger the aperture, the more light enters the camera. In outdoor environments with intense sunlight, the brightness setting of the camera needs to be lowered. For dimly lit scenes, a camera with a larger aperture would be more appropriate. For a moving robot, a faster frame rate can capture moving objects more clearly. Selection for a 30 FPS camera is often due to its cost-effectiveness, while higher frame rate cameras are available at a higher cost. Whether a 30 FPS camera is suitable or not depends on the specific requirements of the application it will be used for. In other words, these selected 30 FPS cameras need to be tested in certain scenarios.

Mini-Computer for AI

This work ultimately chose the Jetson Orin Nano as the "brain" for the robot. At this stage, the Jetson development board not only has very high computing power but also has very strong official support. The Jetson Orin Nano is ideal for learning and prototyping and supports various AI models with a robust software stack. It is the latest and most affordable model in Nvidia's Orin series, the discounted price for students and the educators is $399. It's computing platform with 40 TOPS of AI performance is capable of running LLMs and VLMs that are around 7 billion (7B) parameters in size (NVIDIA, 2024d).

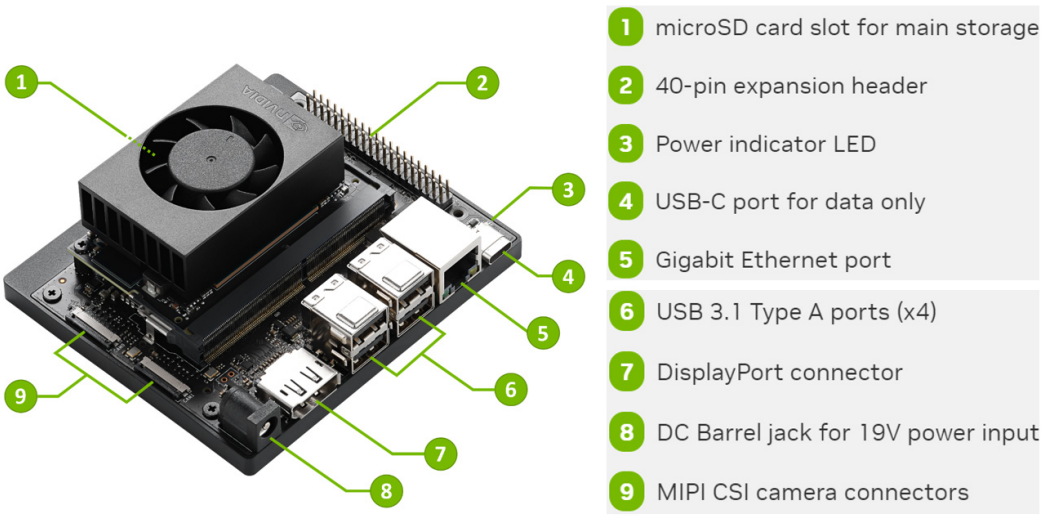


Figure 5-9: Construction of Jetson Orin Nano (NVIDIA, 2024e)

The Jetson Orin Nano requires a 19V DC power supply for operation. Therefore, a 12V lithium battery along with a compatible voltage converter was selected.

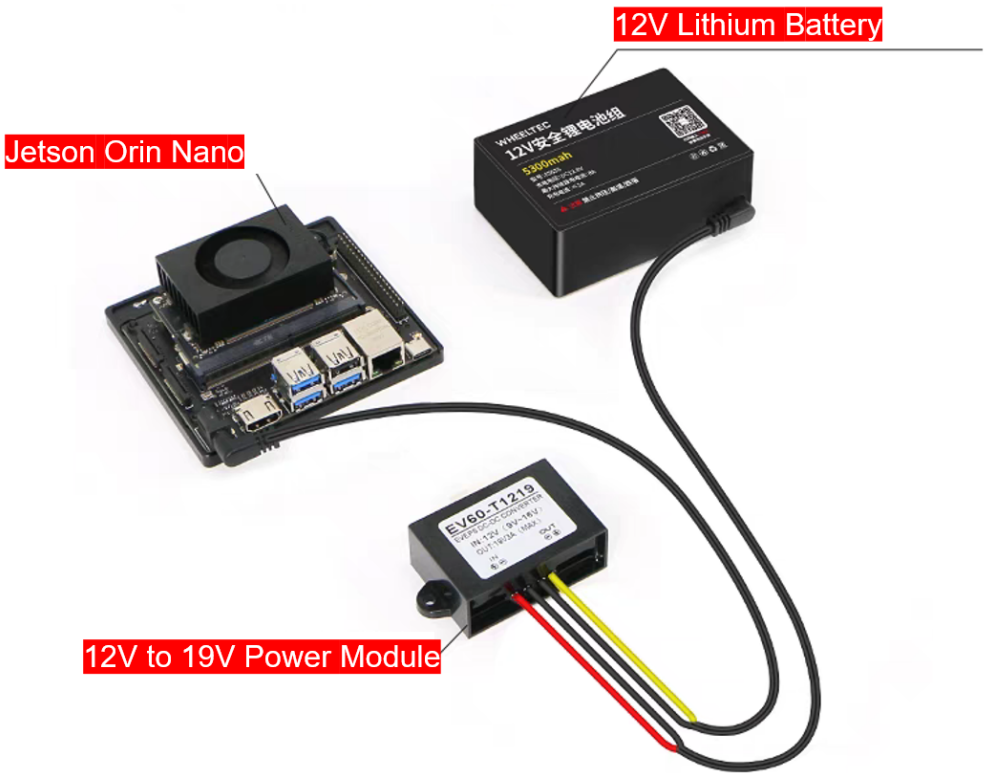


Figure 5-10: Lithium battery and voltage converter for Jetson Orin Nano

5.1.2. Prototype

Figure 5-11 shows the construction of the robot. The Jetson Orin Nano is placed between the robotic arm and the water bottle. Rubber bands are used to secure the device. The outlet of valve is connected to a long spray pole to bring the adjustable nozzle as close to the ground as possible, and it is fixed behind the camera. Because there is a short time interval between the detection of narrow cracks and the activation of the spraying system. And during

this interval, the robot moves slightly forward. The positive and negative terminals of the valve are connected with alligator clip wires for a more secure connection. Before the robot was put into operation, it is necessary to manually apply enough pressure inside the bottle via pressurized rod. The bottle is connected to the valve through a flexible hose. Due to the high pressure inside the hose, hose clamps are required at the joint to ensure a secure and tight connection. For capturing photos of wide cracks, either Intel Realsense 3D camera or Logitech C270 camera can be utilized. The robotic arm is used to hold the 10x zoom-capable USB industrial camera on the front of the robot body, this camera is applied for feeding the real-time video to machine learning model.

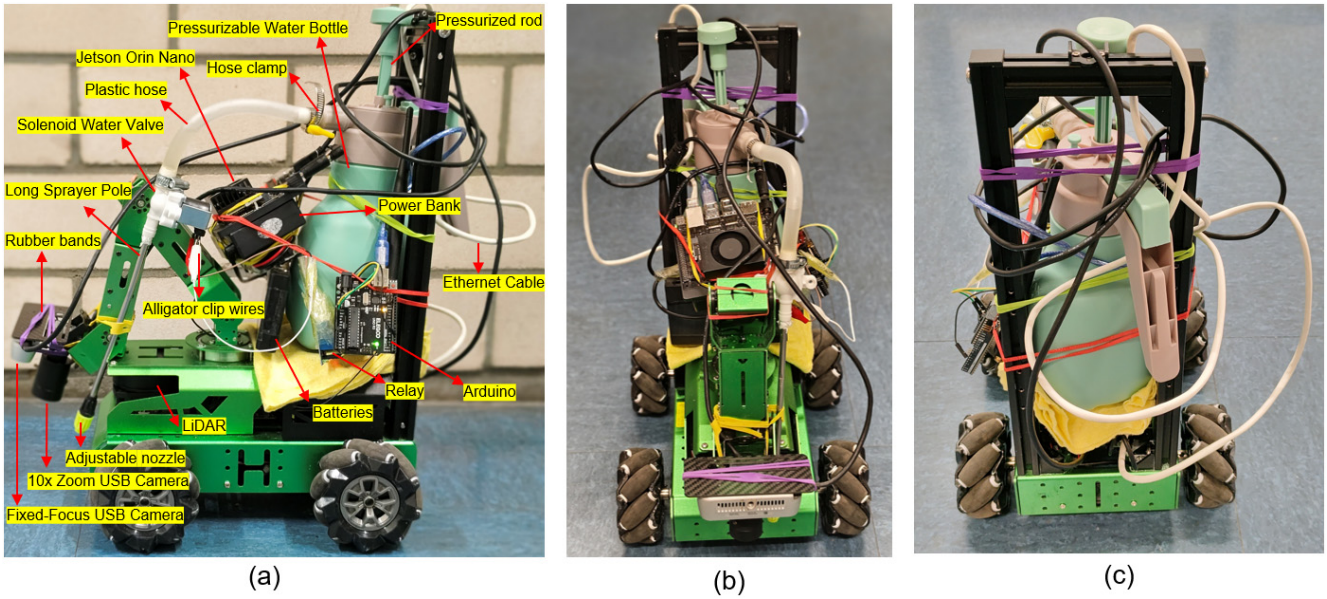


Figure 5-11: Left, front and rear views of the robot

5.1.3. System Connection

The connection between the devices and the transfer of data is shown below.

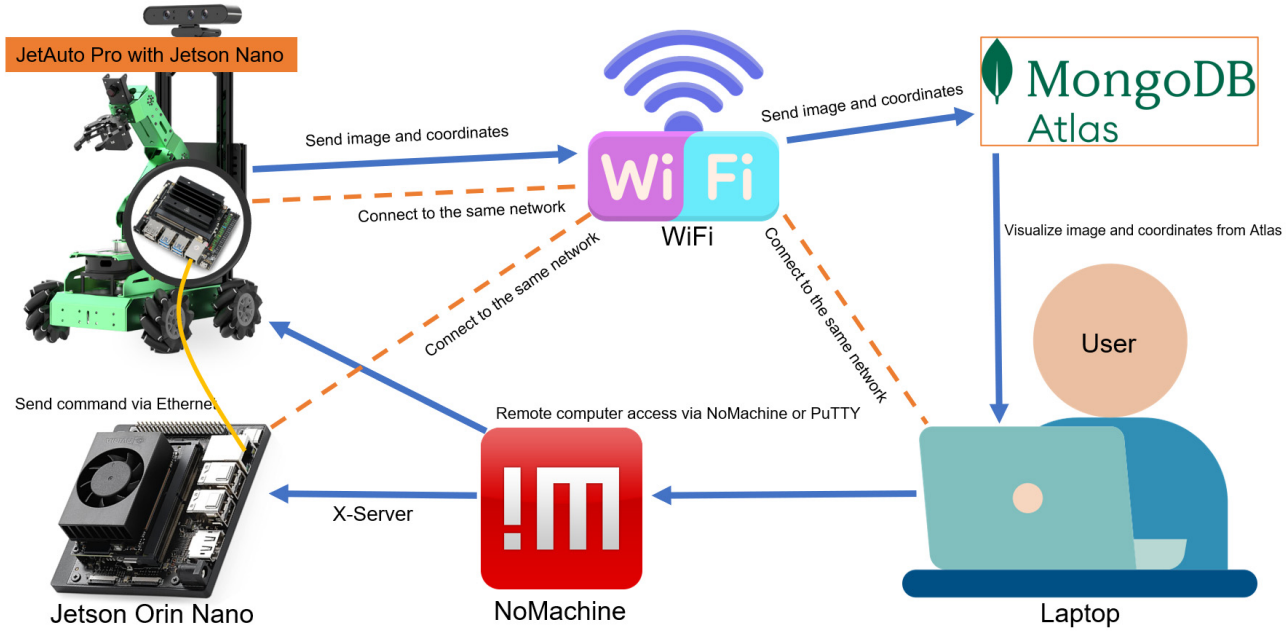


Figure 5-12: Schematic diagram of the connection between the devices and the transfer of data

Wireless Connection

In this work, the robot's control is conducted via a remote computer, primarily running code instructions and visualizing the robot's operation on the remote computer end. NoMachine is an excellent free remote desktop visualization software for this requirement.

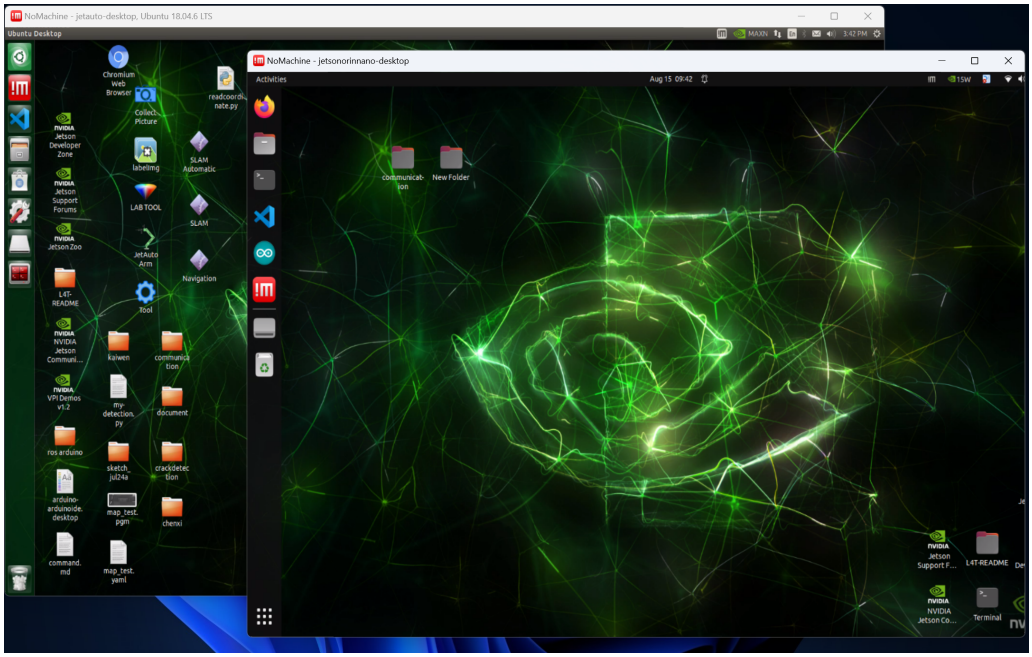


Figure 5-13: Remote desktop control of Jetson Orin Nano and Jetson Nano in NoMachine

To set up remote control functionality, NoMachine software must be installed on the user's laptop, Jetson Nano, and Jetson Orin Nano. Then, all devices must be connected to the same wireless network in order to allow the user to access and control the desktops of Jetson Nano and Jetson Orin Nano via NoMachine installed on the laptop. And through the wireless network, the Jetson Nano can upload the coordinates and photos to a cloud database such as MongoDB Atlas. The wireless network can be provided by the mobile hotspot of a smartphone, which allows the robot to connect to the internet and receive commands from the remote computer without the need for a physical network connection. This setup can be particularly useful in situations where a stable Wi-Fi network is not available or when the robot needs to operate in remote locations. By using a smartphone's hotspot feature, the robot can maintain connectivity and perform its tasks effectively.

An X-server allows user to display graphical problems from a UNIX/Linux machine on local computer. When deploying NoMachine on Jetson Orin Nano, it is necessary to install an X-Server on the Jetson Orin Nano to handle the graphical user interface. Without the X-Server, user will encounter issues with the "headless" problem, which refers to the inability to display a graphical interface without a physical monitor connected to the device. X-Server can create a virtual display in Jetson Orin Nano that allows NoMachine to function correctly even when no physical display is connected to the Jetson Orin Nano. When deploying NoMachine on Jetson Nano, the "headless" problem does not arise. Although the Jetson Orin Nano and Jetson Nano are both single-board computers developed by NVIDIA, there are always some differences in their systems.

Wired Connection

Communication between the Jetson Nano and the Jetson Orin Nano needs to be real-time with no delays. Ethernet is a physical and data link layer technology for the local area network (LAN). It offers high-speed connections

that are typically faster than wireless communications, especially for Gigabit Ethernet and beyond. This results in quicker data transfers and lower latency. Ethernet connections are more reliable than wireless ones because they are less susceptible to interference and signal degradation. Setting up an Ethernet network can be straightforward, often requiring just the cable and proper configuration. It is necessary to configure the respective IP (Internet Protocol Address) addresses of the Jetson Nano and Jetson Orin Nano to establish a LAN via an Ethernet cable. As shown below, the IPv4 (Internet Protocol version 4) Method should be changed to Manual mode. Netmask is set as *255.255.255.0*. Each device has own different IP addresses. The IP address of Jetson Nano *192.168.110* will be used as the address of the information receiver. The IP address of Jetson Orin Nano will be used as the address of the information sender.

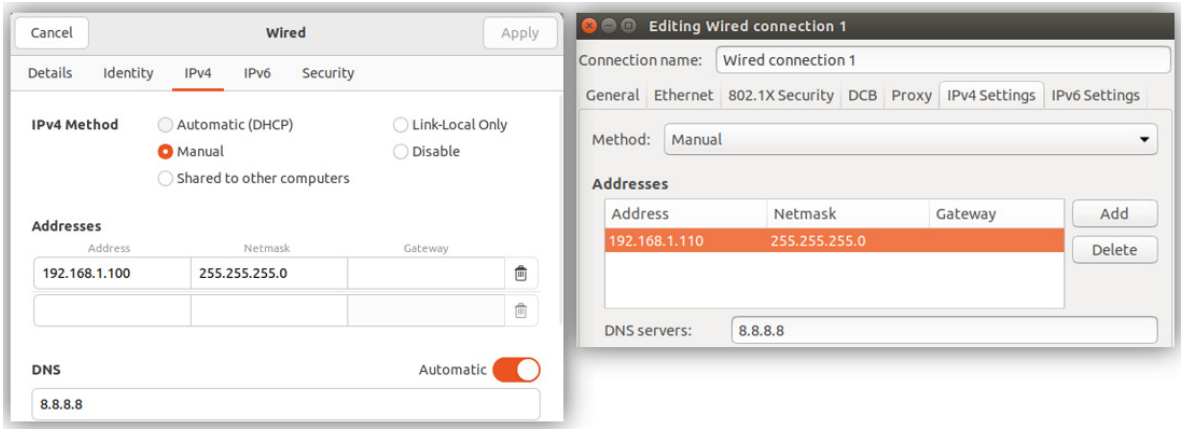


Figure 5-14: IPv4 setting of Jetson Orin Nano (left) and Jetson Nano (right)

For communication, initially, the Python script must be executed on the Jetson Nano to establish the server's general-purpose port, as shown in Figure 5-15 (left). Following this, the Python script on the Jetson Orin Nano should be run to establish the connection and facilitate the sending of information, as shown in Figure 5-15 (right). To maintain a persistent communication link, both scripts implement try and while loops. Without this structure, the network connection would terminate after a single message transmission.

```
import socket

server_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

server_address = ('192.168.1.110', 12345)
server_socket.bind(server_address)

server_socket.listen(1)

print("Waiting for client connection...")
connection, client_address = server_socket.accept()

try:
    print("Connected successfully with:", client_address)

    while True:
        data = connection.recv(1024)
        if data:
            message = data.decode()
            print("Received message:", message)
            connection.sendall(b'acknowledged')

            if message == 'quit':
                break
        else:
            print("Connection with client is interrupted")
            break

finally:
    connection.close()
```

```
xia@xia-desktop:~/communication$ python jetsonnano.py
Waiting for client connection...
('Connected successfully with:', ('192.168.1.100', 46026))
('Received message:', u'hallo world')
('Received message:', u'hi are you ok ?')
```

```
import socket

client_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

server_address = ('192.168.1.110', 12345)
client_socket.connect(server_address)

try:
    while True:
        message = input("Enter your message (type 'quit' to exit): ")
        print("Sending message:", message)
        client_socket.sendall(message.encode())

        if message == 'quit':
            break

        response = client_socket.recv(1024)
        print("Received response:", response.decode())

finally:
    client_socket.close()
```

```
xia@ubuntu:~/communication$ python jetsonorinnano.py
Enter your message (type 'quit' to exit): hallo world
Sending message: hallo world
Received response: acknowledged
Enter your message (type 'quit' to exit): hi are you ok ?
Sending message: hi are you ok ?
Received response: acknowledged
Enter your message (type 'quit' to exit):
```

Figure 5-15: Python code and the messages communication by Jetson Nano (left) and Jetson Orin Nano (right)

In the Python script, the tool socket is used. It is a fundamental tool in network programming, enabling devices to communicate over a network. It serves as an endpoint for sending and receiving data between a server and a client on a network. Python's socket library provides a robust interface, allowing developers to implement connections over both TCP (Transmission Control Protocol) and UDP (User Datagram Protocol) protocols.

5.2. Building the Map for Robot

JetAuto Pro is pre-installed with some mapping methods such as Frontier, Explore_Lite, RRT, Cartographer, Karto, Hector and Gmapping. Frontier map exploration algorithm is semi-automatic for robot coordination. The robot needs to manually specify the target points in RViz before it keeps moving to the nearby frontier to reduce the size of the unknown region. The Explore_lite is an autonomous exploration algorithm. It employs a greedy, frontier-based approach for exploration, where the robot actively seeks out the boundary between known and unknown spaces (frontiers) and moves towards them to uncover new areas of the environment until no more frontiers are detectable. However, mapping using Explore_lite is relatively time-consuming and may get confused by transparent doors as shown below and continuously stuck on the threshold.

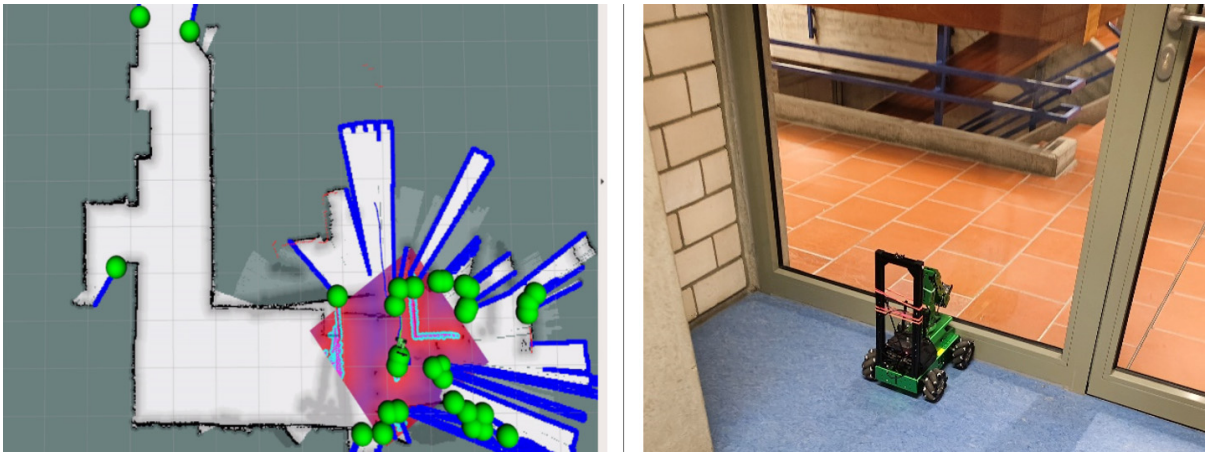


Figure 5-16: Problems encountered when mapping using Explore_lite of JetAuto Pro: bad mapping result (left) and stuck on door (right)

GMapping is a well-known package in the ROS ecosystem, it enables a mobile robot to construct a 2D map using laser and pose data collected during navigation. However, when utilizing GMapping to create maps of complex indoor structures, certain areas of the resulting map exhibit slight distortions, as shown below.

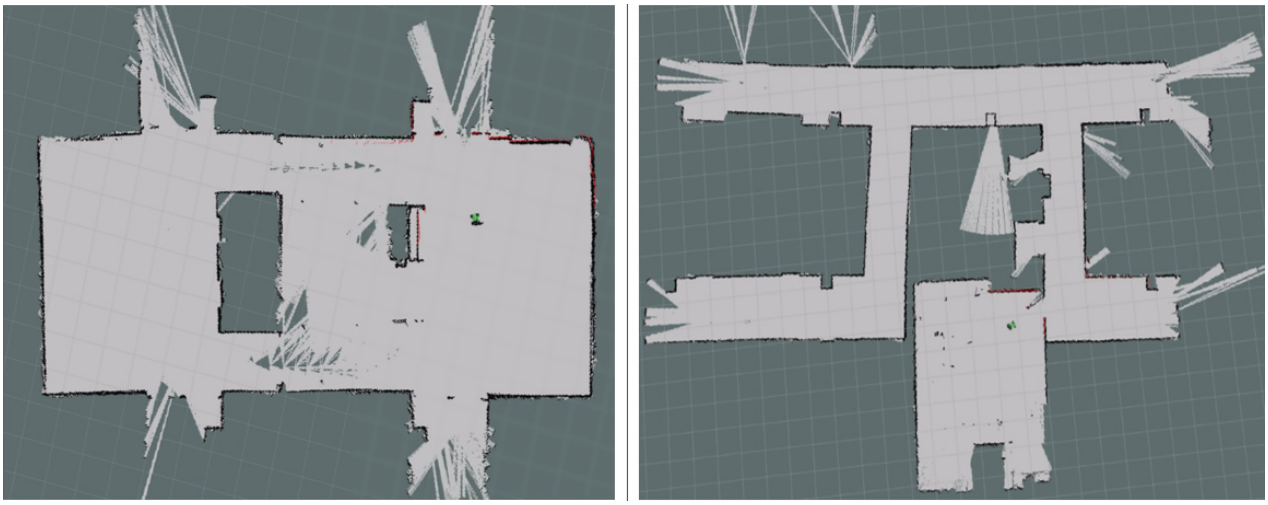


Figure 5-17: Mapping effects in different indoor areas using GMapping of JetAuto Pro

The 2D LiDAR of JetAuto Pro is mounted at a height of 10 centimeters from the ground. If the edge of the area is less than 10 centimeters, the map generated is of low quality, as shown below.

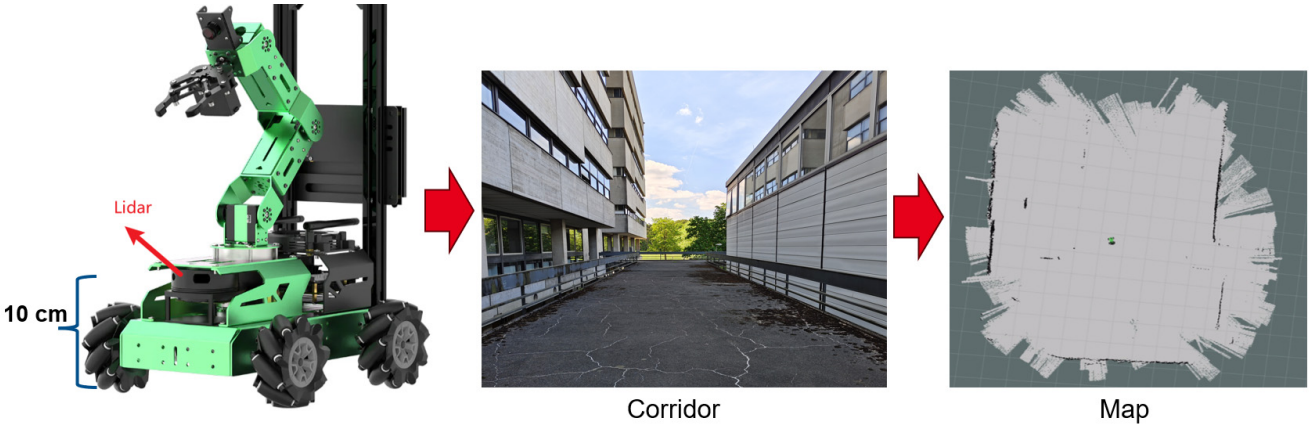


Figure 5-18: Mapping using GMapping of JetAuto Pro by corridor

The 2D LiDAR has an effective measuring range of only 12 meters. It is not suitable for mapping in open outdoor environments. As shown below, the area marked by the red rectangle on the left is the area where the map is planned to be built, and on the right is the result of the generated map which is of very poor quality and is unusable.

Figure 5-19: Mapping using GMapping of JetAuto Pro in open outdoor environment

It can be seen that there are many shortcomings in using the robot's own mapping algorithm, so this work proposes other mapping methods. The IFC format includes information about the geometry, materials, schedules, and quantities of building elements, as well as the spatial relationships between them. If the IFC model of a building is available, the IFC models can be directly converted into 2D robot-readable maps in PGM format, thereby eliminating the need for the machine-generated mapping step altogether. PGM format is a simple grayscale image file format commonly used to store two-dimensional grid data such as maps or images. It can typically represent different regions using grayscale values and can be used in applications such as robot navigation. Tools such as Blender can be used to first convert a IFC model into a 2D PNG (Portable Network Graphics) format image, and then subsequently transform the PNG format image into a PGM format image.

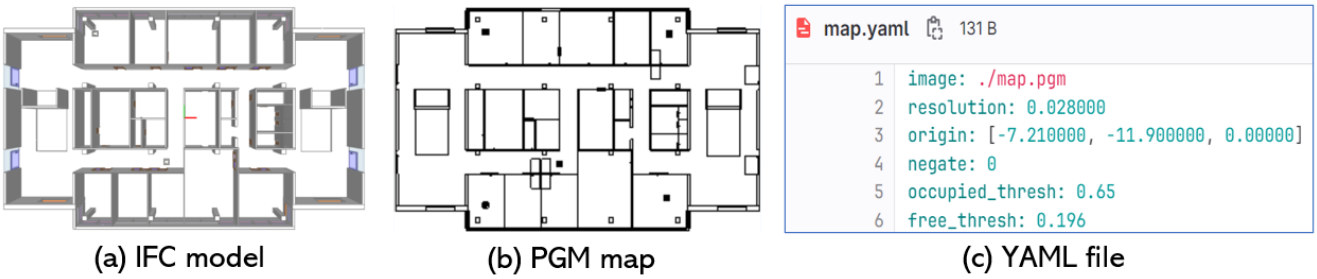


Figure 5-20: Creating PGM map and YAML file from the IFC model of Building L501 of Technical University of Darmstadt

In addition, a YAML file in conjunction with the PGM map is also required. The YAML file provides metadata about the PGM map, which is an image file representing a map layout where the intensity of each pixel corresponds to the occupancy of an area. As shown in Figure 5-20 (c), the first line indicates the path of the PGM map. The second line defines the resolution of the map, which is 0.028 meters per pixel. It specifies the physical distance each pixel in the PGM map represents. The third line defines the map's origin in relation to the map frame. If the coordinates are [0, 0, 0], the robot's initial position is at the bottom right corner of the map. Figure 5-21 illustrates the successful visualization of the PGM map derived from the IFC model within RViz. The origin is [0,0,0], so, the robot model is located in the bottom-right corner.

Figure 5-21: Visualization of the PGM map derived from the IFC model in RViz with the origin [0, 0, 0]

In YAML file, negate has value of 0 or 1 that determines the interpretation of the map's colors. If negate is 0, darker values (closer to 0) are considered obstacles, and lighter values (closer to 255) are considered free space. If negate is 1, this interpretation is reversed. Values in the map data greater than occupancy threshold are considered to represent occupied areas (obstacles). The free_thresh defines what is considered free space. Values in the map data less than this threshold are considered unoccupied or free. It is also a value between 0 and 1. The

combination of these parameters allows robotics software and systems to understand the spatial layout, dimensions, and areas of traversal versus obstacles in the PGM map. This setup is especially common in robotics navigation and simulation tasks.

Relative to the floor plan on the blueprint, the three-dimensional IFC model not only can export clean floor maps but also serve as a carrier for 3D models during the web-based visualization phase. The floor plan on the blueprint typically includes other annotations and icons that need to be manually removed; otherwise, robots would consider their contours as obstacles.

Creating outdoor maps for robots can be achieved by utilizing Google Earth or photos captured by drones. These maps can then be refined using free image editing software such as GIMP. Here's a brief overview of the process:

1. **Map Acquisition**: Obtaining a base map using Google Earth, which provides satellite imagery, or by flying a drone to capture aerial photographs of the area.
2. **Image Editing**: Using GIMP to process the images. This may involve cropping, resizing, deleting objects, enhancing colors, or adjusting contrast to make the map more readable for the robot's navigation system.
3. **Map Processing**: Converting the images into PGM format that can be interpreted by the robot.
4. **Testing**: Testing the robot's ability to navigate using the new map to ensure that it accurately reflects the real-world environment.
5. **Updates**: As the environment changes, the map may need to be updated. This can be done by repeating the process with new satellite or drone imagery.

The second step is very important for creating a detailed and accurate outdoor map that can be used by the robot for navigation and path planning. For example, the shadows of leaves and buildings in satellite imagery of Google Earth significantly impact the quality of the map (Figure 5-22 (a)). The editor GIMP can remove shadows to improve the quality of map derived from satellite imagery of Google Earth (Figure 5-22 (b)).

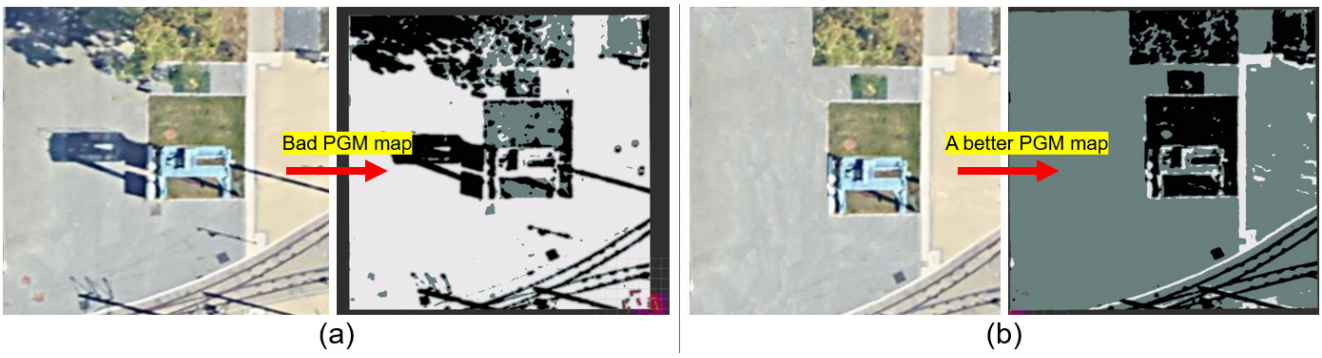


Figure 5-22: Improving the quality of map from satellite imagery of Google Earth using GIMP

If the aerial image captured by drone has too many distracting elements, such as reflections, annotations, grasslands, ditches, etc., manual delineation of drivable and non-drivable zones can be accomplished using GIMP. The non-drivable zones could be filled with black (Figure 5-23 (b)). In PGM map, black is used to indicate an obstacle. Subsequently, the drivable zones could be transformed into solid white using GIMP's threshold tool (Figure 5-23 (c)). The Threshold tool provides a visual graph, a histogram, of the intensity value of the active layer or selection. The Threshold filter can transform the current layer or the selection into a black and white image. Following these procedures, a black-and-white map is ultimately produced.

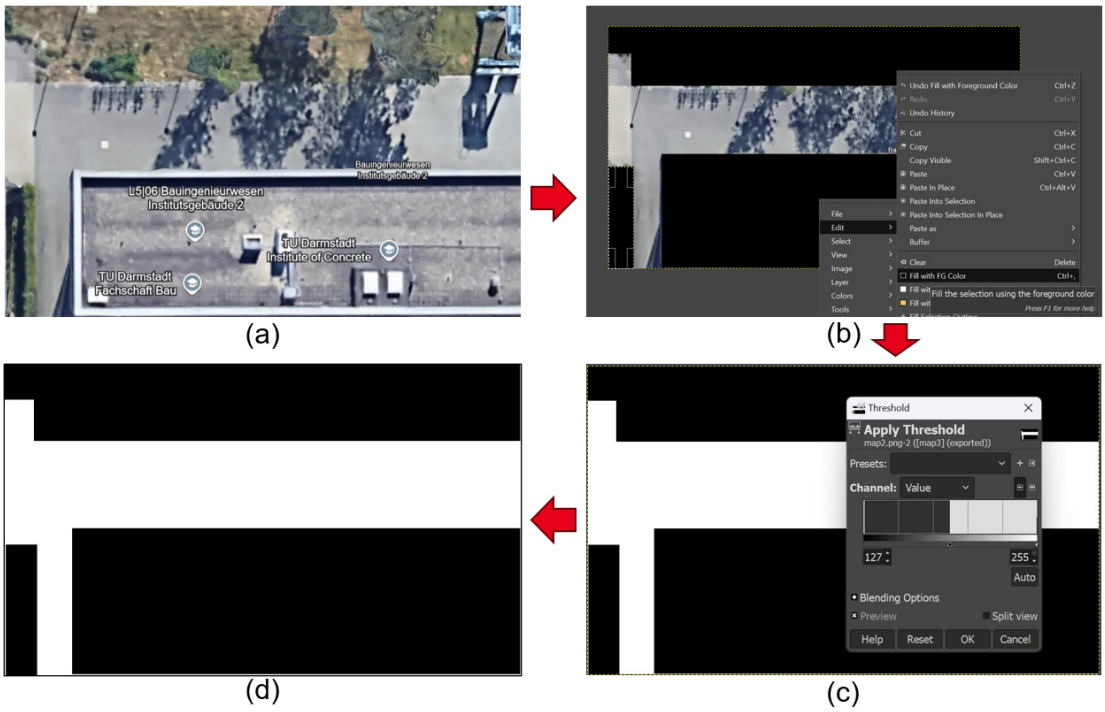
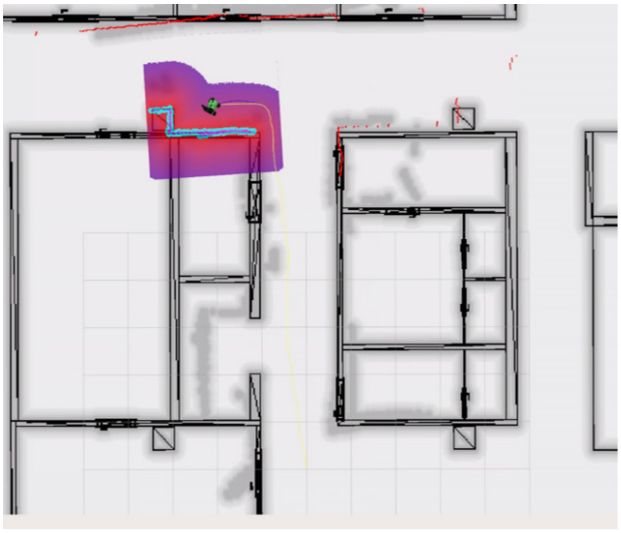


Figure 5-23: Steps to improve the quality of map captured by drone using GIMP

5.3. Path Planning using Python Script

In addition to manually assigning multiple destination points for the robot within RViz, multiple destination points can also be set via Python script using the *MoveBaseAction* and *MoveBaseGoal* messages from the *move_base_msgs* package in ROS. The *move_base_msgs* package contains the messages used to communicate with the *move_base* node. The *move_base* node is a major component of the ROS Navigation stack. It moves the robot from its current position to a goal position. *SimpleActionClient* is used to communicate with *move_base*. The goal point can be obtained from the map within RViz. The for loop is used to iterate over each goal point. This loop allows the script to send each goal sequentially to the *move_base* action server, enabling the robot to navigate to multiple destinations one after the other. The *move_base* node tries to achieve a desired pose by combining a global and a local motion planner to accomplish a navigation task which includes obstacle avoidance.



```python
if __name__ == '__main__':

    rospy.init_node('patrol')

    image_topic = "/jetauto_1/usb_cam/image_raw"

    client = actionlib.SimpleActionClient('/jetauto_1/move_base', MoveBaseAction)
    print("Waiting for move_base action server...")
    client.wait_for_server()

    target_poses = [
    [(2, 0, 0.0), (0.0, 0.0, 0, 1)] ,
    [(2, 0, 0.0), (0.0, 0.0, 0, 1)] ,
    [(8.3, 0, 0.0), (0.0, 0.0, 0, 1)] ,
    [(8.3, 1, 0.0), (0.0, 0.0, 0, 1)] ,
    [(8.3, 1, 0.0), (0.0, 0.0, 0, 1)] ,
    [(8.3, 2, 0.0), (0.0, 0.0, 0, 1)] ,
    [(8.2, 1, 0.0), (0.0, 0.0, 0, 1)] ,
    [(8, 2, 0.0), (0.0, 0.0, 0, 1)] ,
    [(0.000, 0.000, 0.0), (0.0, 0.0, 0, 1)]
    ]

    for target_pose in target_poses:
        goal = goal_pose(target_pose)
        client.send_goal(goal)
        client.wait_for_result()
        print("Patrol script has finished execution.")
```

Figure 5-24: Using Python script (right) to set multiple destination points for robot navigation (left)

The JetAuto Pro is equipped with LiDAR and has the ability to navigate and avoid obstacles in multiple points. It uses global and local path planning algorithms to navigate between points. Global planning algorithm can compute an optimal or feasible path from the robot's current position to a given goal position in map. With the help of LiDAR, the robot can promptly detect new obstacles while moving and, by utilizing local path planning algorithms, avoid obstacles to reach the designated location.

5.4. Crack Segmentation using YOLO Model

The YOLO model series has gained immense popularity in the field of computer vision due to its ability to provide fast and accurate object detection. It is a popular choice for implementing segmentation on robots due to its advanced features and capabilities. It offers high performance for object detection and segmentation, which is crucial for tasks that require understanding the shape and boundaries of objects within an image.

5.4.1. Training YOLO Model using Transfer Learning

In this work, the YOLO model is deployed in Jetson Orin Nano 8G. The concrete crack dataset employed for training is sourced from the publicly available datasets provided by Roboflow (Roboflow, 2023).

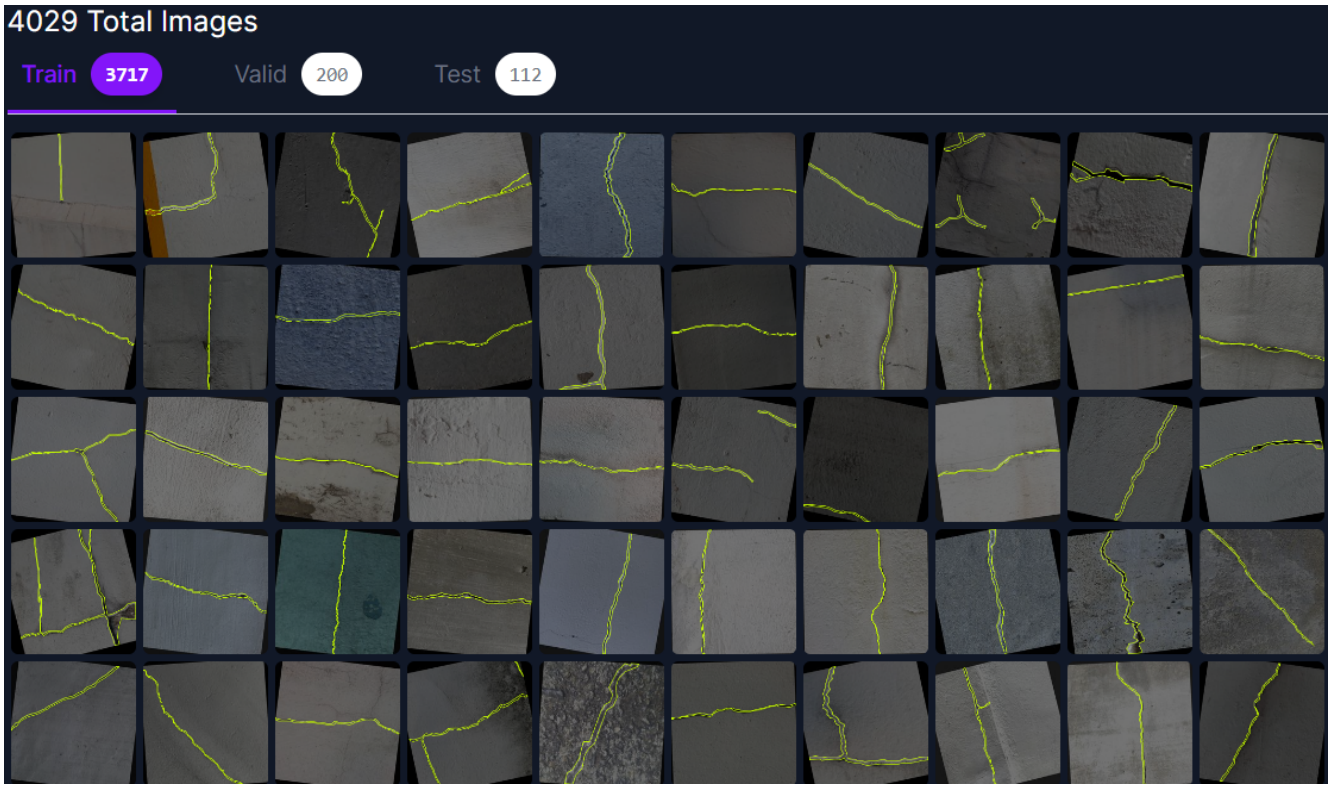


Figure 5-25: Overview crack image dataset from Roboflow

The dataset can be used for training both the YOLOv8 and YOLOv9 segmentation models. The image size for training model is 640, this means that the images will be resized to have their longest dimension set to 640 while maintaining the image's original aspect ratio. When evaluating the real-time inference speed of a model, it is crucial to ensure that the input size of images captured by the camera for visualization does not exceed 640 pixels. Exceeding this size can negatively impact the inference speed. Using 640 image size for training YOLO models is recommended in most cases because it provides a good balance between speed and accuracy and aligns with

the size of pre-trained models. The model could be trained on the Kaggle platform, which offers free GPUs for training purposes. Model training on Kaggle employed a GPU T4x2, running for 100 epochs with a learning rate of 0.01 and batch of 16.

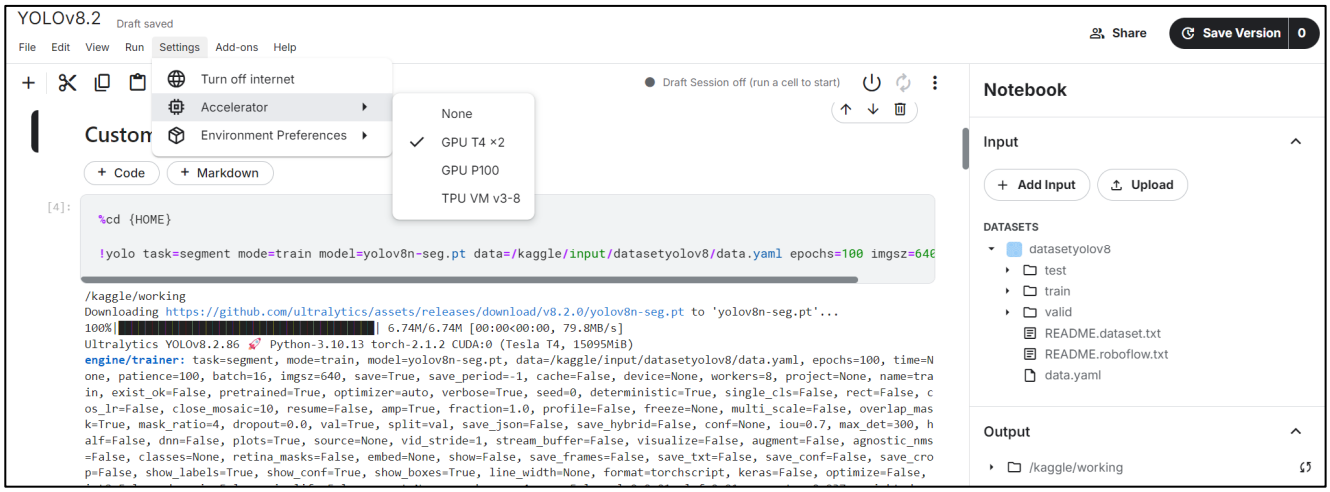


Figure 5-26: The Kaggle interface used for training model

During the training process, once the model achieves its optimal performance, the corresponding weights will be saved. YOLO employs a composite loss function that includes box_loss, cls_loss, seg_loss, and dfl_loss, which work together to efficiently train the model for precise object detection, classification, and segmentation. The individual components are vital for refining particular facets of the model's performance, and their collective use ensures comprehensive learning and optimal task execution within a single framework. Figure 5-27 illustrates that with the increase in training iterations, the loss functions by training and validation consistently decrease.

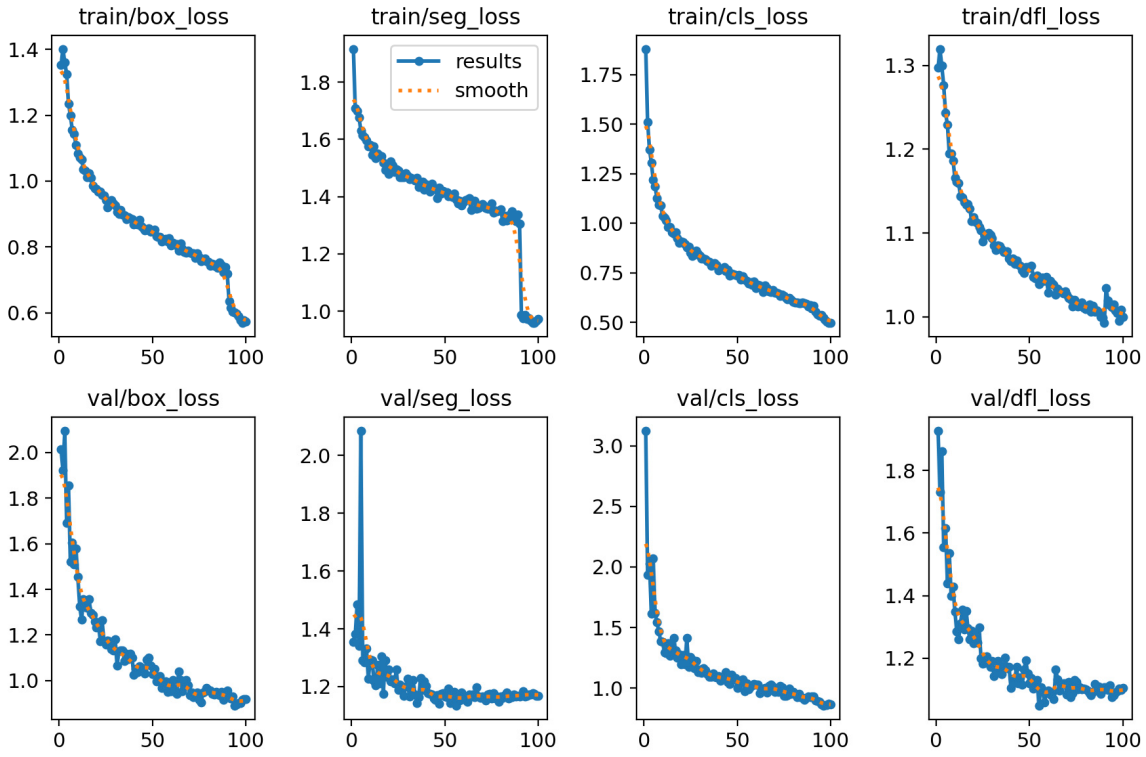


Figure 5-27: The changes of the loss functions by training and validation process of YOLOv8

To evaluate YOLOv8 model performance, important metrics include Precision, Recall, and mAP. As shown in Figure 5-28, as the number of training iterations increased, the Precision, Recall, and mAP also improved. However, after 50 epochs, the rate of improvement in these metrics gradually leveled off.

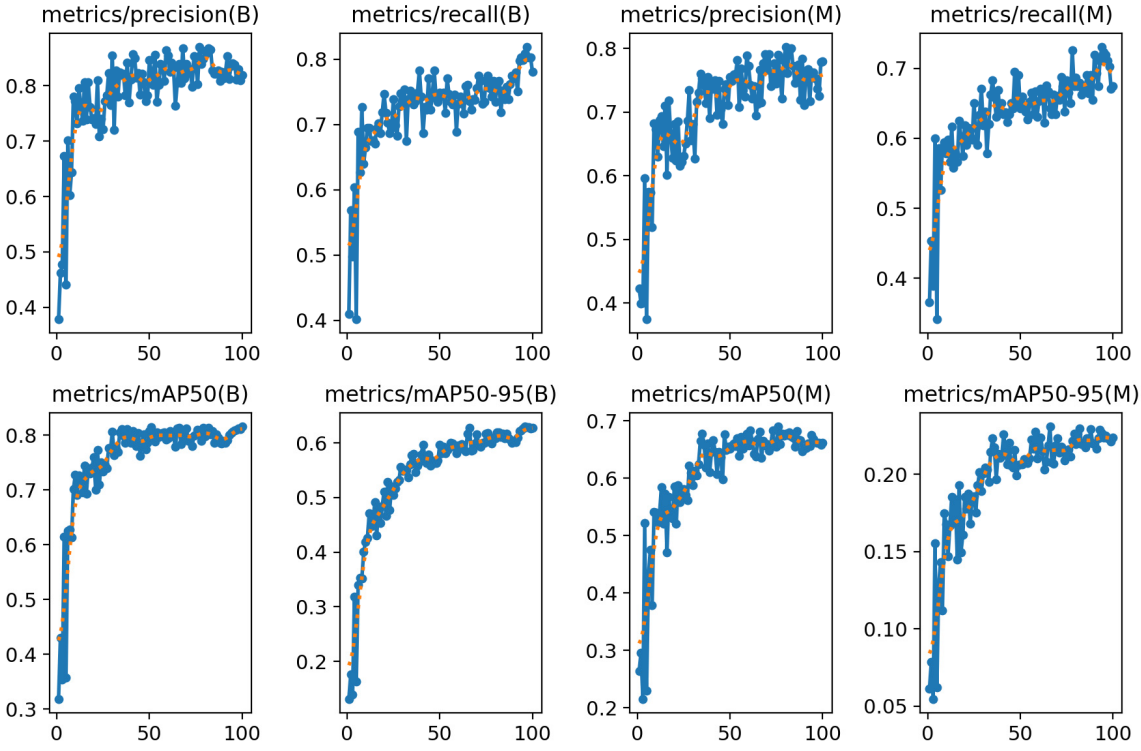


Figure 5-28: The changes of the Precision, Recall and mAP during the training process of YOLOv8

The figure below illustrates crack images with label and the segmentation results obtained using the best-trained model. The comparison demonstrates that the model performs well in segmenting cracks.

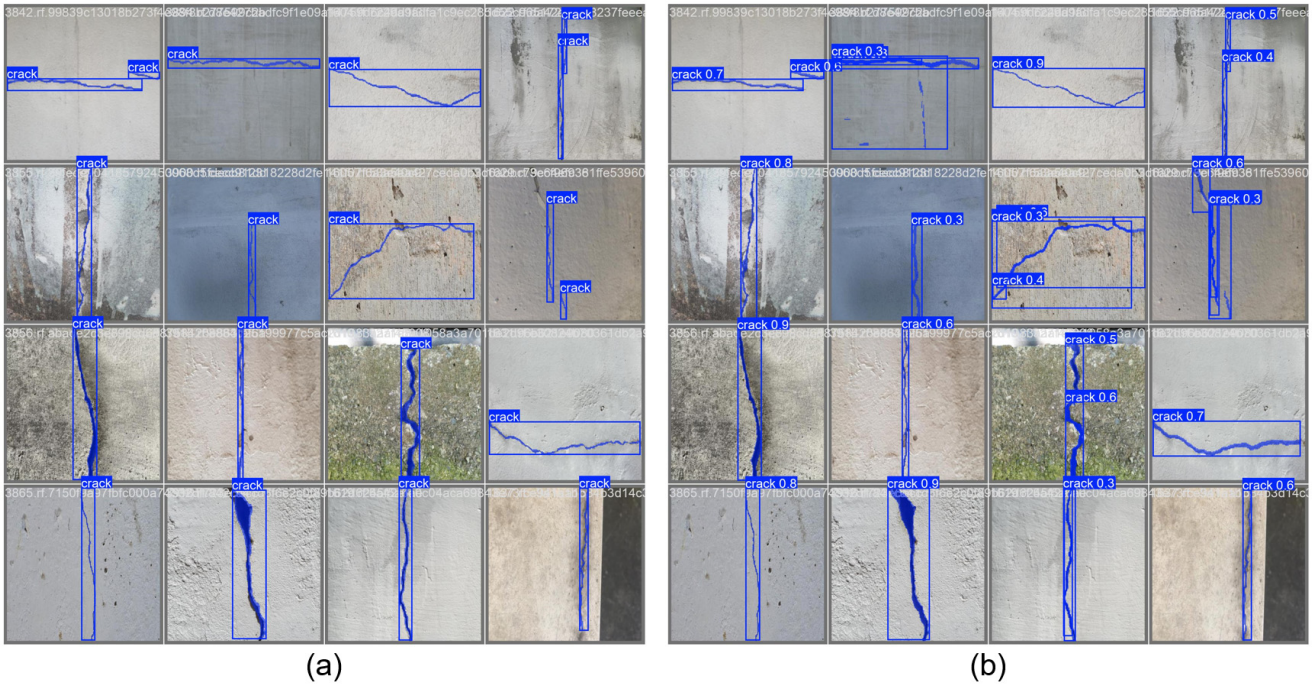


Figure 5-29: Comparison of (a) crack images with label and (b) inference results by best-trained model

The metrics mAP50(M) and mAP50-95(M) are critical in object segmentation scenarios. The following are the validation code and output results about the performance of the best-trained model on Kaggle. The validation dataset contains 200 images. From the output results, it can be seen that mAP50 (M) is 0.679 and mAP50-95 (M) is 0.229.

Figure 5-30: Validating the best YOLOv8n-seg model on the new dataset

The work also trained models such as YOLOv8s-seg, YOLOv8x-seg, and YOLOv9c-seg models using a similar training approach in Kaggle. For example, the best-trained YOLOv8s-seg model's mAP50 (M) and mAP50-95 (M) are higher than those of the best trained YOLOv8n-seg model, which are 0.689 and 0.246, respectively. It can also be seen from the results in the figure below that the best-trained YOLOv8s-seg model is slower in inference speed compared to the best-trained YOLOv8n-seg model.

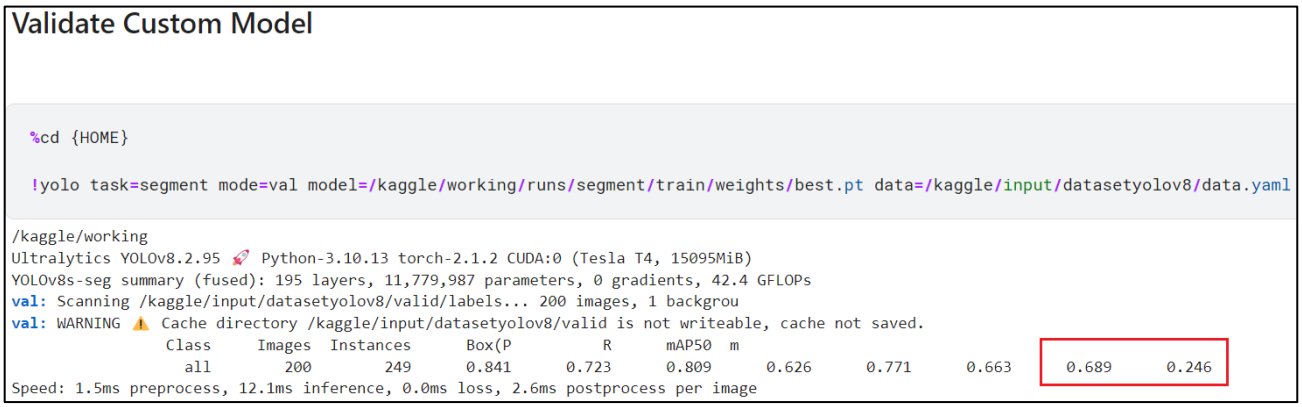


Figure 5-31: Validating the best YOLOv8s-seg model on the new dataset

5.4.2. Deploying the Trained Model in Jetson Orin Nano

Multiple versions of the YOLO model were trained in this work, and deciding which model to deploy requires first comparing the inference speeds, as this is crucial for real-time crack segmentation. To convert a PyTorch weight to a TensorRT engine model, the process involves first exporting the model to ONNX format using the ultralytics API, followed by converting the ONNX model to a TensorRT engine model. This process was guided by an open-source tutorial available on GitHub (Lin, 2024). According to this tutorial, there are two methods to export the TensorRT engine from ONNX model. One is based on TensorRT Python API, the other is based on Trtexec, which are developed by Nividia (NVIDIA, 2019). The original tutorial only provides the Python script *infer-seg.py* for segmenting batches of images. To enable real-time crack segmentation using a USB camera, the script must be modified to include OpenCV's *cv2.VideoCapture* function to open a camera for video streaming. When testing the performance of the best-trained models, the input size of the video frame is set to 640x480, which is also the size of the images used when training the model.

The best-trained YOLOv8n-seg model has a size of 6.8 MB. When this model is converted to a TensorRT engine model, its size increases to 10 MB. The Figure 5-32 illustrates the inference results of best-trained YOLOv8n-seg model with and without TensorRT acceleration. The results indicate that the TensorRT engine exported by Trtexec achieve faster inference speed than the TensorRT engine exported by TensorRT Python API and PyTorch model. However, the acceleration effect of TensorRT is not significant.

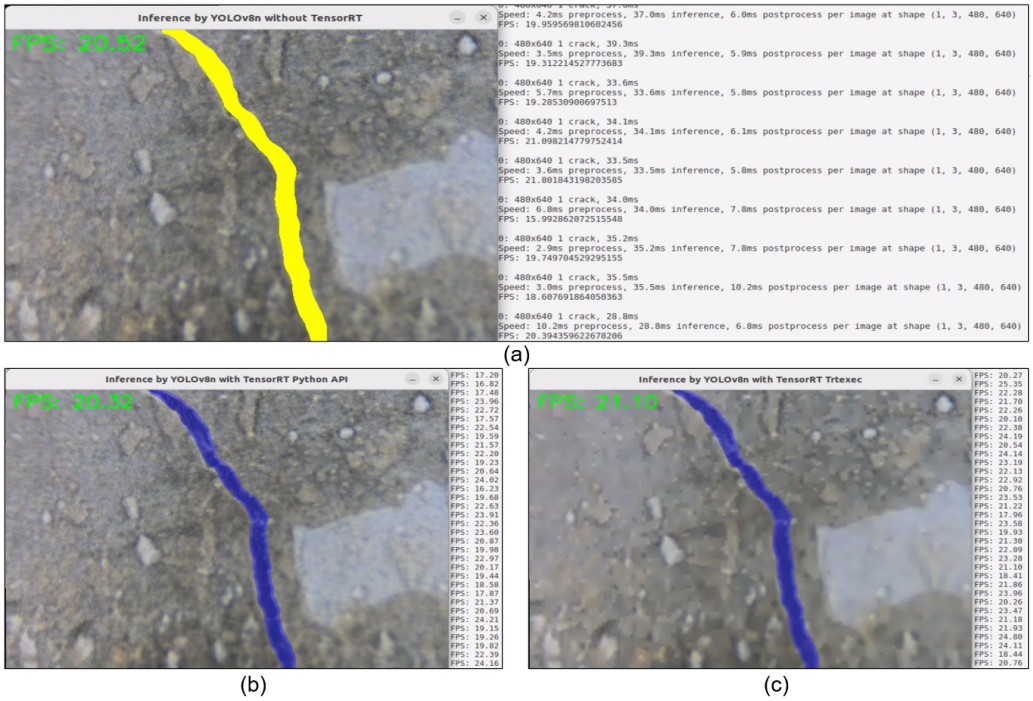


Figure 5-32: Comparison of inference speed with segmentation result and terminal output: (a) without acceleration, (b) acceleration via TensorRT Python API and (c) acceleration via Trtexec

The best-trained YOLOv8s-seg model obtained has a size of 22.7 MB, while the size of corresponding TensorRT model is 26.6 MB. It was found through testing that the inference speed only increased slightly with TensorRT.

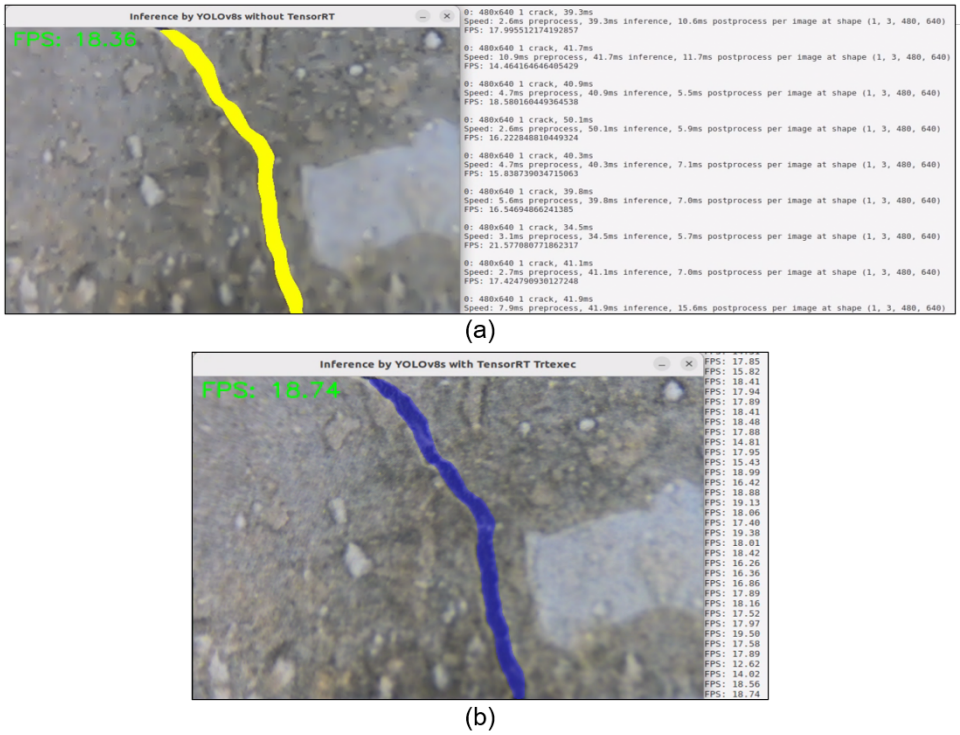
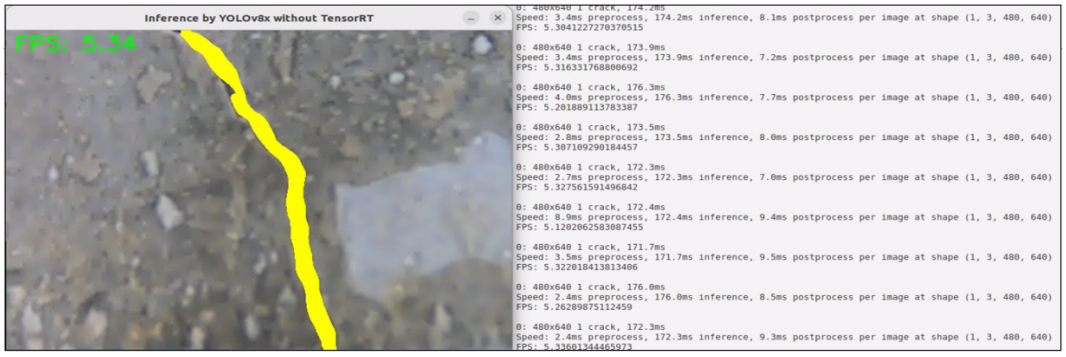
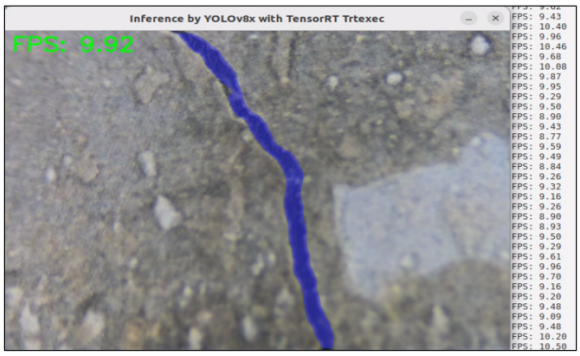


Figure 5-33: Inference speed of the YOLOv8s-seg model (a) without TensorRT acceleration and (b) with TensorRT acceleration

YOLOv8x-seg is the model with the most parameters among the YOLOv8 pre-trained image segmentation models. It can be seen from the Figure 5-34 that the YOLOv8x-seg model is too large, resulting in a maximum inference speed of only about 9.92 FPS. This inference speed is insufficient for practical engineering applications.



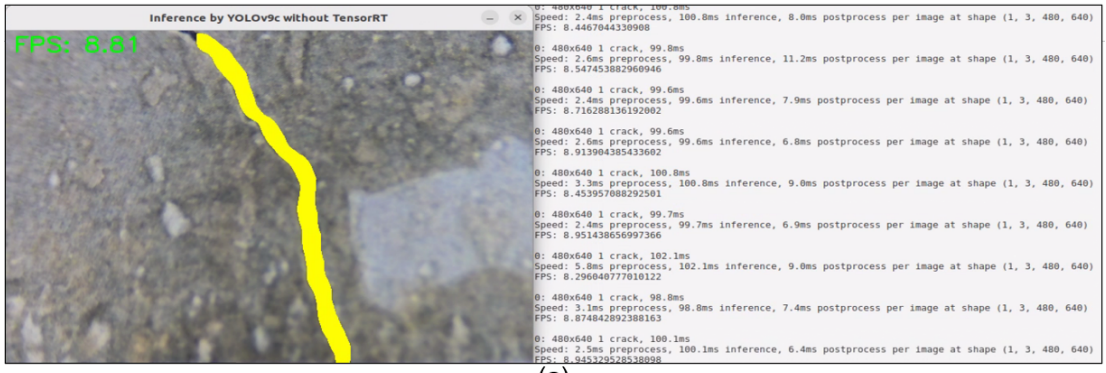
(a)



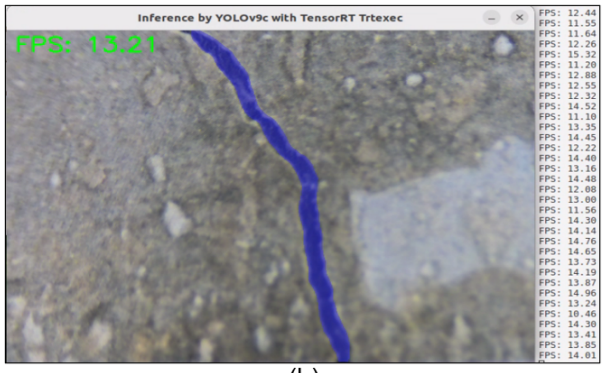
(b)

Figure 5-34: Inference speed of the YOLOv8x-seg model (a) without acceleration and (b) with TensorRT acceleration

The size of corresponding TensorRT model of YOLOv9c-seg is 59.6 MB. As illustrated in the figure below, the YOLOv9c-seg model with TensorRT acceleration can reach a maximum inference speed of about 13 FPS.



(a)



(b)

Figure 5-35: Inference speed of the YOLOv9c-seg model (a) without acceleration and (b) with TensorRT acceleration

When running YOLOv9c-seg with TensorRT acceleration on Jetson Orin Nano, the system on Jetson Orin Nano will pop up a reminder box, notifying that the system is being throttled due to over-current.

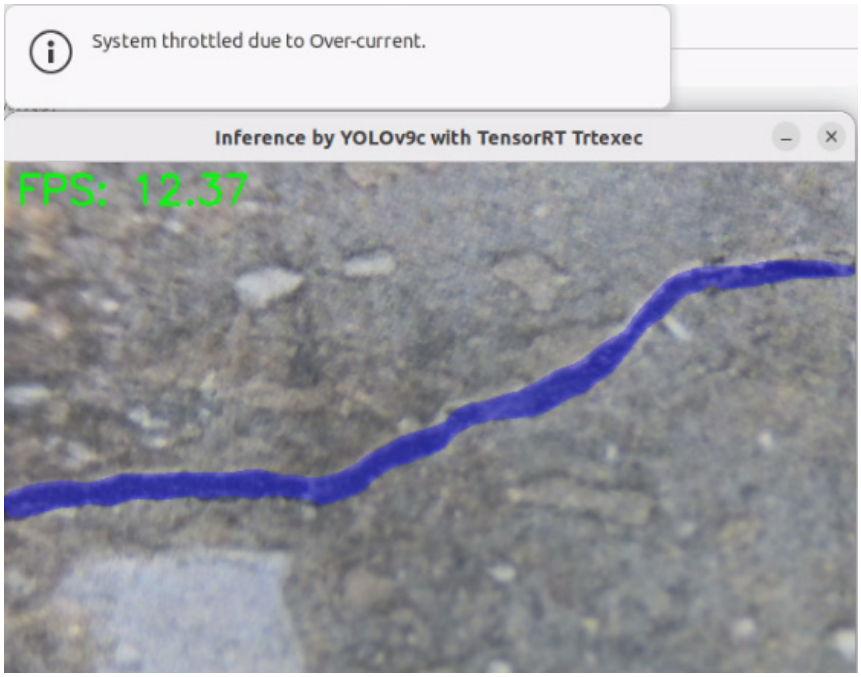


Figure 5-36: Issue about over-current by YOLOv9c-seg model with TensorRT acceleration in Jetson Orin Nano

Through the comparison of inference speeds, the trained YOLOv8n-seg and YOLOv8s-seg models can basically meet the needs of real-time inference, while YOLOv8x-seg has obvious delay and YOLOv9c-seg has problem about over-current. But according to the metrics mAP50(M) and mAP50-95(M), the performance of the best-trained YOLOv8s-seg is better as the best-trained YOLOv8n-seg models. Therefore, this work ultimately selected the best-trained YOLOv8s-seg model for real-time crack segmentation. Due to the lack of significant speed improvement for the YOLOv8n-seg and YOLOv8s-seg models with TensorRT, it is optional to use TensorRT acceleration when deploying. And for crack inference, it is essential to select a confidence threshold that ensures both high precision and high recall. This can be determined using the F1 score confidence curve, which helps in ensuring that the model not only accurately detects the presence of crack but also precisely segments its boundaries. As shown below, the confidence threshold of the best-trained YOLOv8s-seg model is 0.443.

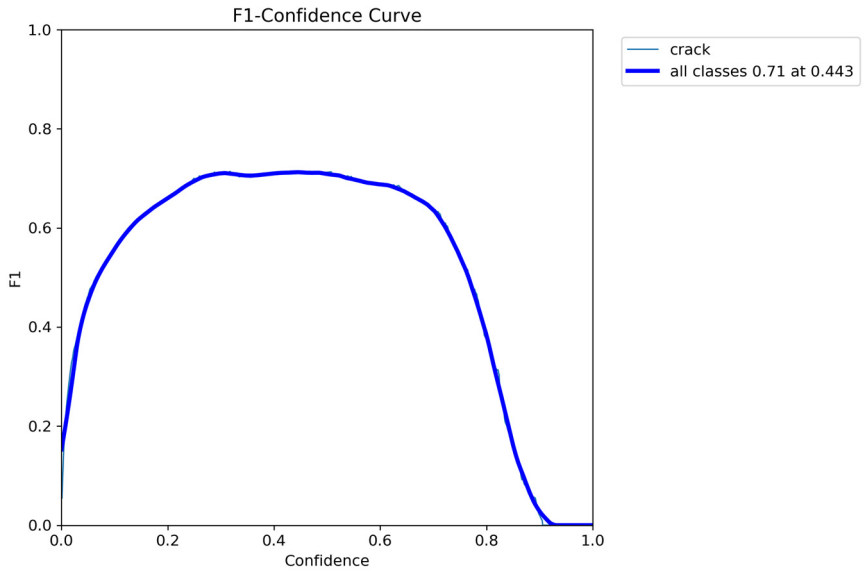


Figure 5-37: F1-Confidence Curve of the best-trained YOLOv8s-seg model

5.5. Determining Crack Width

The calculation of the maximum crack width is based on the mask generated by crack segmentation using YOLOv8. The size of the crack width is the key to trigger the spraying and recording system.

5.5.1. Crack Segmentation Extraction

The YOLOv8 model utilized in this work was developed and released by Ultralytics. Figure 5-38 presents the crack segmentation result utilizing YOLOv8 in the Jetson Orin Nano. Each segmented crack is highlighted with a red label, bounding box, and mask. The maximum width of each crack can be directly determined by measuring the broadest point of the mask. To streamline the process of mask extraction, it is necessary to adjust the YOLOv8's output settings to exclusively retain the masks, while removing all labels and bounding boxes.

Figure 5-38: Crack segmentation result based on YOLOv8n in Jetson Orin Nano

The Figure 5-39 (left) illustrates the segmentation result after modifying the output configuration. Although labels and bounding boxes have been removed, the color of the mask continues to present issues. The mask's transparency is set to semi-transparency by default to facilitate a more effective comparison with the segmented objects in the image. When the regions of the mask overlap with the gray areas of the crack, this interaction results in a spectrum of different colors. The *Pixel Color Counter* tool is a web app that counts the number of pixels in an image per a unique color (Grenon, 2024). Through the tool *Pixel Color Counter*, it can be observed that there are a large number of yellow-tinted pixels in the segmentation result. This makes it difficult for OpenCV tool to accurately capture the mask of the crack based on the specific value of color.

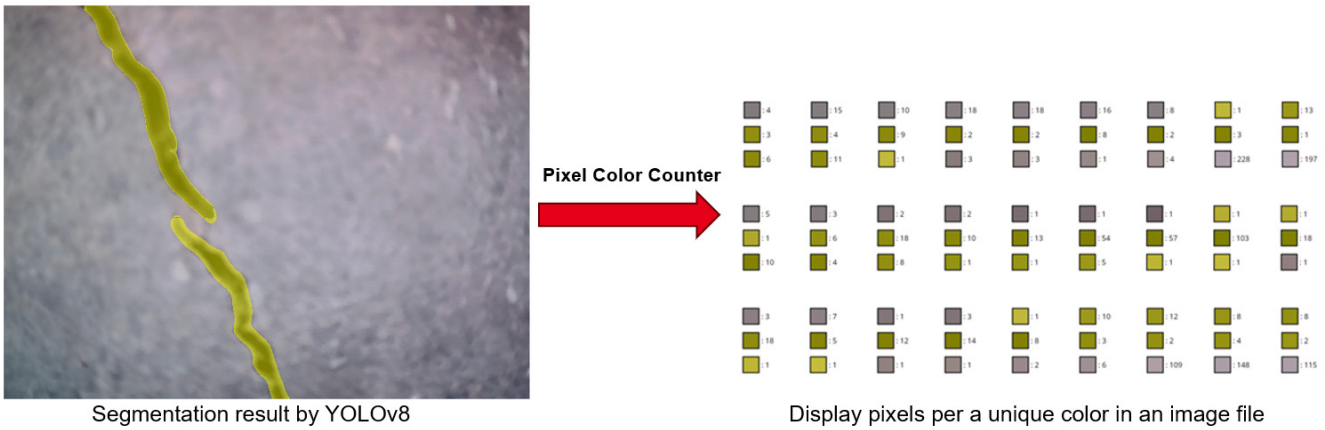


Figure 5-39: Color distribution analysis of the segmentation output using Pixel Color Counter tool

To accurately extract the segmented mask using OpenCV, the specific code in YOLOv8 that controls the mask's transparency need to be adjusted, switching from semi-transparent to opaque. The relevant code is within *plotting.py* file of YOLOv8. The segmentation result after modifying is shown below. The result features a uniform shade of yellow pixels, without variations into darker or lighter yellows.

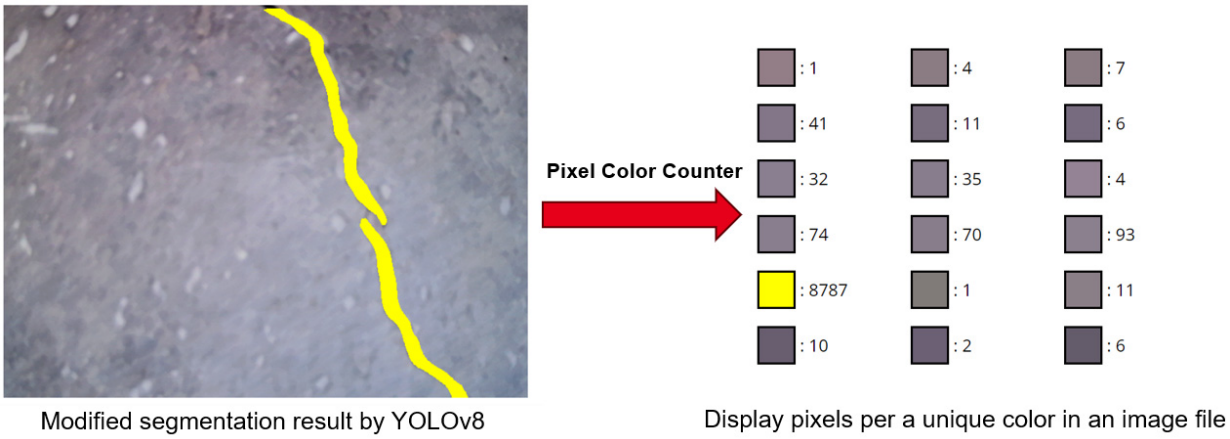


Figure 5-40: Color distribution analysis of the modified segmentation output using Pixel Color Counter tool

5.5.2. Real-Time Calculation of the Maximum Width

Distance transformation of OpenCV is a technique used to calculate the distance of each pixel in an image from the nearest non-zero pixel. It can be used to calculate the largest inscribed circle within the mask of crack. Figure 5-41 illustrates the results of calculating crack maximum width using the proposed technique on the Jupyter Notebook environment in the Jetson platform. The execution time, indicated as 0.0 seconds in the lower-left corner of the interface, demonstrates that the algorithm is capable of determining the maximum width of a crack instantaneously. Such rapid inference capabilities are adequate to fulfill the demands for real-time deployment.

```python
import cv2
import numpy as np

# Load the original image
image_path = 'testimage.png'
image = cv2.imread(image_path)

# Convert image to RGB (OpenCV uses BGR by default)
image_rgb = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)

# Define a range for the color that marks the crack
# The values here are set to detect red but should be adjusted based on the actual image
lower_color = np.array([255, 255, 0])
upper_color = np.array([255, 255, 0])

# Create a mask that captures the areas of the crack
mask = cv2.inRange(image_rgb, lower_color, upper_color)

# Perform a distance transform to find the largest inscribed circle within the crack
# This operation assigns to each pixel a value that represents its distance to the nearest edge
distance_transform = cv2.distanceTransform(mask, cv2.DIST_L2, 5)

# Find the maximum value in the distance transform, which corresponds to the radius of the largest circle
min_val, max_val, min_loc, max_loc = cv2.minMaxLoc(distance_transform)

# The diameter of the largest circle gives the maximum width of the crack
max_diameter = max_val * 2  # Diameter is twice the radius
# Print out the maximum diameter and the location of the circle center
print("Maximum diameter in pixels:", max_diameter)
print("Location of the circle's center (x, y):", max_loc)
```

```
[12] ✓ 0.0s

Maximum diameter in pixels: 20.375198364257812
Location of the circle's center (x, y): (472, 364)
```

Figure 5-41: Code for calculating the maximum width of crack

To attain this level of rapid inference, it is essential to extract the mask of the crack. This step significantly reduces the computational complexity of the algorithm by extracting the crack mask, which enables the algorithm to focus solely on computing the inscribed circle within the mask region. The *cv2.inRange(src, lowerb, upperb)* is the most commonly used function to detect colors in an image. Here, *src* is the input image. *lowerb* and *upperb* denotes the lower and upper boundary of the threshold region. The RGB value of color yellow is (255, 255, 0). Consequently, both the upper and lower boundaries of color are consistently set at (255, 255, 0). Finally, it returns a thresholded image.

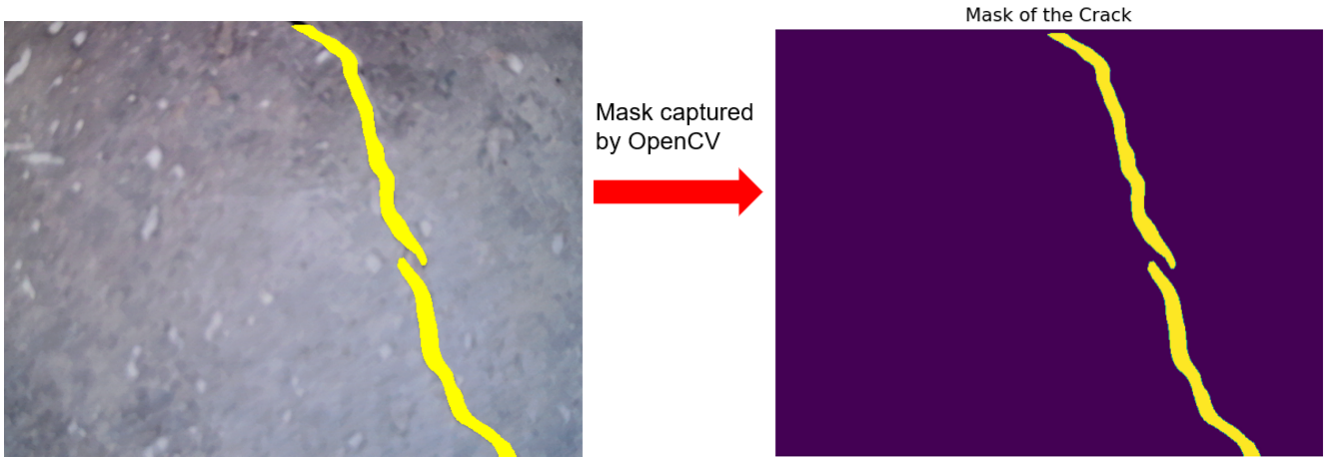


Figure 5-42: Capturing Mask of crack segmentation result through OpenCV

As shown in Figure 5-43 (a), to validate the reliability of the algorithm, a function to draw the largest inscribed circle in green color on the original image was added to the code, and the diameter and coordinates of the largest inscribed circle were outputted. To confirm the accuracy of this computation, the online open-source measurement tool, *Augenmaß*, was employed to manually measure the crack's width at various points. Given a background photo, this tool allows user to draw lines on top of it whose relative lengths in pixel are shown. As shown in Figure 5-43 (b), the manually measured diameter at the position of the largest inscribed circle is indeed the largest, and it closely matches the diameter calculated by the algorithm, with only a negligible difference. This demonstrates the reliability of the employed algorithm.

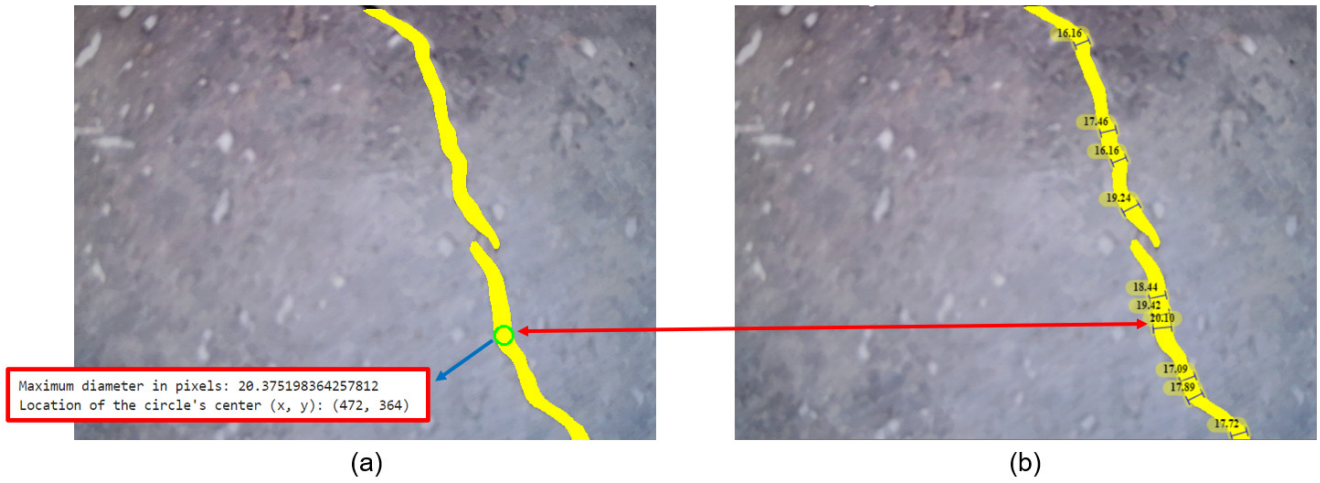


Figure 5-43: Crack width analysis: (a) localization of maximum crack width indicated by inscribed circle and (b) results of manual measurements using Augenmaß tool

The code provided below is essential for calculating the maximum width of cracks in frame by real-time inference using the YOLOv8 segmentation model. *annotated_frame* is the result of segmentation. It contains the real-time image with mask of cracks.

```
# Read a frame from the live camera
success, frame = cap.read()

if success:
    # Run YOLOv8 inference on the frame
    results = model(frame)

    # Visualize the results on the frame without label and boxes
    annotated_frame = results[0].plot(boxes=False)

    # Convert image to RGB (OpenCV uses BGR by default)
    image_rgb = cv2.cvtColor(annotated_frame, cv2.COLOR_BGR2RGB)
```

Figure 5-44: Calculating the maximum width of crack in frame by real-time inference

5.5.3. Using Suitable Camera

The robot employs a USB camera with a 10x zoom capability for crack inspection. This choice is made to overcome the limitations of fixed-focus cameras such as the Intel Realsense 3D Camera and Logitech C270, which have two notable drawbacks: the need for height adjustment and insufficient image clarity.

The Figure 5-45 shows the image segmentation result when the Intel RealSense 3D Camera is 14 cm above the ground. The image is relatively clear, and it can be determined through *Augenmaß* tool that 1 cm in reality corresponds to 44 pixels in the image. In this case, 0.6 mm is equivalent to 2.64 pixels. If the maximum width of the crack exceeds 2.64 pixels, it can be defined as a wide crack.

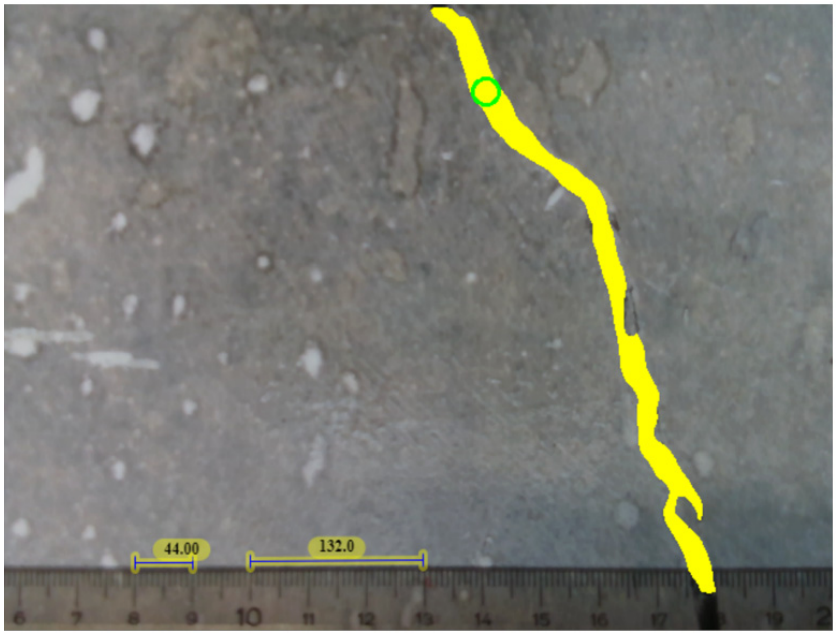


Figure 5-45: Crack segmentation result with fixed-focus camera positioned 14 cm above the ground

However, by testing a nearly 1 mm wide crack with the same height difference (14 cm) between the camera and the ground, it was observed that the segmented mask excessively exceeded the actual size of the cracks. As shown

in Figure 5-46, 1 mm is equivalent to 4.4 pixels. However, by using the *Augenmaß* tool and the inscribed circle-based algorithm to calculate crack width, it is found that a 1 mm wide crack corresponds to a pixel width of approximately 8 pixels. In other words, at a camera height of 14 cm, there is a significant error in the segmentation result for a narrow crack with a width of 1 mm, due to the crack being too small, resulting in the algorithm generating an overly large mask.

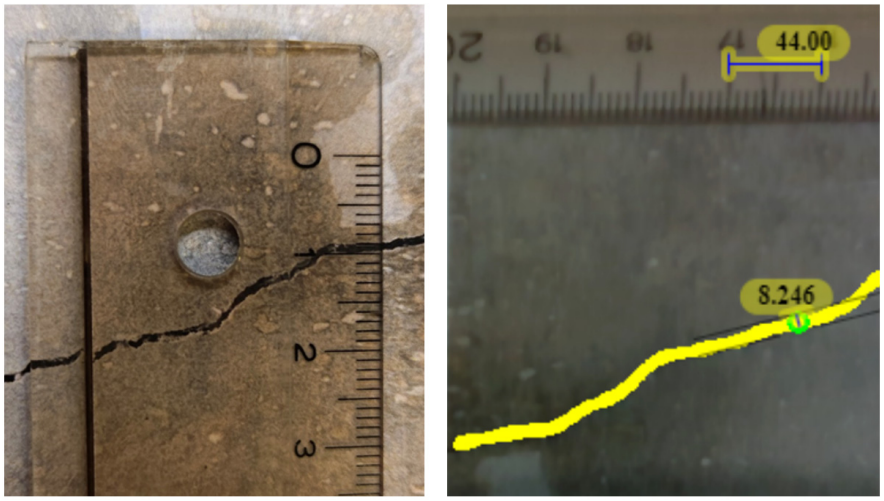


Figure 5-46: Segmentation result of a nearly 1 mm wide crack with camera positioned 14 cm above the ground

When the camera height is lowered to 4 cm above the ground, as demonstrated in the Figure 5-47, 1 mm equates to 15.3 pixels. Using the *Augenmaß* tool, the crack width was found to be approximately 15.62 pixels. Therefore, it can be concluded that, at this camera height, the segmentation of the crack is relatively accurate. After adjusting the mask's opacity to 0.1, it can be seen that the mask is still a bit wider than the actual crack. In this case, a correction factor such as 0.98 can be employed to closely approximate the true width of the crack. Additionally, it was observed that the images captured by the camera lacked sufficient clarity, which will negatively impact the performance of crack segmentation.

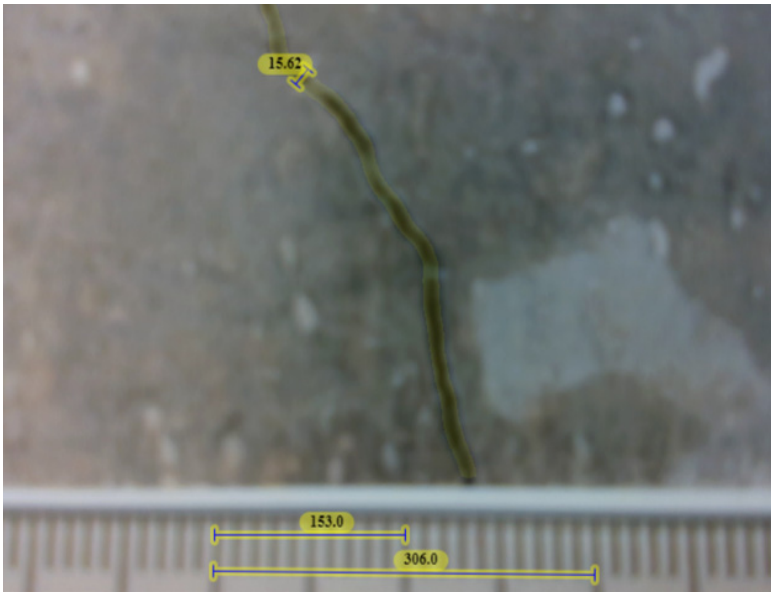


Figure 5-47: Segmentation result of a nearly 1 mm wide crack after adjusting the camera height to 4 cm above the ground

To address the shortcomings of the fixed-focus camera mentioned above, the robot ultimately used a 10x zoom camera for crack segmentation. It can magnify the crack, thereby enhancing the crack segmentation results. The

Figure 5-48 displays the segmentation result of a nearly 1 mm wide crack using a 10x zoom camera. The results indicate that the segmented mask is still slightly larger than the actual crack. A correction factor is also required by calculating the real width of crack. Under the 10x zoom camera, the magnified crack is very clear, and 1 mm equates to 16.8 pixels.

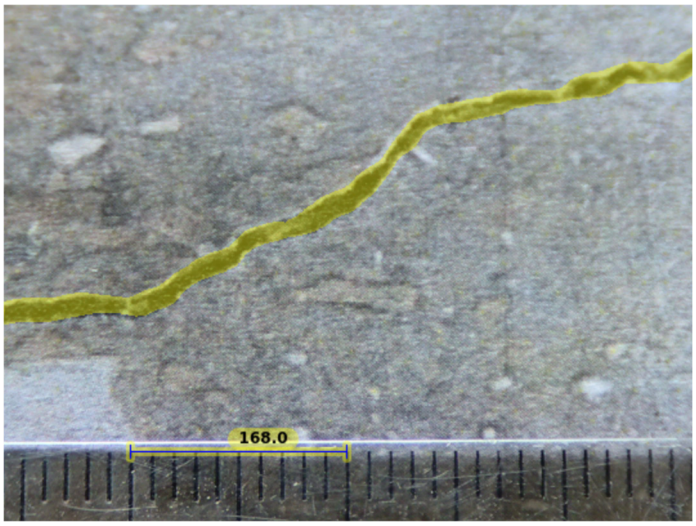
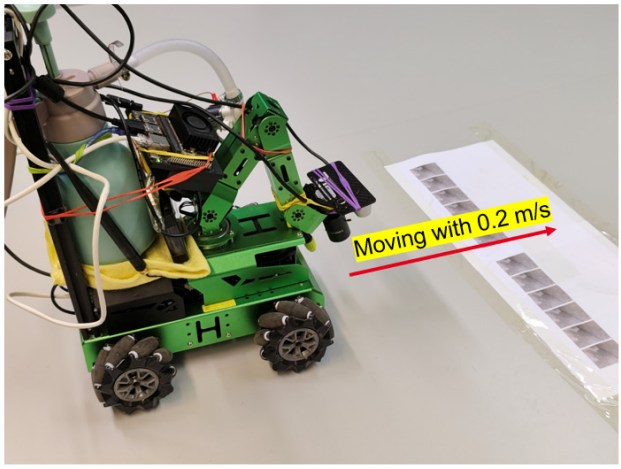


Figure 5-48: Segmentation result of a nearly 1 mm wide crack using a 10x zoom camera

All the segmentation results mentioned are based on a stationary state. It is necessary to test whether the robot can accurately segment narrow cracks using a camera with a frame rate of 30 FPS and YOLOv8s model with an inference speed of 18 FPS at a moving speed of 0.2 m/s. Figure 5-49 (a) shows a simple test scenario, where a 10x zoom camera with a frame rate of 30 FPS was used to magnify the image of a crack that is 0.6 mm wide and the robot would move forward at a speed of 0.2 m/s. In a stationary state, the pixel width corresponding to the 0.6 mm is approximately 19 pixels. Figure 5-49 (b) shows the real-time crack segmentation results output by the Jetson Orin Nano on the terminal. According to the terminal output, 2 cracks with pixel widths of 19.575 and 22.787 pixels were detected.



(a)

```
0: 480x640 (no detections), 39.8ms
Speed: 2.6ms preprocess, 39.8ms inference, 2.2ms postprocess per image at shape (1, 3, 480, 640)
Maximum diameter in pixels: 0.0
Location of the circle's center (x, y): (0, 0)

0: 480x640 (no detections), 39.4ms
Speed: 6.8ms preprocess, 39.4ms inference, 1.7ms postprocess per image at shape (1, 3, 480, 640)
Maximum diameter in pixels: 0.0
Location of the circle's center (x, y): (0, 0)

0: 480x640 1 crack, 34.5ms
Speed: 2.7ms preprocess, 34.5ms inference, 756.5ms postprocess per image at shape (1, 3, 480, 640)
Maximum diameter in pixels: 19.5751953125
Location of the circle's center (x, y): (259, 28)

0: 480x640 1 crack, 55.7ms
Speed: 2.9ms preprocess, 55.7ms inference, 6.0ms postprocess per image at shape (1, 3, 480, 640)
Maximum diameter in pixels: 22.78753662109375
Location of the circle's center (x, y): (408, 38)

0: 480x640 (no detections), 55.6ms
Speed: 2.3ms preprocess, 55.6ms inference, 2.0ms postprocess per image at shape (1, 3, 480, 640)
Maximum diameter in pixels: 0.0
Location of the circle's center (x, y): (0, 0)

0: 480x640 (no detections), 54.2ms
Speed: 11.0ms preprocess, 54.2ms inference, 3.1ms postprocess per image at shape (1, 3, 480, 640)
Maximum diameter in pixels: 0.0
Location of the circle's center (x, y): (0, 0)

0: 480x640 (no detections), 39.7ms
Speed: 13.7ms preprocess, 39.7ms inference, 2.3ms postprocess per image at shape (1, 3, 480, 640)
Maximum diameter in pixels: 0.0
Location of the circle's center (x, y): (0, 0)

0: 480x640 (no detections), 45.8ms
Speed: 2.7ms preprocess, 45.8ms inference, 2.0ms postprocess per image at shape (1, 3, 480, 640)
Maximum diameter in pixels: 0.0
Location of the circle's center (x, y): (0, 0)
```

Detect cracks

(b)

Figure 5-49: Testing real-time crack segmentation with moving robot

The results of the test indicate that when the robot passes over the crack image, the real-time image captured by the camera is sufficiently clear to enable YOLOv8s to accurately segment the crack. Subsequently, the algorithm for calculating the crack width can immediately determine the pixel width of the crack and output it to the terminal.

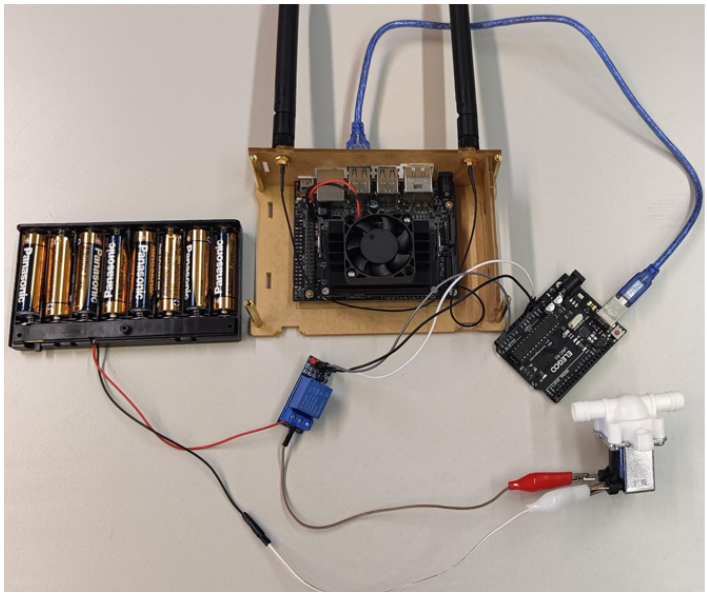
This simple test only preliminarily verifies that the camera with a 30 FPS frame rate, the inference speed of the YOLOv8s model by Jetson Orin Nano, and the algorithm for calculating crack width can meet the real-time segmentation of cracks at a moving speed of 0.2 m/s. The overall performance of the robot will be further tested in multiple scenarios in Chapter 6.

5.6. Control of the Spraying System

When the robot correctly detects a narrow crack, it will activate the spraying system through Arduino.

5.6.1. Controlling the Water Valve via Arduino

The Jetson Orin Nano can control a solenoid water valve through both direct and indirect methods. Direct control can be achieved using the GPIO (General-purpose input/output) pins available on the module, which can be accessed through the Jetson GPIO library or by interfacing with the appropriate kernel drivers. Indirect control via Arduino can be useful for applications that require additional flexibility or when using the Arduino for other purposes in the system. The final solution adopted in this work is to use an Arduino to control the solenoid water valve because there are plans to connect more sensors to the Arduino to further expand the robot's other functionalities. The Jetson devices can be directly connected to an Arduino via a USB cable. The communication between them is achieved through the serial port */dev/ttyAMA0* and *pyserial*, which is a Python library that enables serial communication between computers and various devices. The Figure 5-50 illustrates the connection circuit and relevant code for controlling the valve using Arduino and relay. Pin A5 is connected to the signal pin of the relay. The initial voltage of pin A5 is set to HIGH level. This configuration ensures that when the Arduino is powered on, both the signal pin and the VCC pin of the relay maintain a HIGH voltage, eliminating any potential difference and thus preventing the relay from being activated. When the Jetson Orin Nano detects a narrow crack, it sends command 1 to the Arduino via *pyserial*, switching the voltage at pin A5 to LOW level, thereby activating the relay. Conversely, when the Jetson Orin Nano does not detect a narrow crack, it sends command 0 to the Arduino via *pyserial*, switching the voltage at pin A5 to HIGH level, thereby deactivating the relay.



```
void setup() {
  Serial.begin(9600);
  while (!Serial){
    ;
  }
  // initialize digital pin A5 as an output.
  pinMode(A5, OUTPUT);
  // initialize digital pin A5 as HIGH.
  digitalWrite(A5, HIGH);
}

void loop() {
  char buffer[3];
  // if  get a command, turn the Pumpe on or off:
  if (Serial.available() > 0) {
    char command = Serial.read();
    if (command == '1') {
      digitalWrite(A5, LOW);
    } else if (command == '0') {
      digitalWrite(A5, HIGH);
    }
  }
}
```

Figure 5-50: Using Arduino to control the water valve

After establishing a connection between Jetson Orin Nano and Jetson Nano via Ethernet, and connection between Jetson Orin Nano and Arduino via USB cable, Jetson Orin Nano can issue commands to both Arduino and Jetson Nano simultaneously through a simple Python script based on the calculated size of the cracks.

The Python code is shown in Figure 5-51, if no cracks are present on the concrete surface, the maximum crack width value is 0. The Jetson Orin Nano then sends command 0 to the Jetson Nano. This command sets the voltage of the Arduino pin connected to the relay's signal pin to a HIGH level, thus deactivating the relay. When the robot detects a crack with a width of 0.6 mm or less, it sends command 1 to the Arduino. This command sets the voltage of the Arduino pin connected to the relay's signal pin to a LOW level, thus activating the relay. Here, 12.24 pixels is the threshold to distinguish between narrow and wide cracks, this threshold can be changed depending on the actual situation. When the robot detects a wide crack, it sends instruction 0 to the Arduino. This instruction sets the voltage of the Arduino pin connected to the relay's signal pin to a HIGH voltage, thereby cutting off the relay's power. Meanwhile, the Jetson Orin Nano also sends a "*bigcrack*" instruction to the Jetson Nano through Socket Message.

```
arduino = serial.Serial('/dev/ttyACM0', 9600)

if max_diameter == 0:
    print('no crack')
    arduino.write(str.encode('0'))
    flag = True

elif max_diameter >1 and max_diameter <= 12.24:
    print('small crack')
    arduino.write(str.encode('1'))
    flag = True

elif max_diameter > 12.24 and flag:

    print('big crack')
    arduino.write(str.encode('0'))
    message = 'bigcrack'
    client_socket.send(message.encode())
    flag = False

else:
    arduino.write(str.encode('0'))
```

Figure 5-51: The snippet of Python code by Jetson Orin Nano to issue commands to Jetson Nano

5.6.2. Testing Spraying Capacity

Once the spraying system is established, it is essential to evaluate its spraying capacity. The typical application volume of agent is between 0.25 – 0.3 L/m for crack repair with a direct jet spray. Figure 5-52 shows the time taken to spray 100 milliliters of water using a pressurizable water bottle and valve, measured with a stopwatch and measuring cup, which is approximately 7.05 seconds. Given the robot's default movement speed of 0.2 meters per second, the calculated water output per meter is 0.0709 liters, which is significantly lower than the required application volume of agent.

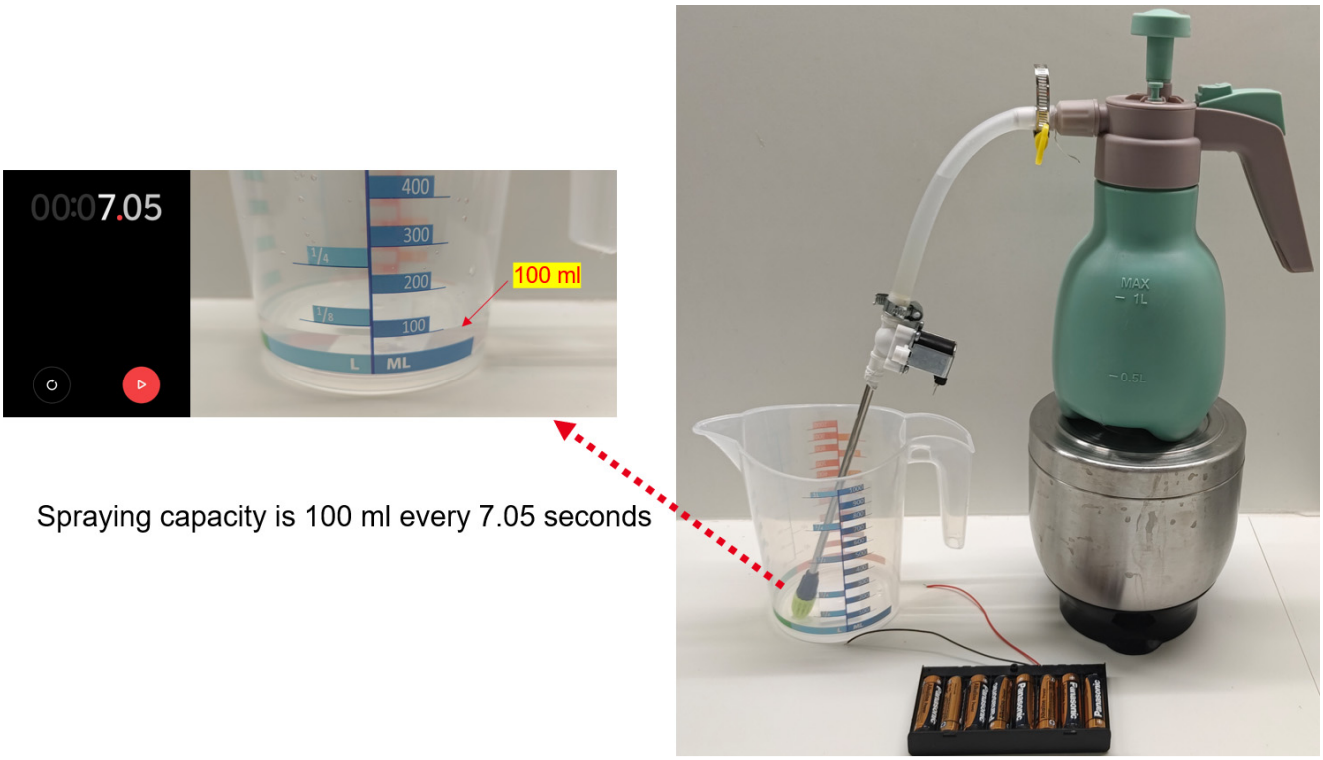


Figure 5-52: Testing the spraying capacity of sprinkler system

There are three solutions to enhance the spraying capacity of the spraying system, thereby meeting the requirement for spraying the repair agent:

1. **Upgrade the Spraying Equipment:** A pressurizable water bottle supporting higher pressure and a water valve with a larger orifice size should be selected to increase the spraying capacity. At present, such suitable pressurizable water bottle and valve are not available on platforms such as Amazon, eBay, or AliExpress. Therefore, custom manufacturing will be required.
2. **Reduce Robot Speed**: Slowing down the robot's movement speed can provide the spraying system with more time to apply a sufficient amount of repair agent onto cracks. JetAuto Pro has a built-in function to control speed. The code for controlling the operating speed of JetAuto Pro is located in the built-in Python script *jetauto_controller_main.py*. After receiving a command at the ROS node */jetauto_controller/cmd_vel*, the function *mecanum.set_velocity* will change the velocity of JetAuto Pro in the x-axis direction, which is represented by *speed_adjust.linear.x*. Reducing the operating speed of the robot also has the advantage of preventing the real-time video captured by the camera from becoming blurry due to excessive movement speed.
3. **Pause for a few Seconds:** It is also possible to develop an algorithm that allows the robot to pause for a few seconds upon detecting a narrow crack, which would give the spraying system more time to apply sufficient repair agent onto the crack. ROS's timer functionality *threading.Timer* can be used to let the robot stay in its current position for few seconds. After the waiting period, the robot's original speed will be resumed to continue moving.

Taking everything into consideration, the solution of reducing the robot's speed is the most feasible and could therefore be adopted.

5.7. Recording Wide Crack by Onboard Computer

The onboard computer of JetAuto Pro, Jetson Nano, is equipped with most of the key frameworks required for robotics, including the ROS, RViz for visualization, and algorithms for controlling both motors and LiDAR sensors. Consequently, the process of retrieving map coordinates is performed on Jetson Nano, which simultaneously handles the task of capturing photos of wide cracks. The gathered coordinates and photos are then uploaded to a cloud-based database.

5.7.1. Frequency Control of the Command from Jetson Orin Nano

The Python code below is deployed on the Jetson Nano. It handles obtaining position of robot through ROS, converting the obtained local coordinates of robot to WGS 84 coordinates, warming-up the camera, capturing photo of wide crack using OpenCV upon receiving commands from the Jetson Orin Nano and uploading coordinates and photo of wide crack to MongoDB Atlas.

```
def main():
    # init ros node
    rospy.init_node('crack_detection_node', anonymous=True)

    # subscribe Jetauto pose
    rospy.Subscriber("/jetauto_1/amcl_pose", PoseWithCovarianceStamped, pose_callback)

    try:
        print("Connected successfully with:", client_address)

        while True:
            data = connection.recv(1024)
            if data:
                message = data.decode()
                print("Received message:", message)
                #connection.sendall(b'acknowledged')

                if message == 'bigcrack' :
                    print('find the crack')
                    cap = cv2.VideoCapture(2)

                    # Warm up the camera, discarding the first few frames
                    for _ in range(30):
                        ret, frame = cap.read()
                    ret, buffer = cv2.imencode('.jpg', frame)
                    print('take the crack photo')

                    # index += 1
                    img_binary = bson.binary.Binary(buffer.tobytes())

                    if current_pose:

                        # for Y
                        lat_intermediate, lon_intermediate = calculate_new_coordinates(initial_lat, initial_lon, -current_pose.position.y, 270)
                        # for X
                        final_lat, final_lon = calculate_new_coordinates(lat_intermediate, lon_intermediate, -current_pose.position.x, 0)

                        fs.put(img_binary, filename="wider crack", X=final_lon, Y=final_lat, H=190, time=datetime.datetime.now())

                        print('insert succesfully')

                    else:
                        print("Current Position is unknown.")
                    cap.release()
                    cv2.destroyAllWindows()


                else :
                    print('find no crack')
            else:
                print("Connection with client is interrupted")
                break
    finally:
        connection.close()
```

Figure 5-53: Key Python code in onboard computer for receiving instructions, reading coordinates, and uploading information of crack

The default movement speed of the robot is 0.2 meters per second, to ensure adequate dispensing of the repair agent, the robot's speed would be reduced to one-third of its original value. At the same time, the camera requires 33.33 milliseconds to capture an image, while the model takes 55 milliseconds to process each image. The robot's relatively slow movement speed, combined with the fast image capture speed of the camera and the rapid inference speed of the model, leads to the following issue: because the robot moves slowly, when a wide crack enters the

camera's view, it takes some time for it to leave the camera's field of vision due to the slow movement. During this time, the camera takes several photos of the crack in a short period, and YOLOv8s also quickly outputs multiple detection results for the wide crack. Consequently, the Jetson Orin Nano sends repeated instructions to the Jetson Nano to record crack information within a short time frame, which results in a large amount of coordinate and image data corresponding to the crack being uploaded to the database. To address this issue, the script deployed on the Jetson Orin Nano will be modified instead of the one on the Jetson Nano, as the former is easier to modify.

As shown in Figure 5-51, a variable *flag* is used to control the frequency of command transmission. When a wide crack is detected, *flag* is set to *false*. In this state, even if the camera continues to identify a wide crack, no instructions will be sent to the Jetson Nano. As the robot moves forward and the camera shifts away from the current wide crack, if a small crack or no crack is detected, *flag* is reset to true. This allows the robot to send instructions to the Jetson Nano when a new wide crack is detected. However, if a location adjacent to the current wide crack is also a wide crack, the *flag* will be not changed, it is still *false*, then the robot is unable to record this adjacent wide crack. The solution is to either raise the height of the USB camera that is used to record wide cracks or use a wide-angle camera. This would expand the camera's field of view, allowing it to cover nearby wide cracks when taking a picture. If the budget allows, a 360-degree camera, such as the Ricoh Theta Z1, can be used to capture all objects within the surrounding 360-degree field of view.

5.7.2. Reading and Converting Coordinates

Due to the weak GNSS signals indoors, it is necessary to utilize the ROS to obtain robot's pose as the coordinate for the wide crack. The robot's pose is typically published on a topic */amcl_pose*. And RViz can show the robot's current pose on the map. When the robot moves, the LiDAR provides real-time feedback on its position, and RViz synchronously updates the robot's position on the map. The obtained coordinates via ROS are actually the distances of robot from the initial point. Therefore, to retrieve the coordinates, the robot must first be positioned at the actual site corresponding to the initial point on the map, as show below.

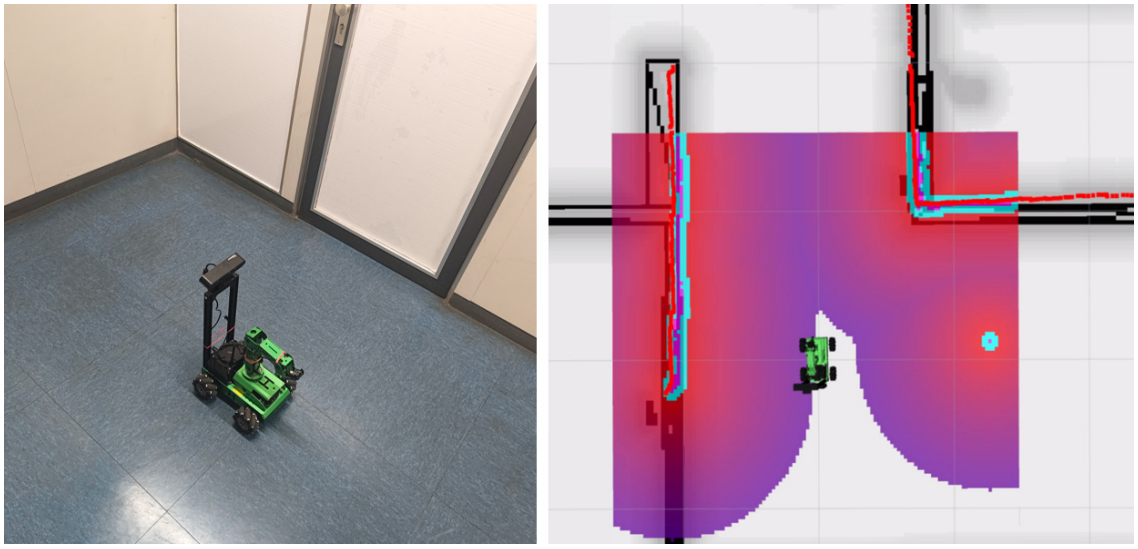


Figure 5-54: Robot initial position comparison: actual Initial location in reality (left) and initial position displayed in RViz (right)

In this process, the robot's LiDAR should be activated and RViz should be launched with the relevant map. The package *geometry_msgs* in ROS provides messages for common geometric primitives such as points, vectors, and

poses. The message type *geometry_msgs/PoseWithCovarianceStamped* of the *geometry_msgs.msg* in ROS can be used to expresse an estimated pose with a reference coordinate frame and timestamp. Once the local coordinates in map are acquired, they must be converted into WGS84 coordinate system format. The spatial reference system used in the 3D geospatial platform Cesium is WGS84, which is a reference system used by the satellite navigation systems like GNSS and is used in various mapping applications. As shown in the figure below, WGS84 models the Earth as an oblate spheroid (flattened at the poles and bulging at the equator).

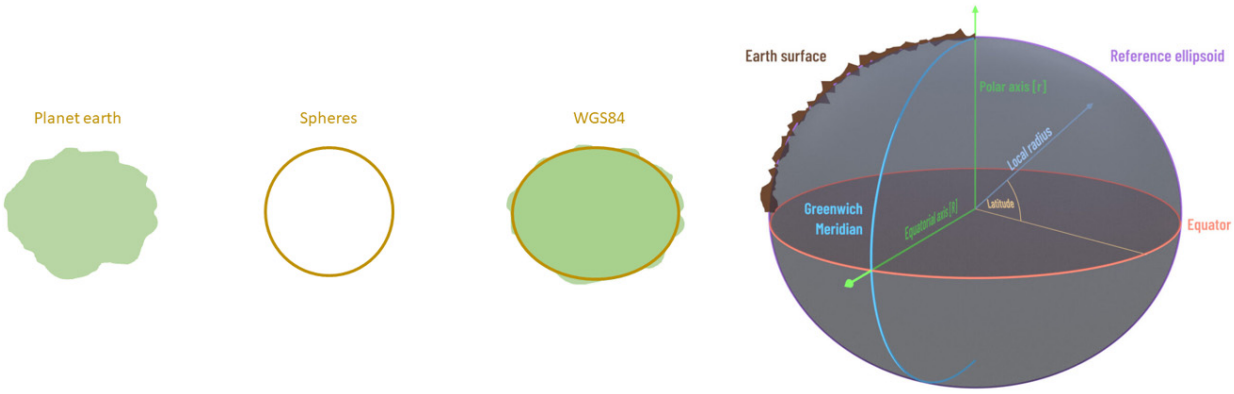


Figure 5-55: The ellipsoid shape of WGS84 (Wagner, 2024)

RViz typically uses a local Cartesian coordinate system centered around a specific origin point, often the starting position of the robot or a designated reference. Since WGS84 models the Earth as an oblate spheroid, the curvature of the Earth must be considered when converting between a local Cartesian coordinate system and WGS84 coordinates, especially over large distances. For relatively small areas (like a building or a small park), the Earth's curvature can often be neglected without causing significant inaccuracies. Thus, it is possible to establish a one-to-one correspondence between the robot's points in the RViz map and points on a Cesium map. For example, as shown in Figure 5-56, in the robot's map, the origin (0, 0, 0) corresponds to the point (8.678653224872748, 49.862085772385974, 225.9496507496025) on the Cesium map. Here, 8.678653224872748 represents the WGS84 longitude, 49.862085772385974 is the WGS84 latitude, and 225.9496507496025 is the height (or elevation) of the corresponding floor in WGS84 system.

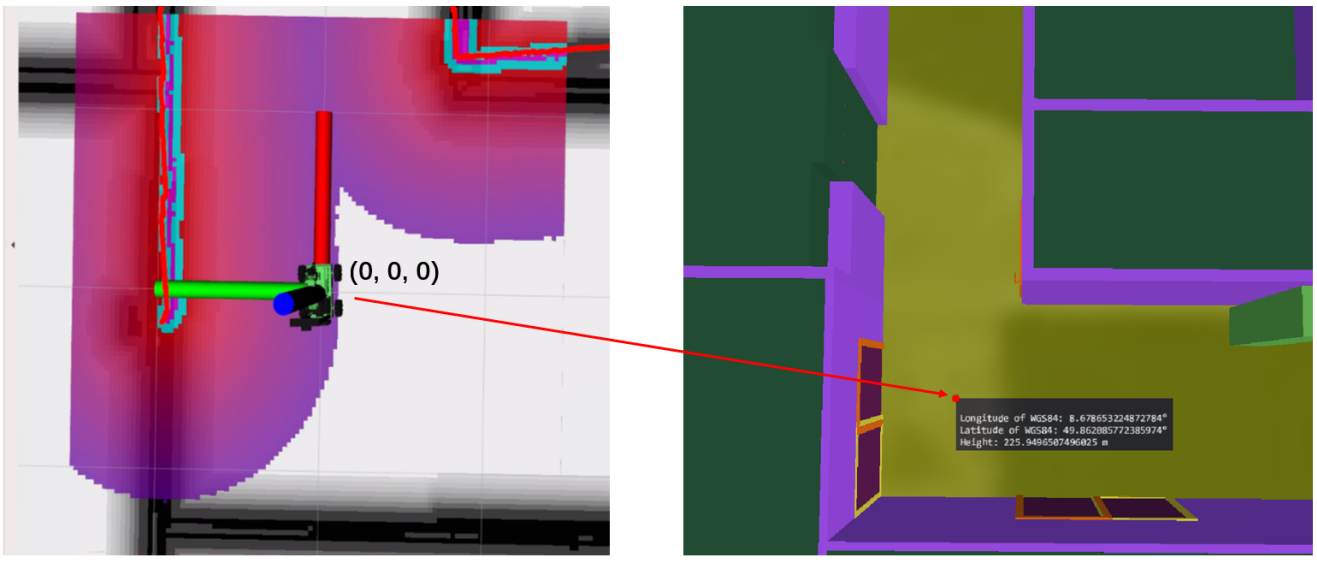


Figure 5-56: The corresponding coordinates of the origin of the robot in Cesium

geopy is a Python client for several popular geocoding web services. The function *destination(point, bearing, distance=None)* of *geopy* can calculate destination point using a starting point, bearing and a distance. The point parameter is the starting point's latitude and longitude, bearing is the direction in degrees from the north (0°), and distance is the distance from the starting point to the destination point. The origin is the starting point for the function. The following is the core algorithm, the initial longitude and latitude can be changed based on the use case.

```
def calculate_new_coordinates(lat, lon, distance, bearing):
    # Define starting point
    start_point = Point(latitude=lat, longitude=lon)
    # Calculate new location
    new_point = geodesic(meters=distance).destination(point=start_point, bearing=bearing)
    return new_point.latitude, new_point.longitude

# starting point
initial_lat = 49.862085772385974
initial_lon = 8.678653224872748

# for Y
lat_intermediate, lon_intermediate = calculate_new_coordinates(initial_lat, initial_lon, -current_pose_position_y, 270)
# for X
final_lat, final_lon = calculate_new_coordinates(lat_intermediate, lon_intermediate, -current_pose_position_x, 0)
```

Figure 5-57: The algorithm to convert the local Cartesian coordinate to WGS84 coordinate

5.7.3. Camera Warm-Up

Upon receiving a command from the Jetson Orin Nano, the Jetson Nano immediately calls the camera to take a photo using OpenCV. If the camera module has not been warmed up at this time, taking a photo immediately after turning on the camera will cause overexposure in bright light conditions (Figure 5-58 (a)), while in low light conditions, the photo will appear very dark (Figure 5-58 (c)). Figure 5-58 (b) shows the normal shooting effect after the camera has been warmed up under strong light.

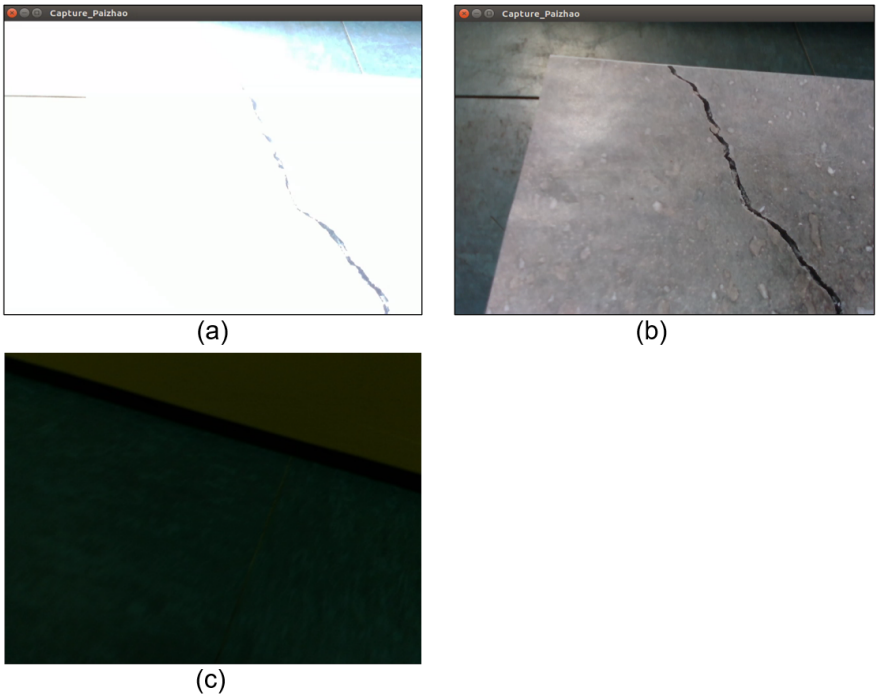


Figure 5-58: Comparison of the image capture quality under conditions with and without camera warm-up

In order to stabilize the camera and ensure optimal image capture, a warm-up process is implemented by discarding an initial set of 30 frames after activating the camera using OpenCV. The exact number of discarded frames, which is typically set to 30, can be adjusted based on the specific characteristics of the camera. Once the camera reaches the specific frame, it has typically stabilized, ensuring the captured image is of standard quality.

```
cap = cv2.VideoCapture(2)

# Warm up the camera, discarding the first few frames
for _ in range(30):
    ret, frame = cap.read()
ret, buffer = cv2.imencode('.jpg', frame)
print('take the crack photo')
```

Figure 5-59: Code for camera warm-up

5.7.4. Storing Data in MongoDB Atlas

Storing data locally on the robot's onboard computer can present challenges for visualization and accessibility. To make the data accessible to everyone, it would be more practical to upload the data to a cloud-based database such as MongoDB Atlas. MongoDB is a non-relational database manager used to store big data files. Unlike SQL databases which sort data into rows and columns, MongoDB stores data in the form of documents and collections. MongoDB Atlas makes it easy to set up, operate, and scale MongoDB deployments in the cloud. User can get started with a MongoDB developer sandbox in MongoDB Atlas for free with basic configuration options. Through a Python script, it is possible to store the recorded time, the coordinates of the crack, and the corresponding picture of the crack into MongoDB Atlas. To ensure the successful upload of data to MongoDB Atlas, it is necessary to add the current IP address to the MongoDB Atlas access list. If the current IP address has not been added, MongoDB Atlas will provide an appropriate notification, as illustrated in the figure below.

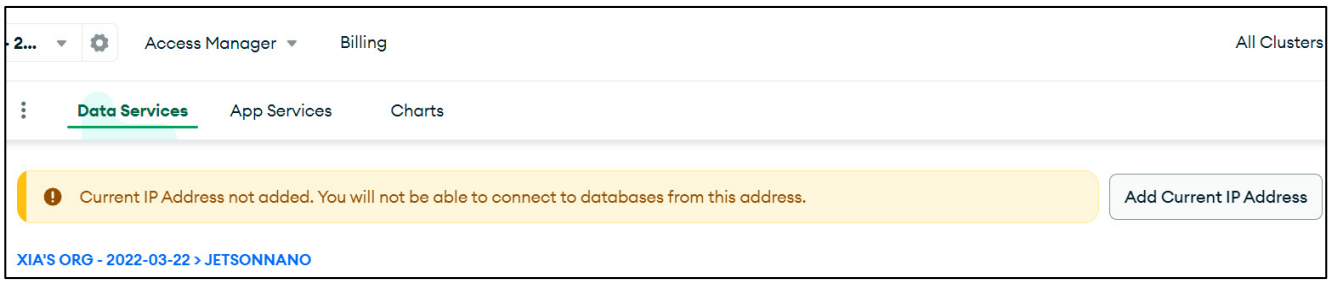


Figure 5-60: Notification about adding current IP address

The picture of the wide crack is stored in binary format, as shown in Figure 5-61. MongoDB GridFS is a great tool for uploading large images files exceeding the BSON-document size limit of 16 MB to MongoDB bucket with minimum configuration. GridFS basically takes a file and breaks it up into multiple chunks which are stored as individual documents in two collections: the chunk collection (stores the document parts), and the file collection (stores the consequent additional metadata). Each chunk is limited to 255 KB in size.

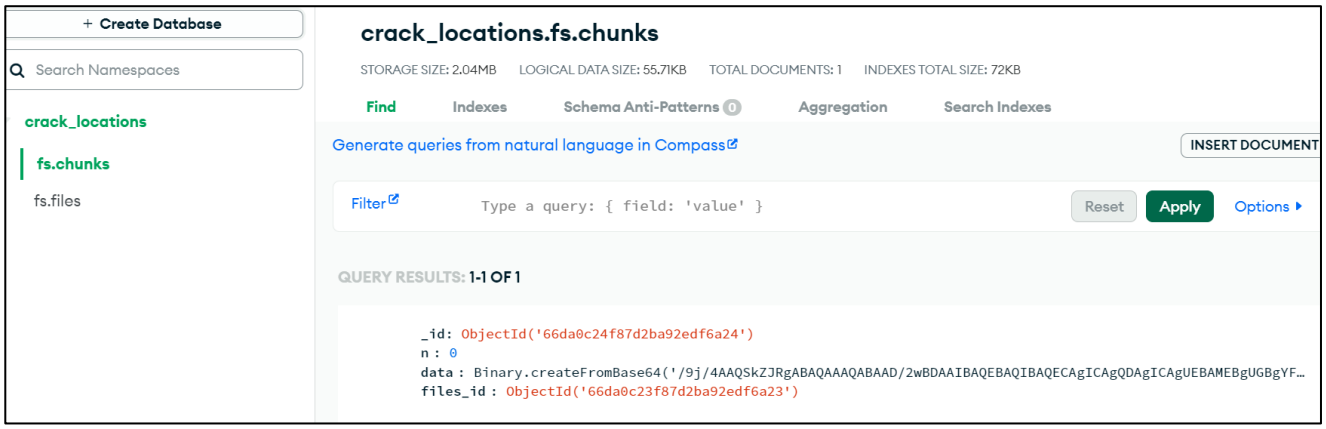


Figure 5-61: Storing images as binary data in MongoDB Atlas

When taking photos of cracks via USB camera, it is important to ensure that the size of the photo is not too large, as MongoDB server supports BSON document sizes up to 16793598 bytes.

```
764, in command
    exhaust_allowed=exhaust_allowed,
  File "/home/jetauto/.local/lib/python3.6/site-packages/pymongo/network.py", li
ne 134, in command
    message._raise_document_too_large(name, size, max_bson_size + message._COMMA
ND_OVERHEAD)
  File "/home/jetauto/.local/lib/python3.6/site-packages/pymongo/message.py", li
ne 1010, in _raise_document_too_large
    " bytes." % (doc_size, max_size)
pymongo.errors.DocumentTooLarge: BSON document too large (24883518 bytes) - the
connected server supports BSON document sizes up to 16793598 bytes.
```

Figure 5-62: Error logs generated when the picture size exceeds the limit of MongoDB server

5.8. Visualizing Wide Cracks in BIM-GIS Web-Application

To visualize the information of cracks recorded in MongoDB Atlas, a web-based application based on Cesium was specifically developed, incorporating many BIM-GIS integration features. Compared to desktop applications, web applications have many advantages. Web apps can run on any device that supports modern web browsers, regardless of the operating system. Users can access web apps directly through a browser without the need to download or install any software. And multiple users can work simultaneously within the same web app, which is ideal for team collaboration.

5.8.1. IFC to 3D Tiles

This work uses the platform Cesium as a GIS viewer. Cesium can visualize and analyze geospatial data with its rendering engine. However, it does not support IFC files natively. To integrate IFC models within Cesium, a common approach involves converting the IFC files into a format compatible with Cesium. Formats such as glTF (Graphics Language Transmission Format) are often used for this purpose. The IFC format is rich in semantic information, which includes not only the geometry and appearance of building elements but also their properties, relationships, and classifications. When IFC files are converted to glTF, the focus is often on the visual aspects,

such as the geometry and textures, which are essential for 3D visualization. But the semantic information can be lost or significantly reduced during the conversion process. This is because glTF is primarily designed for efficient 3D graphics rendering and does not inherently support the complex metadata and relationships found in IFC files. Other significant challenges with converting IFC files to glTF for visualization in Cesium is the loss of interior structure visibility and the inability to interact with individual components of the model. Users is unable to visually pass through walls to observe the internal structures of a building, necessitating entry through openings such as doors and windows for interior access. In IFC viewers, the ability to select individual building elements and view their properties is a key feature due to the rich semantic data embedded within the IFC format. However, when converting IFC files to glTF for visualization in Cesium, this level of interactivity is lost because of the differences in how data is structured and presented in the two formats.

To visualize BIM model within Cesium, the IFC model typically needs to be converted into 3D Tiles, which is developed by Cesium to efficiently render massive 3D content in web browser. This method can address the issues mentioned above. The open-source tool *Py3DTilers* can easily convert 3D Tiles from IFC. It can preserve attributes of IFC model by conversion.

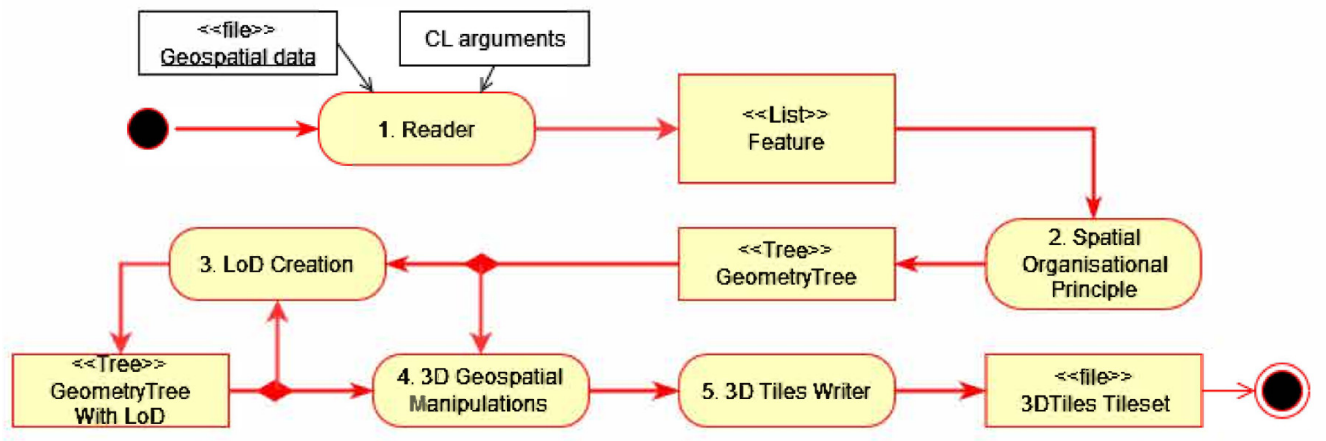


Figure 5-63: Activity diagram of the 3D Tiles creation process (Marnat et al., 2022)

Since IFC models do not inherently contain geographic coordinates for each structure, placing the 3D Tiles converted by *Py3DTilers* in their corresponding correct positions poses a challenge. The cloud-based platform Cesium ion is unable to adjust the location of these converted 3D Tiles. To address these issues, *Cesium.Transforms.eastNorthUpToFixedFrame* is used to create a transformation matrix. Subsequently, *tileset.modelMatrix* applies this transformation to the 3D Tiles converted by *Py3DTilers*, enabling the placement of these 3D Tiles at specific coordinates within the predefined WGS84 coordinate system. The *Cesium.Transforms.eastNorthUpToFixedFrame* function in CesiumJS is used to compute a 4x4 transformation matrix from a reference frame with east-north-up (ENU) axes centered at a provided origin point to the fixed reference frame of a specified ellipsoid, typically the WGS84 ellipsoid. The *tileset.modelMatrix* property in Cesium is used to transform the coordinates of a 3D Tiles. It allows user to apply a 4x4 transformation matrix to the 3D Tiles, which can be used to translate (move), rotate, or scale the 3D Tiles relative to its original position and orientation.

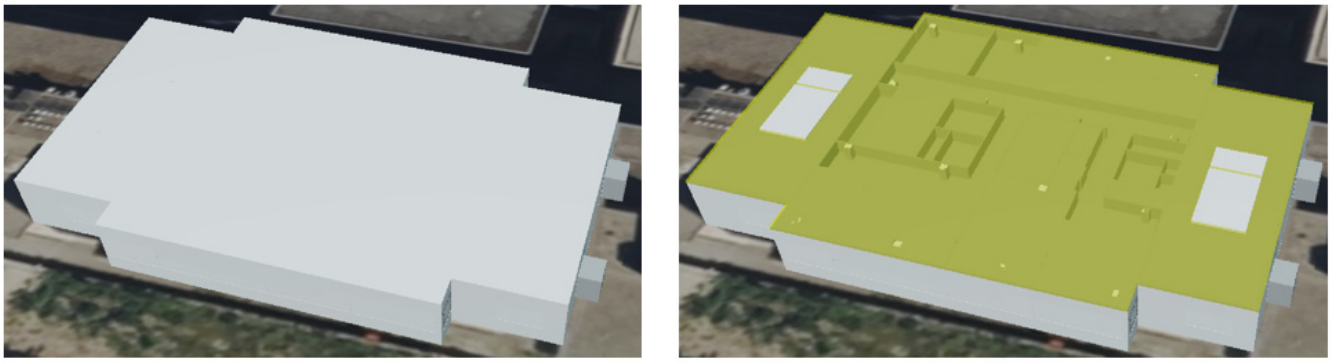
```
//Latitude, longitude in degrees and height in meters
var longitude = 8.678300293;
var latitude = 49.86202121;
var height = 225.5699;
// Convert position to Cartesian3
var position = Cesium.Cartesian3.fromDegrees(longitude, latitude, height);
// Create a translation matrix
var translation = Cesium.Transforms.eastNorthUpToFixedFrame(position);
// Apply the translation matrix to the tileset
tileset.modelMatrix = translation;

viewer.scene.primitives.add(tileset);
```

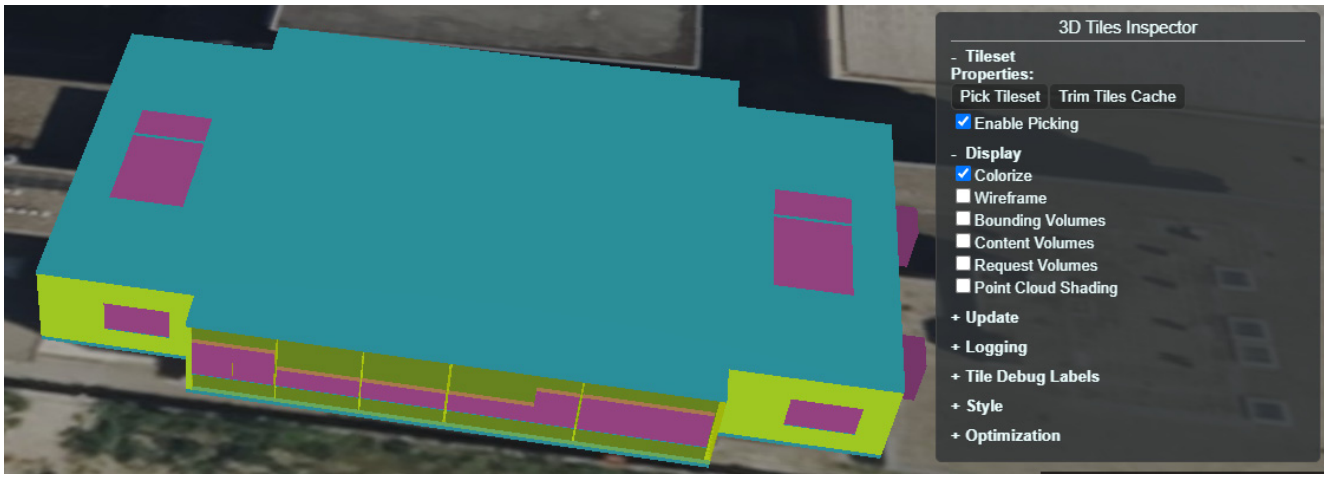
Figure 5-64: JavaScript code to adjust the location of converted 3D Tiles in Cesium

5.8.2. Color and Visibility Settings of Surface

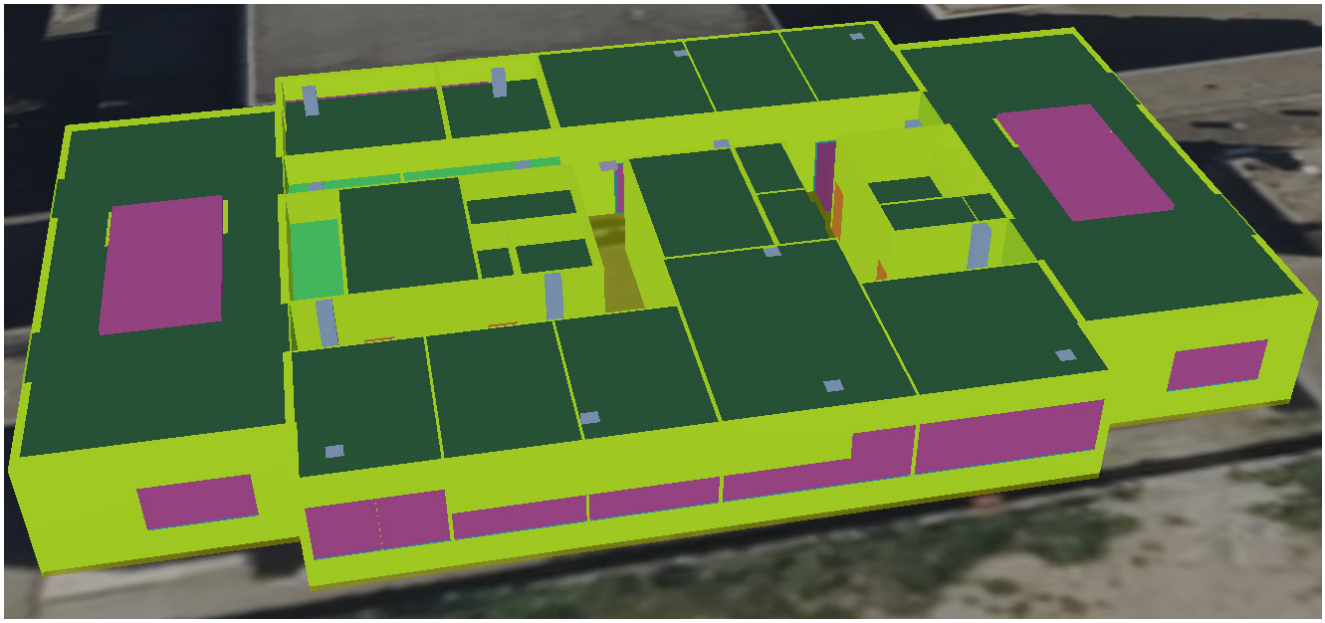
As shown in Figure 5-65 (a), the color of the converted 3D Tiles structure is white. To facilitate the selection of specific structures, when the mouse hovers over a structure, its color changes to a semi-transparent light yellow (Figure 5-65 (b)). This functionality utilizes the *highlighted.feature.color* method of CesiumJS. In 3D Tiles, there are several methods to enhance user interaction experience beyond color changes such as Styling and Conditional Display. User can use the *Cesium3DTileStyle* to style features based on their properties, allowing for dynamic changes in appearance based on conditions such as height (Figure 5-65 (c)). The 3D Tiles converted by *Py3DTilers* is also unable to allow users to visually pass through external structure such as walls to observe the internal structures. To solve this problem, Cesium's built-in functions is utilized to achieve the following functions: when the user hovers the mouse over a surface, it becomes transparent; Upon clicking the middle mouse button, the surface is hidden, enabling the inspection of the infrastructure's internal structure (Figure 5-65 (d)). The hiding feature can be implemented by modifying the show property of *inspectorViewModel.feature.show* to false. This approach is commonly used in CesiumJS that visualize 3D Tiles, to control the visibility of individual features or tiles.



(a) (b)



(c)



(d)

Figure 5-65: Setting style of surface for the converted 3D Tiles in Cesium: (a) converted 3D Tiles, (b) color change by mouse hovering, (c) colorizing structure based on height and (d) hiding surface

5.8.3. Cracks Visualization

To connect to a MongoDB Atlas cluster, a URI (Uniform Resource Identifier) string that includes the necessary connection details is required. The BIM-GIS web application has been designed as a dynamic web application using Flask and can retrieve data stored on MongoDB Atlas. As shown in Figure 5-66, all the wide cracks will be visualized as red points created by CZML format. CZML is a JSON format for describing a time-dynamic graphical scene, primarily for display in a web browser running Cesium. It describes lines, points, billboards, models, and other graphical primitives, and specifies how they change with time. When user clicks on a red point, the photo, description, and coordinates of crack will be displayed in the InfoBox, which is a widget for displaying information or a description in Cesium.

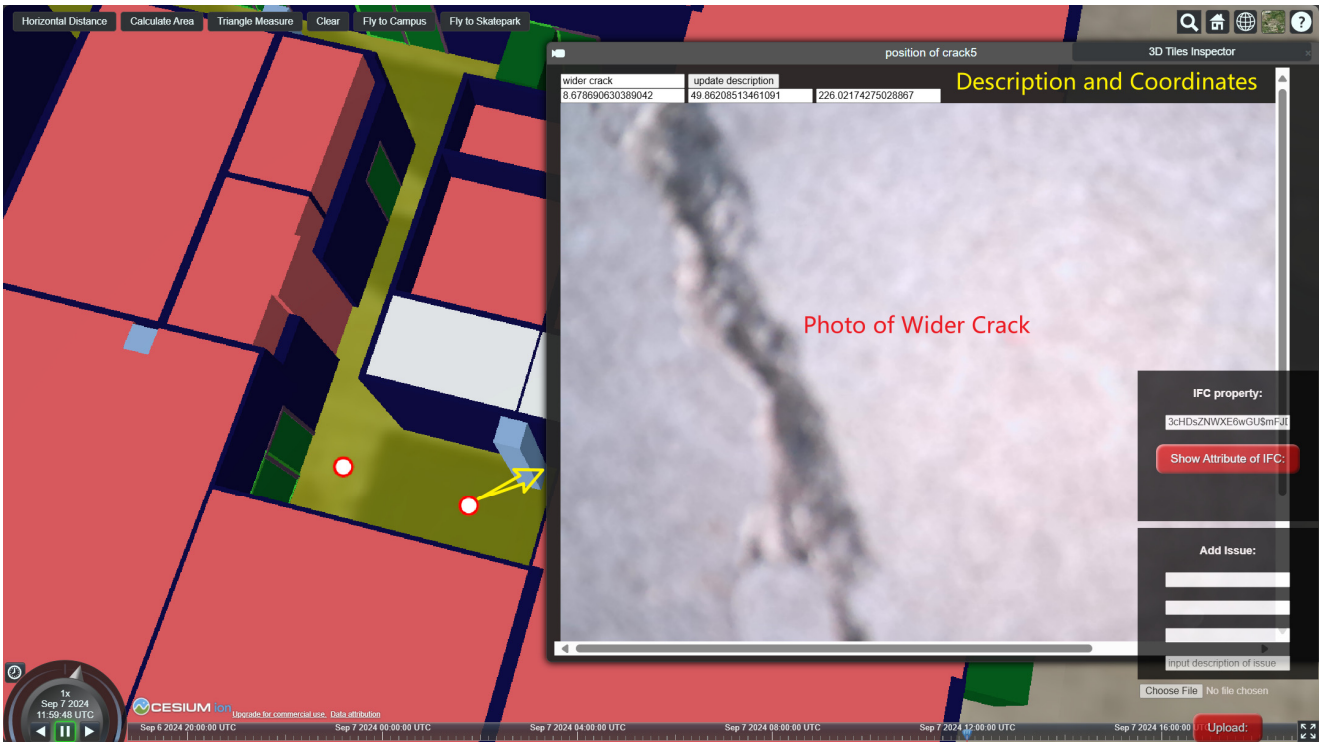


Figure 5-66: Visualization of wide crack and corresponding photo, description, and coordinates

As the code shown in the Figure 5-67, all cracks stored in the database are visualized as red points on the main page of the web application. The coordinates and IDs of the red points come from the backend. During the visualization process, the position of each red point is determined by the coordinates of crack. Information such as the crack's photo, coordinates, and description are dynamically loaded into the InfoBox through an iframe that displays the web page based on HTML file *content.html*. An iframe (short for inline frame) is a HTML (HyperText Markup Language) element used to embed another HTML page within the current webpage. It allows to place a separate document or web application into a portion of web page, effectively creating a small window or frame that can display content from a different source. The *content.html* is placed in the static folder of the project, which serves as a public resource directory easily accessible by the web application. The content of content.html is matched based on the ID of each crack. When a user clicks on a red point representing a crack, the front end of the Flask application sends the ID to the back end, where a function uses this ID to query the database, find the corresponding crack data, and then visualize this data in content.html. Since content.html is embedded in the InfoBox, the InfoBox can display the relevant data of this crack. Upon clicking on red point, the process must instantly retrieve the ID, fetch the corresponding data based on the ID, and then send the retrieved data back to the front end. This requires the use of AJAX. AJAX allows for the asynchronous exchange of data between the client and the server, which means the webpage can update dynamically in response to user actions. To grab the ID of crack by click on it, the jQuery, a popular JavaScript library that simplifies HTML document traversal, is also used.

```
var datas = {{ data | tojson }}
datas = JSON.parse(datas)
const pointsData = datas.map((point, i) => {
  return {
    id: "point " + i,
    name: "position of crack" + i,
    description:
      `
        <iframe src="/static/content.html?id=${point.id}" width="1000" height="800" frameborder="0"></iframe>\
      `,
    position: {
      cartographicDegrees: [point.X, point.Y, point.H],
    },
    point: {
      color: {
        rgba: [255, 255, 255, 255],
      },
      outlineColor: {
        rgba: [255, 0, 0, 255],
      },
      outlineWidth: 4,
      pixelSize: 20,
    },
  }
})
```

Figure 5-67: Key JavaScript code to visualize different points associated with corresponding photo of crack

5.8.4. Additional Features

To enhance user experience, additional useful features have been also developed for this web application.

Quick Switching between Multiple Projects

Some projects are spaced far apart, making manual switching cumbersome. To address this, the *viewer.camera.flyTo* method is used in conjunction with buttons on the main page, allowing users to quickly switch between projects with a single click. The *flyTo* method smoothly transitions the camera from its current position to a specified destination, creating a flight-like animation. The destination can be defined by various parameters, such as a target position or a simple object. *orientation* is an optional parameter that allows user to define the camera's rotation when it reaches the destination. As shown below, when the user clicks the "Fly to Campus" button, the screen will switch to a view above the school's buildings. Similarly, when the user clicks the "Fly to Skatepark" button, the screen will switch to a view above the park.

Figure 5-68: GUI of the quick switching between multiple projects

Measuring Tool

In CesiumJS, the measuring tool is a feature that allows users to measure distances, areas, and volumes within the virtual globe or map. This tool is particularly useful for geographic analysis, site surveys, and any application where accurate measurements are required. However, the measurement and analytics widgets are not part of CesiumJS. They're part of the commercial Analytics SDK. This work referenced an open-source JavaScript script, *cesium-measure.js*, from GitHub and integrated it into the BIM-GIS web application (zhangti, 2020). With the buttons "Calculate Area", "Triangle Measure", and "Horizontal Distance", corresponding functions can be called up, as shown in Figure 5-69. After clicking the 'Clear' button, the measurement records are deleted, allowing for re-measurement. To ensure smooth measurement, users must be able to select any point in the 3D scene. To achieve this, *cesium-measure.js* utilizes several CesiumJS picking methods, such as *scene.drillPick, scene.pickPosition, scene.camera.getPickRay, scene.globe.pick, and scene.camera.pickEllipsoid*.

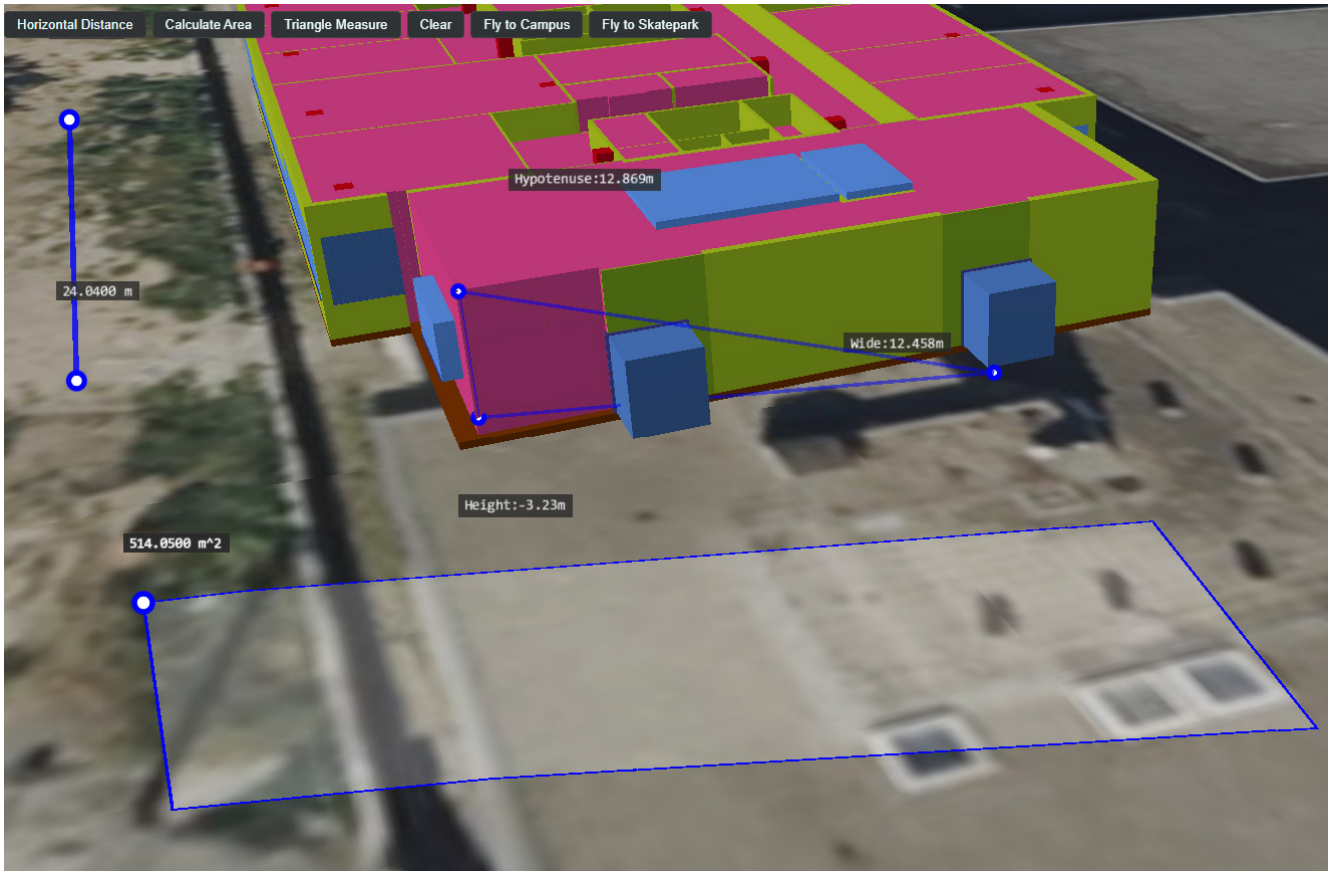
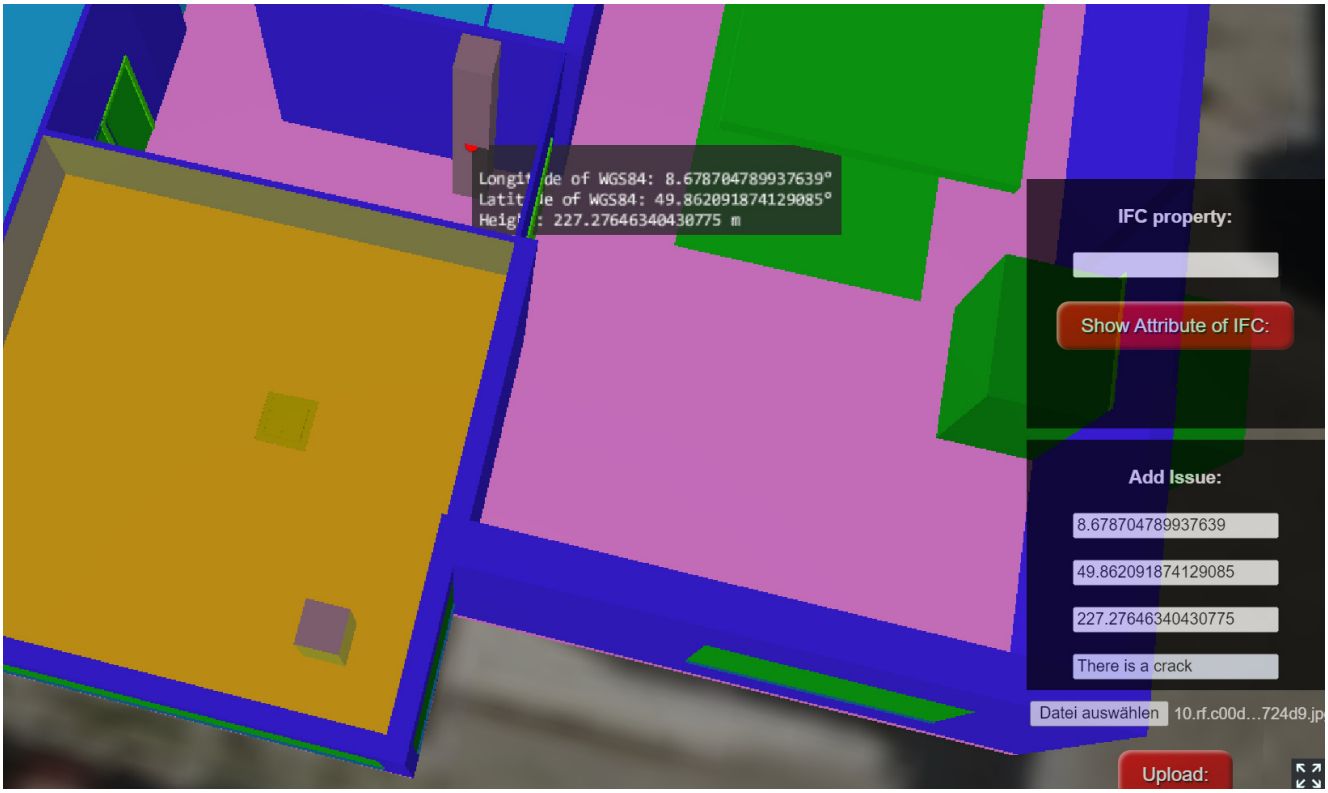


Figure 5-69: Effects of using measuring tools

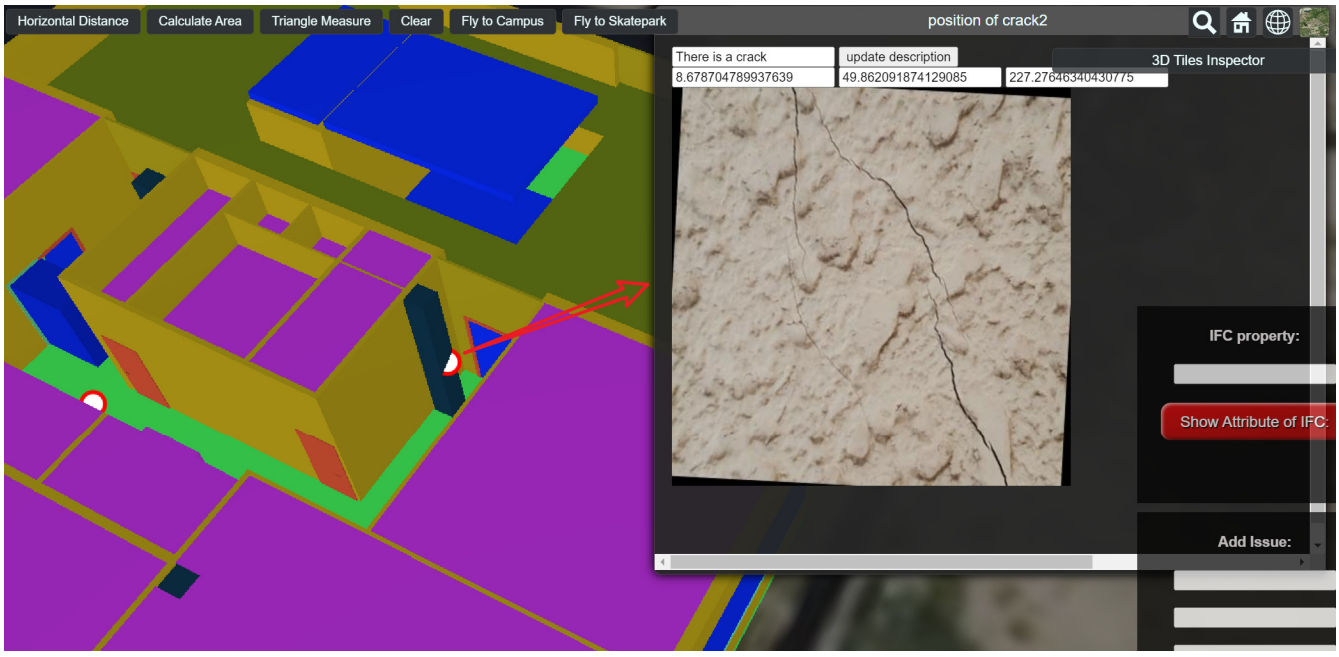
Adding Description of Issues

In the 3D scene, issues may arise at any location, so users must be able to click on any point in the 3D scene to add annotations. The previously used *cesium-measure.js* already includes this functionality, and it can be directly invoked. The coordinates of the selected point are displayed in a pop-up window, and they will be automatically filled into the input fields of the "Adding Issue" form. The user can manually enter a description of the issue, upload the relevant image of the issue, and click the "Upload" button to complete the annotation, as shown in Figure 5-70 (a). All the information will be stored in MongoDB Atlas and the annotation will be visualized as red point in the viewer. When user clicks this red point, all the data of the annotation will be displayed, as shown in Figure 5-70 (b). For existing issue annotations, the description can be modified later. For example, if the initial

crack has been repaired, other users can edit the annotation. After clicking the “update description” button, the description of the annotation will be updated accordingly.



(a)



(b)

Figure 5-70: Adding issue: (a) selecting a location and adding description and (b) visualizing the issue in Cesium

Property Search

In CesiumJS viewer, clicking on each component of the 3D Tiles converted by *Py3DTilers* allows the corresponding attributes to be displayed in Cesium’s InfoBox. However, the InfoBox can only display a subset of

the properties. Each component of the converted 3D Tiles retains the global ID from the corresponding IFC component. According to this feature, when a component is clicked, its ID will be retrieved and passed to the backend of Flask. *IfcOpenShell* is a powerful IFC engine which allows users to read, write, and manipulate IFC files, as well as extract information such as geometry, properties, and relationships from these files. With the help of *IfcOpenShell*, the attributes of the corresponding IFC component can be parsed using its ID. These parsed attributes are then sent back to the Cesium frontend. As a result, users can view the attributes of the 3D Tiles component. As shown in Figure 5-71, when a user clicks on a building's roof, its ID will automatically be filled into the IFC property input field. After clicking the "Show Attribute of IFC" button, the roof's name will appear below.

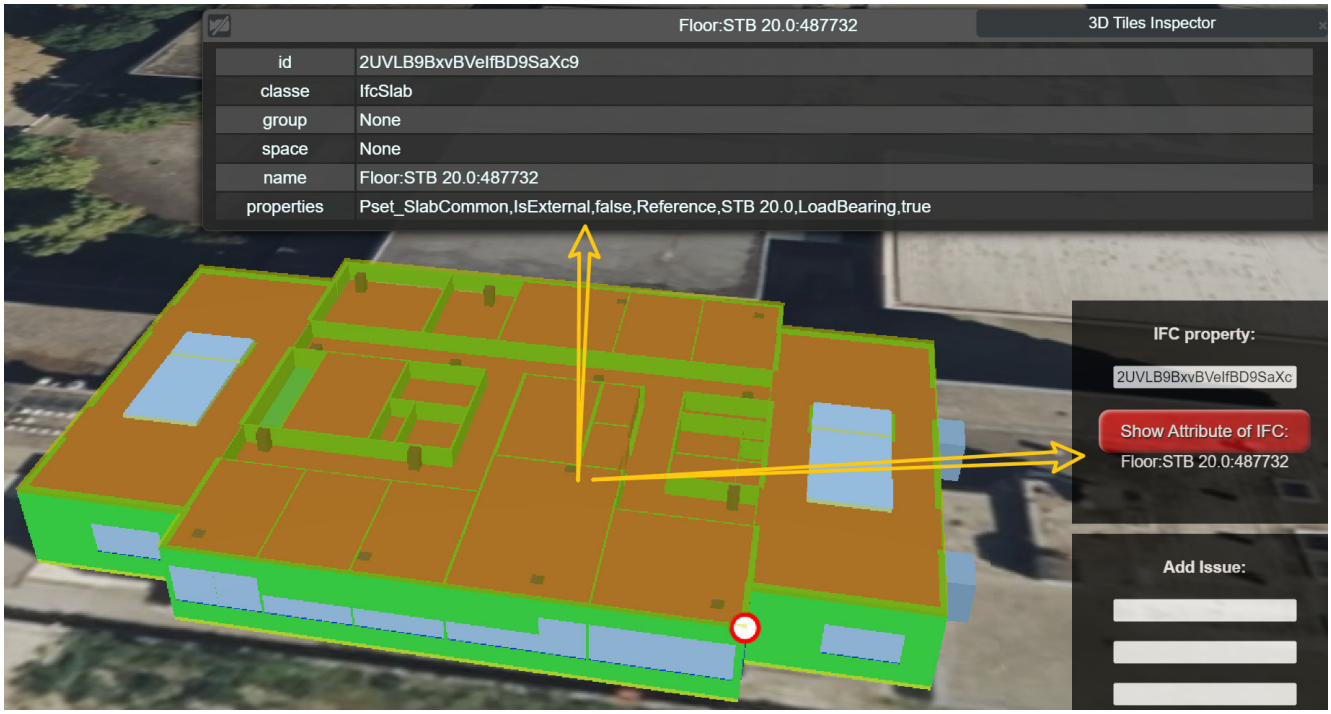


Figure 5-71: Display IFC attribute in Cesium using IfcOpenShell

Users can also customize the backend code as shown in Figure 5-72 to display any other attributes they need to the front end.

```
@app.route("/showifc", methods=['GET','POST'])
def showifc():
    if request.method == "POST":
        global_id = request.form['ifc_element']

        ifc_file = ifcopenshell.open("static/3d_model/iib.ifc")

        object_selected = ifc_file.by_guid(global_id)
        name=object_selected.Name

        if object_selected:
            return jsonify({'output_ifc': name})
    return render_template('map.html')
```

Figure 5-72: Key Python code for retrieving the name of selected IFC element using IfcOpenShell

6. Evaluation and Application Example

Testing is a critical component of any research project, including a thesis. The primary goal of testing in a thesis is to evaluate the performance and effectiveness of a hypothesis, theory, or model. Through testing, developer can validate the accuracy and reliability of findings, as well as identify any limitations or potential areas for improvement. The focus of this work is on the design and prototyping of a robot for repairing narrow cracks, as well as a system for automatically recording and visualizing wide cracks. To verify the usability of the developed solutions, tests will be conducted under various operating conditions. Initially, the robot's crack segmentation and spraying capabilities will be tested using concrete samples with cracks. Subsequently, the robot's autonomous navigation, recording and visualization of wide cracks will be tested indoors. Finally, the entire system will be tested at a local skate park.

6.1. Practical Specific Tests for Certain Functions

Before full deployment, it is common to test and optimize individual functionalities one by one. If certain functions do not operate properly or do not meet the requirements during testing, these need to be addressed before proceeding. Once all functionalities have passed their individual tests, they can be tested collectively to ensure they coordinate and operate harmoniously together.

6.1.1. Test on Concrete Samples with Crack

To evaluate the robot's capabilities in crack segmentation and agent spraying, multiple concrete samples featuring pre-formed cracks were prepared for testing. These cracks were generated by applying compressive or tensile forces from mechanical equipment or human. The concrete slab shown in the figure below was purchased from the BAUHAUS store, and cracks were induced by striking with stones. When the robot identifies a narrow crack, it will spray repair agent onto crack.

Figure 6-1: Testing robot on concrete slab with narrow crack

Figure 6-2 (a) illustrates the segmentation result of the crack on the concrete slab before the application of repair agent. The segmentation performance is good. When the solution of component B is sprayed onto the crack, the resultant gel impacts the crack segmentation, as shown in Figure 6-2 (b). With the continued spraying the solution of component B, the gel will completely cover the crack, and the robot will be unable to identify the crack anymore, as shown in Figure 6-2 (c). To maintain the moisture of the crack, it is necessary to regularly spray water to the crack. Figure 6-2 (d) shows the segmentation result of the same crack detected by the robot one day after treatment. The white material in the crack and in the surrounding of crack is the dried gel. Although machine learning models can still identify crack on the surface of concrete, the appearance of the crack has been altered, and the segmentation result is now significantly different from the initial result.

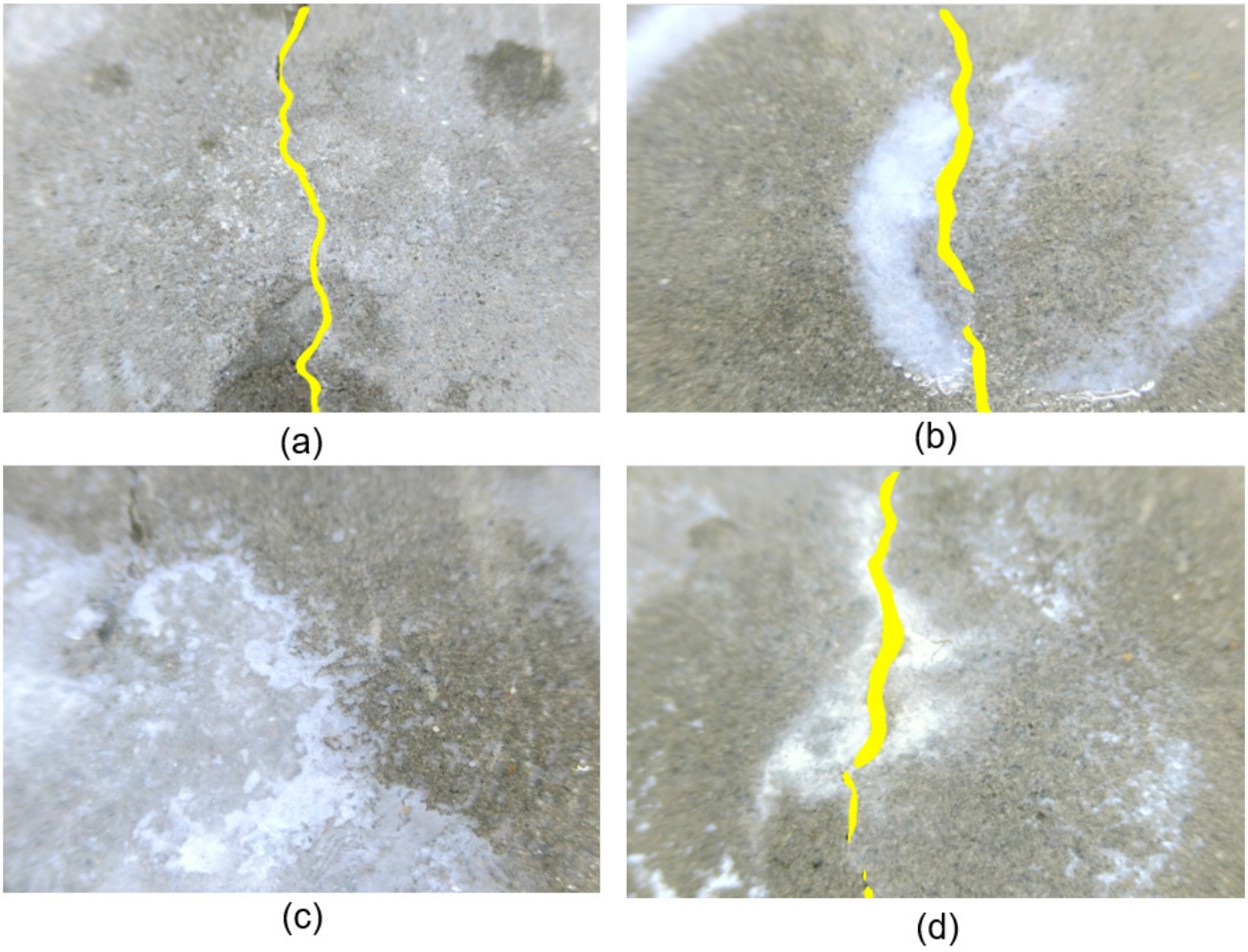


Figure 6-2: The segmentation results of the cracks on the concrete slabs at various stages

Figure 6-3 shows the effects of the crack repair. Six weeks later, the narrow crack was filled with a white substance, and the concrete slab that was originally broken into two halves was tightly bonded together. Even when lifted by hand, it would not break apart. Considerable force was required to break it apart. Although verifying the effectiveness of the repair agent is not the task of this work, it can be inferred that the repair agent does indeed induce bacteria to produce calcium carbonate, as advertised, and seal narrow cracks of concrete.

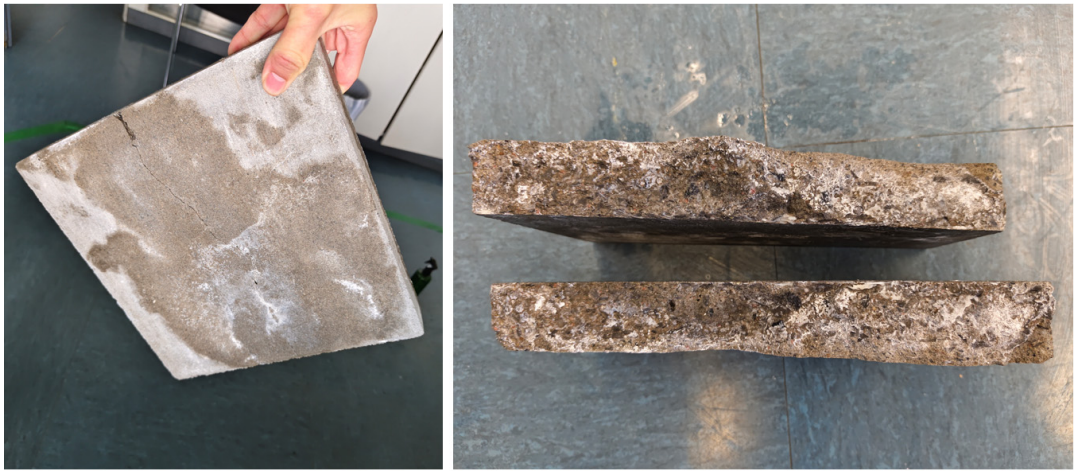


Figure 6-3: Effects of the crack repair after 6 weeks

To verify the repair effect of the repair agent, the Institute of Construction and Building Materials at the Technical University of Darmstadt produced several concrete sticks with cracks, as shown in Figure 6-4 (a). The concrete stick was placed at the end of the wooden tray. Figure 6-4 (b) shows the segmentation results displayed on the notebook's remote desktop via NoMachine. The robot mistakenly identified the gap between the concrete stick and the wooden tray as a crack. Therefore, a paper with a concrete surface texture needs to be applied over the gap, as shown as Figure 6-4 (c). Figure 6-4 (d) shows application of repair agent, the robot correctly identified the narrow cracks, sprayed the repair agent, and then moved back.

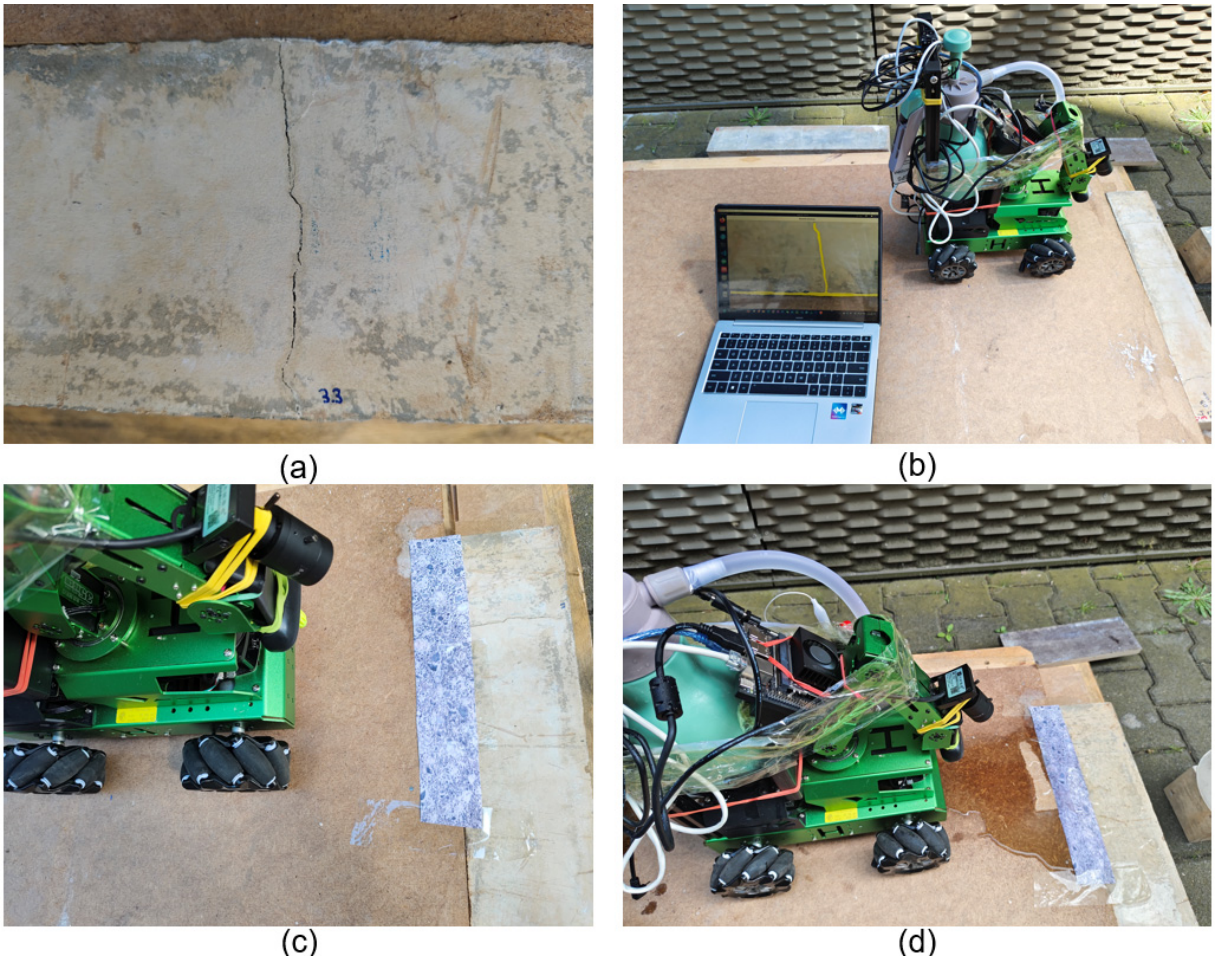


Figure 6-4: Testing robot on concrete stick with cracks

The Figure 6-5 shows the condition of concrete samples after treatment with Basilisk ER7 and being maintained indoors with timely water spraying to maintain humidity. Bacteria gradually induced the formation of calcium carbonate, which adheres to the inner walls of the cracks or fills the entire gap of the cracks.

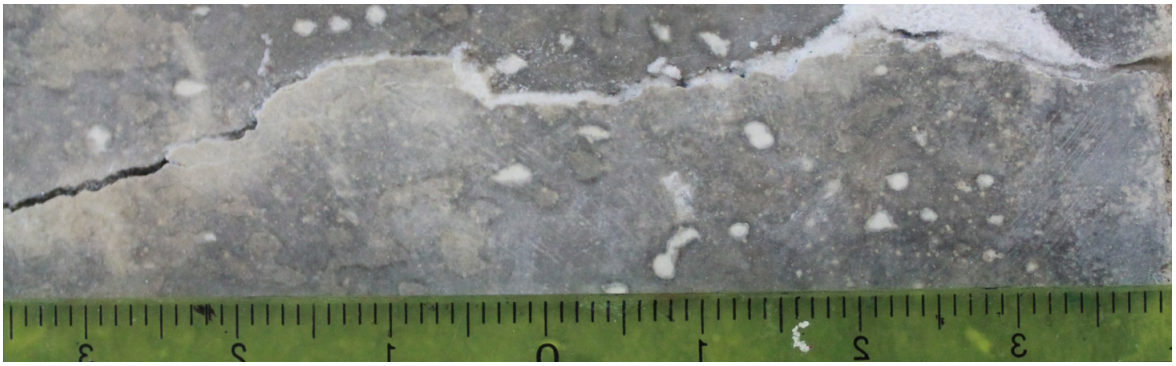


Figure 6-5: Calcium carbonate was induced under indoor maintenance

It was found in the test that if the solenoid water valve passage is too small or if the solenoid water valve is not cleaned in time, the solution of component A or the gel produced by contact of residual repair agent with solution of component B will block the solenoid water valve.

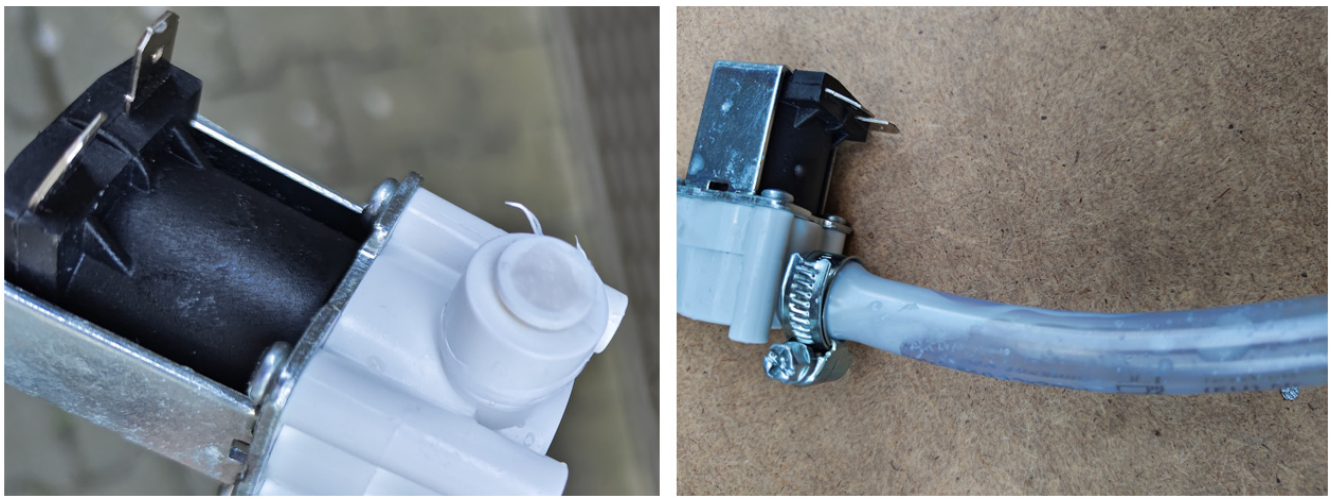


Figure 6-6: Outlet of solenoid water valve clogged by suspension of the solution of component A

During the above tests, the image quality of the Logitech C270 camera was also evaluated and found to be unsatisfactory. Therefore, for both indoor and outdoor tests, the Intel RealSense 3D camera will be used to capture images of the wider cracks.

6.1.2. Test Indoors

The indoor testing primarily aims to verify the robot's multi-point navigation based on Python scripts, its ability to read the coordinates of wider cracks and convert those coordinates, take photos of the wider cracks, upload the recorded data, and visualize the recorded wider cracks on the web interface. The indoor testing site was situated on the second floor of the Building L501 of Technical University of Darmstadt. The tiled floor does not exhibit natural concrete cracks; however, the best-trained YOLOv8s model mistakenly interprets the tile joints as cracks. During testing, this property will be utilized by printing cracks wider than the tile joints and adhering them to the floor to represent wider cracks, while the actual tile joints will be treated as narrow cracks.

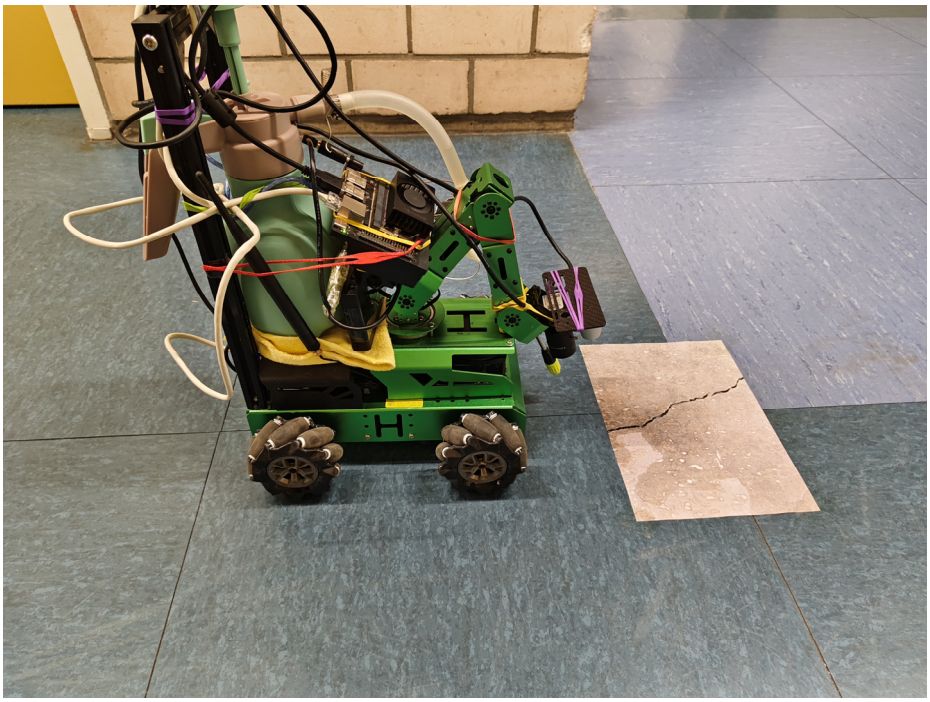


Figure 6-7: Printing crack and the tile joints represent as cracks for the robot

Figure 6-8 (a) illustrates the YOLOv8 segmentation result of the wide crack and the corresponding backend output, with the diameter of the wide crack being approximately 40 pixels. In Figure 6-8 (b), YOLOv8 misidentifies the tile joint as a crack, the diameter of the joint being approximately 20 pixels. For this test, the threshold between narrow and wide cracks was set to 30 pixels, ensuring that the robot only records the information of the wide crack.

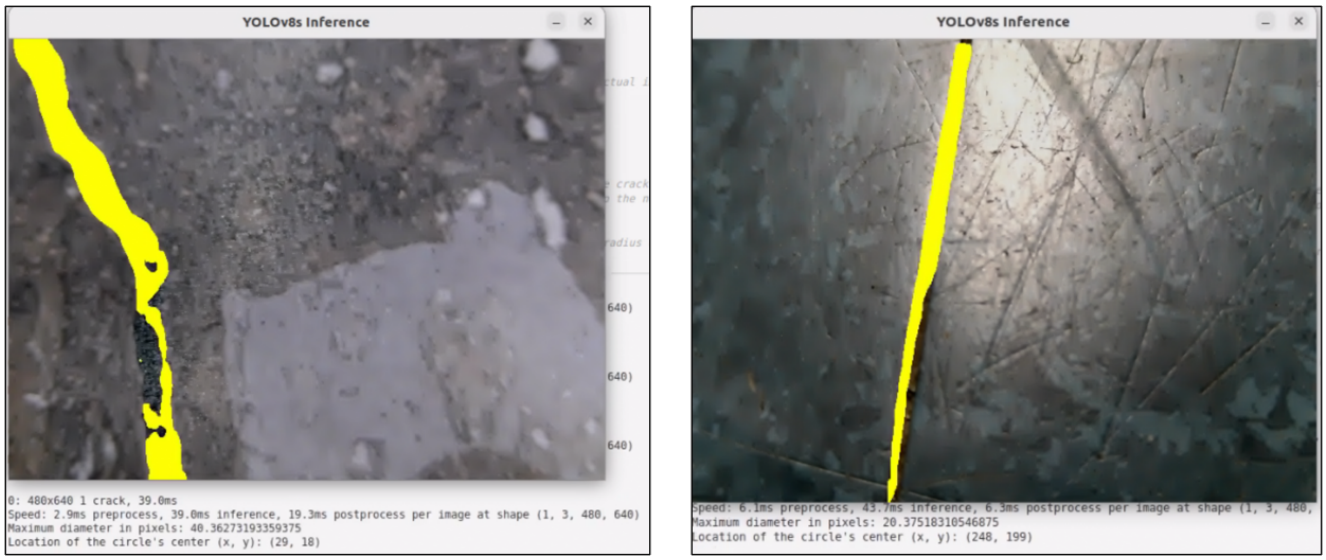


Figure 6-8: YOLOv8 segmentation result of (a) printing crack and (b) tile joint

For navigation, Figure 6-9 (a) illustrates seven target points defined within the map. The numbers visible in the figure were manually annotated post-screenshot, whereas the actual target points, defined by a Python script, are invisible on the map. Figure 6-9 (b) shows the real-time position of the robot in RViz along with the terminal output, where it records the coordinates and captures images when the robot passes over the paper with printing

wide crack. The LiDAR scan used to generate the map shows some discrepancies compared to the actual map, which is attributed to the limited precision of the LiDAR system.

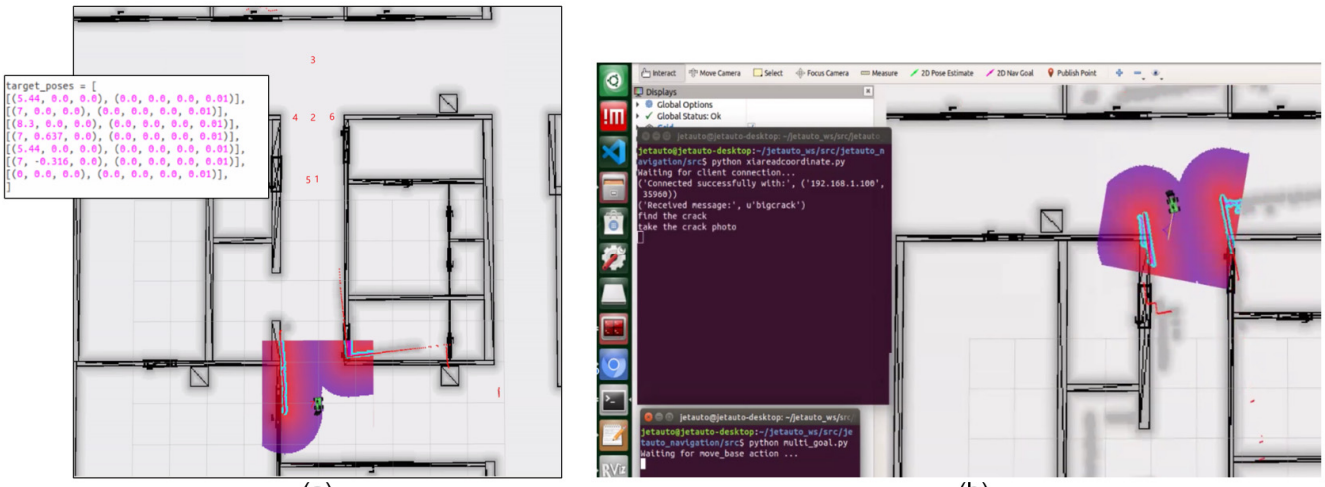


Figure 6-9: Setup and operation of multi-point navigation

The information of wide crack stored in MongoDB Atlas will be visualized in Cesium, where the red point represents the location of the wide crack. As shown in Figure 6-10, there is no significant deviation between the crack position displayed on Cesium and the actual crack position. This means that the robot has accurately captured the coordinates of the wider crack, and the coordinate transformation algorithm has successfully achieved the conversion between local Cartesian coordinates and WGS84 coordinates. The photo of the wider crack taken by the Intel Realsense 3D camera is clear, indicating that the Intel Realsense 3D camera with a frame rate of 30 FPS is suitable for capturing images of wide cracks when the robot moves at a speed of 0.2 m/s.

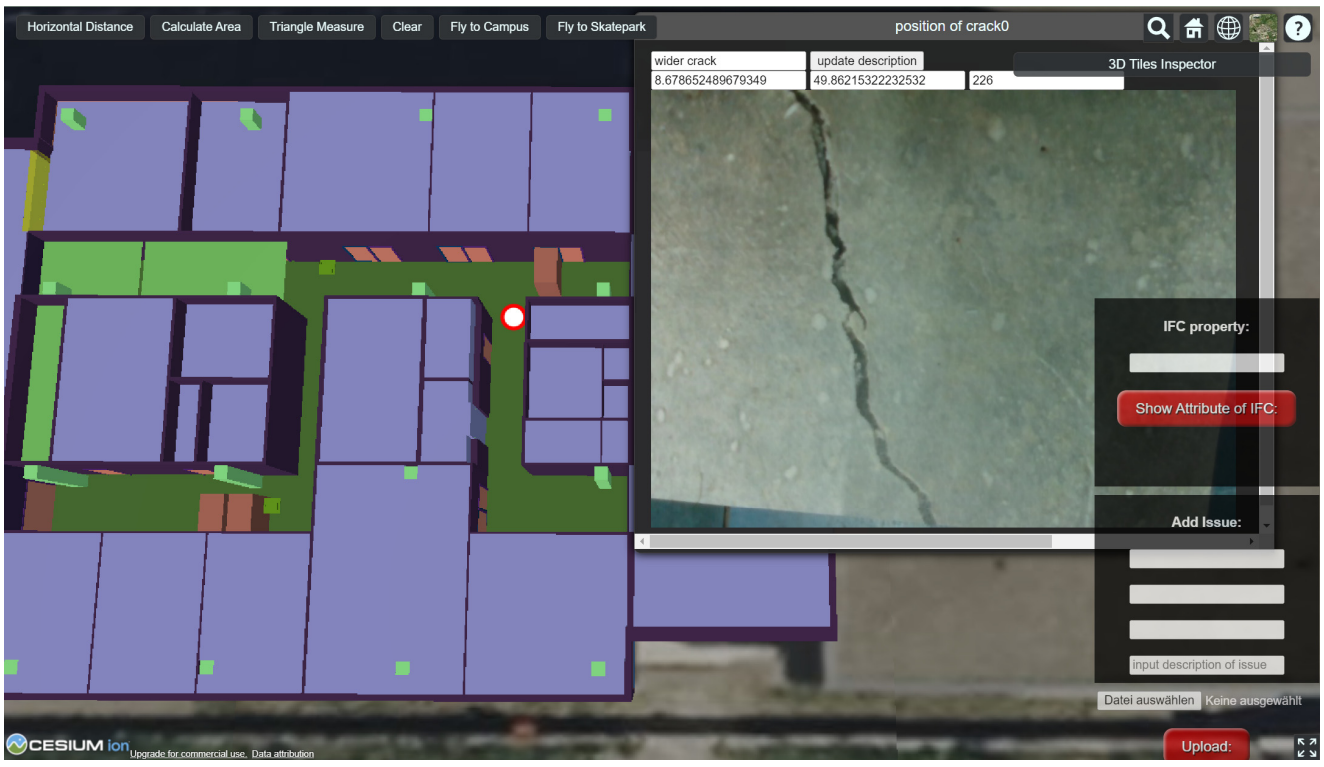


Figure 6-10: Display the information of recorded wide crack in Cesium

6.2. Application Example

After testing on concrete samples and in indoor environments, it was verified that all the robot's functions performed well as expected. Finally, it is necessary to test all the functions of the robot on a concrete surface with cracks to see whether these functions will conflict or can work in harmony to successfully complete the crack repair and recording task. The testing will take place at the Darmstadt Skate Park, where the concrete surface was specifically designed for skateboard enthusiasts. The liquid sprayed during the test was water, as the site is a public area where unauthorized crack repair is not allowed.

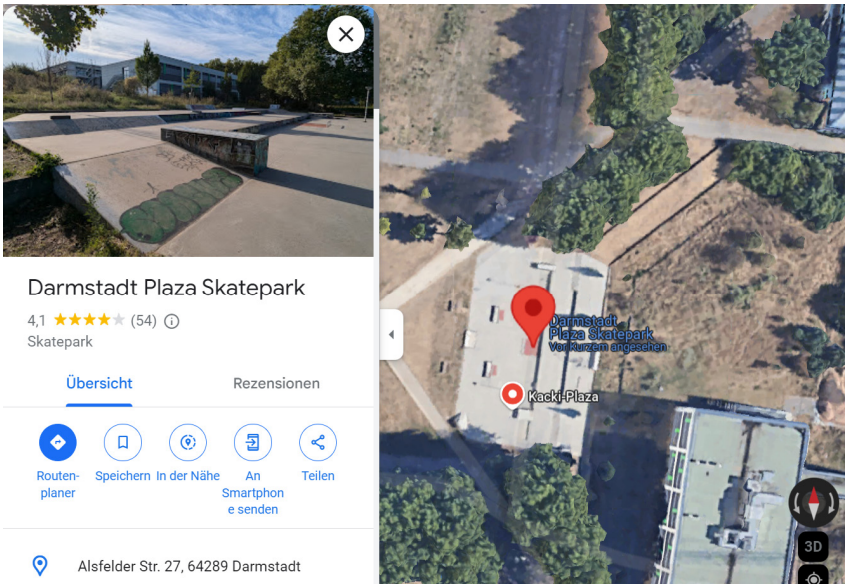


Figure 6-11: The location of Darmstadt skate park in Google map

Figure 6-12 (a) shows an aerial view of the skate park on Google Earth, but this view has several issues. First, the resolution is insufficient, and second, there are multiple tree shadows. Figure 6-12 (b) is an aerial view taken by a drone on a cloudy day, which is relatively clear and free of shadow issues. Therefore, the aerial view taken by the drone will be used to create the map for the robot.

Figure 6-12: Aerial view of the skate park from (a) Google Earth and (b) drone

A portion of the skate park was selected as the robot's testing area. Next, the aerial image of the testing area captured by the drone was converted into a PGM map using the free image editor GIMP, where the brick-paved surface and concrete box structures were considered impassable obstacles.

Figure 6-13: Creating map using drone: (a) is aerial image captured by drone and (b) is corresponding PGM map processed by image editor GIMP

Before operating the robot, it is necessary to determine the thresholds for narrow and wider cracks. As illustrated in the Figure 6-14, the first step involved using a measuring ruler to select a crack with a width of approximately 1 millimeter. Subsequently, the robot's crack segmentation and maximum width algorithms were employed to determine the corresponding pixel width for this crack. Based on the calculated results from the terminal, the pixel width of this 1mm wide crack corresponds to 14 pixels, which means the pixel width of a 0.6 mm wide crack is equivalent to 8.4 pixels. Since the segmented masks are often slightly larger than the actual size of the cracks, the threshold for narrow and wider cracks can be set to 9 pixels. In this case, a crack with a pixel width greater than 9 pixels is classified as a wider crack, while a crack with a pixel width less than 9 pixels is classified as a narrow crack.

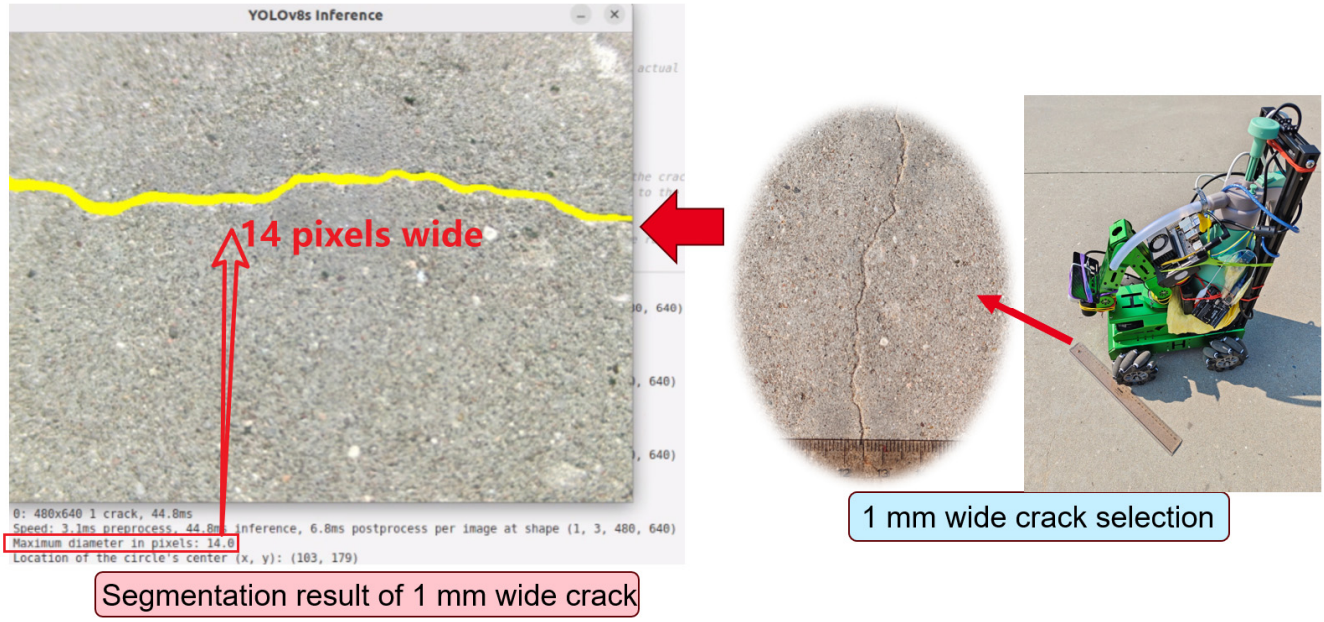


Figure 6-14: Determining the threshold using ruler for distinguishing between narrow and wide cracks

During the test, a series of target points were defined using a Python script, and the robot would sequentially navigate to each of these points. Upon detecting narrow cracks via its camera, the robot would activate the

spraying system. For wide cracks, it would record these wide cracks for further analysis. The seven target points in the figure below were manually labeled by the screenshot. From the records displayed in the terminal, it could be seen that the robot detected a wide crack when moving towards the first target point and performed the recording function accordingly. The recorded information of wide cracks would be stored in MongoDB Atlas.

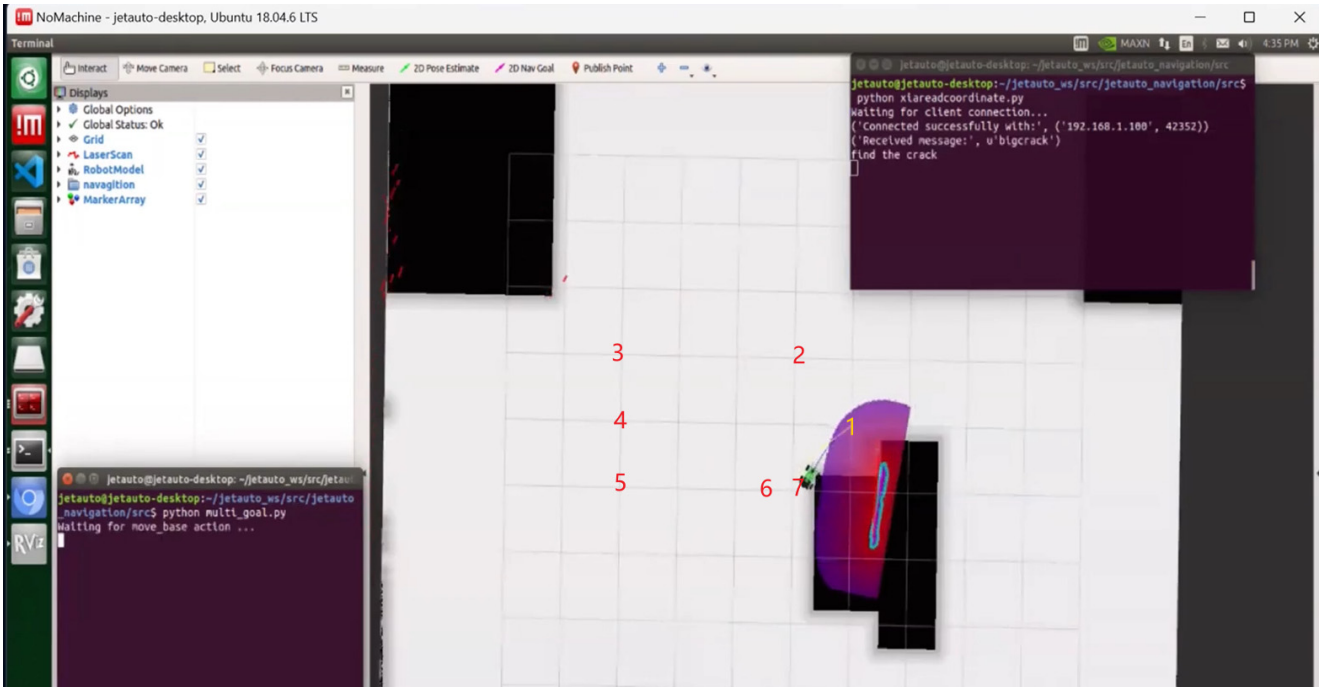
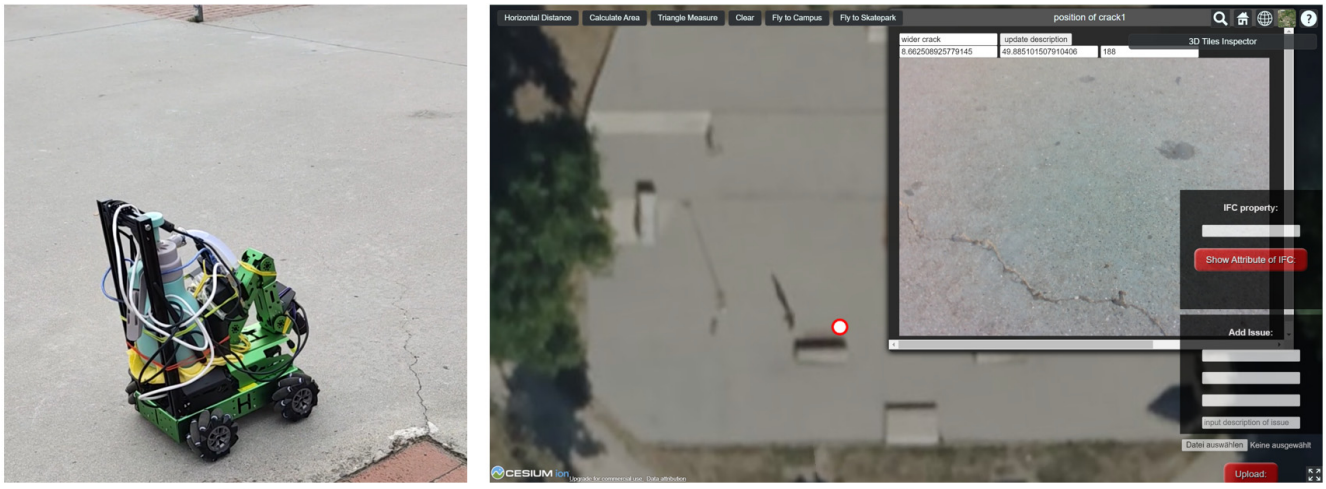


Figure 6-15: Visualizing robot operation on NoMachine

Figure 6-16 (a) shows the actual location of this wide crack on the testing area. Figure 6-16 (b) presents the visualization of the recorded wider crack in Cesium. The photo of the crack is relatively clear, but only a portion of it was captured. This suggests that using a 360-degree panoramic camera instead of Intel Realsense 3D camera for capturing the wider cracks would be more effective, allowing the entire crack to be captured in a single photo.



(a) (b)

Figure 6-16: The recorded wide crack on the testing area: (a) actual location and (b) visualization in Cesium

During the robot's autonomous driving process, when a narrow crack was detected, it would automatically spray water onto the crack.

Figure 6-17: Spraying water on narrow cracks upon detection

It was found during testing that the cracks often extended like tree roots, with some sections being wide and others being narrow.

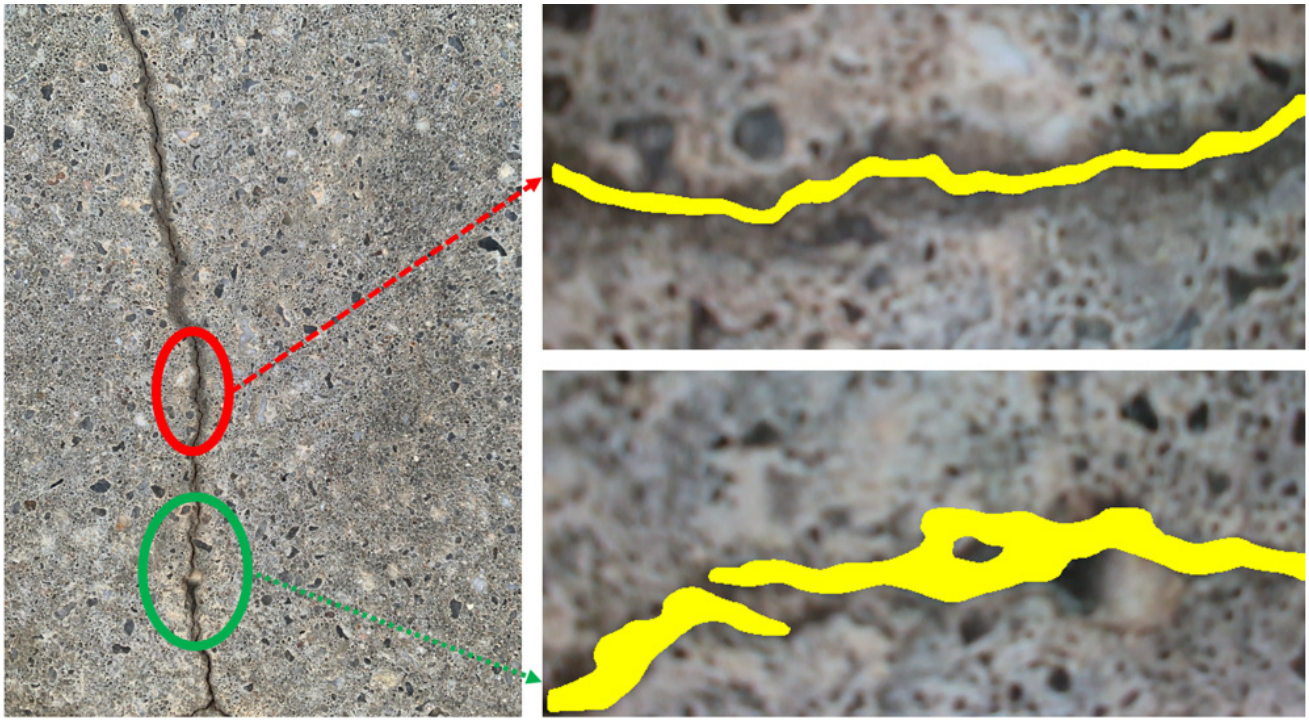


Figure 6-18: Segmentation of a crack with wide section (green region) and narrow section (red region)

Due to the limited field of view of the robot's camera, different sections of the long crack could activate different functions of robot. For example, when the robot passed through the narrow section, it would trigger the spraying system, while in the wider section, robot would record the crack. There are some potential solutions:

1. **Multi-Camera System**: Implement a system with multiple cameras that have overlapping fields of view. This can ensure comprehensive coverage of the area being inspected, reducing the chances of missing any part of the crack.
2. **Wide-Angle Lens**: Use a camera with a wide-angle lens to increase the field of view. This can help capture larger areas in a single frame, though it may slightly distort the image.

3. **Motorized Pan-Tilt Mechanism**: Equip the camera with a motorized pan-tilt mechanism that allows it to move and focus on different parts of the crack dynamically.
4. **Collaborative Robots**: If multiple robots are used, they can be programmed to work in tandem, each covering different areas and sharing data for a complete analysis.

During the robot testing process, it was also found that the best-trained YOLOv8s model identified black stains as cracks. When the features of black stains are very similar to those of cracks, it will identify the former as the latter. This suggests that the accuracy of the best-trained model requires further improvement.

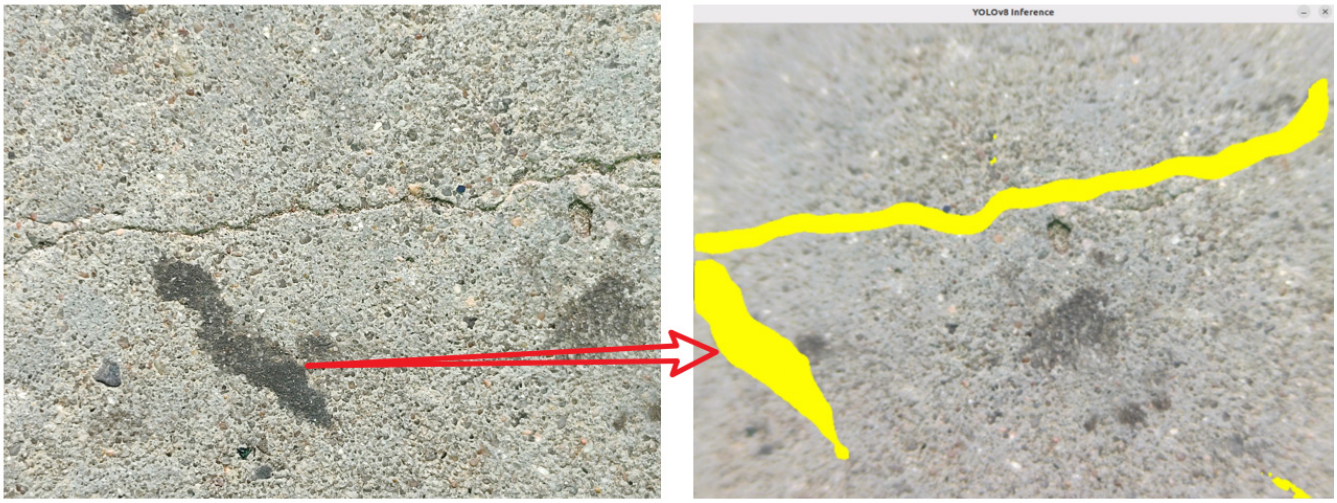


Figure 6-19: Misidentifying black stains as crack

7. Summary and Outlook

This chapter serves as a summary of the entire work and proposes future perspectives based on the achievements of the current work.

7.1. Conclusion

Narrow cracks in concrete structures are inevitable, conventional methods for narrow crack repair by large concrete structures are both time-consuming and labor-intensive, and the repair materials also have environmental impacts. Although the Basilisk ER7 repair agent offers the advantages of being environmentally friendly and easy to use, it is typically applied by spraying over the entire surface of the concrete being repaired, which results in significant waste of the agent. This work proposes the development of a compact, low-cost and AI-based robot to repair the narrow crack drying- and shrinkage cracking and wear on flat concrete surfaces using bio-concrete agent Basilisk ER7 and open-source robotic platform. The robot can detect narrow cracks on the concrete surface and only sprays repair agent onto those narrow cracks, thus conserving the use of the repair agent. The robot also has the ability to record wide cracks, and the recorded information will be visualized on a web application that integrates BIM and GIS, making it convenient for engineers to analyze and take appropriate actions later. The robot's control module was developed using a distributed architecture, with separate components handling distinct tasks: a compact computer Jetson Orin Nano with strong AI performance running machine learning models YOLOv8, an onboard computer Jetson Nano managing autonomous navigation and recording the information of wide cracks, and a microcontroller Arduino overseeing the spraying system with relay. This modular approach improves the robot's scalability and paves the way for future upgrades, including the deployment of VLMs, facilitating the development of embodied intelligence. In testing, these modules operated seamlessly in coordination. The robot's control is conducted via a remote computer using the NoMachine software, which allows the user to remotely access and operate the Jetson devices. The robot performs full-area scanning by navigating multiple target points defined via Python scripts. The maps used for this process are derived from IFC model and drone-captured aerial imagery.

To verify the reliability of the developed robotic system, the robot was tested both indoors and outdoors. According to the test results, the robot was able to identify narrow cracks on the concrete surface and sprayed liquid onto them. For wider cracks, the robot could record the coordinates and image information of these cracks, which could then be visualized on the web application. However, during the testing phase, it was observed that the best-trained crack segmentation model YOLOv8 mistakenly identified ground stains, tile joints, gap between concrete samples and wooden tray as cracks. This reveals a significant limitation of CNNs, namely its heavy reliance on local features making them sensitive to complex backgrounds and noise. The presence of noise or irrelevant details in the background can interfere with the feature extraction process and may lead to false detections. And if the training data doesn't adequately represent the variety of backgrounds and noise levels encountered in real-world scenarios, the CNN may exhibit challenges in generalizing. Potential solutions include introducing more training samples, especially those that are easily misidentified, which can help the model learn a broader range of features, thereby enhancing its generalization capabilities. Alternatively, employing more advanced algorithms could be beneficial. Consequently, in future plans, efforts will be made to employ the VLM

for crack identification tasks. By robot testing, it was also found that the frame rate and real-time imaging quality of cameras are crucial for ensuring that the ground conditions provided to machine learning model are up-to-date, accurate, and clear. High frame rates allow for smoother video capture, which is essential for detecting fast-moving objects. Meanwhile, high-quality imaging ensures that the details necessary for accurate analysis are captured, which is vital for the machine learning model to make reliable predictions and detections.

This work demonstrates that the intelligent robot can autonomously perform time-consuming, labor-intensive, and tedious tasks such as repairing narrow concrete cracks and recording wider concrete cracks, thereby saving labor costs on construction sites. Although the various challenges faced in robot development, such as ensuring hardware-software compatibility and addressing interoperability between different software platforms, while also requiring multiple rounds of testing to optimize performance, once completed, the AI-based robot could effectively serve as a reliable workforce capable of carrying out specific engineering tasks.

7.2. Future Work

For higher buildings' exterior walls, crack inspection is also very meaningful because these building structures will also have crack issues, and manual inspection is time-consuming and labor-intensive. In future work, the crack inspection and repair methods mentioned above will be deployed on drones. Nowadays, multimodal AI is becoming increasingly popular, and with the support of multimodal models, robots are becoming more intelligent. Multimodal models leverage information from multiple data types. This allows for a richer and more comprehensive feature representation compared to CNNs. With the integration of language models, multimodal models can perform complex reasoning tasks that are difficult for CNNs. Thus, another future plan is to deploy multimodal models on the robot to make the robot smarter while being able to perform more complex tasks.

7.2.1. Deployment on Drone

Cracks in the exterior walls of high-rise buildings can be difficult to detect early on, especially if they are not easily visible or if regular inspections are not conducted. As these cracks widen, they can compromise the structural integrity of the building or bridge, leading to increased risk of failure and a significant rise in repair costs. Early detection and repair are crucial to prevent further damage and to maintain safety and structural stability. Advanced monitoring systems and regular inspections can help in timely detection and management of such issues. Engaging in high-altitude work is inherently hazardous and can lead to severe accidents. Such incidents not only result in substantial economic damages but also lead to numerous casualties. Consequently, the deployment of wall-climbing robots to substitute for manual high-altitude tasks is imperative. These robots, which take inspiration from the climbing mechanisms of insects and animals, are intelligent, biomimetic devices. They are capable of carrying specialized equipment to perform operations on walls and execute hazardous tasks more efficiently than firefighters or inspection personnel, thereby enhancing safety and operational effectiveness (Fang et al., 2022). While the negative pressure wall-climbing robot, which utilizes suction cups to adhere to vertical surfaces, is adept at navigating walls, glass, and other smooth surfaces of buildings, it may not offer the same level of convenience as drones. Drones provide aerial mobility and the ability to access areas that may be difficult or impossible for ground-based robots, such as inspecting the tops of tall buildings or reaching areas affected by natural disasters. Drones can quickly and easily access various parts of a structure without the need for physical

contact. This makes them ideal for inspecting areas that are difficult or dangerous to reach. And drones can cover large areas in a relatively short amount of time. This rapid assessment capability allows for quicker identification of potential issues, which is crucial for the timely maintenance and safety of the structures.

The DJI Matrice 350 RTK is an impressive enterprise drone for the industry (DJI, 2024). It features an all-new video transmission system. It utilizes the DJI O3 Enterprise Transmission system, which supports triple-channel 1080p live feeds and boasts a maximum transmission distance of 20 kilometers. This robust system ensures reliable communication between the drone and the remote controller. Its max. Load capacity is 2.7 kg. It has good flight performance with a maximum airspeed of 23 m/s, maximum Wind resistance of 15 m/s and maximum Flight time of 55 min. However, the software and hardware of DJI drones are not open source. M350 RTK supports Payload SDK, Mobile SDK, and Cloud API. It has E-Port. The Payload SDK supports E-Port, which greatly reduce the payload development lifecycle and maximize the potential of payloads in more diverse scenarios. E-Port supports Payload SDK V3 and later versions and is compatible with third-party Payload SDK payloads of the Matrice 30 Series. This means that machine learning models can be deployed on the Jetson Orin Nano, and the results of real-time inference of the machine learning models can be used to control the M350 RTK through the E-Port and Payload SDK. As shown in Figure 7-1, in the future plan, the drone will carry a lithium battery to power the Jetson Orin Nano, and a zoom-capable USB high-definition camera provides high-quality video input to Jetson Orin Nano for crack detection. The drone will also carry small canisters of repair agent for crack repair. The M350 RTK is equipped with a functionality that allows it to follow a predetermined flight path. During its operation, upon detection of cracks by the Jetson Orin Nano, the M350 RTK will receive commands from Jetson Orin Nano to hover in place. This hovering capability facilitates the precise application of repair agents and detailed inspection of the detected cracks.

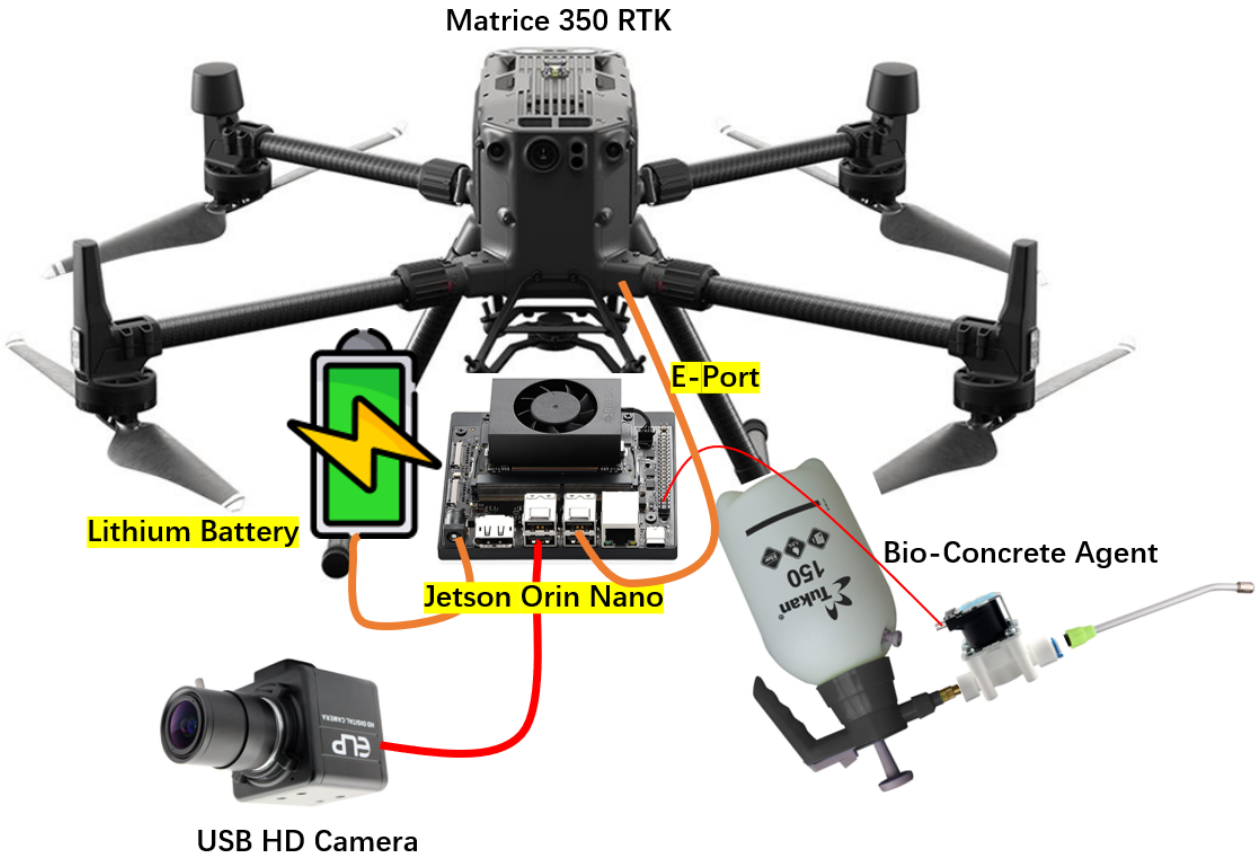


Figure 7-1: Schematic diagram about the integration with Jetson Orin Nano in Matrice 350 RTK drone

While the performance of this DJI drone is impressive, its price is quite high. Therefore, an alternative option under consideration is an open-source drone. The P600 Advanced Version Drone is based on the P600 basic model drone, equipped with different types of onboard computers (sunrise, Jetson Orin Nano) for computational expansion, supporting more sensors and advanced technical functions. The P600 Advanced Version Drone is a professional research drone platform based on the PX4 open-source flight controller and can be paired with high-precision RTK positioning modules, efficient data transmission links, and multiple types of onboard computers. It is suitable for application development in the drone industry and algorithm verification of drones in outdoor environments. The P600 drone's onboard system is based on the autonomous drone software platform Prometheus, providing control code frameworks, communication code frameworks, and corresponding secondary development interfaces. Users can control the drone through the Prometheus ground station, implement control modes set by Prometheus, and quickly switch from Prometheus's simulation demo to actual flight, enhancing research efficiency from simulation to real-world application. The maximum payload of the P600 drone is 0.6 kg (AMOVLAB, 2024). Since this is an open-source drone, subsequent enhancements can be made by replacing with motors and propellers that provide higher lift, to increase load capacity. The flight time is 36 minutes when fully loaded.

Figure 7-2: P600 Advanced Version Drone (AMOVLAB, 2023)

7.2.2. Integrating Multimodal AI Model

The rise of multimodal AI like OpenAI's GPT series has ushered in a revolution where AI can simultaneously comprehend various modalities, such as text, image, speech, facial expressions, physiological gestures, etc. These models are trained on diverse datasets that cover a broad spectrum of knowledge, allowing them to generalize across tasks. LLMs are evolving rapidly, with new versions and models frequently being released.

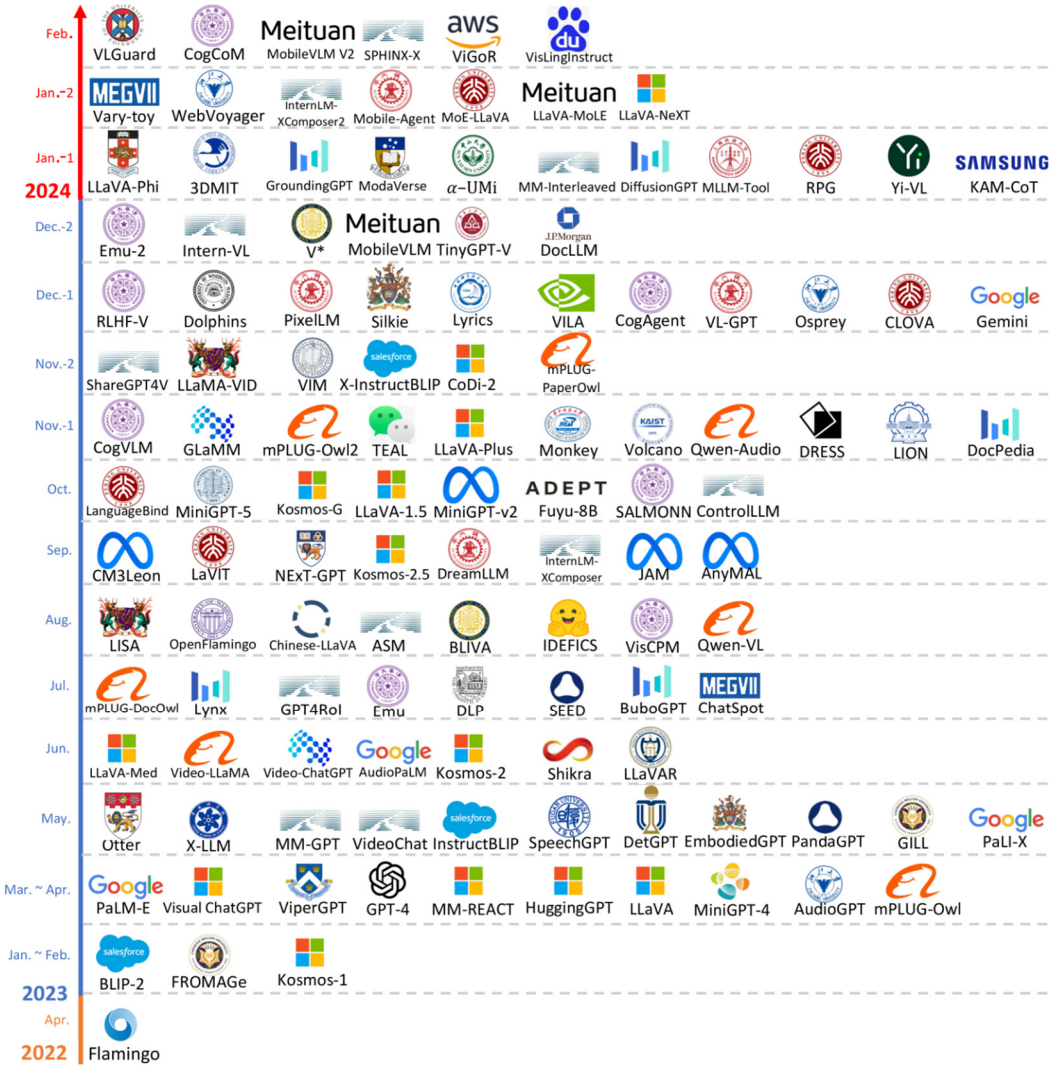


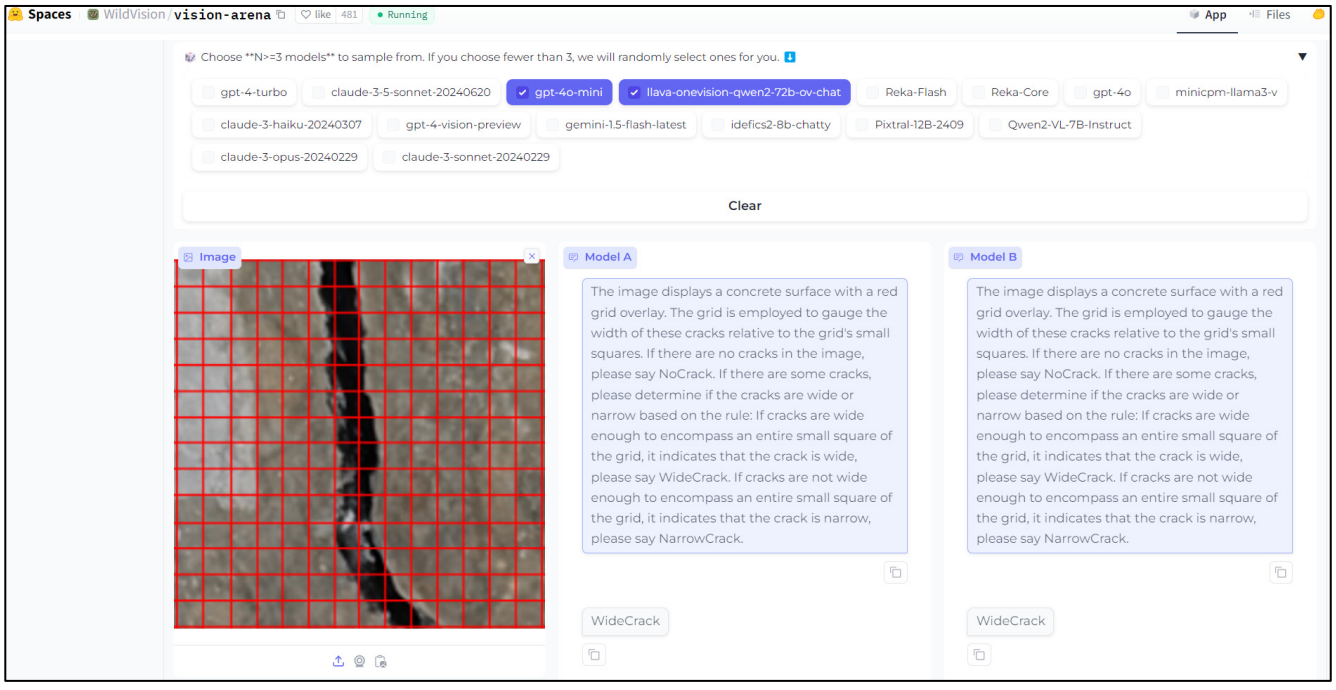
Figure 7-3: The timeline of MM-LLMs from 2022 to Feb. 2024 (Zhang et al., 2024)

One exciting application of multimodal AI is VLM. These models are trained on diverse datasets that allow them to comprehend various aspects of visual data, providing a comprehensive understanding that can be applied to a wide range of tasks such as describing the scene depicted in images and object detection. The YOLO model used in this work for identifying and segmenting cracks does not understand the scene; it merely classifies objects as cracks or non-cracks based on their features. To distinguish between narrow and wide cracks, the crack segmentation mask must be extracted for quantitative calculation, which is a rather cumbersome process. In addition, CNNs are highly effective at capturing local features in images. However, this reliance on local features can make CNNs more sensitive to complex backgrounds and noise. Complex backgrounds may introduce patterns that resemble the target features, potentially confusing the network. Similarly, noise can disrupt the local patterns that CNNs rely on, leading to decreased performance. Compared to CNNs, VLMs have several advantages. For example, while CNNs focus primarily on local features through convolutional filters, VLMs can capture global context using architectures like Transformers. This allows them to understand the overall scene and relationships between objects, not just individual features. In future work, VLMs' ability to understand the scenes in image will be utilized to distinguish between narrow and wide cracks. The principle involves adding a grid of squares with a certain width as a reference to the images of cracks captured by the camera. To achieve this, a grid made of thin lines could be hung close to the ground below the camera, or a grid could be drawn on the captured images through

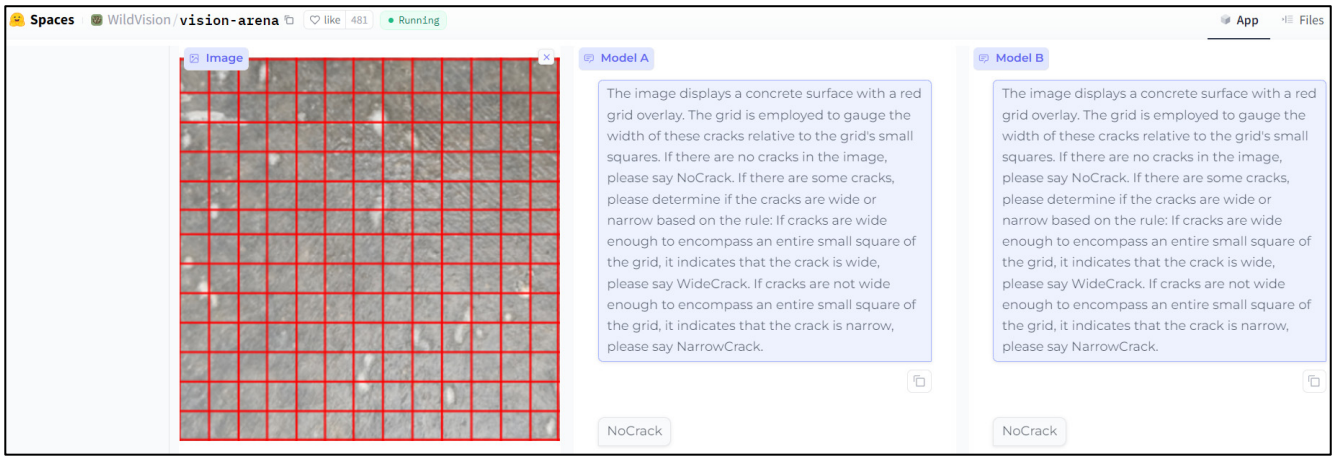
an algorithm, and the images with the added grid could then be sent to a VLM. The width of these squares serves as the threshold to distinguish between narrow and wider cracks. If there are visible cracks present and the cracks are wide enough to encompass an entire small square of the grid, it indicates that the crack is wide; otherwise, it's a narrow crack. After correctly understanding the scene, the VLM outputs three keywords such as NoCrack, NarrowCrack, and WideCrack. Subsequently, the spraying system will be activated based on the keyword response.

The WildVision Team has built a platform on Hugging Face that integrates several VLMs to compare their image analysis capabilities with each other (huggingface, 2024). The following is a brief test conducted on this platform. The prompt for interacting with the models is like this: "*The image displays a concrete surface with a red grid overlay. The grid is employed to gauge the width of these cracks relative to the grid's small squares. If there are no cracks in the image, please say NoCrack. If there are some cracks, please determine if the cracks are wide or narrow based on the rule: If cracks are wide enough to encompass an entire small square of the grid, it indicates that the crack is wide, please say WideCrack. If cracks are not wide enough to encompass an entire small square of the grid, it indicates that the crack is narrow, please say NarrowCrack*".

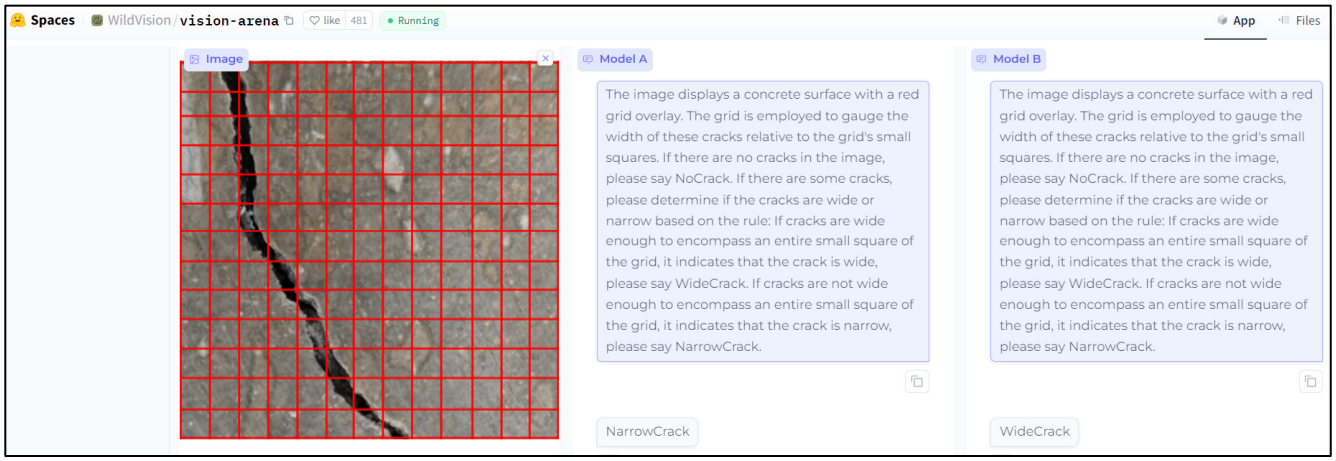
As shown in Figure 7-4 (a), user can select two models, upload an image, and send prompt to these two models. The Model A is gpt-4o-mini, and Model B is llava-onevision-qwen2-72b-ov-chat model. According to the results in Figure 7-4 (a), Figure 7-4 (b) and Figure 7-4 (c), gpt-4o-mini provided accurate answers for scenarios represented by three types of images: wide cracks, no cracks, and narrow cracks. However, the llava-onevision-qwen2-72b-ov-chat model misidentified narrow cracks as wide cracks. The test results demonstrate that the VLM can successfully interpret images of concrete surfaces, accurately identifying whether a crack exists. Additionally, it can use the red grid as a reference to distinguish between narrow and wide cracks. Nevertheless, certain VLMs still have limited capacity for image analysis, which makes them prone to errors. Therefore, when deploying VLMs on robot, selecting the appropriate model is crucial for ensuring optimal performance. And to effectively enhance the accuracy and generalization of VLMs on specific scenarios, several strategies can be employed, such as fine-tuning the model with task-specific dataset of the target scenario.



(a)



(b)



(c)

Figure 7-4: Analyzing crack images through gpt-4o-mini (Model A) and llava-onevision-qwen2-72b-ov-chat model (Model B)

NVIDIA Jetson AI Lab presented a case where the LLaVA-3b model was deployed on a Jetson Orin device to enable real-time video analysis (NVIDIA, 2024d). As illustrated in the Figure 7-5, in this use case, users can continuously supply prompt "*What changes occurred in the video"* to the model through a Python3 script, and the model will keep responding to the queries.

```
jetson-containers run $(autotag nano_llm) \
  python3 -m nano_llm.vision.video \
    --model Efficient-Large-Model/VILA1.5-3b \
    --max-images 8 \
    --max-new-tokens 48 \
    --video-input /data/my_video.mp4 \
    --video-output /data/my_output.mp4 \
    --prompt 'What changes occurred in the video?'
```

Figure 7-5: Prompt and inference result of live LLaVA-3b model running on Jetson Orin device (NVIDIA, 2024d)

Multimodal AI models are also capable of understanding user needs through various inputs, which can then be used to automatically control robots. These models can process and integrate information from different modalities to enhance the performance and accuracy of tasks. They are particularly useful in scenarios where robots need to interact with humans in a more natural and intuitive way. To facilitate the autonomous navigation of the CrackRepairBot, this work utilized Python scripts to set multiple target points for the robot. Additionally, to activate certain functions of the robot, commands must be entered in the terminal every time, leading to a poor human-computer interaction experience. Moreover, some cracks extend to surrounding areas like tree roots, but the cameras and nozzles of the robot developed in this work are fixed at the front of the robot body. Therefore, in order to handle these surrounding areas, the robot needs to patrol these places to detect and deal with the cracks. To overcome these challenges, future work aims to integrate multimodal AI models and employ a robotic arm to control the movements of both the camera and the spray nozzle.

As shown in Figure 7-6, a chat interface with multimodal AI model will be integrated into the web application, enabling users to directly communicate their requirements to the model for more efficient interaction. The multimodal AI model could be deployed on a computer connected to the robot's onboard system, enabling it to indirectly control both the robot's movement and the motion of the robotic arm. In terms of navigation, users can directly select an area on the map presented by Cesium and use the dialog box to instruct a robot to patrol the selected area. The multimodal model could be capable of understanding commands as well as interpreting map information, subsequently autonomously executing the user's instructions. The camera and nozzle would be mounted on a robotic arm. When the robot detects a crack on the ground, it will pause for a moment. During this time, the robotic arm will move left and right to inspect the surrounding area for any additional cracks. If it finds any, it will process them immediately. If not, the robotic arm will return to its original position, and the robot will continue on its path.

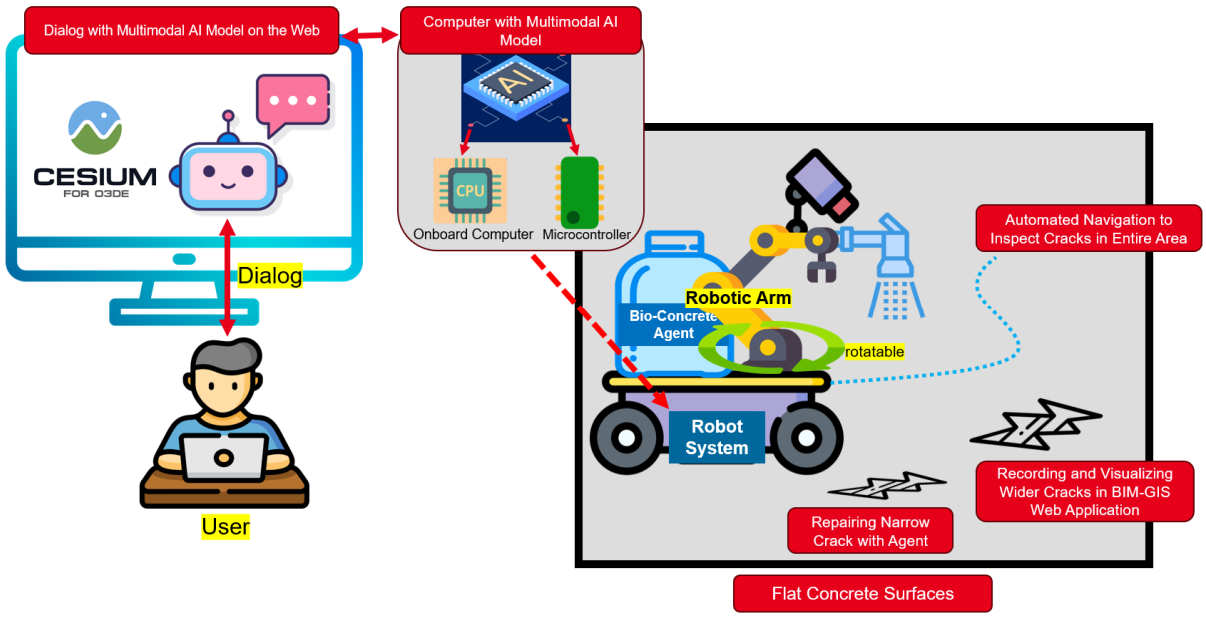


Figure 7-6: Schematic illustrating the principle of controlling the robot for crack repair using a multimodal model

In civil engineering, robots equipped with multimodal models and robotic arms have a wide range of applications beyond crack repair. These include detecting and repairing steel corrosion, patrolling for fire inspection and extinguishing, and even performing simple transportation tasks on construction sites. The potential for development of multimodal intelligent robots is enormous.

List of Abbreviations

AI	Artificial Intelligence
ARM	Advanced RISC Machines
AMCL	Adaptive Monte Carlo Localization
API	Application Programming Interface
ALU	Arithmetic Logic Units
ACI	Airports Council International
AECO	Architecture, Engineering, Construction, and Operations
AC	Alternating Current
AP	Average Precision
BIM	Building Information Modeling
BPU	Branch Processing Unit
CSI	Camera Serial Interface
COCO	Common Objects in Context
CO_2	Carbon Dioxide
$CaCO_3$	Calcium Carbonate
CNN	Convolutional Neural Network
CPU	Central Processing Unit
CUDA	Compute Unified Device Architecture
$CaCl_2$	Calcium Chloride
COM	Common Contact
DRAM	Dynamic Random Access Memory
DCNN	Deep Convolutional Neural Network
DC	Direct Current
DSRM	Design Science Research Methodology
FPS	Frames Per Second
GIS	Geographic Information System
GPU	Graphics Processing Unit
GFLOPS	Floating Point Operations Per Second
GPS	Global Positioning System
GND	Ground
GUI	Graphical User Interface
GPR	Ground Penetrating Radar
GLONASS	Global Navigation Satellite System (Russia)
GNSS	Global Navigation Satellite Systems
GPIO	General-purpose input/output
glTF	Graphics Language Transmission Format
HTML	HyperText Markup Language
IOU	Intersection over Union
IoT	Internet of Things
IDE	Integrated Development Environment
IP	Internet Protocol Address
IPv4	Internet Protocol version 4
ICSP	In Circuit Serial Programming
IFC	Industry Foundation Classes
LiDAR	Light Detection and Ranging
IMU	Inertial Measurement Unit
IMUs	Inertial Measurement Units
LAN	Local Area Network
LLM	Large Language Model
LLMs	Large Language Models
MICP	Microbiologically Induced Calcium Carbonate Precipitation

mAP	Mean Average Precision
MEP	Mechanical, Electrical, and Plumbing
MB	Mbytes
NPU	Neural Network Processing Unit
NO	Normally Open
NC	Normally Closed
OGC	Open Geospatial Consortium
ONNX	Open Neural Network Exchange
ORB	Oriented FAST and Rotated BRIEF
pH	Potential of hydrogen
PGM	Portable Gray Map
PWM	Pulse Width Modulation
QZSS	Quasi Zenith Satellite System
RAM	Random Access Memory
RViz	Reconstruct Visualization Interface
ROS	Robot Operating System
RTK	Real-Time Kinematic
RFID	Radio Frequency Identification
SDK	Software Development Kit
SLAM	Simultaneous Localization and Mapping
SIFT	Scale Invariant Feature Transform
TCP	Transmission Control Protocol
TOPS	Tera Operations Per Second
TPU	Tensor Processing Unit
TensorRT	Turing Tensor R-Engine
ToF	Time of Flight
UDP	User Datagram Protocol
URI	Uniform Resource Identifier
USB	Universal Serial Bus
VOC	Volatile Organic Compound
VLA	Vision Language Action
VLM	Vision Language Model
VLMs	Vision Language Models
VCC	Voltage Collector Collector
WGS84	World Geodetic System 1984
WiFi	Wireless Fidelity
4G	Fourth Generation

List of Tables

Table 1: Byproducts and its effect of different metabolic pathways involved in MICP (Jain et al., 2021).....12
Table 2: Performance Metrics of YOLOv8 models trained on COCO (Ultralytics, 2024e) 56

List of Figures

Figure 1-1: Corrosion of steel reinforcement cracks (Happho, 2024).....1
Figure 1-2: System overview of the robot and the web application for crack repair and recording 3

Figure 1-3: Steps of the DSRM process model (left) in the structure of the work (right)....4
Figure 2-1: Causes of concrete cracking according to ACI committee 224 (Poursaee and Ross, 2022) 5
Figure 2-2: Steps to repair cracks by epoxy injection (Strong-Tie Company, 2024)....7
Figure 2-3: Steps to repair cracks by polyurethane injection (SealBoss, 2024)....7
Figure 2-4: Steps to repair cracks by routing and sealing (Goodwin, 2009)....8
Figure 2-5: Steps to repair a crack wide than 0.64 cm in stair with a vinyl concrete patcher (wikiHow, 2023)....8
Figure 2-6: Steps to repair cracks by dry pack (Jovi, 2023)....9
Figure 2-7: MICP mechanism diagram at a single-cell level (Sun et al., 2021)....10
Figure 2-8: Accumulation of calcium carbonate at the multicellular level (Choi et al., 2017) 10
Figure 2-9: Visual evaluation of crack repair (Van Tittelboom et al., 2010) 11
Figure 2-10: Schematic drawing of healing process of concrete cubes with narrow and wide cracks through the MICP treatment (Zhang et al., 2023)....11
Figure 2-11: Remediation of mortar crack with a width of 0.4 mm (Jongvivatsakul et al., 2019)....12
Figure 2-12: Stereomicroscopic images of crack-healing process in control mortar specimen (Wiktor and Jonkers, 2011)....13
Figure 2-13: Microscopic image of: (a) crack remediated area, (b) enlarged portion of crack remediated area showing calcite precipitation, and (c) Electron micrograph showing rod shaped bacteria embedded in crack remediated area (Achal et al., 2013)....13
Figure 2-14: Microscopic images of control (A), PC (B), MC-Ao (C) and MC-Aa (D) crack healing processes (Zhang et al., 2019) 14
Figure 2-15: Component A and B of Basilisk ER7 15
Figure 2-16: Schematic diagram of Basilisk ER7 for crack sealing (Basilisk, 2021a)....15
Figure 2-17: Application examples of Basilisk ER7 (Basilisk, 2021b)....16
Figure 2-18: Crack repairs at Milan-Malpensa MXP International Airport using polyurethane injection (Glewwe, 2017) 17
Figure 2-19: Overall features of the planned robot 18
Figure 3-1: Using a crack width ruler for approximating crack sizes in visual inspections (OSNDT, 2024)....19
Figure 3-2: Comparison among computer vision tasks (SuperAnnotate, 2023)....20
Figure 3-3: A comparison between a human neuron and a neural network neuron (Roffo, 2017) 20
Figure 3-4: Illustration of the correspondence between the regions associated with the primary visual cortex and the layers in CNN (Roffo, 2017) 21
Figure 3-5: A typical CNN architecture (Sahai, 2024)....21
Figure 3-6: Crack recognition using CNN: (a) Input image; (b) classification: image patches classified as crack or noncrack; (c) object detection: bounding boxes generated around areas that contain cracks; and (d) segmentation: pixels classified as crack or noncrack (Hsieh and Tsai, 2020)....22
Figure 3-7: Traditional machine learning vs Transfer learning (Martinez, 2020)....23
Figure 3-8: Examples of model underfitting, model fitting good, and model overfitting (SHRIVASTAVA, 2020) 24
Figure 3-9: Calculation of Precision, Recall and Accuracy in the confusion matrix (MA et al., 2019)....25
Figure 3-10: IoU: Measuring Overlap Between Ground Truth and Prediction Segments (Kukil, 2022)....26
Figure 3-11: Edge Computing hierarchy (Bhandari, 2023)....26

Figure 3-12: Jetson modules and AI performance (NVIDIA, 2024a) 27
Figure 3-13: Comparison of CPU and GPU architectures (Paz-Gallardo and Plaza, 2011) 28
Figure 3-14: ONNX: Flexible framework for model deployment and compatibility in machine learning (Microsoft, 2024) 29
Figure 3-15: TensorRT: A programmable inference accelerator (NVIDIA, 2018) 30
Figure 3-16: Comparison chart of the inference time per image by model size and format (Ultralytics, 2024b) 31
Figure 3-17: Arduino family 31
Figure 3-18: Raspberry Pi 5 (left) and Yahboom Raspberry Pi 5 robot (right) 32
Figure 3-19: Comparison chart of inference time per image for YOLOv8n (Ultralytics, 2024c) 33
Figure 3-20: Horizon Sunrise X3 and Ultra Board (Yahboom, 2024) 33
Figure 3-21: Orange Pi AIPro (left) and Orange Pi 5 Plus 8GB (right) 34
Figure 3-22: Schematic of ROS communication mechanism (RS et al., 2023) 35
Figure 3-23: Visualizing the robot model and laser data in RViz 37
Figure 3-24: GUI of RViz plugin Navi_waypoints (Zou, 2024) 38
Figure 3-25: SLAM processing flow (MathWorks, 2024) 38
Figure 3-26: System overview of the vision-based adaptive robotics for autonomous surface crack repair (Genova et al., 2024) 41
Figure 3-27: Automated navigation and data collection in the indoor and outdoor environment (Khan et al., 2023) 42
Figure 3-28: Self-reconfigurable pavement inspection and cleaning robot Panthera (Ramalingam et al., 2021) 42
Figure 3-29: Sensory, electrical, programming and control units in Panthera (Ramalingam et al., 2021) 43
Figure 3-30: Independent trial for the mobile mapping system and the defect identification: (a) Defect location on global map and (b,c) detected defects and patches (Ramalingam et al., 2021) 43
Figure 3-31: Sensor configuration of robot system (Gui and Li, 2020) 44
Figure 3-32: (a) LLMs generate code, which interacts with VLMs, to produce a sequence of 3D affordance maps and constraint maps grounded in the observation space of the robot. (b) The composed value maps then serve as objective functions for motion planners to synthesize trajectories for robot manipulation (Huang et al., 2023) 44
Figure 3-33: Vision-language model is trained with Internet-scale vision-language datasets. During inference, the text tokens are de-tokenized into robot actions, enabling closed loop control (Chebotar and Yu, 2023) 45
Figure 3-34: An example about BIM-GIS Integration with 3D Tiles converted from BIM model 46
Figure 3-35: The new Design Tiler and Revit Add-In improve workflows and capabilities that place AECO content in a 3D geospatial context (Braig, 2024) 47
Figure 3-36: Sample IFC data including structure and MEP as 3D Tiles (Braig, 2024) 47
Figure 4-1: Two application methods of Basilisk ER7: (a) jet spray and (b) complete surface treatment (Basilisk, 2021g) 48
Figure 4-2: The appearance of the crack after spaying the solution of component A 49
Figure 4-3: Gel formation due to the application of component B: (a) the immediate effect after application and (b) the condition observed one day after the application 49
Figure 4-4: Timeline of application Basilisk ER7 (Basilisk, 2022) 50
Figure 4-5: Simplified presentation of overall concept of the robot 51

Figure 4-6: The principle of converting machine learning results into robot control commands 52
Figure 4-7: The connection schematic of the distributed system 54
Figure 4-8: Comparisons of YOLO networks in terms of latency-accuracy (left) and size-accuracy (right) trade-of (A. Wang et al., 2024) 55
Figure 4-9: YOLO supports a full range of vision AI tasks (Jocher et al., 2023) 55
Figure 4-10: Two strategies for capturing narrow cracks on the ground with fixed-focus and zoomable camera57
Figure 4-11: Image processing: (a) crack segmentation result and (b) applying orthogonal skeleton line method (SubChange, 2022) 58
Figure 4-12: The calculation results of the inscribed circle algorithm (Jiang, 2022) 58
Figure 4-13: The logic of the system operation for cracks 60
Figure 4-14: A single channel 5V relay module (Microcontrollers Lab, 2021) 60
Figure 4-15: The connection schematic of the entire sprinkler system. 61
Figure 4-16: Adjustable nozzle and sprayer pole (Cozi Life, 2024) 61
Figure 4-17: Schematic diagram about spraying system constructed with a water pump 62
Figure 4-18: Schematic diagram about spraying system built by combining a water valve with a pressurizable spray bottle 63
Figure 4-19: Normally closed valve: (a) unpowered and (b) powered states (BOLA, 2024) 63
Figure 4-20: Solutions for checking the flow rate 64
Figure 4-21: Methods to create PGM map for robot 65
Figure 4-22: Different algorithms executing in cyclic order in the robot's cleaning cycle (Hasan et al., 2014)...66
Figure 4-23: Automated navigation of robot AMSEL: (a) Algorithm for automated navigation and inspection and (b) the diagram of the survey area for the robot (Khan et al., 2023) 67
Figure 4-24: Schematic of the robot path planning on the airport runway (Gui and Li, 2020) 68
Figure 4-25: Schematic of the robot path planning for cracks inspection via multi-point navigation method 68
Figure 4-26: System overview of wide cracks recording and visualizing 69
Figure 4-27: Yahboom GNSS module (left) and the coordinate data acquired by this GNSS module on the Jetson Nano terminal (right) 71
Figure 4-28: Schematic diagram of the crack localization principle by the robot 71
Figure 5-1: Key Components and size of JetAuto Pro (Hiwonder, 2024) 72
Figure 5-2: Schematic of mounting the pressurizable water bottle on robot 73
Figure 5-3: The size of solenoid water valve used in this work (AliExpress, 2024) 73
Figure 5-4: Plastic battery storage box and 8 x 1.5V dry cell 74
Figure 5-5: ELEGOO Arduino Uno R3 (left) and 5V relay module (right) 74
Figure 5-6: Compatibility issues with the built-in camera of JetAuto Pro 75
Figure 5-7: Recommended compatible USB cameras for the robot 75
Figure 5-8: Different effects of zooming in on images with different types of zoomable camera 76
Figure 5-9: Construction of Jetson Orin Nano (NVIDIA, 2024e) 77
Figure 5-10: Lithium battery and voltage converter for Jetson Orin Nano 77
Figure 5-11: Left, front and rear views of the robot 78
Figure 5-12: Schematic diagram of the connection between the devices and the transfer of data 78
Figure 5-13: Remote desktop control of Jetson Orin Nano and Jetson Nano in NoMachine 79

Figure 5-14: IPv4 setting of Jetson Orin Nano (left) and Jetson Nano (right) 80
Figure 5-15: Python code and the messages communication by Jetson Nano (left) and Jetson Orin Nano (right) 80
Figure 5-16: Problems encountered when mapping using Explore_lite of JetAuto Pro: bad mapping result (left) and stuck on door (right) 81
Figure 5-17: Mapping effects in different indoor areas using GMapping of JetAuto Pro....81
Figure 5-18: Mapping using GMapping of JetAuto Pro by corridor....82
Figure 5-19: Mapping using GMapping of JetAuto Pro in open outdoor environment 82
Figure 5-20: Creating PGM map and YAML file from the IFC model of Building L501 of Technical University of Darmstadt 83
Figure 5-21: Visualization of the PGM map derived from the IFC model in RViz with the origin [0, 0, 0] 83
Figure 5-22: Improving the quality of map from satellite imagery of Google Earth using GIMP....84
Figure 5-23: Steps to improve the quality of map captured by drone using GIMP....85
Figure 5-24: Using Python script (right) to set multiple destination points for robot navigation (left) 85
Figure 5-25: Overview crack image dataset from Roboflow 86
Figure 5-26: The Kaggle interface used for training model 87
Figure 5-27: The changes of the loss functions by training and validation process of YOLOv8....87
Figure 5-28: The changes of the Precision, Recall and mAP during the training process of YOLOv8 88
Figure 5-29: Comparison of (a) crack images with label and (b) inference results by best-trained model....88
Figure 5-30: Validating the best YOLOv8n-seg model on the new dataset....89
Figure 5-31: Validating the best YOLOv8s-seg model on the new dataset 89
Figure 5-32: Comparison of inference speed with segmentation result and terminal output: (a) without acceleration, (b) acceleration via TensorRT Python API and (c) acceleration via Trtexec....90
Figure 5-33: Inference speed of the YOLOv8s-seg model (a) without TensorRT acceleration and (b) with TensorRT acceleration....90
Figure 5-34: Inference speed of the YOLOv8x-seg model (a) without acceleration and (b) with TensorRT acceleration....91
Figure 5-35: Inference speed of the YOLOv9c-seg model (a) without acceleration and (b) with TensorRT acceleration....91
Figure 5-36: Issue about over-current by YOLOv9c-seg model with TensorRT acceleration in Jetson Orin Nano 92
Figure 5-37: F1-Confidence Curve of the best-trained YOLOv8s-seg model 92
Figure 5-38: Crack segmentation result based on YOLOv8n in Jetson Orin Nano 93
Figure 5-39: Color distribution analysis of the segmentation output using Pixel Color Counter tool 93
Figure 5-40: Color distribution analysis of the modified segmentation output using Pixel Color Counter tool ...94
Figure 5-41: Code for calculating the maximum width of crack....94
Figure 5-42: Capturing Mask of crack segmentation result through OpenCV....95
Figure 5-43: Crack width analysis: (a) localization of maximum crack width indicated by inscribed circle and (b) results of manual measurements using Augenmaß tool....95
Figure 5-44: Calculating the maximum width of crack in frame by real-time inference 96
Figure 5-45: Crack segmentation result with fixed-focus camera positioned 14 cm above the ground....96

Figure 5-46: Segmentation result of a nearly 1 mm wide crack with camera positioned 14 cm above the ground 97
Figure 5-47: Segmentation result of a nearly 1 mm wide crack after adjusting the camera height to 4 cm above the ground....97
Figure 5-48: Segmentation result of a nearly 1 mm wide crack using a 10x zoom camera....98
Figure 5-49: Testing real-time crack segmentation with moving robot....98
Figure 5-50: Using Arduino to control the water valve....99
Figure 5-51: The snippet of Python code by Jetson Orin Nano to issue commands to Jetson Nano....100
Figure 5-52: Testing the spraying capacity of sprinkler system....101
Figure 5-53: Key Python code in onboard computer for receiving instructions, reading coordinates, and uploading information of crack....102
Figure 5-54: Robot initial position comparison: actual Initial location in reality (left) and initial position displayed in RViz (right)....103
Figure 5-55: The ellipsoid shape of WGS84 (Wagner, 2024)....104
Figure 5-56: The corresponding coordinates of the origin of the robot in Cesium....104
Figure 5-57: The algorithm to convert the local Cartesian coordinate to WGS84 coordinate....105
Figure 5-58: Comparison of the image capture quality under conditions with and without camera warm-up ...105
Figure 5-59: Code for camera warm-up....106
Figure 5-60: Notification about adding current IP address....106
Figure 5-61: Storing images as binary data in MongoDB Atlas....107
Figure 5-62: Error logs generated when the picture size exceeds the limit of MongoDB server....107
Figure 5-63: Activity diagram of the 3D Tiles creation process (Marnat et al., 2022)....108
Figure 5-64: JavaScript code to adjust the location of converted 3D Tiles in Cesium....109
Figure 5-65: Setting style of surface for the converted 3D Tiles in Cesium: (a) converted 3D Tiles, (b) color change by mouse hovering, (c) colorizing structure based on height and (d) hiding surface....110
Figure 5-66: Visualization of wide crack and corresponding photo, description, and coordinates....111
Figure 5-67: Key JavaScript code to visualize different points associated with corresponding photo of crack .112
Figure 5-68: GUI of the quick switching between multiple projects....112
Figure 5-69: Effects of using measuring tools....113
Figure 5-70: Adding issue: (a) selecting a location and adding description and (b) visualizing the issue in Cesium....114
Figure 5-71: Display IFC attribute in Cesium using IfcOpenShell....115
Figure 5-72: Key Python code for retrieving the name of selected IFC element using IfcOpenShell....115
Figure 6-1: Testing robot on concrete slab with narrow crack....116
Figure 6-2: The segmentation results of the cracks on the concrete slabs at various stages....117
Figure 6-3: Effects of the crack repair after 6 weeks....118
Figure 6-4: Testing robot on concrete stick with cracks....118
Figure 6-5: Calcium carbonate was induced under indoor maintenance....119
Figure 6-6: Outlet of solenoid water valve clogged by suspension of the solution of component A....119
Figure 6-7: Printing crack and the tile joints represent as cracks for the robot....120
Figure 6-8: YOLOv8 segmentation result of (a) printing crack and (b) tile joint....120

Figure 6-9: Setup and operation of multi-point navigation 121
Figure 6-10: Display the information of recorded wide crack in Cesium 121
Figure 6-11: The location of Darmstadt skate park in Google map 122
Figure 6-12: Aerial view of the skate park from (a) Google Earth and (b) drone 122
Figure 6-13: Creating map using drone: (a) is aerial image captured by drone and (b) is corresponding PGM map processed by image editor GIMP 123
Figure 6-14: Determining the threshold using ruler for distinguishing between narrow and wide cracks 123
Figure 6-15: Visualizing robot operation on NoMachine 124
Figure 6-16: The recorded wide crack on the testing area: (a) actual location and (b) visualization in Cesium 124
Figure 6-17: Spraying water on narrow cracks upon detection 125
Figure 6-18: Segmentation of a crack with wide section (green region) and narrow section (red region) 125
Figure 6-19: Misidentifying black stains as crack 126
Figure 7-1: Schematic diagram about the integration with Jetson Orin Nano in Matrice 350 RTK drone 129
Figure 7-2: P600 Advanced Version Drone (AMOVLAB, 2023) 130
Figure 7-3: The timeline of MM-LLMs from 2022 to Feb. 2024 (Zhang et al., 2024) 131
Figure 7-4: Analyzing crack images through gpt-4o-mini (Model A) and llava-onevision-qwen2-72b-ov-chat model (Model B) 133
Figure 7-5: Prompt and inference result of live LLaVA-3b model running on Jetson Orin device (NVIDIA, 2024d) 133
Figure 7-6: Schematic illustrating the principle of controlling the robot for crack repair using a multimodal model 134

Bibliography

Achal, V., Mukerjee, A., Sudhakara Reddy, M., 2013. Biogenic treatment improves the durability and remediates the cracks of concrete structures. Constr. Build. Mater. 48, 1–5. https://doi.org/10.1016/j.conbuildmat.2013.06.061

ACI Committee 224, 2007. 224.1R-07: Causes, Evaluation, and Repair of Cracks in Concrete Structures [WWW Document]. URL https://www.concrete.org/publications/internationalconcreteabstractsportal/m/details/id/18555 (accessed 9.8.24).

AliExpress, 2024. Dc 12V 24V 220V Plastic Electric Solenoid Valve Normally Closed Pressure Solenoid Valve Inlet Valve Inlet Flow Switch [WWW Document]. aliexpress. URL //www.aliexpress.com/item/3256805291266809.html?src=ibdm_d03p0558e02r02&sk=&aff_platform=&aff_trace_key=&af=&cv=&cn=&dp= (accessed 5.27.24).

AMOVLAB, 2024. Hardware Introduction - P-Series Drone Documentation [WWW Document]. URL https://wiki.amovlab.com/public/prometheuswiki/ (accessed 6.7.24).

AMOVLAB, 2023. P600 Advanced Research Drone [WWW Document]. URL https://www.amovlab.com/product/detail?pid=38 (accessed 6.7.24).

Antony, A., Low, J.H., Gray, S., E. Childress, A., Le-Clech, P., Leslie, G., 2011. Scale formation and control in high pressure membrane water treatment systems: A review. J. Membr. Sci. 383, 1–16. https://doi.org/10.1016/j.memsci.2011.08.054

Bang, S.S., Galinat, J.K., Ramakrishnan, V., 2001. Calcite precipitation induced by polyurethane-immobilized *Bacillus pasteurii*. Enzyme Microb. Technol. 28, 404–409. https://doi.org/10.1016/S0141-0229(00)00348-3

Bang, S.S., Lippert, J.J., Yerra, U., Mulukutla, S., Ramakrishnan, V., 2010. Microbial calcite, a bio-based smart nanomaterial in concrete remediation. Int. J. Smart Nano Mater. 1, 28–39. https://doi.org/10.1080/19475411003593451

Basilisk, 2022. PDS Product Data Sheet [WWW Document]. Prod. Doc. URL https://basiliskconcrete.com/en/downloads/ (accessed 5.21.24).

Basilisk, 2021a. Parking garage Apeldoorn [WWW Document]. URL https://basiliskconcrete.com/en/portfolio-items/parkeergarage-apeldoorn/ (accessed 9.13.24).

Basilisk, 2021b. Projects of Basilisk Liquid Repair System ER7 [WWW Document]. URL https://basiliskconcrete.com/en/portfolio_category/basilisker7-en/ (accessed 9.13.24).

Basilisk, 2021c. Life span concrete bus lane extended by 15 years [WWW Document]. URL https://basiliskconcrete.com/en/portfolio-items/life-span-concrete-bus-lane-extended-by-15-years/ (accessed 9.13.24).

Basilisk, 2021d. Repair of a 1000 shrinkage cracks [WWW Document]. URL https://basiliskconcrete.com/en/portfolio-items/reparatie-van-1000-krimpscheuren/ (accessed 9.13.24).

Basilisk, 2021e. Cracks in precast elements for ProRail repaired with Basilisk Liquid System [WWW Document]. URL https://basiliskconcrete.com/en/portfolio-items/cracks-in-precast-elements-for-prorail-repaired-with-basilisk-liquid-system/ (accessed 9.13.24).

Basilisk, 2021f. Durability issue due to concrete cracks in balconies, solved with Basilisk ER7 [WWW Document]. URL https://basiliskconcrete.com/en/portfolio-items/durability-issue-due-to-concrete-cracks-in-balconies-solved-with-basilisk-er7/ (accessed 9.13.24).

Basilisk, 2021g. Easy concrete repair with Basilisk Liquid Repair System ER7 [WWW Document]. Basilisk Self-Heal. Concr. URL https://basiliskconcrete.com/en/products/product-liquid-concrete-repair-system-er7/ (accessed 5.21.24).

Bhandari, B., 2023. Edge Computing with Raspberry Pi. Medium. URL https://medium.com/@thebinayak/edge-computing-with-raspberry-pi-ec28912bc3df (accessed 5.24.24).

BOLA, spol s.r.o., 2024. Bola Systems [WWW Document]. Bola Syst. URL https://www.bolasystems.com/help-advice/what-are-solenoid-valves (accessed 5.27.24).

Braig, D., 2024. Join the Cesium AECO Tech Preview Program [WWW Document]. Cesium. URL https://cesium.com/blog/2024/09/03/join-the-cesium-aeco-tech-preview-program/ (accessed 9.8.24).

Brunner, D., 2023. Frame Rate: a Beginner's Guide. TechSmith Blog. URL https://www.techsmith.com/blog/frame-rate-beginners-guide/ (accessed 6.6.24).
Cha, Y.-J., Choi, W., Büyüköztürk, O., 2017. Deep Learning-Based Crack Damage Detection Using Convolutional Neural Networks. Comput.-Aided Civ. Infrastruct. Eng. 32, 361–378. https://doi.org/10.1111/mice.12263
Chebotar, Y., Yu, T., 2023. RT-2: New model translates vision and language into action [WWW Document]. Google Deep. URL https://deepmind.google/discover/blog/rt-2-new-model-translates-vision-and-language-into-action/ (accessed 5.27.24).
Choi, S.G., Chu, J., Brown, R.C., Wang, K., Wen, Z., 2017. Sustainable Biocement Production via Microbially Induced Calcium Carbonate Precipitation: Use of Limestone and Acetic Acid Derived from Pyrolysis of Lignocellulosic Biomass. ACS Sustain. Chem. Eng. 5, 5183–5190. https://doi.org/10.1021/acssuschemeng.7b00521
Cozi Life, 2024. Karcher Pressure Washer Adapter 1/4 Quick Connect 5 Nozzle - Temu Germany [WWW Document]. temu. URL https://www.temu.com/de-en/karcher-pressure-washer-adapter-1-4-quick-connect-with-5-nozzle-tips-g-601099519228334.html (accessed 5.27.24).
De Muynck, W., Cox, K., Belie, N.D., Verstraete, W., 2008a. Bacterial carbonate precipitation as an alternative surface treatment for concrete. Constr. Build. Mater. 22, 875–885. https://doi.org/10.1016/j.conbuildmat.2006.12.011
De Muynck, W., Debrouwer, D., De Belie, N., Verstraete, W., 2008b. Bacterial carbonate precipitation improves the durability of cementitious materials. Cem. Concr. Res. 38, 1005–1014. https://doi.org/10.1016/j.cemconres.2008.03.005
DJI, 2024. Matrice 350 RTK [WWW Document]. URL https://enterprise.dji.com/de/matrice-350-rtk (accessed 9.27.24).
Fang, Y., Wang, S., Bi, Q., Cui, D., Yan, C., 2022. Design and Technical Development of Wall-Climbing Robots: A Review. J. Bionic Eng. 19, 877–901. https://doi.org/10.1007/s42235-022-00189-x
Genova, J., Cabrera, E., Hoskere, V., 2024. Vision-Based Adaptive Robotics for Autonomous Surface Crack Repair. https://doi.org/10.48550/arXiv.2407.16874
Glewwe, K., 2017. Concrete Mender crack repair at the Milan-Malpensa Airport (MXP) | Roadware Incorporated. URL https://concretemender.com/crack-repair-at-the-milan-malpensa-airport-mxp/ (accessed 9.8.24).
Goodwin, F., 2009. Floor Cracking: How, What, Where? - ppt download [WWW Document]. URL https://slideplayer.com/slide/1347306/ (accessed 6.4.24).
Grenon, A., 2024. townsean/canvas-pixel-color-counter.
Gui, Z., Li, H., 2020. Automated Defect Detection and Visualization for the Robotic Airport Runway Inspection. IEEE Access 8, 76100–76107. https://doi.org/10.1109/ACCESS.2020.2986483
Gupta, P., Dixit, M., 2022. Image-based crack detection approaches: a comprehensive survey. Multimed. Tools Appl. 81, 40181–40229. https://doi.org/10.1007/s11042-022-13152-z
Happho, 2024. Corrosion of Steel Reinforcement: Causes, Effects and Remedies – happho. URL https://happho.com/corrosion-steel-reinforcement-causes-effects-remedies/ (accessed 8.20.24).
Hasan, K., Nahid, A., Reza, K.J., 2014. Path planning algorithm development for autonomous vacuum cleaner robots, in: 2014 International Conference on Informatics, Electronics and Vision, ICIEV 2014. p. 6. https://doi.org/10.1109/ICIEV.2014.6850799
Hiwonder, 2024. JetAuto Pro ROS Robot Car with Vision Robotic Arm Powered by Jetson Na – Hiwonder [WWW Document]. URL https://www.hiwonder.com/products/jetauto-pro?variant=40040875229271 (accessed 5.24.24).
Horizon, 2024. hobot_llm [WWW Document]. GitHub. URL https://github.com/HorizonRDK/hobot_llm (accessed 5.26.24).
Hsieh, Y.-A., Tsai, Y.J., 2020. Machine Learning for Crack Detection: Review and Model Performance Comparison. J. Comput. Civ. Eng. 34, 04020038. https://doi.org/10.1061/(ASCE)CP.1943-5487.0000918
Huang, W., Wang, C., Zhang, R., Li, Y., Wu, J., Fei-Fei, L., 2023. VoxPoser: Composable 3D Value Maps for Robotic Manipulation with Language Models. https://doi.org/10.48550/arXiv.2307.05973
huggingface, 2024. WildVision Arena: Benchmarking Multimodal LLMs in the Wild [WWW Document]. URL https://huggingface.co/spaces/WildVision/vision-arena

Jain, S., Fang, C., Achal, V., 2021. A critical review on microbial carbonate precipitation via denitrification process in building materials. Bioengineered 12, 7529–7551. https://doi.org/10.1080/21655979.2021.1979862
Jiang, B., 2022. Crack width detection algorithm based on opencv (algorithm for calculating the maximum tangent circle of a contour)_Crack Detection Algorithm-CSDN Blog [WWW Document]. URL https://blog.csdn.net/oJiWuXuan/article/details/120160916 (accessed 9.9.24).
Jocher, G., Chaurasia, A., Qiu, J., 2023. Ultralytics YOLO [WWW Document]. URL https://github.com/ultralytics/ultralytics (accessed 5.24.24).
Jongvivatsakul, P., Janprasit, K., Nuaklong, P., Pungrasmi, W., Likitlersuang, S., 2019. Investigation of the crack healing performance in mortar using microbially induced calcium carbonate precipitation (MICP) method. Constr. Build. Mater. 212, 737–744. https://doi.org/10.1016/j.conbuildmat.2019.04.035
Jonkers, H., Thijssen, A., Muyzer, G., Copuroglu, O., Schlangen, E., 2010. Application of bacteria as self-healing agent for the development of sustainable concrete. Ecol. Eng. 36 2010 230–235.
Jouppi, N., 2017. Quantifying the performance of the TPU, our first machine learning chip [WWW Document]. Google Cloud Blog. URL https://cloud.google.com/blog/products/gcp/quantifying-the-performance-of-the-tpu-our-first-machine-learning-chip (accessed 5.26.24).
Jovi, 2023. DRY POUR CONCRETE for CRACK REPAIR! Part 2! [WWW Document]. URL https://www.youtube.com/watch?app=desktop&v=iiBQ5ldw_ok&ab_channel=Jovi (accessed 6.5.24).
Khaliq, W., Ehsan, M.B., 2016. Crack healing in concrete using various bio influenced self-healing techniques. Constr. Build. Mater. 102, 349–357. https://doi.org/10.1016/j.conbuildmat.2015.11.006
Khan, M.A.-M., Harseno, R.W., Kee, S.-H., Nahid, A.-A., 2023. Development of AI- and Robotics-Assisted Automated Pavement-Crack-Evaluation System. Remote Sens. 15, 3573. https://doi.org/10.3390/rs15143573
Kukil, 2022. Intersection Over Union IoU in Object Detection Segmentation [WWW Document]. URL https://learnopencv.com/intersection-over-union-iou-in-object-detection-and-segmentation/ (accessed 9.8.24).
Lin, S., 2024. triple-Mu/YOLOv8-TensorRT [WWW Document]. YOLOv8-TensorRT. URL https://github.com/triple-Mu/YOLOv8-TensorRT/blob/main/docs/Segment.md (accessed 6.23.24).
MA, J., Ding, Y., Cheng, J., Tan, Y., Gan, V., ZHANG, J., 2019. Analyzing the Leading Causes of Traffic Fatalities Using XGBoost and Grid-Based Analysis: A City Management Perspective. IEEE Access PP, 1–1. https://doi.org/10.1109/ACCESS.2019.2946401
Marnat, L., Gautier, C., Colin, C., Gesquière, G., 2022. PY3DTILERS: AN OPEN SOURCE TOOLKIT FOR CREATING AND MANAGING 2D/3D GEOSPATIAL DATA. ISPRS Ann. Photogramm. Remote Sens. Spat. Inf. Sci. X-4-W3-2022, 165–172. https://doi.org/10.5194/isprs-annals-X-4-W3-2022-165-2022
Martinez, D., 2020. Is Transfer Learning the final step for enabling AI in Aviation? Datascience.aero. URL https://datascience.aero/transfer-learning-aviation/ (accessed 5.24.24).
MathWorks, 2024. Was ist SLAM (Simultane Lokalisierung und Kartierung) – MATLAB & Simulink [WWW Document]. URL https://de.mathworks.com/discovery/slam.html (accessed 5.29.24).
Microcontrollers Lab, 2021. 5V Single Channel Relay Module [WWW Document]. Microcontrollers Lab. URL https://microcontrollerslab.com/5v-single-channel-relay-module-pinout-working-interfacing-applications-datasheet/ (accessed 5.27.24).
Microsoft, 2024. ONNX Runtime | Events [WWW Document]. URL https://onnxruntime.ai/events (accessed 5.26.24).
Moropoulou, A., Bakolas, A., Anagnostopoulou, S., 2005. Composite materials in ancient structures. Cem. Concr. Compos., Cement and Concrete Research in Greece 27, 295–300. https://doi.org/10.1016/j.cemconcomp.2004.02.018
Nguyen, S.D., Tran, T.S., Tran, V.P., Lee, H.J., Piran, Md.J., Le, V.P., 2023. Deep Learning-Based Crack Detection: A Survey. Int. J. Pavement Res. Technol. 16, 943–967. https://doi.org/10.1007/s42947-022-00172-z
NVIDIA, 2024a. Jetson Modules, Support, Ecosystem, and Lineup [WWW Document]. NVIDIA Dev. URL https://developer.nvidia.com/embedded/jetson-modules (accessed 5.24.24).
NVIDIA, 2024b. NVIDIA Isaac ROS [WWW Document]. URL https://developer.nvidia.com/isaac/ros (accessed 10.1.24).
NVIDIA, 2024c. NVIDIA Isaac Sim [WWW Document]. URL https://developer.nvidia.com/isaac/sim (accessed 10.1.24).

NVIDIA, 2024d. Live LLaVA - NVIDIA Jetson AI Lab [WWW Document]. URL https://www.jetson-ai-lab.com/tutorial_live-llava.html (accessed 5.26.24).
NVIDIA, 2024e. Jetson Orin Nano Developer Kit Getting Started Guide [WWW Document]. NVIDIA Dev. URL https://developer.nvidia.com/embedded/learn/get-started-jetson-orin-nano-devkit (accessed 9.9.24).
NVIDIA, 2019. TensorRT Command-Line Wrapper: trtexec [WWW Document]. URL https://github.com/NVIDIA/TensorRT/tree/main/samples/trtexec (accessed 10.1.24).
NVIDIA, 2018. Deep Learning SDK Documentation [WWW Document]. URL http://docs.nvidia.com/deeplearning/sdk/tensorrt-developer-guide/index.html (accessed 5.26.24).
Oh, F., 2012. What Is CUDA? [WWW Document]. NVIDIA Blog. URL https://blogs.nvidia.com/blog/what-is-cuda-2/ (accessed 5.25.24).
ONNX Runtime developers, 2018. ONNX Runtime [WWW Document]. URL https://github.com/microsoft/onnxruntime (accessed 5.26.24).
OSNDT, 2024. Top 3 Methods For Crack Depth Measurement In Concrete [WWW Document]. OnestopNDT. URL https://www.onestopndt.com/ndt-articles/crack-depth-measurement-in-concrete (accessed 6.6.24).
Paz-Gallardo, A., Plaza, A., 2011. A New Morphological Anomaly Detection Algorithm for Hyperspectral Images and its GPU Implementation, Proceedings of SPIE - The International Society for Optical Engineering. https://doi.org/10.1117/12.892282
Peffers, K., Tuunanen, T., Rothenberger, M., Chatterjee, S., 2007. A design science research methodology for information systems research. J. Manag. Inf. Syst. 24, 45–77.
Poursaee, A., Ross, B., 2022. Chloride-Induced Corrosion of Carbon Steel in Cracked Concrete [WWW Document]. URL https://encyclopedia.pub/entry/24073 (accessed 9.8.24).
Ramachandran, S.K., Ramakrishnan, V., Bang, S.S., 2001. Remediation of Concrete Using Microorganisms. Mater. J. 98, 3–9. https://doi.org/10.14359/10154
Ramalingam, B., Hayat, A.A., Elara, M.R., Félix Gómez, B., Yi, L., Pathmakumar, T., Rayguru, M.M., Subramanian, S., 2021. Deep Learning Based Pavement Inspection Using Self-Reconfigurable Robot. Sensors 21, 2595. https://doi.org/10.3390/s21082595
Roboflow, 2023. crack Image Dataset [WWW Document]. Roboflow. URL https://universe.roboflow.com/university-bswxt/crack-bphdr (accessed 5.30.24).
Roffo, G., 2017. Ranking to Learn and Learning to Rank: On the Role of Ranking in Pattern Recognition Applications. https://doi.org/10.48550/arXiv.1706.05933
ROS Wiki, 2018. ROS/Introduction - ROS Wiki [WWW Document]. URL https://wiki.ros.org/ROS/Introduction (accessed 9.28.24).
RS, S., Mohanty, S., Elias, S., 2023. Control and Coordination of a SWARM of Unmanned Surface Vehicles using Deep Reinforcement Learning in ROS. https://doi.org/10.48550/arXiv.2304.08189
Sahai, N., 2024. Convolutional Neural Network: Layers, Types, & More. Blogs Updat. Data Sci. Bus. Anal. AI Mach. Learn. URL https://www.analytixlabs.co.in/blog/convolutional-neural-network/ (accessed 5.22.24).
Schmugge, S.J., Rice, L., Lindberg, J., Grizziy, R., Joffey, C., Shin, M.C., 2017. Crack Segmentation by Leveraging Multiple Frames of Varying Illumination, in: 2017 IEEE Winter Conference on Applications of Computer Vision (WACV). Presented at the 2017 IEEE Winter Conference on Applications of Computer Vision (WACV), IEEE, Santa Rosa, CA, USA, pp. 1045–1053. https://doi.org/10.1109/WACV.2017.121
Schmugge, S.J., Rice, L., Nguyen, N.R., Lindberg, J., Grizzi, R., Joffe, C., Shin, M.C., 2016. Detection of cracks in nuclear power plant using spatial-temporal grouping of local patches, in: 2016 IEEE Winter Conference on Applications of Computer Vision (WACV). Presented at the 2016 IEEE Winter Conference on Applications of Computer Vision (WACV), IEEE, Lake Placid, NY, USA, pp. 1–7. https://doi.org/10.1109/WACV.2016.7477601
SealBoss, 2024. Concrete Crack Injection - Polyurethane - SealBoss Corp. [WWW Document]. URL https://sealboss.com/concrete-crack-injection-polyurethane/ (accessed 6.4.24).
SHRIVASTAVA, A., 2020. Underfitting Vs Just right Vs Overfitting in Machine learning | Kaggle [WWW Document]. URL https://www.kaggle.com/discussions/getting-started/166897 (accessed 5.24.24).
Siddique, R., Chahal, N.K., 2011. Effect of ureolytic bacteria on concrete properties. Constr. Build. Mater. 25, 3791–3801. https://doi.org/10.1016/j.conbuildmat.2011.04.010
Slamtec, 2024. Slamtec RPLIDAR A1 - SLAMTEC Global Network [WWW Document]. URL https://www.slamtec.ai/product/slamtec-rplidar-a1/ (accessed 8.26.24).

Spencer, C.A., van Paassen, L., Sass, H., 2020. Effect of Jute Fibres on the Process of MICP and Properties of Biocemented Sand. Materials 13, 5429. https://doi.org/10.3390/ma13235429
Strong-Tie Company, 2024. Crack Injection Guide [WWW Document]. URL https://www.strongtie.com/products/anchoring-systems/technical-notes/anchoring-adhesives/epoxy-injection-guide (accessed 6.4.24).
SubChange, 2022. Image-based Crack Segmentation and Crack Width Calculation (Orthogonal Skeleton Line Method) - CSDN Blogs [WWW Document]. URL https://blog.csdn.net/Subtlechange/article/details/118523710 (accessed 9.9.24).
Sun, X., Chen, J., Lu, S., Liu, M., Chen, S., Nan, Y., Wang, Y., Feng, J., 2021. Ureolytic MICP-Based Self-Healing Mortar under Artificial Seawater Incubation. https://doi.org/10.3390/su13094834
SuperAnnotate, 2023. Image segmentation detailed overview [WWW Document]. URL https://www.superannotate.com/blog/image-segmentation-for-machine-learning (accessed 5.22.24).
tmall, 2024. 10x 12M manual zoom usb industrial camera [WWW Document]. URL https://detail.tmall.com/item.htm?_u=t2mke2fc799c&id=736851577968&skuId=5262437867021&spm=a1z09.2.0.0.2d942e8dfMaofU (accessed 9.29.24).
Ultralytics, 2024a. TensorRT [WWW Document]. URL https://docs.ultralytics.com/integrations/tensorrt (accessed 5.26.24).
Ultralytics, 2024b. NVIDIA Jetson [WWW Document]. URL https://docs.ultralytics.com/guides/nvidia-jetson (accessed 9.2.24).
Ultralytics, 2024c. Raspberry Pi [WWW Document]. URL https://docs.ultralytics.com/guides/raspberry-pi (accessed 9.2.24).
Ultralytics, 2024d. ultralytics/assets [WWW Document]. GitHub. URL https://github.com/ultralytics/assets/releases (accessed 6.17.24).
Ultralytics, 2024e. YOLOv8 [WWW Document]. URL https://docs.ultralytics.com/models/yolov8 (accessed 5.26.24).
Van Tittelboom, K., De Belie, N., De Muynck, W., Verstraete, W., 2010. Use of bacteria to repair cracks in concrete. Cem. Concr. Res. 40, 157–166. https://doi.org/10.1016/j.cemconres.2009.08.025
Wagner, T., 2024. Position information and their coordinate systems WGS84 and ETRS. GeneSys Elektron. GmbH. URL https://genesys-offenburg.de/support/application-aids/gnss-basics/position-information/ (accessed 6.1.24).
Wang, A., Chen, H., Liu, L., Chen, K., Lin, Z., Han, J., Ding, G., 2024. YOLOv10: Real-Time End-to-End Object Detection. https://doi.org/10.48550/arXiv.2405.14458
Wang, C.Y., Yeh, I.-H., Liao, H.-Y.M., 2024. YOLOv9: Learning What You Want to Learn Using Programmable Gradient Information. arXiv.
wikiHow, 2023. 3 Ways to Fix Concrete Cracks [WWW Document]. URL https://www.wikihow.com/Fix-Concrete-Cracks (accessed 6.5.24).
Wiktor, V., Jonkers, H.M., 2011. Quantification of crack-healing in novel bacteria-based self-healing concrete. Cem. Concr. Compos. 33, 763–770. https://doi.org/10.1016/j.cemconcomp.2011.03.012
Xu, F., 2021. ONNX Runtime 1.8: mobile, web, and accelerated training [WWW Document]. Microsoft Open Source Blog. URL https://cloudblogs.microsoft.com/opensource/2021/06/07/onnx-runtime-1-8-mobile-web-and-accelerated-training/ (accessed 5.26.24).
Yahboom, 2024. RDK X3 ROS2 Robot Car with Mecanum Wheel [WWW Document]. Yahboom. URL https://category.yahboom.net/products/rdk-x3-robot (accessed 5.27.24).
Zhang, D., Yu, Y., Dong, J., Li, C., Su, D., Chu, C., Yu, D., 2024. MM-LLMs: Recent Advances in MultiModal Large Language Models. https://doi.org/10.48550/arXiv.2401.13601
Zhang, J., Zhao, C., Zhou, A., Yang, C., Zhao, L., Li, Z., 2019. Aragonite formation induced by open cultures of microbial consortia to heal cracks in concrete: Insights into healing mechanisms and crystal polymorphs. Constr. Build. Mater. 224, 815–822. https://doi.org/10.1016/j.conbuildmat.2019.07.129
Zhang, K., Tang, C.-S., Jiang, N., Pan, X.-H., Liu, B., Wang, Y.-J., Shi, B., 2023. Microbial-induced carbonate precipitation (MICP) technology: a review on the fundamentals and engineering applications. Env. Earth Sci 82 229 2023 Httpsdoiorg101007s12665-023-10899-. https://doi.org/10.1007/s12665-023-10899-y
Zhang, L., Yang, F., Zhang, Y., Zhu, Y., 2016. Road crack detection using deep convolutional neural network. https://doi.org/10.1109/ICIP.2016.7533052
zhangti, 2020. 3D measurement plug-in based on Cesium [WWW Document]. URL https://github.com/zhangti0708/cesium-measure (accessed 9.6.24).

Zou, X., 2024. whi_rviz_plugins [WWW Document]. URL https://github.com/xinjuezou-whi/whi_rviz_plugins