



Palestine Polytechnic University
College of Information Technology and Computer Engineering
Computer Engineering Department

Automated Building Inspection System

Project Members:

Yara Sarhna 217092

Ayat Abu Allan 191060

Raghad Abu Samor 217099

Supervisor:

Eng. Wael Altakrouri

Hebron - Palestine

January - 2024

Acknowledgment

The Giver of Light, the Weaver of Paths, the Ever-magnificent, the Ever-Thankful, Alhamdulillah, praise be to Allah the Almighty of God the most Gracious and the most Merciful, for all the blessings you have given us. The ability, courage, endurance, and patience you put inside us strengthened us in completing this Project.

First and foremost, we want to remember and honour the brave people and martyrs in Gaza who spent their lives for freedom and justice. Their strength reminds us of challenges beyond our studies, and we salute their memory. They remain alive in our hearts and our history.

Our deepest appreciation to our mothers for their precious and continuous guidance and support throughout our lives, and for all the daily prayers that you have dedicated to us. To our fathers and family members who have supported us so far and who have been the source of inspiration, thank you for always being by our side.

Heartfelt appreciation to Eng. Wael Takrouri, our project supervisor, who has been a guiding light, leading us down the right path and offering invaluable direction. His continuous guidance, encouragement, and unwavering support throughout the semester have been instrumental to our success.

To those who stood by us through thick and thin, making awesome memories in tough and good times. A big thanks to our special friends Maria, Zainah, Duha and Nagham for always being there, encouraging us, and making this journey truly special.

ABSTRACT

This work aims to find innovative solutions to the problem of manually detecting defects or damages in building's walls that require significant resources and time. The proposed solutions aim to create an automated inspection system that can effectively and efficiently detect areas that require maintenance or repair.

In terms of implementation, a mobile robot used along with ROS packages and tools to manage data from sensors and integrate different algorithms to perform analysis and inspections. It will incorporate machine learning algorithms to analyze the input images from the camera and identify any cracks or structural defects.

The automated inspection system's data and results can be accessed remotely, providing a comprehensive report with detailed information about their building's condition. The solution aims to enhance building safety, improve maintenance, and prevent long-term damage to the structure.

Keywords: ROS, mobile robot, defects finding, reports, 2D map, Localization, Navigation, SLAM.

الملخص

يهدف هذا المشروع إلى إيجاد حلول مبتكرة لمشكلة الكشف اليدوي عن العيوب والأضرار في جدران المباني التي تتطلب موارد كبيرة ووقتاً طويلاً، كما تهدف الحلول المقترحة إلى إنشاء نظام تفتيش آلي يمكنه الكشف بفعالية وكفاءة عن المناطق التي تحتاج إلى صيانة أو إصلاح. سيستخدم النظام مجموعة متنوعة من حزم ROS وأدواتها لإدارة البيانات من الحساسات ودمج الخوارزميات المختلفة للقيام بالتحليل والتفتيش. يتضمن النظام خوارزميات تعلم الآلة لتحليل النموذج الثلاثي الأبعاد وتحديد أي تشققات أو فجوات أو عيوب أخرى في الهيكل.

يمكن الوصول إلى بيانات النظام ونتائجها عن بعد من خلال موقع إلكتروني، كما يمكن تحميل تقرير شامل لحالة المبنى، مما يوفر معلومات دقيقة لأصحاب المباني والمتخصصين عن حالة مبناهم، الذي يؤدي إلى تحسين سلامة المبني وصيانتها ومنع تهالكها على المدى الطويل.

كلمات مفتاحية: نظام تشغيل الروبوت، العيوب الإنسانية، كشف الأضرار، الصيانة والإصلاح، أضرار المبني .

Table of Contents

List of Figures.....	6
List of Tables.....	8
List of Acronyms.....	9
Chapter 1: Introduction.....	10
1.1 Preface.....	10
1.2 Project Aims and Objectives.....	10
1.3 Problem Statement.....	10
1.3.1 Problem Definition.....	10
1.3.2 Problem Significance And Motivation.....	10
1.4 Project Requirements.....	11
1.4.1 Functional Requirements:.....	11
1.4.2 Non-Functional Requirements:.....	11
1.5 Project Limitations and Constraints.....	12
1.6 Project Expected Output.....	12
Chapter 2: Theoretical Background.....	13
2.1 Preface.....	13
2.2 Theories.....	13
Simultaneous Localization and Mapping.....	13
Gmapping.....	13
Localization.....	14
Obstacles Avoidance.....	14
Machine Learning.....	15
Image Processing.....	15
2.3 Literature Review.....	16
2.4 System Components.....	18
Chapter 3: System Design.....	19
3.1 Overview.....	19
3.2 Design Options.....	19
3.2.1 Hardware Components Options.....	19
3.2.2 Software Components Options.....	25
3.3 General Block Diagram.....	28
3.4 Conceptual System Description.....	29
3.5 Pseudo-Code.....	30
Chapter 4: Implementation.....	31
4.1 Overview.....	31
4.2 Hardware Components.....	31
4.3 Software Components.....	32
4.3.1 Installing Ubuntu Mate 20.04 Operating System.....	32
4.3.2 Installing ROS Noetic.....	32

4.3.3 TurtleBot Installation.....	32
4.4 Mapping using SLAM.....	33
4.5 Navigation.....	34
4.5.1 Path Planning.....	34
4.5.2 Obstacle Avoidance.....	35
4.6 Image Classification.....	35
4.6.1 Teachable Machine For Image Classification.....	35
4.6.2 Classification Of Defects.....	36
4.7 ROS Nodes Graph.....	37
4.8 Web Application Implementation.....	38
4.8.1 MQTT Protocol.....	38
4.8.2 User Interface.....	38
4.8.3 Deployment Process on AWS Lightsail.....	42
4.9 Implementation Issus.....	44
Chapter 5: Testing.....	45
5.1 Overview.....	45
5.2 ROS Installation.....	45
5.3 Testing The TurtleBot.....	45
5.3.1 Testing Motor In Turtlesim.....	45
5.4 Testing The Kinect.....	46
5.4.1 Default 3D sensor.....	46
5.4.2 Test OpenNI Driver.....	46
5.4.3 Test Kinect Stream.....	46
5.4.4 Test Image Data.....	47
5.4.5 Test Depth Data.....	47
5.4.6 Autonomous Motion and Avoiding Obstacles.....	47
5.5 Testing The ML Model.....	48
5.6 Testing The Web App.....	50
5.7 Unit Testing.....	51
5.8 System Validation.....	52
Chapter 6: Conclusion and Future work.....	54
6.1 Conclusion.....	54
6.2 Future work.....	54

List of Figures

Figure 2.1	General schematic for mobile robot localization	13
Figure 2.2	Autonomous robot navigation pipeline	14
Figure 3.1	Kobuki robot - Turtlebot2	18
Figure 3.2	Clearpath Jackal	19
Figure 3.3	Kinect camera	23
Figure 3.4	System block diagram	27
Figure 3.5	Conceptual system design	28
Figure 4.1	System components	30
Figure 4.2	Connect Kinect sensor to Turtlebot	30
Figure 4.3	Generated 2D map	32
Figure 4.4	Creating a map using Rviz.	33
Figure 4.5	Example of model development with a teachable machine	34
Figure 4.6	ROS Nodes Graph	36
Figure 4.7	Sign in page	37
Figure 4.8	Welcome page	38
Figure 4.9	Map and defects page	38
Figure 4.10	Map with defects	39
Figure 4.11	Generated pdf report	40
Figure 4.12	Launched Lightsail instance	41
Figure 4.13	Created Database	41
Figure 4.14	Remote database connection	42
Figure 4.15	Remote database password hashing	42
Figure 5.1	Turtlesim robot simulation	44
Figure 5.2	RGB camera data stream	45
Figure 5.3	Test Kinect depth data	46
Figure 5.4	Test the ML model	47
Figure 5.5	Cross-validated calibration plot	47
Figure 5.6	Confusion matrix	48
Figure 5.7	Result of testing and scoring the model	48
Figure 5.8	Test broker connection and topic subscription	49

Figure 5.9	Test data transmission	49
Figure 5.10	Robot position transmission	51
Figure 5.11	Redundant data transmission	51
Figure 5.12	Defects localization results	52

List of Tables

Table 2.1	Literature Review	16
Table 3.1	Differences Between TurtleBot 2 and Clearpath Jack	19
Table 3.2	Differences Between The Raspberry Pi 3 and Laptop	20
Table 3.3	Differences Between The Kinect Camera and Logitech Camera	21
Table 3.4	Differences Between The Kinect Sensor and LIDAR Sensor	22
Table 3.5	Differences between the mobile application and web application	24
Table 3.6	Differences between MRPT and ROS Noetic	25
Table 4.1	Dataset classes and sample images	35
Table 5.1	Software Testing Table	50

List of Acronyms

2D	Two Dimensional.
CSS	Cascade Style Sheets.
DHT	Digital Humidity and Temperature.
AWS	Amazon Web Services.
FoV	Field of View.
HAT	Hardware Attached On Top.
HTML	HyperText Markup Language.
HDD	Hard Disk Drive.
HDMI	High-Definition Multimedia Interface.
IMU	Inertial Measurement Unit.
LiDAR	Light Detection and Ranging.
MQTT	Message Queuing Telemetry Transport.
MRPT	Mobile Robot Programming Toolkit.
PHP	Hypertext Preprocessor.
RAM	Random Access Memory.
RDS	Relational Database Service.
ROS	Robot Operating System.
Rviz	Robot Visualization.
RFID	Radio Frequency Identification.
RQT	ROS Qt GUI toolkit
SLAM	Simultaneous Localization and Mapping.
SSD	Solid-State Drive.
SSH	Secure Shell
SQL	Structured query language.
URG	Ultrasonic Ranging And Guidance.
YOLO	You Only Look Once.

Chapter 1: Introduction

1.1 Preface

There is a need to create an automated inspection system that can effectively detect areas that require maintenance or repair. The lack of such a system has resulted in many buildings suffering from neglected maintenance, leading to structural damage or deterioration.

1.2 Project Aims and Objectives

This project focuses on developing an automated inspection system for detecting and analysing structural defects and damage in buildings. Using sensors such as cameras and laser imaging and detection, the system will generate a 2D model of the building's structure and identify areas that require maintenance or repair. The system will be built using a Robot Operating System (ROS) and aims to provide an accurate, efficient, and effective solution for detecting and analysing structural defects in buildings.

1.3 Problem Statement

1.3.1 Problem Definition

Problem Significance: Structural defects and damage in buildings can be hazardous and costly to homeowners. Traditional methods of identifying these defects rely on human factors that are time-consuming, expensive, and prone to errors. And those in charge of detection and monitoring may not be as accurate as required, in addition to people underestimating and abstaining from periodic maintenance and follow-up of buildings and facilities.

1.3.2 Problem Significance And Motivation

The proposed system will provide an accurate, efficient, and effective solution for detecting and analyzing structural defects in buildings, improving the inspection process by reducing inspection times, lowering costs, and providing more accurate and consistent results. The motivation behind the project is to create a practical and innovative solution for detecting and analyzing structural defects in buildings, making them safer and more habitable for all.

1.4 Project Requirements

1.4.1 Functional Requirements:

The functional requirements of an automated inspection system play a crucial role in ensuring that the system can detect and analyze structural defects and damage accurately, efficiently, and effectively. These requirements outline the specific capabilities and features that the system must have to meet its objectives and provide value to users. The following is a list of functional requirements :

1. It should be able to generate a map of the building structure .
2. The system should be able to identify and locate structural defects and damage within the 2D map.
3. It should be able to provide feedback to users when structural defects or damage are detected.
4. It should be able to navigate the targeted area autonomously without constant user interaction.
5. It should be able to avoid obstacles during its navigation.

1.4.2 Non-Functional Requirements:

1. The system should be reliable and stable, with minimal downtime or system failures, It should also have a backup and recovery mechanism in case of unexpected system failures.
2. It should be user-friendly and easy to use, with a simple interface. It should also provide clear reports and results that are easy to read.
3. System's web application should be secure and protect sensitive data from unauthorised access or theft.
4. It should be compatible with different hardware and software environments, including sensors, platforms, and communication protocols.
5. Acceptable response time.

1.5 Project Limitations and Constraints

1. Environmental Conditions: The robot will be designed and tested to operate in indoor environments and it is not suitable for outdoor use.
2. WiFi connectivity constraint in order to make the system functional.
3. The robot can only navigate through a single floor environment and can't go up or down stairs.
4. The brightness of lighting can impact the accuracy of defect classification.

1.6 Project Expected Output

A mobile robot, operating on ROS, is designed to construct a map of a building's layout, autonomously navigate through the environment, and identify and localize structural damages, categorizing them by type. The robot then transmits all gathered data to the cloud. This information is accessible through a web application, offering a comprehensive display of the building's status. Additionally, the system possesses the capability to generate reports outlining the detected damages.

Chapter 2: Theoretical Background

2.1 Preface

This chapter provides a theoretical background and literature review for the project. It briefly outlines the system components, including the sensors, and provides information on the methodology and machine learning models that will be used to automate the building inspection process. Overall, it aims to provide the necessary background information to understand the system and its functioning.

2.2 Theories

This section explores the fundamental technologies and concepts of navigation, localization, SLAM, and machine learning/image processing that will be used in the system to detect structural defects such as cracks, humidity, or structural damage.

Simultaneous Localization and Mapping

Simultaneous Localization and Mapping (SLAM) is a fundamental concept in robotics that involves constructing a map of the environment while simultaneously estimating the robot's position within that map. SLAM combines localization and mapping to enable a robot to autonomously explore and navigate in unknown environments. This algorithm utilizes sensor data, such as Light Detection and Ranging (LiDAR) or camera motor encoders as inputs, to incrementally build the map and refine the robot's position estimate [1].

Gmapping

This package contains the ROS wrapper for Gmapping for OpenSlam. The laser-based gmapping package provides SLAM, as a ROS node called `slam-gmapping`. You can create a 2D occupancy network map (like a building floor plan) from laser data and an image collected by an animated robot. The map is drawn by the movement of the robot in a specific location, with LIDAR sensors the distances between the robot and nearby obstacles, and during the drawing the robot everything around is being discovered[2].

Localization

Localization is the process of determining the precise position of a robot within its environment. It involves estimating the robot's coordinates (e.g., x, y, and z) as shown in Figure 2.1, localization is crucial for the robot to understand its position relative to the surrounding objects and to accurately navigate and interact with the environment depth sensor and cameras are used as main sensors to obtain information about the surrounding environment and there are common localization methods, including simultaneous localization and mapping [3].

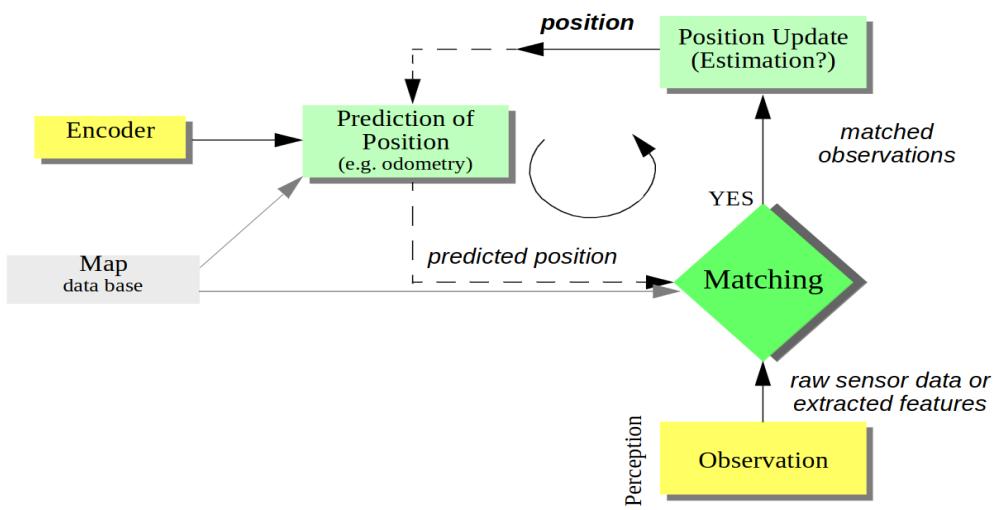


Figure 2.1 : General schematic for mobile robot localization [4].

Obstacles Avoidance

Fundamental component of robot navigation plays a crucial role in ROS implementation designed for autonomous robots. It enables the robot to safely navigate its environment by detecting and reacting to obstacles and barriers. In this project, which includes sensors like ultrasonic and Kinect's IR, the robot identifies obstacles like staircases or objects in its path. When an obstacle is detected, the robot initiates a series of actions based on predefined algorithms [5]. For example, if the robot encounters a staircase, it may utilize its sensors to detect the change in elevation and depth. Using this information, it can determine the presence of a staircase and its dimensions. Then, the robot's navigation algorithms come into play. These algorithms are designed to plan an alternative route around the obstacle, whether it's a staircase, object, or any other impediment. The robot will then proceed with navigation while avoiding the obstacle, ensuring it reaches its intended destination safely and efficiently.

Navigation

Navigation refers to the process of guiding a robot from one location to another in a given environment. Figure 2.3 provides a representation of the process that involves determining the robot's path, avoiding obstacles, and reaching the desired destination. Various algorithms and techniques are used for navigation, including path planning, object detection, and recognition [6].

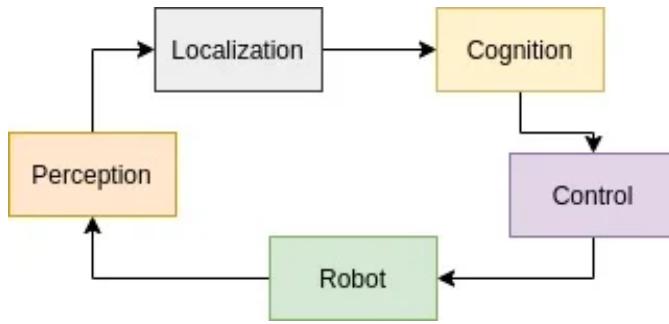


Figure 2.2: Autonomous robot navigation pipeline [7].

Machine Learning

Machine learning algorithms, such as deep neural networks, can be employed for tasks like image classification, semantic segmentation, and object localization. These techniques enable the robot to interpret visual data and make informed decisions based on the analyzed information [8].

Furthermore, the project will leverage the Teachable Machine platform for model training.

Image Processing

Image processing techniques play a crucial role in various aspects of the project. Image processing algorithms are used to analyze and extract relevant information from camera inputs, such as object detection, recognition, and tracking. Is well-established and supported by packages like OpenCV and perception packages in ROS [9].

2.3 Literature Review

In [10], the authors compared the performance of several machine learning algorithms for automated defect detection in building structures using image data captured by cameras ,and found that random forests had the highest accuracy and efficiency for detecting defects such as cracks, humidity, or structural damage.

However, the authors noted that the algorithm's performance can be affected by the quality of the image data and the size and complexity of the building structure.

Authors [11] found that ROS can be used to integrate different sensors and devices for building automation and inspection, such as cameras, LiDAR sensors, and thermal sensors. They also noted that ROS can be used to develop intelligent algorithms for analyzing sensor data and detecting defects or anomalies.

However, the authors noted that the scalability and reliability of the system can be affected by the complexity of the sensor network and the communication protocol used.

The limitations of this literature review are that it focused mainly on the integration of ROS and IoT technologies for smart building systems and did not discuss other important aspects of automated inspection systems, such as navigation and communication of the inspection robots. Additionally, the review did not provide a detailed evaluation of the existing literature on defect detection algorithms and their limitations.

Simulation of a mobile robot done by [12], that combines mapping, localization, and object detection using the Robot Operating System (ROS). The robot uses a LIDAR sensor to scan the environment, which is transformed into an occupancy grid map through ROS. The robot then localizes itself using the Monte Carlo localization algorithm, navigates through the Dijkstra algorithm, and detects objects using the You Only Look Once algorithm. The simulation was conducted using the Gazebo software, and the robot's performance was evaluated through 100 simulated localization experiments across 10 maze environments, resulting in a success rate of 62% and an average time of 139 seconds for successful attempts.

In [13] the Storing Robot is a mobile robot with an arm attached to its base that classifies objects based on their RFID tags. It moves along a predefined path and uses IR range finders to locate objects and move towards them dynamically. This robot is designed to put objects on their designated shelves, and it achieves this task through a series of specific

moves. This technology is useful in various settings, such as warehouses and storage facilities.

The "Object Finder" [14] project integrates cutting-edge technologies to enable a robot's autonomous search and detection capabilities. Employing a mapping algorithm, the robot generates a comprehensive map of its environment, and directives from a mobile application guide its search for specific objects. A specialised camera facilitates object detection using the YOLO algorithm and the Robot Operating System (ROS). Navigating and avoiding obstacles, the robot communicates successful identifications and precise coordinates to the mobile app. This seamless integration showcases the project's efficiency and sophistication in executing targeted search and detection tasks.

Table 2.1: Literature review.

Literature	Author & Year	Key Points
Literature 1 [10]	Authors: Y. Sun Year: 2019	<ul style="list-style-type: none"> Comparison of machine learning algorithms for defect detection in building structures using image data. Random forests showed the highest accuracy and efficiency for detecting defects. Algorithm performance can be affected by image quality and building structure complexity.
Literature 2 [11]	Authors: M.A.H. Chowdhury Year: 2020	<ul style="list-style-type: none"> Integration of ROS and IoT for building automation and inspection. ROS can integrate different sensors and develop intelligent algorithms for detecting defects. Scalability and reliability can be affected by sensor network complexity and communication protocol used.

Literature 3 [12]	Authors: Firas Mohtaseb, Abdalmenem Amleh Mohammad Mohtaseb Year: 2020	<ul style="list-style-type: none"> Development of a simulation of a mobile robot that combines mapping, localization, and object detection using ROS. LIDAR sensor used for environment scanning and Monte Carlo Localization algorithm used for self-localization. Dijkstra algorithm used for navigation and YOLO algorithm used for object detection. Success rate of 62% and average time of 139 seconds for successful attempts.
Literature 4 [13]	Authors: Mariam E'mar Alaa Shaheen Year: 2013	<ul style="list-style-type: none"> Storing Robot is a mobile robot with an arm that classifies objects based on RFID tags. IR range finders used for locating objects and dynamic movement towards them. Designed for putting objects on designated shelves in settings such as warehouses and storage facilities.
Literature 5 [14]	Authors: Hamza Dwaik Moayad Hrebat Motaz Natsheh Year: 2023	<ul style="list-style-type: none"> Cutting-Edge Object Detection. Seamless User Interaction: The integration of a mobile application facilitates user directives, enabling a smooth interaction between the user and the autonomous robot during search missions. Effective System Communication.

2.4 System Components

1. Hardware: The system will use different components to capture data about the building structure and to identify any defects in the walls, this includes:

- Mobile robot.
- Processing unit.
- Camera.
- Depth sensor.

2. Software: It includes packages to control the robot and other tools for image processing and ML and create a user-friendly app, like:

- ROS .
- OpenCV.
- Teachable Machine.
- WebTools.

Chapter 3: System Design

3.1 Overview

This chapter will cover the overall design of the system and the integration of its components, showing the block diagram and schematic diagram, as well as details about the algorithms used.

3.2 Design Options

By comparing the available components and evaluating different hardware and software choices, the aim is to identify the most suitable components that align with the project requirements and objectives.

3.2.1 Hardware Components Options

1. Mobile Robot

The mobile robot plays a crucial role in the system as it serves as the physical platform for navigation and inspection tasks. Various options are available for selecting a suitable mobile robot for the project, considering factors such as features, functionalities, and compatibility with the project requirements.

1.A. Kobuki Robot - Turtlebot2

It provides a reliable and customizable hardware base with integrated sensors, including a camera, laser scanner, and inertial measurement unit as shown in Figure 3.1.

The TurtleBot's compatibility with ROS allows for seamless integration with various software libraries and algorithms, enabling advanced functionalities such as mapping, localization, and path planning [15].



Figure 3.1: Kobuki robot - Turtlebot2 [16].

1.B. Clearpath Jackal

The Clearpath Jackal is a compact and agile mobile robot designed for outdoor and indoor environments. It comes with an array of sensors, including a 3D LiDAR, and offers robust localization and mapping capabilities as shown in Figure 3.2 [17].

And the key differences between TurtleBot 2 and Clearpath Jack, presented in Table 3.1.



Figure 3.2: Clearpath Jackal [17].

Table 3.1: Differences between TurtleBot 2 and Clearpath Jack.

Mobile Robot		
Characteristic	Clearpath Jackal[18]	Turtlebot 2[19]
Size	Bigger Size	Smaller Size
Max Speed	2.0m/s	0.65 m/s
Weight	17Kg	6.3 Kg
Cost	Higher cost	Low cost
Number Of Motors	4	2
Battery Capacity	270000 mAh	2200-3000 mAh

For our project, the TurtleBot has been chosen as the mobile robot platform due to its low cost, lightweight design, it offers a customizable hardware base with integrated sensors like a camera, laser scanner, and inertial measurement unit and it is available. The turtlebot's compatibility with ROS allows for seamless integration with various software libraries and algorithms, enabling advanced functionalities such as mapping, localization, and path planning.

2. Processing Unit

The processing unit is the component that drives the system's functionality. It receives and processes sensor data, executes algorithms, and controls system components.

When comparing and evaluating two choices, a Raspberry Pi 3 and a laptop, the selection of the processor depends on specific characteristics shown in Table 3.2.

Table 3.2: Differences between the Raspberry Pi 3 and Laptop.

Processing Unit		
Characteristic	Raspberry Pi 3[20]	Laptop[21]
Image		
Cost	Low (Started from \$40)	Higher(Started from \$300)
Size	Small (85mm x 56mm x 17 mm) [22]	Larg (330mm x 220mm)
Memory	1GB RAM	8GB - 16GB or more RAM
Storage	MicroSD card “up to 1TB”	HDD /SSD “256GB-1TB”
Speed	Quad-core ARM Cortex-A72	Multi-core x86
Ports	HDMI, USB, Ethernet, GPIO	HDMI, USB, Ethernet, various peripherals
Power Consumption	Low (2.8 to 3.1 watts)	High (10–100 watts)

After careful consideration and reviewing previous work, including "Object Finder and Storing Robot" [23], chose to use a laptop as a controller. This decision was influenced by concerns related to lagging issues that were observed in the project utilizing the Raspberry Pi.

3. Camera

The camera is used to capture images of the building structure, enabling detailed inspection and analysis for the detection of structural defects. There are two available options mentioned in Table 3.3:

Table 3.3: Differences Between the Kinect camera and Logitech camera.

Camera		
Characteristic	Kinect Camera[24]	Logitech Camera[25]
Image	 [26]	
Cost	\$160	\$70
FoV	60 Degree	78 Degree
Depth Resolution	512*424	Not Available
Multiple Sensors per PC	Yes	Yes
External Power Supply	12V DC adapter	powered through USB

Kinect camera chosen because it captures detailed images of the environment, not only RGB images but also depth information, allowing the robot to build a comprehensive understanding of its surroundings. Also its high-resolution imaging enhances the ability to detect and analyze structural defects[27].

4. Depth Sensor

A depth sensor is used to measure the distance between the sensor and objects in its environment. It enables the creation of three-dimensional representations by capturing depth information and provides simultaneous localization and mapping capabilities. This technology automatically detects any object nearby and measures the distance to it on the go. These features allow devices to move autonomously by making real-time decisions[28]. There are two available options for depth sensors to choose from shown in Table 3.4.

Table 3.4: Differences between the Kinect sensor and LIDAR URG 04lx sensor.

Depth Sensor		
Characteristic	Kinect Sensor [29]	LIDAR URG 04lx sensor[30]
Image	 A photograph of the Microsoft Kinect sensor, which is a black rectangular device with a camera lens and two infrared sensors at the top, connected to a white Xbox 360 console below it.	 A photograph of the LIDAR URG 04lx sensor, which is a small, compact black circular device mounted on a black base.
Cost	\$150	\$100
Detection Range	0.5 m to 4.5 m	4 m
Interference from Light	Infrared-based, less affected by visible light	Sensitive to ambient light
Direction	Depth information in a wide-angle (horizontal 70 degrees, vertical 60 degrees)	Forward-facing (0 to 240 degrees adjustable)
Mapping Capability	Yes	Yes
Accuracy	Sub-millimeter accuracy in depth measurements	±1% of measured distance

Kinect was chosen because it can detect in a range from 0.5m to 4.5 m, covering what is needed for mapping and navigation. It's not too expensive, and it has good accuracy.

Moreover, the Kinect sensor as shown in Figure 3.3 ,captures RGB data while using IR depth sensor to enable depth measurement. In this process, the emitter releases infrared light beams, and the depth sensor reads the reflections of these beams then translates it into depth information, measuring the distance between an object and the sensor.

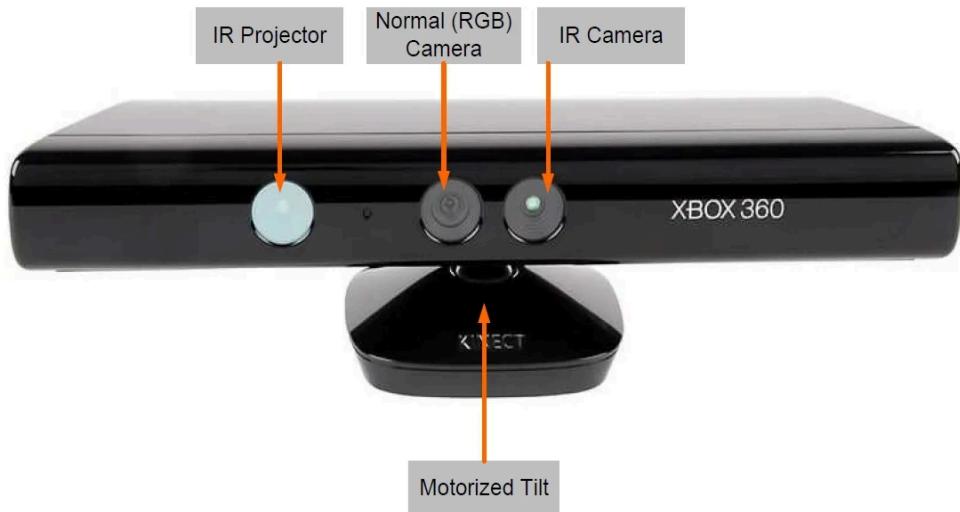


Figure 3.3: Kinect camera [32].

This capability enables the sensor to capture features and generate accurate maps of the building; these maps will be used for navigation and obstacle avoidance within the specified space[33].

3.2.2 Software Components Options

1. Application For Reporting.

Considered two options that act as platforms for displaying the inspection report and giving a thorough explanation of the issues found in the building: a web application and a mobile application

2. Development Tool.

The mobile app or a web app each option has different technologies and languages to ensure a simple user experience, Table 3.5 shows the differences between them.

Table 3.5: Differences between the mobile application and web application.

Mobile app and Web app [34] [35]		
Characteristic	Web app	Mobile app
Access	Accessible across all devices via web browsers.	Not immediately accessible until installed
Offline Access	Require a proper internet connection	Can be accessed even offline.
Loading Speed	Generally faster browsing experience.	May take a while to load, depending on network conditions.
Language	Frontend :Javascript ,HTML,CSS. Backend :Php,Django.	Frontend: Flutter. Backend: Python,Java.
Database	MySQL,MongoDB.	MySQL.

After careful comparison, a web application was chosen, it can be accessed from any device with a web browser, including desktop computers, laptops, tablets, and smartphones. This ensures a broader reach and accessibility for users, as they can access the application using their preferred device without the need to download and install a dedicated mobile app.

3. Robot Operating System.

ROS is a flexible framework for developing robot software. It provides a wide range of libraries, tools, and drivers that can be used for building inspection tasks. ROS supports various programming languages and offers a rich ecosystem of pre-built packages that can be leveraged for perception, mapping, navigation, and other functionalities.

There are several other options for robot operating system frameworks that can be considered for the project, including:

A. ROS Noetic

ROS Noetic is a framework and toolset designed for the development of robotic software. It supports component-based architecture and programming in various languages [36].

B. MRPT

MRPT is a collection of C++ libraries and algorithms for mobile robotics applications. It offers localization, mapping, path planning, and other essential functionalities[37].

The differences between MRPT and ROS Noetic shown in Table 3.6.

Table 3.6: Differences between MRPT and ROS Noetic.

Robot Operating System [38]		
Characteristic	ROS Noetic	MRPT
Inter-platform operability	Multi-language support	Does not support multi language platform
High-end capabilities	Yes	Yes
Support high-end sensors and actuators	Yes	Limited
Tools	Tons of tools	Inbuilt tools and external packages available
Modularity & Active community	Yes	Yes

There are two programming languages available for development within the ROS noetic framework: **C++** and **Python**.

C++ is a powerful and efficient programming language widely used in robotics,

Python was chosen because it offers a wide range of libraries and tools for development and prototyping.

4. OpenCV.

OpenCV is very useful in visualization and analyzing purposes. Its primarily responsible for image preprocessing and enhancing the quality of images recorded by the Kinect camera, This could involve reducing noise, adjusting contrast, and optimizing the images for analysis. The library provides algorithms for image segmentation, pattern recognition, and feature extraction, all of which are vital for identifying structural defects in buildings [39].

5. Visual Studio Code.

An integrated development environment. widely used, free, and open-source code editor with extensive features and support for various programming languages.

6. Teachable Machine.

Teachable Machine is a web-based tool designed to simplify the creation of machine learning models, enabling individuals of varying expertise to efficiently train models, utilizing Convolutional Neural Networks .

This contributes to the enhancement of the system's ability to detect and analyze structural defects, showcasing the dedication to incorporating accessible and innovative technologies into the project [40].

3.3 General Block Diagram

The system design in Figure 3.4 shows a general overview of the system's main components and the connections between them. The camera provides visual data of the building's structure, while the controller acts as the central processing unit, handling data processing. The depth sensor detects nearby objects and obstacles, providing valuable information for navigation. The controller creates a detailed 2D map of the building's environment, enabling precise measurements and the identification of structural defects.

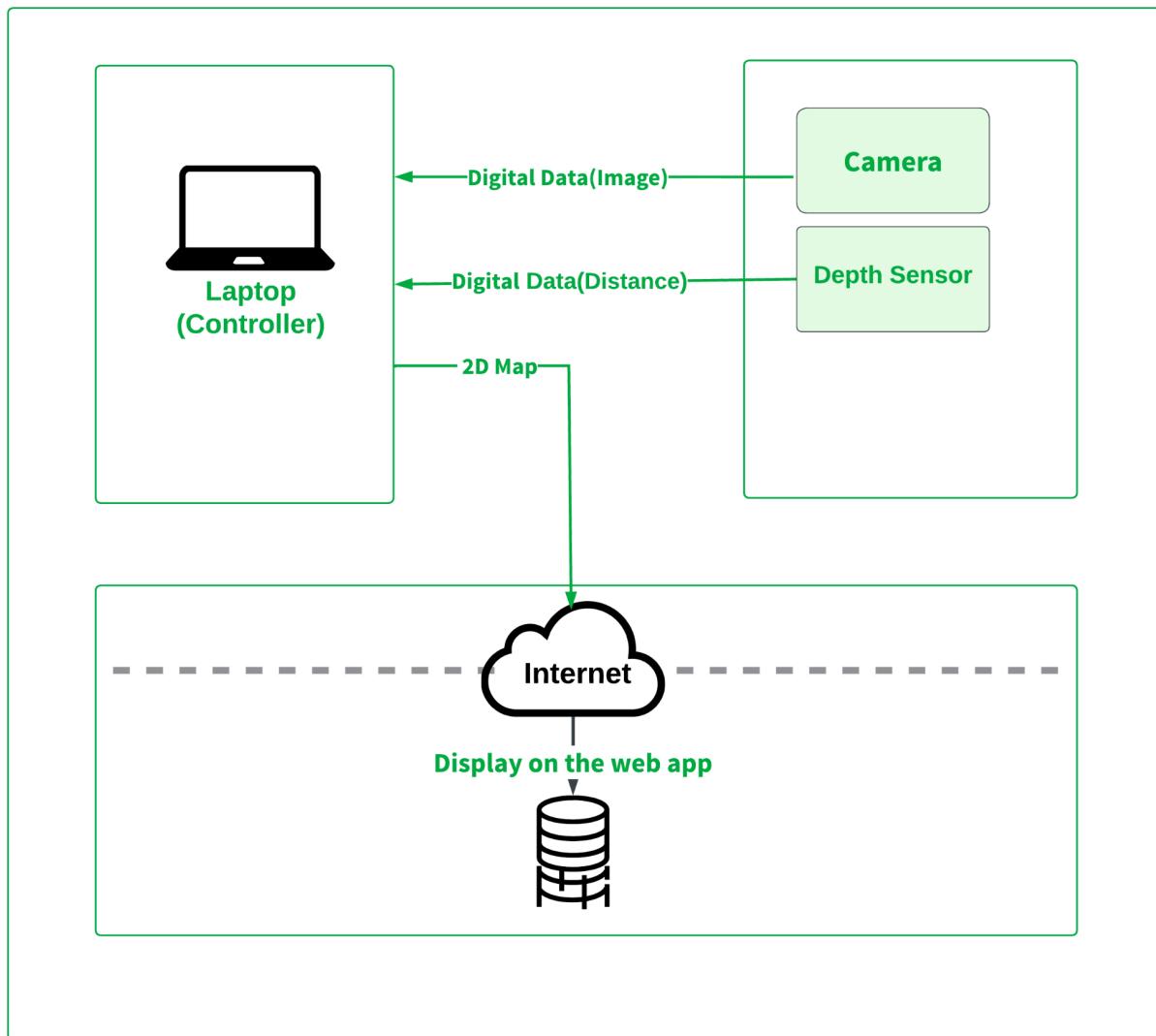


Figure 3.4: System block diagram.

3.4 Conceptual System Description

The conceptual design integrates a mobile robot, camera and depth sensor for capturing images, obstacle avoidance, and 2D modeling as shown in Figure 3.5.

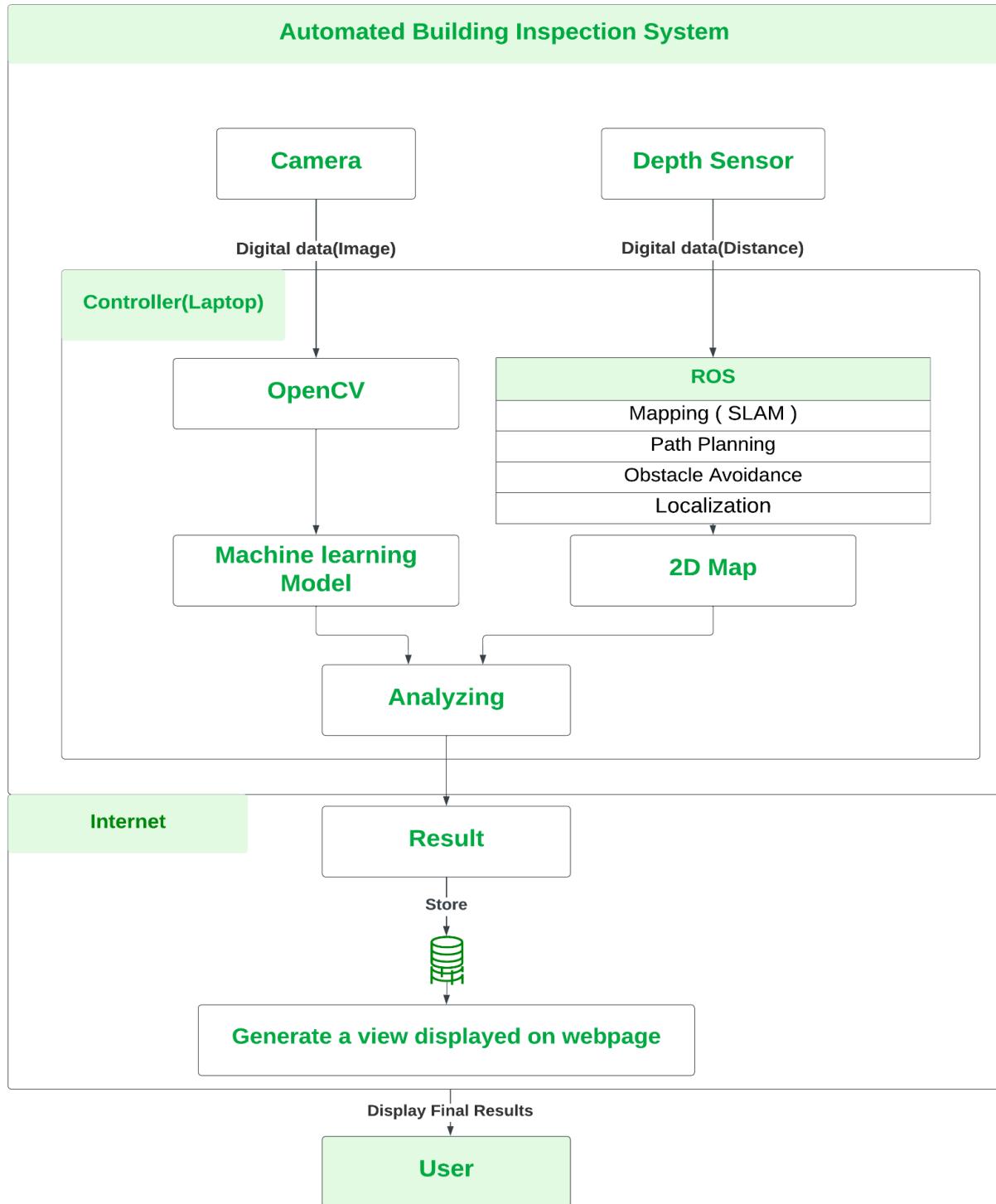


Figure 3.5: Conceptual system design.

3.5 Pseudo-Code

This Pseudo-Code helps a robot navigate, avoiding obstacles and detecting defects. It captures images, checks for obstacles, and uses OpenCV to find defects. Defect details are sent to the cloud, The process continues until the user stops it or the robot reaches its goal.

Algorithm 1:Pseudo-Code for mobile robot navigation and defect detection localization.

```
Begin
Obstacle_range = 0.5
while True:
    image = capture image from camera
    distance_data = capture depth sensor data

    if distance_data < Obstacle_range // There is an obstacle
        Stop Robot
        Turn
    else
        Continue

    detected = defect_detection_algorithm_OpenCV(image)
    If detected == True
        Get x,y coordinates from ROS_Node
        Get image from camera
        Get defect_type from Machine_learning_model
    End if
    Publish_using_MQTT_to_broker({x,y}coordinates,defect_type)
    If user_input == "stop" or robot_reached_goal():
        break

    OUTPUT website_subscribe_to_broker_and_display_the_data
End While
```

Chapter 4: Implementation

4.1 Overview

This chapter describes the implementation part of the project in more detail. It dives deep into the different hardware components of the system and its software with all of its modules.

4.2 Hardware Components

The main component is the laptop, it is linked to various other system components as follow:

- The Turtlebot is connected to the laptop via a USB cable as shown in Figure 4.1.



Figure 4.1: System components.

- The laptop is connected to the Kinect Sensor via a USB cable, and it receives power through an adapter utilizing the Turtlebot's 12V/5A cable, as shown in Figure 4.2.



Figure 4.2: Connect Kinect sensor to Turtlebot.

4.3 Software Components

This section covers the initial setup of the software environment, including the installation of the Ubuntu operating system and the ROS. This is a critical starting point for the software implementation, as it ensures that the system is ready to support the development and operation of the robotic application.

4.3.1 Installing Ubuntu Mate 20.04 Operating System

Ubuntu 20.04 Mate, ensuring stability and compatibility with ROS noetic[41].

4.3.2 Installing ROS Noetic

The system runs on ROS Noetic. It contains all the necessary packages like *ros_enviroment* and *catkin* to operate the Turtlebot robot and Kinect sensor. Additionally, it includes tools for creating maps like *GMapping* [42].

Basic installation commands:

- The following line of command will install the latest ROS Noetic on Ubuntu mate 20.04

```
$ Wget  
https://raw.githubusercontent.com/qboticslabs/ros_install_noetic/  
master/ros_install_noetic.sh && chmod +x ./ros_install_noetic.sh  
&& ./ros_install_noetic.sh
```

```
$ sudo apt install ros-noetic-desktop-full
```

- Environment setup

```
$ echo "source /opt/ros/noetic/setup.bash" >> ~/.bashrc
```

```
$ source ~/.bashrc
```

4.3.3 TurtleBot Installation

This involves installing essential TurtleBot packages that didn't come with the basic ROS Noetic installation. These additional packages include things like *turtlebot_apps*, *launch files*, *turtlebot_viz*, and *turtlebot_bringup*. They provide extra functionalities and tools that enhance the capabilities of TurtleBot for various tasks and applications [43].

The following command used to get the all required packages:

```
$ git clone https://github.com/hanruihua/Turtlebot_on_noetic.git
```

4.4 Mapping using SLAM

The algorithm chosen for this project is GMapping which is a laser-based SLAM that builds a 2D map by following these steps [44]:

Gmapping commands:

1. On Turtlebot, Launch the basic TurtleBot drivers and hardware interface.

```
$ rosrun turtlebot_bringup minimal.launch
```

2. Start the Gmapping algorithm for SLAM to create a map using TurtleBot's sensors.

```
$ rosrun turtlebot_navigation gmapping_demo.launch
```

3. Launch RViz to visualize the navigation stack and display the generated map along with the robot's pose.

```
$ rosrun turtlebot_rviz_launchers view_navigation.launch
```

4. Activate the keyboard teleoperation to manually control the TurtleBot's movements using the keyboard.

```
$ rosrun turtlebot_teleop keyboard_teleop.launch
```

5. While driving the Turtlebot around, it will begin to generate a map as shown in Figure 4.3.

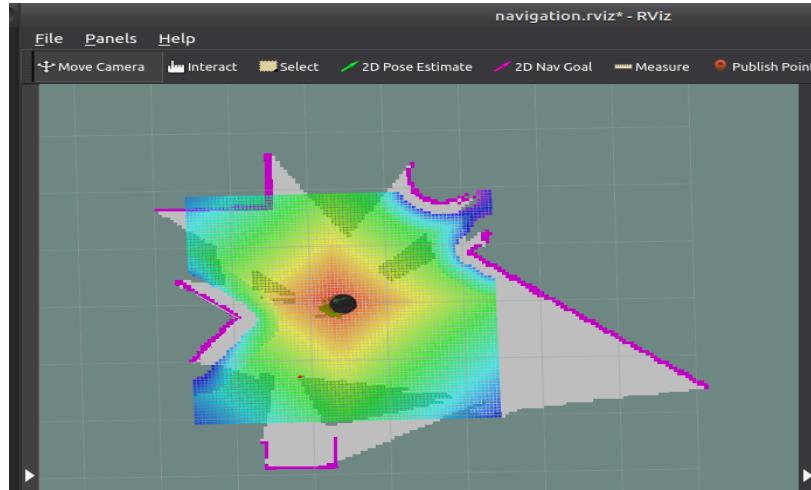


Figure 4.3: Generated 2D map.

6. Save the Map: To save the pre-mapped environment, utilize the `map_saver` tool. Execute the following command, replacing `/path/to/save/the/map` with the desired file path:

```
$ rosrun map_server map_saver -f /path/to/save/the/map
```

4.5 Navigation

The generated map serves as a reference for the robot's position. The robot then plans a path from its current location to the goal using this map. During navigation, the Kinect's sensor continuously scans the surroundings for obstacles. If obstacles are detected, the robot dynamically adjusts its path in real time to avoid collisions[30].

4.5.1 Path Planning

The robot's destination is defined using a script, and it will then generate a path to reach that specified goal. This path planning typically relies on pre-existing maps that were generated[36].

Before beginning path planning, it is necessary to load the map by following these steps:

1.Load the Saved Map

```
$ export TURTLEBOT_MAP_FILE=~/gmapping_01.yaml
```

2.Localize The Turtlebot

start the necessary nodes for the `amcl` localization node including map server, odometry, and sensor nodes.

```
$ roslaunch turtlebot_navigation amcl_demo.launch
```

Start RViz and display the map, the estimated position of the Turtlebot using green arrows.

```
$ roslaunch turtlebot_rviz_launchers view_navigation.launch  
--screen
```

Using a python script sends a navigation goal to the robot, to lead the robot to move to a specified location on the map as shown in the Figure 4.4.

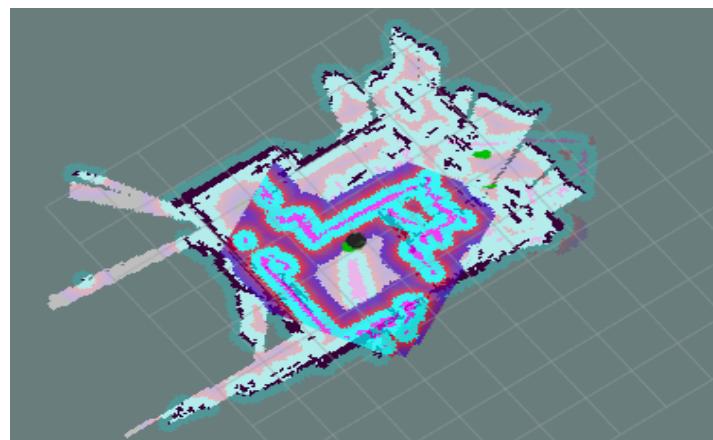


Figure 4.4: Creating a map using Rviz.

4.5.2 Obstacle Avoidance

Using a Python script with the navigation package `move_base`, the robot achieves autonomous movement by planning and executing paths while actively avoiding obstacles. The `move_base` package employs a global planner for high-level path planning and a local planner for real-time adjustments based on sensor feedback. To enhance obstacle avoidance capabilities, specific parameters are set, including `max_obstacle_height` at 0.6m and `obstacle_range` at 0.5 m [45].

4.6 Image Classification

4.6.1 Teachable Machine For Image Classification

Using the Teachable Machine tool as shown in Figure 4.5, the model trained on a dataset of structural defect images to recognize and categorize various structural defects, we chose a batch size of 16, ran 50 epochs, and set the learning rate to 0.001 for optimal training. The dataset consist about 800 images each sized at 170*250 pixels. This model will be utilized in conjunction with OpenCV and the Kinect's camera for real-time structural defect recognition [46].

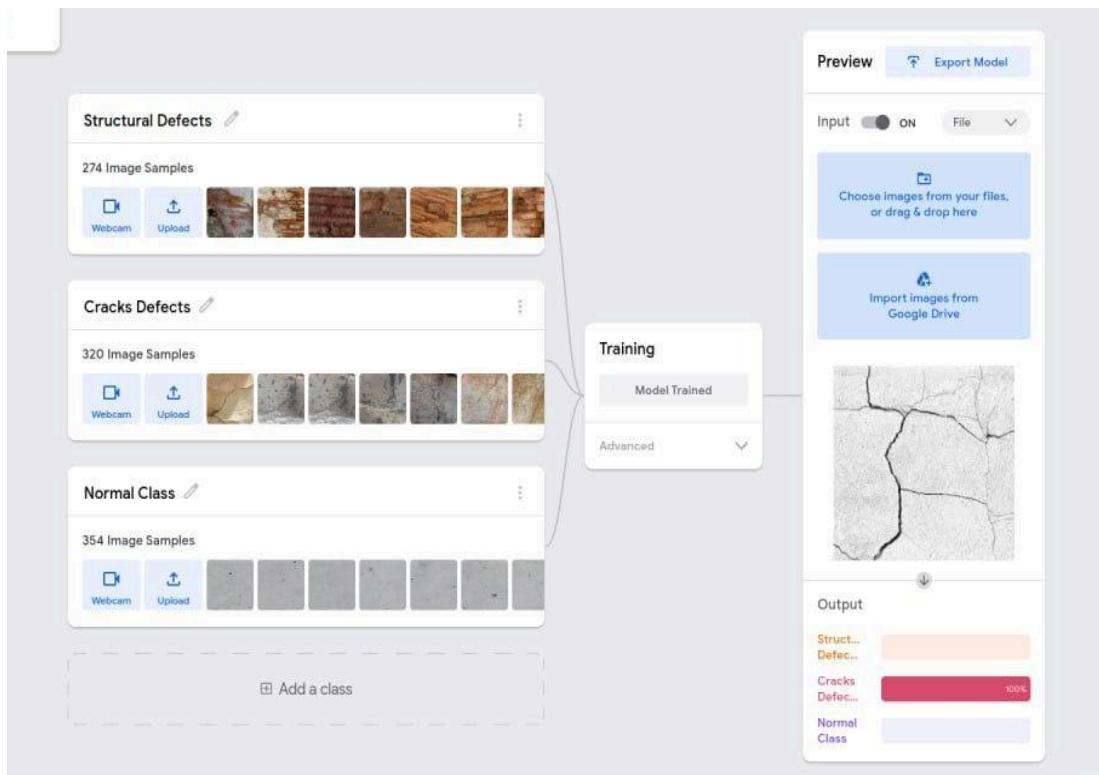


Figure 4.5: Example of model development with a teachable machine.

4.6.2 Classification Of Defects

The dataset is organized into three main classes:

1. **Normal Class:** representing undamaged structures, serving as a reference point for evaluating structural integrity.
2. **Structural Defects Class:** this class represents more extensive and critical harm to structures. Examples include images of buildings with severe damage, such as significant structural cracks, collapsing walls, or any issues posing a substantial threat to the structural integrity of the building.
3. **Cracks Class:** This class contains examples like images of cracks in walls, pavements, or other surfaces. These are relatively minor structural issues when compared to more extensive damage.

This classification system enables the models to distinguish between varying levels of structural integrity, with structural damage representing more extensive and critical harm compared to cracks as shown in Table 4.1.

Table 4.1: Dataset classes and sample images.

Class Name	Sample Images
Normal Class	
Cracks Class	
Structural Defects	

4.7 ROS Nodes Graph

We observed running ROS nodes and their interconnections and communication pathways by executing the following command:

```
$ rosrun rqt_graph rqt_graph
```

The visual representation of nodes and their interaction are shown in RQT Graph in Figure 4.6

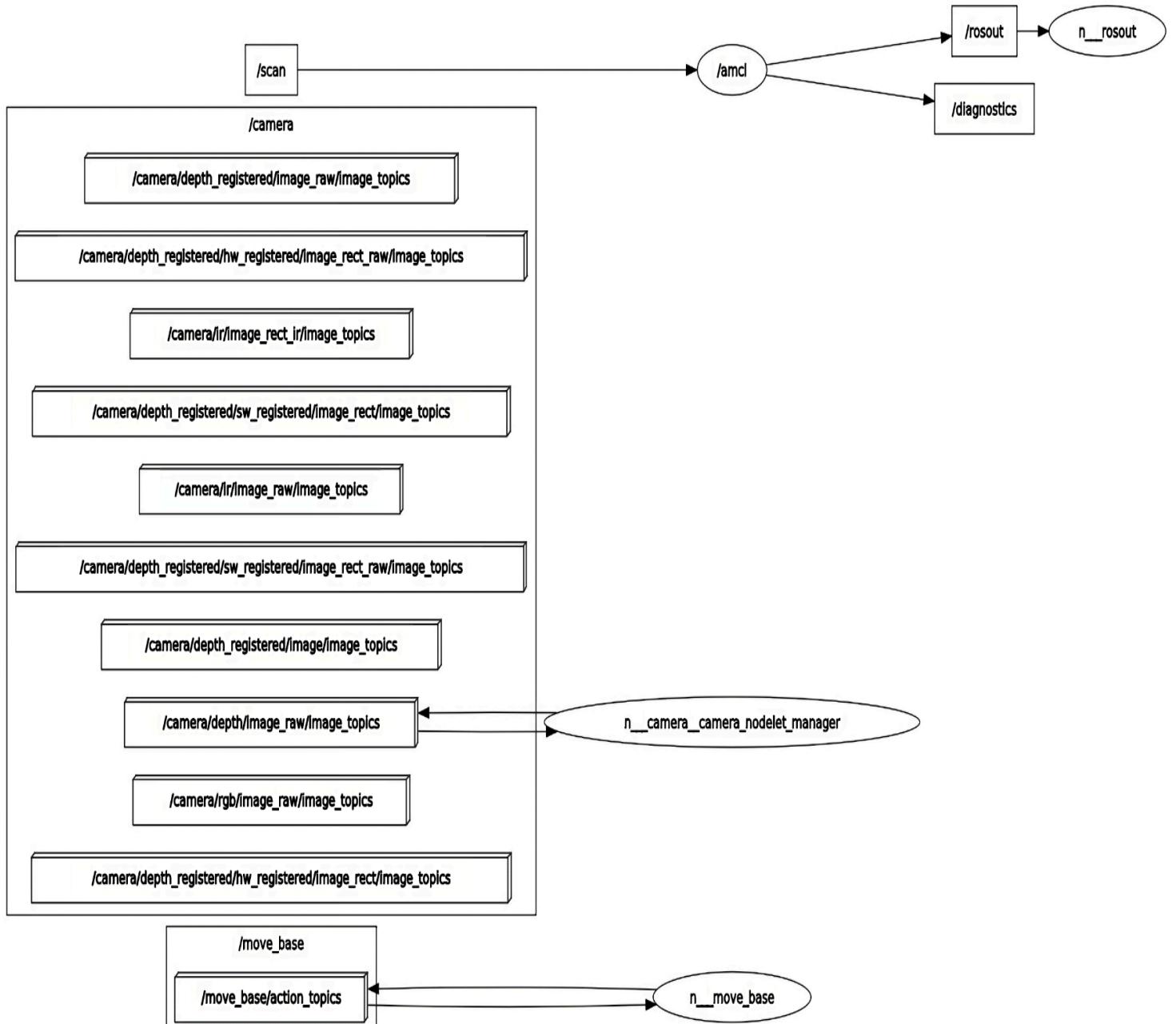


Figure 4.6: ROS Nodes Graph.

4.8 Web Application Implementation

4.8.1 MQTT Protocol

MQTT is a lightweight open messaging protocol, which employs a publish/subscribe communication pattern used for machine-to-machine communication, it supports messaging between devices to the cloud and the cloud to the device[47].

It works as below:

- The MQTT client establishes a connection with the MQTT broker.
- Once connected, the client can either publish messages, subscribe to specific messages, or do both.
- When the MQTT broker receives a message, it forwards it to subscribers who are interested.

A Python script to send the data from the ROS node to the application using this protocol was written. This code is available in Appendix B.

4.8.2 User Interface

The project integrates PHP and SQL for the server side, JavaScript, CSS, and HTML for the client side to create a dynamic application. Additionally, MQTT protocol is employed to facilitate communication with the robot, the system contains three main pages:

- Sign In page shown in Figure 4.7: The entry point for users to access the system. Users can be authenticated using their credentials, and administrators without an account can create a new one.



Figure 4.7: Sign in page.

- Welcome page in Figure 4.8: The central hub following a successful sign-in, providing users with access to key functionalities and navigation to the final page containing the map and additional details

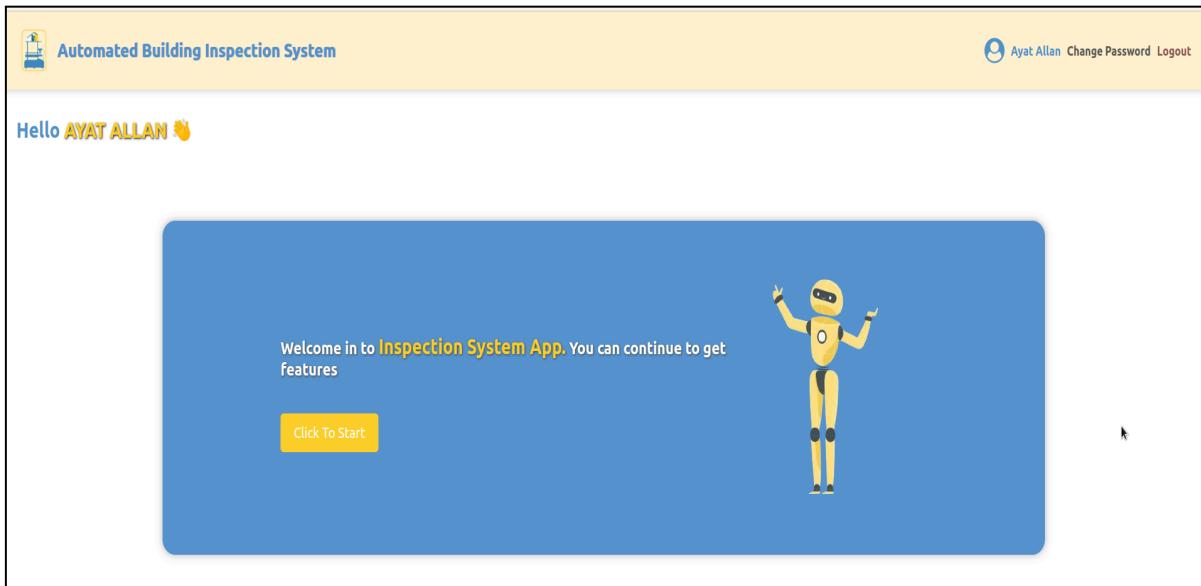


Figure 4.8: Welcome page.

- Map and defects page in Figure 4.9: It is the final page containing the building's map, highlighting various defects identified during the inspection. To enhance clarity, a colour-coded key has been provided to specify the types of defects and their corresponding representations on the map, **yellow dot** for Structural Defects and **blue dot** for Cracks.



Figure 4.9: Map and defects page.

The previously saved map, created using Gmapping algorithm, is uploaded to the application as a PGM file. When the robot navigates within the mapped area, it starts searching for defects on the room walls. If a defect is detected , the robot transmits the (x, y) coordinates of the defect to the application. To ensure accurate defect positioning, these coordinates are converted from metres to pixels using the scaling factor of 35.37 (Scale_meter_to_pixel). Additionally, determining the robot's origin point involves placing it at 162 pixels from the left and 277 pixels from the top in the updated map.

The application, upon receiving the defect information, dynamically responds by drawing on the canvas of the uploaded map. The position of the defect is marked based on its scaled (x, y) coordinates, and the type of defect determines the specific representation on the canvas. For example, structural defects represented by a yellow mark, while cracks depicted with a blue mark.

It's also including six buttons:

1. Map Saving Button: Users can save the updated map, including the positioned canvases marking detected defects as shown in Figure 4.10.
2. Change Map Button: Allows users to switch between different maps.
3. Reset Button: Provides an option to remove all defect markings from the map.

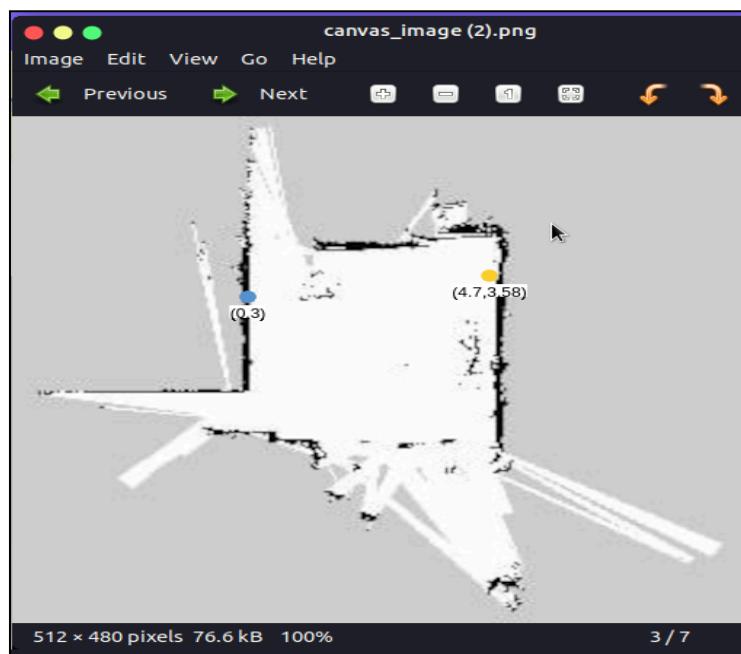


Figure 4.10: Map with defects.

4. Generate PDF Button: Generating a downloadable PDF report with the inspection details includes defect locations, a map representation, and the current date and time using the html2pdf library, this report shown in Figure 4.11.



Figure 4.11: Generated pdf report.

5. Connect / Disconnect Buttons: The "Connect" button establishes communication between the web app and the robot via the MQTT protocol, while the "Disconnect" button terminates this communication.

4.8.3 Deployment Process on AWS Lightsail

The project is deployed using AWS Lightsail service, selected for its strong security protocols. Data is automatically encrypted at rest and in motion for increased security, it's completely managed by AWS, the service provides integrated backup features, enhancing reliability in the hosting environment.

1. Instance Creation: create a new lightsail instance by selecting the desired instance specifications and configuration options, then create a new key pair that is used for SSH access to the instance, then launch it as shown in Figure 4.12, after that create a MySQL managed database on it as shown in Figure 4.13.

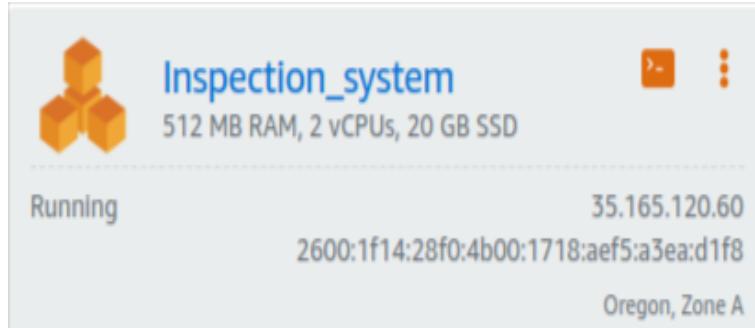


Figure 4.12: Launched Lightsail instance.

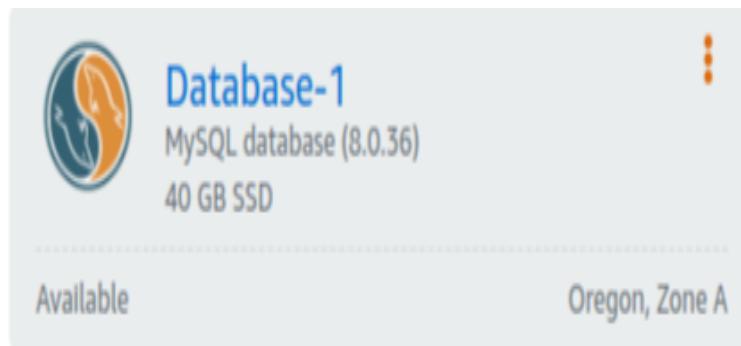


Figure 4.13: Created Database.

The instance can be accessed via HTTP and HTTPS using its IP address. However, data transmission is limited to HTTP when using the MQTT protocol.

2. Files Upload: FileZilla is used to transfer the project files from the local machine to the instance.

3. Install MySQL Server: after use SSH to connect to the instance, install the server by applying this command:

```
$ sudo apt install mysql-server
```

Then the MySQL shell will be accessible to create a new database, after that in the server-side code establish a connection to a MySQL database hosted on Amazon RDS as shown in Figure 4.14.

```
$host = 'ls-c518f8b51062a200e07267aba15aa47cef9f0c81.clui2eio8yhz.us-west-2.rds.amazonaws.com';
$database_name = 'inspectionSys';
$username = 'inspection';
$password = 'ayatallan123';

$connect = new mysqli($host, $username, $password, $database_name);
```

Figure 4.14: Remote database connection.

In AWS Lightsail, user passwords are stored in hashed representations as shown in Figure 4.15. This cryptographic technique significantly enhances security to prevent unauthorized access. Additionally, It employs key pairs for secure access to the instance, consisting of a public key for accessing it and a private key securely held by the user. This asymmetric encryption method strengthens overall security, thus ensuring the confidentiality and integrity of user data within the database[48].

```
MySQL [inspectionSys]> SELECT * FROM users;
+----+-----+-----+-----+
| id | fname | lname | loginname | password |
+----+-----+-----+-----+
| 193 | Ayat | Allan | ayat | 40bd001563085fc35165329ea1ff5c5ecbdbbeef |
| 194 | Yara | Sarahna | Yarasa | 22dace98fac01ff897bf82dfbfe8f2a2a9cae4 |
| 195 | Raghad | Samoor | raghad | 40bd001563085fc35165329ea1ff5c5ecbdbbeef |
+----+-----+-----+-----+
3 rows in set (0.001 sec)

MySQL [inspectionSys]> █
```

Figure 4.15: Remote database password hashing.

4.9 Implementation Issues

1. Kinect Camera Identification:

We faced issues with Kinect connection, it's using a different ID than the default one, causing another device to connect to the USB port instead. Resolved this by adjusting the device's default ID to match the designated Kinect ID within the OpenNI file .

`<arg name="device_id" default="#2" />` sets the device ID for the camera. The value is set to "#2", which suggests using the second device found. It can be specified in various formats (serial number, bus, and address) to uniquely identify the camera device.

2. Velocity Adjustment for Defect Detection:

During navigation robot motion was fast. So it's hard for the robot to detect the defect. This issue was solved by reduction of robot speed by changing the maximum x velocity (`Max_vel_x =0.55`) for the robot to `0.1m/s`.

3. Camera-Robot Synchronization:

Faced delays between camera input and robot movement. Successfully fixed the synchronization gap, improving real-time coordination by adding this line of code .

```
if rospy.Time.now() - self.last_image_time < self.min_interval:  
    return  
self.last_image_time = rospy.Time.now()
```

4. Defect Position Correction:

We faced a problem where the '`odom`' topic was displaying the robot's position instead of the defect's position. So we used the Kinect depth sensor to calculate the closest point coordinates to the robot when it detected a defect, then added these coordinates to the robot's actual position. This adjustment ensures accurate defect positioning in the x and y coordinates.

5. OpenNI and AMCL Conflict:

Facing a conflict between `OpenNI` and `AMCL` packages while subscribing to image topic '`/camera/rgb/image_color`' which is needed to run OpenNI node, So we used the '`/camera/rgb/image_raw`' topic instead. This topic is already functional within the `amcl_nav` package and does not necessitate the operation of OpenNI.

Chapter 5: Testing

5.1 Overview

This chapter explains the project component testing methodology and displays the project system implementation outcomes.

5.2 ROS Installation

You can check the installed version by opening a terminal and running:

```
roscore
```

ROS is installed correctly if you see: *started core service [/rosout]*

5.3 Testing The TurtleBot

Make sure the turtlebot is connected to the laptop using the cables and connectors. Check the power connection and ensure that the robot is receiving power. After run the following command, a sound will be emitted to indicating power connection :

```
roslaunch turtlebot_bringup minimal.launch
```

5.3.1 Testing Motor In Turtlesim

Run a python script for driving turtlebot in linear motion such as moving in a square or zigzag path, angular like moving in a circle, the Figure 5.1 shown applying the same script in Turtlesim.

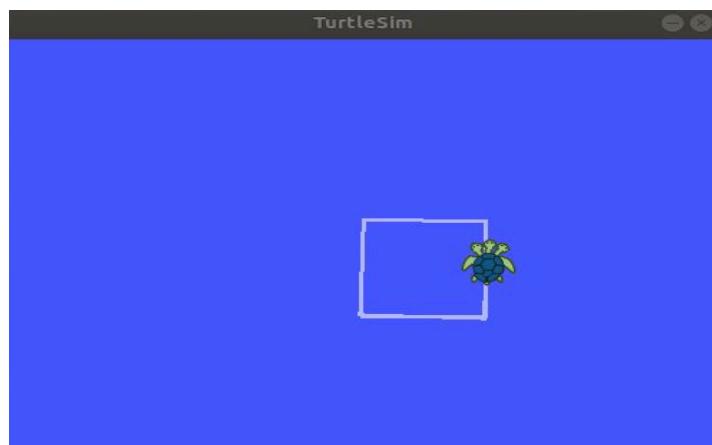


Figure 5.1 :Turtlesim robot simulation.

5.4 Testing The Kinect

5.4.1 Default 3D sensor

check the default 3D sensor of TurtleBot by printing an environment variable and confirm the output:

```
$ echo $TURTLEBOT_3D_SENSOR  
# Output: kinect
```

5.4.2 Test OpenNI Driver

Launch the OpenNI driver for ROS, initializing the OpenNI camera and making its data available for further processing in the ROS ecosystem.

```
$ rosrun openni_launch openni.launch
```

5.4.3 Test Kinect Stream

Open a new terminal, and check the list of topics being published:

```
$ rostopic list
```

This command will display the RGB camera data being published on that topic in the terminal as shown in Figure 5.2:

```
$ rostopic echo /camera/rgb/image_raw
```

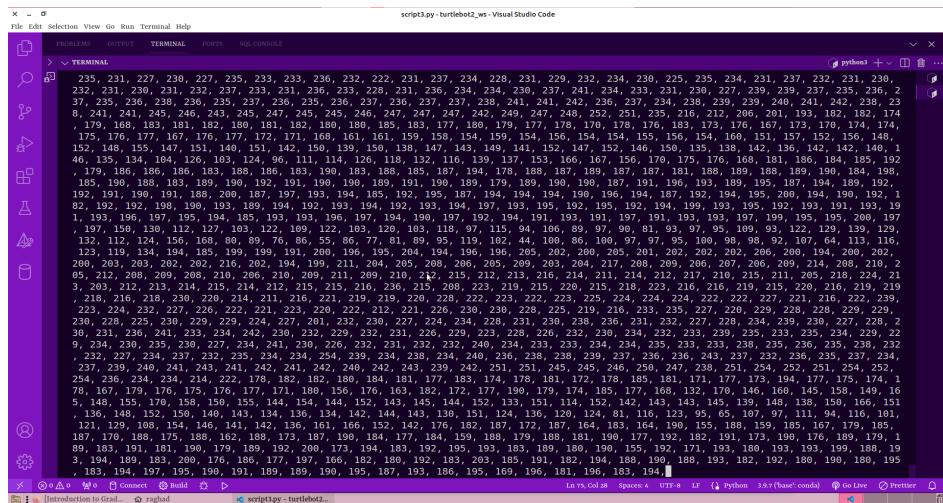


Figure 5.2: RGB camera data stream.

5.4.4 Test Image Data

Using the following command will display a live video stream from the Kinect:

```
rosrun image_view image_view image:=/camera/rgb/image_color
```

5.4.5 Test Depth Data

This command test of depth data and showed visual representation of distance where objects closer to the Kinect appear darker as shown in Figure 5.3:

```
rosrun image_view image_view image:=/camera/depth/image
```

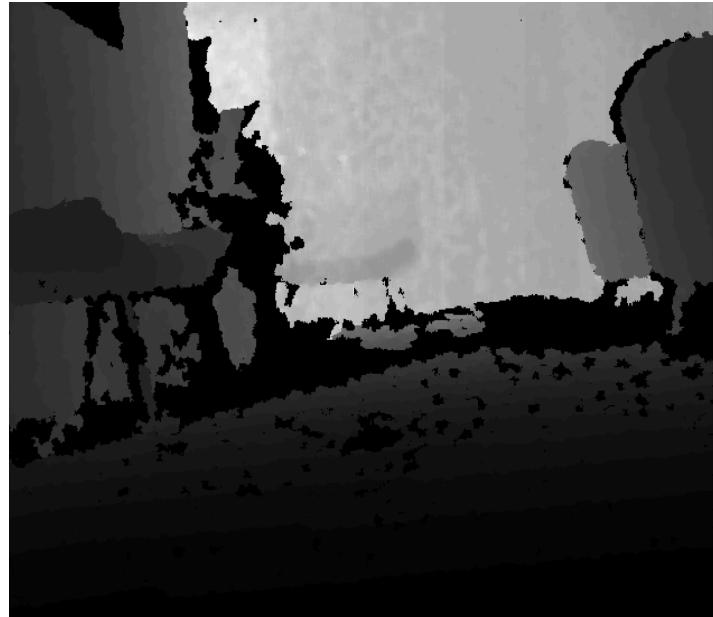


Figure 5.3: Test Kinect depth data.

5.4.6 Autonomous Motion and Avoiding Obstacles

A self-motion test was executed to verify the robot's correct movement. A Python script for obstacle avoidance during movement was written. This code is available in Appendix A.

5.5 Testing Machine Learning Model

After training the mode using Teachable Machine, this model is utilized in a python script available in Appendix C to perform real-time image classification for wall's defects .

Running the script provides the predicted classes with a clear output as shown in Figure 5.4:

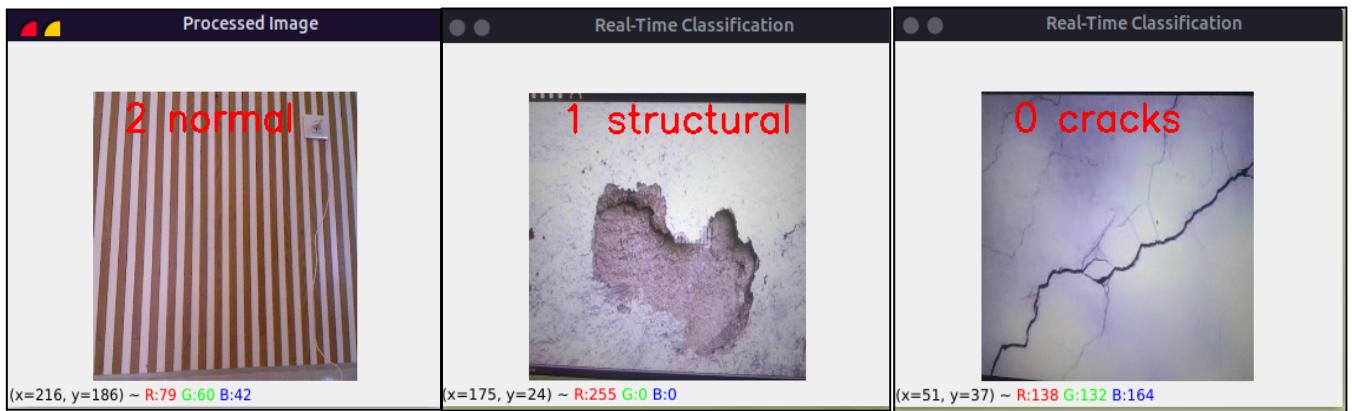


Figure 5.4: Test the ML model.

Orange tool that shown in Figure 5.5 used to evaluate the accuracy of the model by drag and drop components (called widgets) to create a data analysis pipeline, This tool generates a detailed confusion matrix for each class. The model gives 95% of accuracy in classifying different structural defect categories.

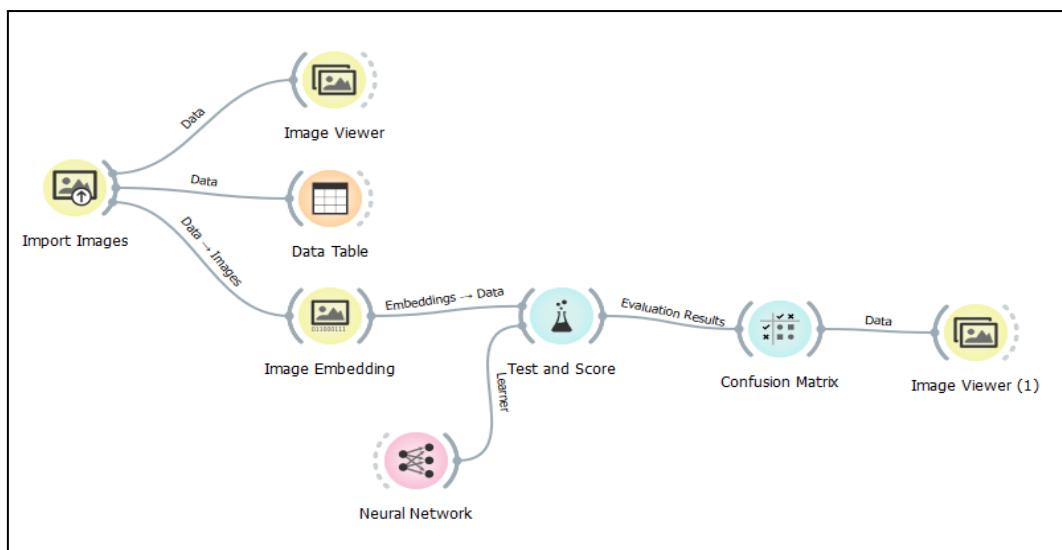


Figure 5.5: Cross-validated calibration plot.

The overall accuracy for the ML model is shown in Figure 5.7 and as it shown in Figure 5.6 the confusion matrix display the accuracy for each class as follows:

- Cracks - 93.5%
- Normal - 98.8%
- Structural - 89.5%

		Predicted			
		Cracks	Normal	Structural defects	Σ
Actual	Cracks	247	2	15	264
	Normal	2	350	2	354
	Structural defects	10	7	146	163
Σ		259	359	163	781

Figure 5.6: Confusion matrix.

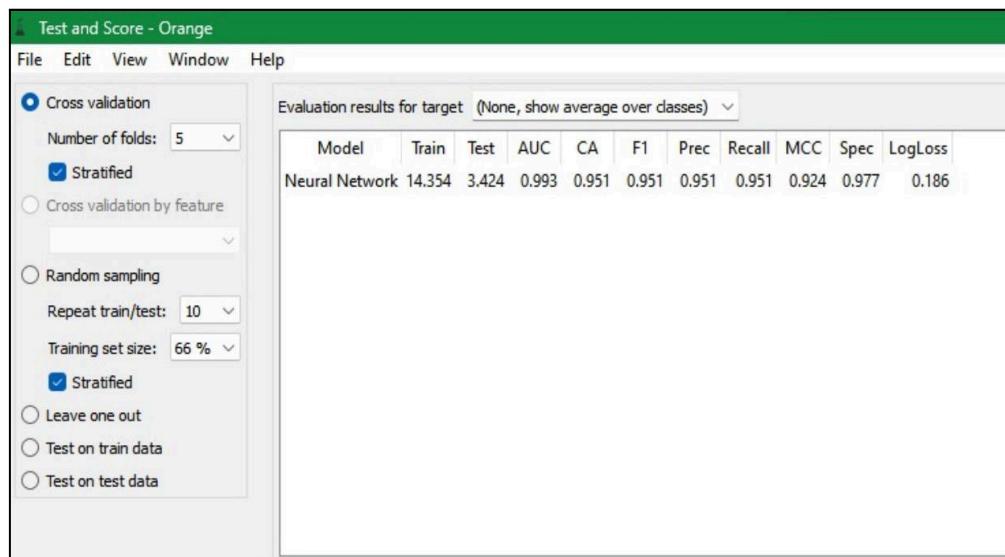
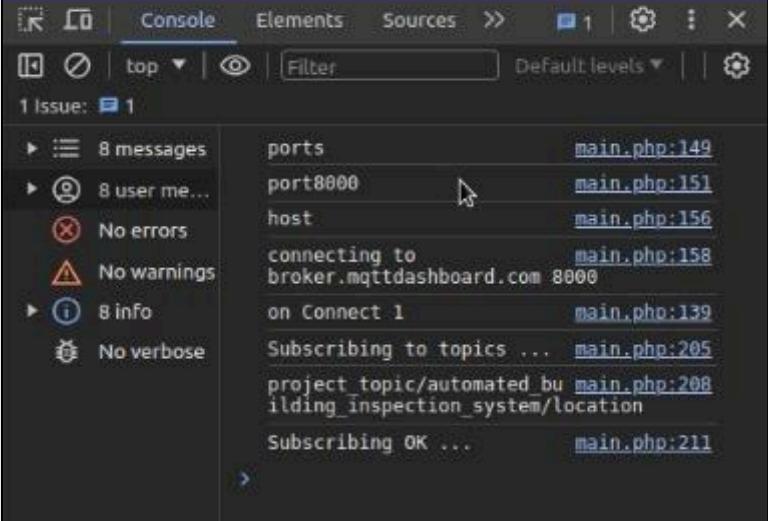


Figure 5.7: Result of testing and scoring the model.

5.6 Testing The Web App

After successfully establishing a connection with the Lightsail instance at IP address 35.165.120.60, the application connects with the robot. Using the MQTT protocol for data transmission, the console logs and displays information about the connection status and the defects.

The application successfully established a connection with the broker and subscribed to the topic "project_topic/automated_building_inspection_system/location" as shown in figure 5.8:

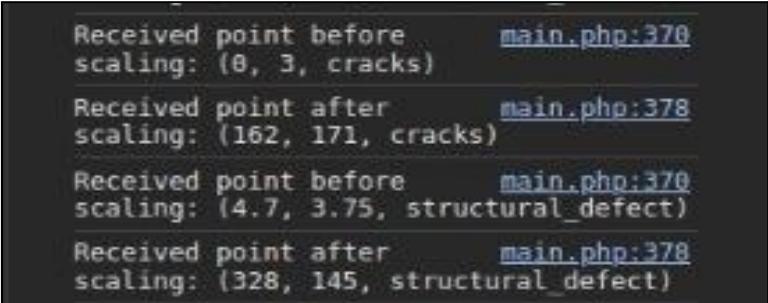


A screenshot of a browser's developer tools Console tab. The tab title is 'Console'. Below it are buttons for 'Elements', 'Sources', and a search bar with 'Filter' and 'Default levels' dropdowns. A message '1 Issue: 1' is displayed. The log area shows the following entries:

```
ports main.php:149
port8000 main.php:151
host main.php:156
connecting to main.php:158
broker.mqttdashboard.com 8000
on Connect 1 main.php:139
Subscribing to topics ... main.php:205
project_topic/automated_building_inspection_system/location main.php:208
Subscribing OK ... main.php:211
```

Figure 5.8: Test broker connection and topic subscription.

It effectively received scaled points through these subscriptions, providing information about defects type and its location as shown in figure 5.9:



A screenshot of a browser's developer tools Console tab showing received MQTT data points. The log area displays the following entries:

```
Received point before main.php:370
scaling: (0, 3, cracks)
Received point after main.php:378
scaling: (162, 171, cracks)
Received point before main.php:370
scaling: (4.7, 3.75, structural_defect)
Received point after main.php:378
scaling: (328, 145, structural_defect)
```

Figure 5.9: Test data transmission.

5.7 Unit Testing

Each part of the system was tested individually by connecting it to the laptop and running a specific python script. The results of these tests were recorded in Table 5.1.

Table 5.1: Software Testing Table.

Case	Result
Generate 2D map of the building structure	2D map with nearly the same real building structure.
Navigate the mapped area autonomously	Robot moved smoothly and avoided obstacles with the derived path in the saved map.
Identify the defects and its type.	Robot can detect the defect and identify its type with 93.5% accuracy for cracks and 89.5% for structural defects.
Locate defect's position in the map.	Robot can exactly locate the defect position, convert to pixels, and display it on the scaled map with its type.
Connect to MQTT	Robot connecting to the web server by MQTT.
Send necessary data to the web application.	Map, defects location and its type sent successfully to the application.
Provide comprehensive report	Users can easily get and read the building status using a pdf report.

5.8 System Validation

In this section, we check the complete system component to make sure it meets the functional requirements under various test scenarios. These include:

Case 1: The robot navigates through an environment containing images of defects. It faces difficulty detecting defects, when the light is inadequate at certain points, also when the robot moves fast.

Case 2: The robot navigated one more time and it was able to detect the defects type, but it printed its own position instead of transmitting the defects' positions. This resulted in printing incorrect data on the map as shown in Figure 5.10:

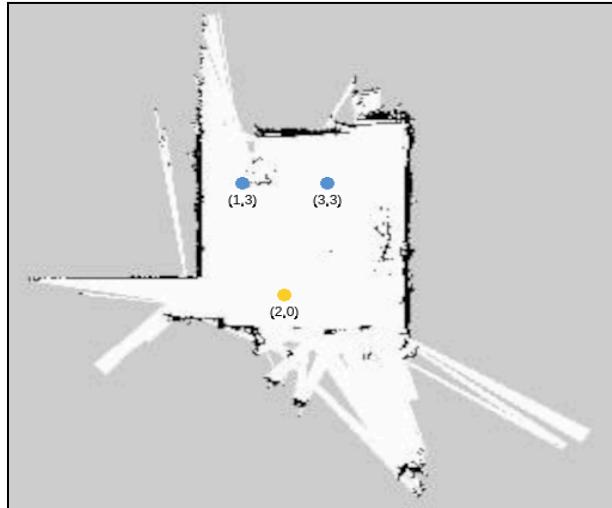


Figure 5.10: Robot position transmission.

Case 3: The robot now navigates on the same environment and it was able to detect the defects type and its location, but it's transmitted redundant data for each identified defect, leading to generate multiple close points and it goes out of the scaled map, as shown in Figure 5.11:

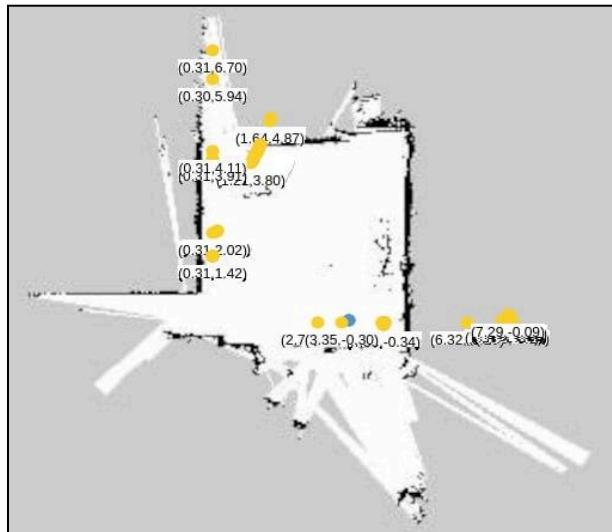


Figure 5.11: Redundant data transmission.

Case 4: the robot navigates again, after many attempts, successfully drew a map of where it is located and the detected defects are mapped correctly with its coordinates on the scaled map in the web application as shown in Figure 5.12, However, the transmitted data includes non-defective objects due to machine learning model accuracy.

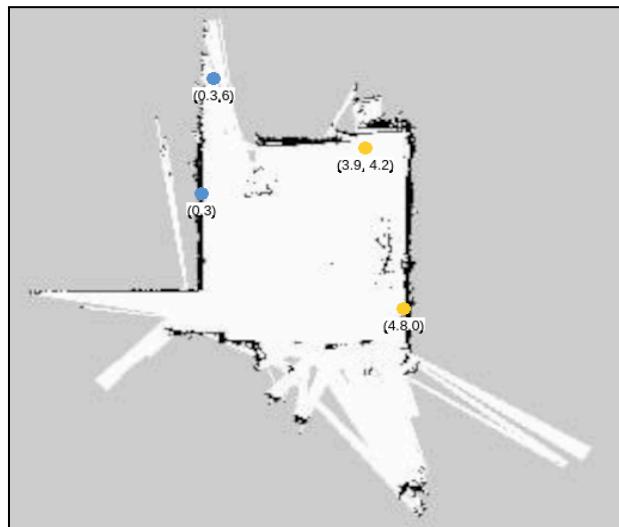


Figure 5.12: Defects localization results.

Chapter 6: Conclusion and Future work

6.1 Conclusion

In this project, we have presented an automated inspection system that combines robotics, sensor technology, computer vision, and machine learning to create a mobile robot system capable of autonomously inspecting building walls. The system's output includes a detailed 2D map, defect identification, and a clear report accessible by the user using a web application.

6.2 Future work

In the future development of this project, several enhancements can be implemented to improve the system's functionality, this considerations include:

- Integrating ultrasonic sensors to prevent fall down the stairs.
- Enabling outdoor functionality to inspect outer walls of buildings.
- Train the ML model with a larger dataset. to enhance the ability to detect more structural defects types.

Appendices

Appendix A

Robot path with Obstacle Avoidance Script:

```
import rospy
from move_base_msgs.msg import MoveBaseAction, MoveBaseGoal
class GoForwardAvoid():
    def __init__(self):
        rospy.init_node('nav_test', anonymous=False)
        rospy.on_shutdown(self.shutdown)
        self.move_base = actionlib.SimpleActionClient("move_base",
                                                      MoveBaseAction)
        rospy.loginfo("wait for the action server to come up")
        self.move_base.wait_for_server(rospy.Duration(5))
        goal = MoveBaseGoal()
        goal.target_pose.header.frame_id = 'base_link'
        goal.target_pose.header.stamp = rospy.Time.now()
        goal.target_pose.pose.position.x = 3.0
        goal.target_pose.pose.orientation.w = 1.0
        self.move_base.send_goal(goal)
        success = self.move_base.wait_for_result(rospy.Duration(60))
        if not success:
            else:
                state = self.move_base.get_state()
                if state == GoalStatus.SUCCEEDED:
                    rospy.loginfo("Hooray, the base moved 3 metres forward")
                if __name__ == '__main__':
                    try:
                        GoForwardAvoid()
                    except rospy.ROSInterruptException:
                        rospy.loginfo("Exception thrown")
```

Appendix B

Connect MQTT protocol :

```
import time
import paho.mqtt.client as paho
from paho import mqtt
import threading
import random

def on_connect(client, userdata, flags, rc, properties=None):
    print("CONNACK received with code %s." % rc)
    if rc == 0:
        print("Connected to MQTT Broker!")
    else:
        print("Failed to connect, return code %d\n", rc)
def on_publish(client, userdata, mid, properties=None):
    print("publish OK => mid: " + str(mid))
def on_subscribe(client, userdata, mid, granted_qos,
properties=None):
    print("Subscribed Ok => mid: " + str(mid) + " " +
str(granted_qos))
def on_message(client, userdata, msg):
    subscribe_function(str(msg.topic),str(msg.payload))
client_name_id="set_here_cliente_id_"+str(random.randint(1,
100))+ "_" +str(random.randint(1, 100))
client = paho.Client(client_id=client_name_id, userdata=None,
protocol=paho.MQTTv31)
client.on_connect = on_connect
client.connect("mqtt-dashboard.com", 1883)
client.on_subscribe = on_subscribe
client.on_message = on_message
client.on_publish = on_publish
client.subscribe("any_topic_here_to_subsecribe/in/our/project",
qos=1)
def subscribe_function(topic,msg):
    print("topic:"+topic+ " msg:" +msg)

def publish_mqtt(topic,msg):
    client.publish(topic, payload=msg, qos=1)

def demo_test_publish():
```

```
while 1:  
    msg="0,3,structural_defect"  
    msg2="4.7,3.58,cracks"  
publish_mqtt("project_topic/automated_building_inspection_system  
/location",msg)  
    time.sleep(2)  
publish_mqtt("project_topic/automated_building_inspection_system  
/location",msg2)  
    time.sleep(2)  
t1 = threading.Thread(target=lambda :client.loop_forever(),  
args=())  
t1.start()  
demo_test_publish()
```

Appendix C

ML model working on Kinect RGB stream

```
import rospy
from sensor_msgs.msg import Image
from cv_bridge import CvBridge
import cv2
import numpy as np
from tensorflow import keras
from keras.preprocessing import image
# Load the model from the desktop
model =
keras.models.load_model('/home/raghad/turtlebot2_ws/src/my_kinec
t_stream/keras_model.h5')

# Compile the model (you can adjust the optimizer, loss, and
metrics based on your original compilation)
model.compile(optimizer='adam', loss='categorical_crossentropy',
metrics=['accuracy'])
# Load the class labels from the desktop
with
open('/home/raghad/turtlebot2_ws/src/my_kinect_stream/labels.txt
', 'r') as file:
    class_labels = [line.strip() for line in file]

# Initialize the CvBridge
bridge = CvBridge()
# Define a fixed region of interest (ROI) within the image
roi_x = 100 # X-coordinate of the top-left corner of the ROI
roi_y = 100 # Y-coordinate of the top-left corner of the ROI
roi_width = 224 # Width of the ROI
roi_height = 224 # Height of the ROI

# ROS Subscriber for the Kinect RGB image
def image_callback(msg):
    try:
        # Convert the ROS Image message to an OpenCV image
        frame = bridge.imgmsg_to_cv2(msg, desired_encoding="bgr8")

        # Extract the fixed region of interest (ROI) from the image
        roi = frame[roi_y:roi_y + roi_height, roi_x:roi_x +
roi_width]
```

```

# Preprocess the ROI for prediction
roi = cv2.resize(roi, (224, 224))
img_array = image.img_to_array(roi)
img_array = np.expand_dims(img_array, axis=0)
img_array = img_array / 255.0 # Normalize the image data

# Make a prediction using the loaded model
predictions = model.predict(img_array)

# Get the class with the highest probability
predicted_class_index = np.argmax(predictions[0])
predicted_class = class_labels[predicted_class_index]

# Print the predicted class
print(f"Predicted class: {predicted_class}")

# Display the frame with the predicted class name
cv2.putText(frame, f" {predicted_class}", (10, 30),
cv2.FONT_HERSHEY_SIMPLEX, 1, (0, 0, 255), 2)

# Show the processed image (optional)
cv2.imshow("Processed Image", frame)
cv2.waitKey(1)

except Exception as e:
    print(f"Error processing image: {e}")

def main():
    rospy.init_node('image_classifier_node', anonymous=True)

    # ROS Subscriber for the Kinect RGB image
    rospy.Subscriber('/camera/rgb/image_raw', Image,
image_callback)

    rospy.spin()

if __name__ == "__main__":
    main()

```

References

- [1] mathworks.com, What Is SLAM (Simultaneous Localization and Mapping) – MATLAB & Simulink, Date accessed:May 21, 2023, Available from:<https://www.mathworks.com/discovery/slam.html>.
- [2] Medium, Localization, Mapping and SLAM using GMapping, Date accessed:October 28,2023, Date published:December 28, 2022, Available from:<https://softillusion-robotics.medium.com/gmapping-d26c13b1b69>.
- [3] ieeexplore.ieee.org, Localization and Navigation for Indoor Mobile Robot Based on ROS, Date accessed :May 21, 2023, Available from:<https://ieeexplore.ieee.org/abstract/document/8623225/>.
- [4] [cs.cmu.edu](https://www.cs.cmu.edu/~rasc/Download/AMRobots5.pdf), Mobile robot localization, Date accessed:November 12,2023; Available from: <https://www.cs.cmu.edu/~rasc/Download/AMRobots5.pdf> .
- [5] researchgate, Robot navigation with obstacle avoidance in unknown environment, Date accessed: October 28, 2023, Available from: https://www.researchgate.net/publication/328774686_Robot_navigation_with_obstacle_avoidance_in_unknown_environment.
- [6] researchgate, Robot navigation with obstacle avoidance in unknown environment , Date accessed: October 28, 2023, Available from: https://www.researchgate.net/publication/328774686_Robot_navigation_with_obstacle_avoidance_in_unknown_environment.
- [7] control.com, An Introduction to Simultaneous Localization and Mapping (SLAM) for Robots, Date accessed: January 9, 2024 ,Available from: <https://control.com/technical-articles/an-introduction-to-simultaneous-localization-and-mapping-slam-for-robots/> .
- [8] weka.io,Machine Learning vs. Neural Networks (Differences Explained), Date accessed:May 21, 2023, Date published:May 01, 2023, Available from: <https://www.weka.io/learn/ai-ml/machine-learning-vs-neural-networks/>
- [9] neptune.ai, Image Processing in Python: Algorithms, Tools, and Methods You Should Know, Date accessed:May 22, 2023, Available from:<https://neptune.ai/blog/image-processing-python>.
- [10] Y. Sun, A Comparative Study of Machine Learning Algorithms for Automated Defect Detection in Building Structures, Journal of Building Engineering, Volume 33, 2021, 101816, ISSN 2352-7102, <https://doi.org/10.1016/j.jobe.2020.101816>.
- [11] S. Chowdhury, K.D. Saha, C.M. Sarkar, O.V. Gnana Swathika,Smart Buildings Digitalization,Boca Raton,CRC Press,2022

- [12] A.Amleh ,F.Mohtaseb, M. Mohtaseb, An intelligent Mobile Robot that localises itself Inside a Maze,2020.
- [13] A. Shaheen, M. E'mar,Storing and Classification Robot, 2013.
- [14] H.Dwaik, M.Hrebat, M.Natsheh, palestine polytechnic university, Object Finder Robot , 2023 .
- [15] clearpathrobotics.com, Jackal UGV - Small Weatherproof Robot - Clearpath, Date accessed:May 23, 2023,Date published:December 20, 2022, Available from:<https://clearpathrobotics.com/jackal-small-unmanned-ground-vehicle/>.
- [16] warf.com,TURTLEBOT 2 COMPLETE KIT, Date Accessed: 21 May 2023,Available from :http://www.warf.com/view.TURTLEBOT_2_COMPLETE_KIT-10162.html .
- [17] allaboutcircuits.com,Teardown Tuesday: Microsoft's Xbox 360 Kinect, Date Accessed: December 3, 2023, Available From <https://www.allaboutcircuits.com/news/teardown-tuesday-microsofts-xbox-360-kinect/>.
- [18] clearpathrobotics.com, Jackal UGV - Small Weatherproof Robot - Clearpath , Date accessed: January 9, 2024 ,Available from: <https://clearpathrobotics.com/jackal-small-unmanned-ground-vehicle/>.
- [19] clearpathrobotics.com, TurtleBot 2 - Open source personal research robot, Date accessed: January 9, 2024 ,Available from: <https://clearpathrobotics.com/turtlebot-2-open-source-robot/>.
- [20] raspberrypi.com, Buy a Raspberry Pi 3 Model B, Date accessed: January 9, 2024 , Available from: <https://www.raspberrypi.com/products/raspberry-pi-3-model-b/> .
- [21] wikipedia.org, Laptop, Date accessed : January 5, 2024, Available from : <https://en.wikipedia.org/wiki/Laptop>.
- [22] Raspberrypi-spy.co.uk, Introducing the Raspberry Pi 3 B+ Single Board Computer, Date accessed : February 12, 2024 available from : <https://www.raspberrypi-spy.co.uk/2018/03/introducing-raspberry-pi-3-b-plus-computer/>.
- [23] A. Shaheen, M. E'mar,Storing and Classification Robot, 2013.
- [24] researchgate.net, Technical specifications of the Kinect v2, Date accessed : January 5, 2024,Available from:https://www.researchgate.net/figure/Technical-specifications-of-the-Kinect-v2_tbl1_321048476.
- [25] Logitech.com, 4K PRO MAGNETIC WEBCAM, Date accessed : January 5, 2024, Available from: <https://www.logitech.com/en-us/product/4k-pro-magnetic-webcam>.

- [26] Konga.com, Microsoft Xbox 360 Kinect Sensor | Konga Online Shopping, Date accessed : January 5, 2024, Available from:<https://www.konga.com/product/microsoft-xbox-360-kinect-sensor-2473996>.
- [27] jameco.com, How It Works: Xbox Kinect - Jameco Electronics, Date accessed:December 3,2023, Available from :<https://www.jameco.com/Jameco/workshop/Howitworks/xboxkinect.html>.
- [28] terabee.com, Depth sensors: Precision & personal privacy, Date accessed:January 4, 2024, Available from : <https://www.terabee.com/depth-sensors-precision-personal-privacy/>.
- [29] Researchgate.net, Features of Kinect Sensor V2, Date accessed:January 5, 2024, Available from : https://www.researchgate.net/figure/Features-of-Kinect-Sensor-V2_tbl1_318646867.
- [30] Researchgate.net, Characteristics of the lidar sensor ,Date accessed:January 5, 2024, Available from : https://www.researchgate.net/figure/Characteristics-of-the-lidar-sensor_tbl1_340344846.
- [31] Researchgate.net, 2D Laser scanners (left) URG-04LX (middle) RPLidar A1 (right) UTM-30LX, Date accessed:January 5, 2024, Available from : https://www.researchgate.net/figure/2D-Laser-scanners-left-URG-04LX-middle-RPLidar-A1-right-UTM-30LX_fig1_362316600 .
- [32] allaboutcircuits.com,Teardown Tuesday: Microsoft's Xbox 360 Kinect, Date Accessed: December 3, 2023, Available From <https://www.allaboutcircuits.com/news/teardown-tuesday-microsofts-xbox-360-kinect/.>
- [33] ijste.org, Indoor SLAM using Kinect Sensor, Date accessed:January 6, 2024, Available from:<https://www.ijste.org/articles/IJSTEV2I10306.pdf>.
- [34] codeinstitute.net,What is the Difference Between Web App & Mobile App? ,Date accessed:January 10, 2024, Available from:<https://codeinstitute.net/global/blog/web-app-vs-mobile-app/>.
- [35] techtarget.com, Mobile website vs. app: What's the difference?, Date accessed:January 10, 2024, Available from:<https://www.techtarget.com/whatis/feature/Mobile-website-vs-app-Whats-the-difference>
- [36] wiki.ros.org, ROS Noetic Ninjemys, Date Accessed: December 3, 2023, Available From:<https://wiki.ros.org/noetic>
- [37] docs.mrpt.org, MRPT, Date accessed:May 23, 2023, Date published:May 22, 2023, Available from:<https://docs.mrpt.org/reference/latest/index.html>.
- [38] iaacblog.com, APPLICATION OF SMALL SCALE ROBOTS IN THE CONSTRUCTION – Applied Theory, Date accessed:January 10, 2024, Available

from:<https://www.iacblog.com/programs/application-of-small-scale-robots-in-%CF%84%CCE%B7%CE%B5-construction-site/>.

[39] Medium, Preprocess The Images Using Python Opencv, Date accessed: October 28, 2023, Date published:December 28, 2022, Available from:[Preprocess The Images Using Python Opencv](#).

[40] Google. (Date Accessed: 21 May 2023). Available at: [teachablemachine](#).

[41] Ubuntu releases, Ubuntu MATE 20.04.6 LTS (Focal Fossa), Date accessed:October 20, 2023, Available from:[ubuntu-mate releases](#).

[42] ros.org, ROS, Date accessed: October 20, 2023, Available from:[ros](#).

[43] ros.org, Turtlebot Installation, Date accessed: October 20, 2023, Available from:[Turtlebot Installation](#).

[44] Medium, ROS Autonomous SLAM using Randomly Exploring Random Tree (RRT), Date accessed:October 20, 2023, Available from: [ROS Autonomous SLAM](#).

[45] wiki.ros.org, move_base, Date accessed:January 6, 2024, Available from: http://wiki.ros.org/move_base.

[46] Medium,Train image classification model using Teachable Machine, Date accessed: October 28, 2023 Date published:September,2,2022, Available from:[Train image classification model using Teachable Machine](#).

[47] aws.amazon.com,What is MQTT, Date accessed: January 10, 2024, Available from:<https://aws.amazon.com/what-is/mqtt/>.

[48] repost.aws, Encryption of LightSail Instances, Date accessed: february 18, 2024, Available:https://repost.aws/questions/QUisyn04kSS_2oOxwRchpRAA/encryption-of-lightsail-instances.