

THE UNIVERSITY OF DANANG
UNIVERSITY OF SCIENCE AND TECHNOLOGY
FACULTY OF ADVANCED SCIENCE AND TECHNOLOGY



CAPSTONE PROJECT REPORT

Autonomous Robot for Temperature Monitoring Using a Thermal Camera and SLAM-based Localization with LiDAR

Students:	Huynh Ngoc Lam	Class:20PFIEV2
	Tran Duc Cong	Class:20PFIEV2
	Nguyen Duy Minh	Class:20PFIEV2

Supervisor:	Dr. Nguyen Le Hoa
	Mr. Mai Hai An

Danang, 06/2025

ABSTRACT

Abstracts: This project presents an autonomous mobile robot designed for temperature monitoring using a thermal camera and LiDAR-based SLAM. The system enables real-time thermal imaging to detect heat anomalies and uses SLAM algorithms for self-localization and mapping in unknown environments. Implemented with ROS 2, the robot can navigate and monitor indoor spaces autonomously, making it suitable for applications such as fire prevention, equipment inspection, and environmental assessment. Experimental results confirm the system's effectiveness in accurate thermal detection and reliable autonomous navigation.

WORK CONTRIBUTION

Student name	Content
Huynh Ngoc Lam	- Designed and implemented the autonomous vehicle control system, including path planning, obstacle avoidance, and motor control algorithms. - Developed and integrated sensor fusion logic, combining data from IMU, GPS, and wheel encoders for accurate localization. - Conducted extensive testing of the self-driving logic in both simulated and real-world environments. - Documented the navigation and control algorithms, speed profiling, and turning logic.
Nguyen Duy Minh	- Assisted in the data collection process during field testing of the autonomous robot - Provided feedback on the design of the user interface displaying sensor information Participated in model testing and performance validation under real-world conditions Contributed to the formatting, layout, and visual presentation of the final report

Tran Duc Cong	- Led the development of the thermal and RGB camera-based object detection module. - Integrated YOLOv8 for real-time detection of key events (e.g., fire, smoking, human presence) using RGB camera, and used AMG8833 thermal camera for heat-based monitoring. - Trained custom YOLOv8 models on Roboflow and optimized them for deployment using Google Colab and ONNX format. - Developed a real-time Tkinter-based GUI for live video analysis and alert visualization. - Handled all preprocessing, data augmentation, and ONNX-based inference pipeline on Raspberry Pi 5.
---------------	--

Da Nang , June 2025

Head of Division

Supervisor

PREFACE

In recent years, the integration of robotics and intelligent sensing has become increasingly important in both research and real-world applications. Autonomous robots are now widely used for monitoring, inspection, and operations in hazardous or complex environments.

This thesis presents the development of an autonomous robot equipped with a thermal camera and 2D LiDAR sensor. The system performs real-time temperature monitoring and self-localization using LiDAR-based SLAM.

The robot is designed for tasks such as fire detection, industrial safety inspection, aiming to reduce human risk and improve efficiency in challenging conditions.

Despite the limited time frame, with sincere dedication and effort, we have completed the project in line with the objectives set out. However, some shortcomings are inevitable, and we sincerely welcome any constructive feedback from lecturers and reviewers to help improve the final result.

On this occasion, we would like to express our deepest gratitude to Mr. Nguyen Le Hoa, PhD, for his dedicated guidance, professional support, and constant encouragement throughout the project. His mentorship provided us with both a solid technical foundation and a productive working environment.

We also thank Mr. Mai Hai An, our supervisor at the company, for providing technical support and sharing practical experience during our internship and project implementation.

Lastly, we are truly thankful to all the lecturers from the Faculty of Advanced Science and Technology, who have devotedly taught and equipped us with the essential knowledge and skills that enabled us to complete this thesis.

Danang, June 19th , 2025

DECLARATION

We hereby declare that this graduation project is the result of our group's own research, design, and implementation efforts. Throughout the process of developing, assembling, and testing products, we have complied with the standards of academic integrity. All external sources of data, references, and ideas have been properly cited and acknowledged in accordance with academic guidelines.

All calculations and decisions made during the project were based on theoretical knowledge learned throughout our studies, as well as on practical experiments and testing.

We understand that real-world conditions may involve inaccuracies or deviations, and we have made every effort to reflect those realistically in the report. We take full responsibility for the content of this project and affirm our commitment to academic honesty and ethical engineering practices

Students perform:

Huynh Ngoc Lam – Tran Duc Cong – Nguyen Duy Minh

CONTENT

ABSTRACT	2
WORK CONTRIBUTION	1
PREFACE	3
DECLARATION	4
LIST OF FIGURES	viii
LIST OF TABLES	11
LIST OF ABBREVIATIONS	12
CHAPTER 1: INTRODUCTION	12
I. Urgency of the topic	13
II. Existing solutions and related commercial products	14
1. Technical solutions on the market	14
2. Some commercial products	17
3. Proposed preliminary solution	19
3.1 Expected functional objectives	20
3.2 Limitations of the topic	20
CHAPTER 2: HARDWARE ANALYSIS AND DESIGN FOR ROBOT	22
I. Overview	22
II. Tools and Equipment needed for implementation	24
1. Brushes Motor (Surpass Hobby 540)	24
2. BTS7960 43A Motor Driver Circuit	25
3. Arduino	26
4. Raspberry Pi 5	27
5. OV5647 camera	28
6. Thermal camera	29
7. LiDAR A1M8	30
8. MG996R RC Servo Motor	31
9. Power Supply	32
III. ROBOT kinematics	34
IV. Realistic model	36

CHAPTER 3: ROBOT OPERATING SYSTEM (ROS2) AND SLAM ALGORITHM IN AUTONOMOUS ROBOT	38
I. Robot Operating System (ROS 2):	38
1. Overview	38
2. Main Nodes used in the this project	39
3. COORDINATE TRANSFORMATION IN ROS2	40
II. Slam Algorithm	41
1. Overview	41
2. Navigation Stack blocks used in robot positioning and planning:	42
3. Algorithms used:	43
3.1 Mapping algorithm:	43
3.2 AMCL algorithm:	45
3.3 Robot navigation algorithm	48
3.3.1 Costmap 2D	48
3.3.2 Dijkstra's algorithm for global planner	49
3.3.3 Pure Pursuit trajectory tracking algorithm	53
III. Obtained result:	54
1. TEST ENVIRONMENT	54
2. Mapping process and map results	55
3. Locate the robot on the map	57
3. Navigation for robots	58
CHAPTER 4: INTEGRATED OV5647 CAMERA FOR VISUAL IMAGE MONITORING	62
I. Ideas and approaches	62
1. RGB image surveillance system concept using OV5647 camera	62
1.1. System Concept for RGB Image Monitoring using OV5647 Camera	62
1.2. Overall Methodology and Design Strategy	62
II. Theoretical Background	63
1. OV5647 Image Sensor	63
2. YOLOv8n Object Detection Model	65
2.1 Overview of Computer Vision and YOLOv8	65
2.2 YOLOv8 Architecture: A Deep Dive	67

2.3 Introduction to Supporting Tools for Model Training – Roboflow and Google Colab	71
2.3.1 Roboflow – A Platform for Dataset Management and Computer Vision Acceleration	71
2.3.2 Google Colaboratory (Colab) – A Cloud-based Environment for AI Model Training	72
III. System Integration Workflow	73
1 Model Preparation and Training with Roboflow and Google Colab	73
2. Hardware Integration and Video Streaming Configuration	77
3. ONNX Inference and Real-Time GUI Deployment	79
IV. Results	81
1. Training and Evaluation Metrics	81
2. Qualitative Results on Real-time Input	84
CHAPTER 5: THERMAL IMAGING MONITORING USING AMG8833 SENSOR	88
I. Ideas and Approaches	88
II. Theoretical Overview	88
1. AMG8833 Sensor	88
2. Image interpolation - resize and distort example[7]	92
III. Program Design for AMG8833 Thermal Image Processing	94
1. Software Environment Configuration	94
2. Program implementation	94
IV. Result	99
REFERENCES	105

LIST OF FIGURES

Figure 1 : IR thermal camera.....	3
Figure 2: Thermal security robot.....	4
Figure 3 : Substation monitoring robot.....	6
Figure 4: Boston Dynamics Spot.....	7
Figure 5 : Block diagram of system.....	10
Figure 6 : Surpass Hobby 540 motor.....	12
Figure 7 : BTS7960 4.....	12
Figure 8: Raspberry Pi5 8GB.....	14
Figure 9: OV5647 camera.....	15
Figure 10: Thermal camera AMG8833.....	16
Figure 11 : RP LiDAR A1	18
Figure 12 : MG996R RC Servo Motor.....	19
Figure 13 : 18650 Li-ion Battery Pack.....	20
Figure 14 : UPS HAT E.....	21
Figure 15 : Ackerman kinematic model.....	22
Figure 16 : Realistic model.....	24
Figure 17 : Nodes on ROS2.....	27
Figure 18 : TF transformation.....	28
Figure 19 : Navigation diagram on ROS2.....	30
Figure 20 : Catographer node.....	31
Figure 21 : Describe the operating principle of AMCL.....	32
Figure 22 : Localization algorithm AMCL.....	34
Figure 23 : Weight map values influence robot behavior.....	35
Figure 24 : System structure diagram on ROS2.....	36
Figure 25 : Select node.....	36
Figure 26 : Shortest distance.....	37
Figure 27 : Minimum distance.....	37

Figure 28 : Distance traveled.....	37
Figure 29 : Consider node F next.....	38
Figure 30 : Path planning using Dijkstra's algorithm.....	39
Figure 31 : Pure Pursuit trajectory tracking.....	40
Figure 32 : Test environment.....	41
Figure 33 : Robot location and initial map.	41
Figure 34 : Mapping process.....	42
Figure 35 : Final map.....	42
Figure 36 : Costmap 2D.....	43
Figure 37 : AMCL algorithm.....	44
Figure 38 : Initial position.....	45
Figure 39 : Navigation and path planning.....	45
Figure 40 : New location after moving.....	46
Figure 41 : Changing location.....	46
Figure 42 : CMOS_Bayer_Array.....	50
Figure 43 : CMOS Sensor Pixel Filtering with Bayer Pattern.....	50
Figure 44 : CMOS BayerFilter Mosaic Structure and Demosaicing Principle.....	50
Figure 45 : YOLOv8 Architecture, visualisation made by GitHub user RangeKing.....	53
Figure 46 : Fire class dataset.....	58
Figure 47 : Person class dataset.....	59
Figure 48 : Smoking class dataset.....	59
Figure 49 : Results when training model.....	60
Figure 50 : Results when exporting to ONNX.....	60
Figure 51 : Results when streaming MJPEG video frames.....	62
Figure 52 : Results when streaming to a Virtual Webcam.....	63
Figure 53 : Results displayed with Tkinter gui.....	64
Figure 54 : F1 Score vs Confidence Threshold Curve.....	65

Figure 55 : Precision, Recall, and mAP per Class.....	66
Figure 56 : Confusion Matrix of Detection Results Across Classes.....	67
Figure 57 : YOLOv8n Detection of 'Person' Class.....	68
Figure 58 : YOLOv8n Detection of 'Fire' Class.....	68
Figure 59 : YOLOv8n Detection of 'Smoking' Class.....	69
Figure 60 : Circuit diagram of AMG8833.....	72
Figure 61 : AMG8833 and Raspberry Pi Connection.....	74
Figure 62 : Interpolation example.....	75
Figure 63 : Operation diagram of the temperature measurement system with AMG8833.....	77
Figure 64 : Program Flowchart AMG8833.....	78
Figure 65 : GUI snapshot during real-time operation.....	79
Figure 66 : Real-Time Workflow.....	81
Figure 67 : GUI with visual overheat warning.....	82
Figure 68 : Measure non-overheated objects with 8x8 native resolution....	82
Figure 69 : Measure non-overheated objects with 64x64 interpolation.....	83
Figure 70 : Measure overheated objects with 8x8 native resolution.....	84

LIST OF TABLES

Table 1 : Evaluate solutions.....	5
Table 2 : Evaluate solutions.....	10
Table 3 : Specifications of motor.....	10
Table 4 : Comparison of some popular Arduinos.....	13
Table 5 : OV5647 Camera Specifications.....	15
Table 6 : Comparison between Lidar sensors from manufacturers.....	17
Table 7 : Value MG996R RC Servo Motor.....	18
Table 8 : Algorithm result.....	38
Table 9 : Multi-Scale Feature Maps and Target Object Sizes.....	55
Table 10 : Model parameters after training.....	66
Table 11 : Connection between AMG8833 and Raspbberry.....	74
Table 12 : Supported Interpolation Modes.....	79
Table 13 : Technical Analysis and Performance Evaluation.....	86

LIST OF ABBREVIATION

Abbreviation	Full Term	Description
AI	Artificial Intelligence	Simulates human intelligence in machines to perform tasks like learning.
AMCL	Adaptive Monte Carlo Localization	Probabilistic localization using particle filter in robot navigation.
BBox	Bounding Box	A rectangle that encloses an object in an image.
CNN	Convolutional Neural Network	Deep learning model highly effective for image-related tasks.
CSI	Camera Serial Interface	High-speed interface for connecting camera modules to processors.
CV	Computer Vision	Enables machines to interpret and process visual data.
DC	Direct Current	Unidirectional flow of electric charge, used in electronics and motors.
FN	False Negative	Actual object missed by the detection model.
FP	False Positive	Incorrect object detection when none exists.
FPS	Frames Per Second	Number of video frames processed per second.
GPU	Graphics Processing Unit	Parallel processor for accelerating graphics and AI computation.
GUI	Graphical User Interface	User interface with visual components for interaction.
LIDAR	Light Detection and Ranging	Remote sensing method using lasers to measure distances.
mAP	Mean Average Precision	Average of precision scores across all classes and thresholds.
mAP@0.5	mAP at IoU = 0.5	Mean Average Precision at IoU threshold 0.5.
mAP@0.5:0.95	mAP from IoU 0.5 to 0.95	Averaged mAP across IoU thresholds from 0.5 to 0.95.
ONNX	Open Neural Network Exchange	Cross-platform model format for AI deployment.
OpenCV	Open Source Computer Vision Library	Library for real-time computer vision and image processing.
PF	Particle Filter	Probabilistic method for estimating system state, used in localization.
Precision	Precision	Proportion of correct positive detections among all detections.
PWM	Pulse Width Modulation	Technique to control power delivered to electrical devices.
Recall	Recall	Proportion of actual positives correctly detected.
ROS	Robot Operating System	Flexible framework for writing robot software.
SLAM	Simultaneous Localization and Mapping	Builds a map while localizing the robot within it.
Smoking	(Class label)	Refers to detection of cigarette usage in the image dataset.
TP	True Positive	Correct object detection.
TPU	Tensor Processing Unit	AI accelerator developed by Google for deep learning tasks.
UART	Universal Asynchronous Receiver/Transmitter	Serial communication protocol for microcontrollers.

CHAPTER 1: INTRODUCTION

I. Urgency of the topic

Industrial safety and monitoring in industry

In industrial factories, monitoring the temperature of equipment plays an important role in maintenance and fire prevention. Continuously operating equipment can overheat, leading to reduced life or fire. Using thermal cameras helps detect areas with abnormal temperatures, however, traditional methods require manual inspection by staff, which is time-consuming and dangerous. In addition, positioning and navigation in an environment without GPS is a big challenge.

In many industrial equipment warehouses, equipment such as motors, electrical cabinets, and compressors can operate continuously, generating high heat. If not monitored promptly, the risk of electrical short circuits and explosions is very high.

According to statistics from the NFPA (National Fire Protection Association) and other safety organizations, 30–40% of industrial fires and explosions originate from electrical faults or equipment overheating.

The damage from these fires and explosions can amount to billions of VND, and seriously affect production progress and human safety.

Difficulties of manual supervision

The method of patrolling with security guards and visual inspection requires continuous human resources, lacks precision, and easily misses areas.

With a large area and complex structure (high shelves, narrow aisles), full monitoring and early detection of temperature abnormalities becomes even more difficult.

The demand of combining security - management - safety

In addition to fire and explosion risks, warehouses also require security monitoring (detecting intrusion, tracking people entering and leaving) and equipment management (tracking pallet and forklift movements).

A self-driving robot with an integrated conventional camera (for object identification, security monitoring) and a thermal camera (for measuring equipment temperature, hot warning) will simultaneously solve these goals.

Automation in the context of Industry 4.0

The increased use of mobile robots is an essential part of the fourth industrial revolution (Industry 4.0) that emphasizes IoT, AI, robotics. Autonomous robots (AGV, AMR) are widely applied to save manpower, optimize the flow of goods. However, most of the existing robots focus on transportation or line - thermal cameras are not popular.

II. Existing solutions and related commercial products

1. Technical solutions on the market

a. Using fixed thermal cameras

Advantages:

Fixed thermal cameras are installed at important locations in factories and warehouses to monitor the temperature of the environment and equipment in real time. Popular camera lines include FLIR, Hikvision, Dahua, Axis... with advantages such as:

- Continuous monitoring 24/7 without human intervention.
- High accuracy, can measure temperature from a long distance (~5-50m depending on the model).
- Easy to integrate into the security monitoring system, can transmit data to the control center.
- No need for regular maintenance, stable operation.
- Can combine AI to analyze data and detect abnormalities

Disadvantages:

- Fixed observation range, cannot be moved to another area.
- High cost, some high-end cameras cost from 2000 - 10000 USD.
- Depending on the installation location, if installed in the wrong location, it can cause blind spots.
- Difficult to use for large areas without investing in many cameras.



Figure 1 : IR thermal camera

<https://eigroup.biz/en/industrial-and-professional-thermal-cameras/51-timage-high-resolution-fixed-thermal-camera.html>

b. Drone with thermal camera

Advantages:

Drones equipped with thermal cameras help monitor from above, popular in fire fighting, power line inspection ...

- Wide monitoring range, can fly to many different areas.
- Flexible, easy to adjust the viewing angle as required.
- High efficiency in outdoor monitoring, especially in emergency situations.
- Can be programmed to fly automatically or remotely.
- Good application in difficult-to-access environments such as forests, large industrial zones.

c. Self-propelled robot with thermal camera

Advantages:

Mobile robots equipped with thermal cameras help move automatically in factories, hospitals, cold storage ... to continuously monitor temperature.

- Flexible movement, can go to any area that needs monitoring.

- Automatic operation, no human intervention required.
- Can be combined with LiDAR for positioning and SLAM, helping to avoid obstacles.
- Easily integrated with other sensors (humidity, smoke ...) to enhance monitoring capabilities. - Can transmit real-time data to the control center.
- Integrated AI helps analyze thermal images and provide early warning.

Disadvantages:

- High cost, specialized robots like Boston Dynamics Spot cost from \$74,500.
- Requires a flat surface or wide enough aisle, difficult to operate in complex terrain.
- Slower movement speed than drones, not suitable for large areas.
- Requires complex programming and integration of appropriate control software.



Figure 2: Thermal security robot

https://smprobotics.com/security_robot/thermal-security-robot/

d. IoT thermal sensor system

Advantages:

IoT temperature sensors (DHT22, FLIR Lepton...) are installed in many locations, transmitting data to the system via WiFi.

- Low cost, IoT sensors cost only \$10 - \$200.
- Can be deployed widely by installing many sensors.
- Energy saving, can operate on batteries for a long time.
- Easy to integrate into AI and IoT systems for remote monitoring.

- No need for regular maintenance, just change the battery or check periodically.

Disadvantages:

- Limited flight time (~30-45 minutes), need to charge or change the battery continuously.
- High cost, specialized drones cost from 5000 - 20000 USD.
- Not suitable for indoor surveillance or areas with many obstacles.
- Need to have a human operator or pre-programmed route, not fully automatic.
- Affected by weather, it is difficult to operate in strong wind and rain conditions.

Solution	Monitoring scope	MoveMent	Accuracy	Automat-ion	Cost	Applications
Fixed thermal camera	Narrow	None	High	High	High	Factory
Drone with thermal camera	Wide	High	High	Medium	Very high	Fire, power lines
Autonomous robot with thermal camera	Medium	High	High	High	High	Factories, hospitals, warehouses
IoT Thermal Sensor	Narrow	None	Medium	Medium	Low	Office, smart home

Table 1: Evaluate solutions.

2. Some commercial products

a. Substation monitoring robot of Thua Thien Hue Power Company

Monitoring system: The robot is equipped with a regular camera and a thermal camera to monitor and inspect equipment in the substation, as shown in Figure 1.

This system helps to detect early abnormalities of the equipment thanks to the thermal camera.

Application: The robot is used to monitor the operation of an unmanned 110kV substation, helping operators proactively monitor the operation of the entire system, including the working status of the equipment, warning signals and early detection of abnormalities.

Advantages:

- Cost savings compared to purchasing robots from abroad, with a price of about 50,000 USD.
- Improving the skills of staff and engineers in accessing and using modern technologies

Disadvantages:

It takes time and resources to research, design and manufacture robots.

Maintenance and upgrading can be difficult if there is a lack of specialized technical support.



Figure 3: Substation monitoring robot

<https://husta.vn/cong-trinh-de-tai/thiet-ke-che-tao-robot-tu-hanh-dieu-khien-xa-phuc-vu-cong-tac-giam-sat-van-hanh-tram-bien-ap-110kv-khong-nguoi-truc/>

b. Boston Dynamics Spot

Product description: Spot is a four-legged robot developed by Boston Dynamics, famous for its flexible mobility and adaptability to various terrains. Weighing about 25 kg, Spot can carry sensors, including thermal cameras, to perform monitoring and inspection tasks.

Detailed review:

- Spot can climb stairs, overcome rough terrain and move in narrow spaces, suitable for monitoring in complex environments.
- Spot can be equipped with a variety of sensors, including thermal cameras, to detect hot spots or fire risks in buildings.



Figure 4: Boston Dynamics Spot

<https://bostondynamics.com/products/spot/>

3. Proposed preliminary solution

Objectives and ideas for implementing the topic

Building an autonomous robot (small size) that can move in an industrial equipment storage warehouse, monitor security and warn of fire risks through a thermal camera.

Combine sensors: LiDAR, regular camera for observation, thermal camera for temperature measurement

3.1 Expected functional objectives

- Automatic patrol: The robot moves in the warehouse, scans the environment, and warns when detecting abnormalities (obstacles, high temperatures).
- Manual control: The operator can switch to manual mode, driving the robot to observe suspicious areas.
- Video monitoring: The robot transmits images from a regular camera (and possibly a thermal camera) to the center.
- Early warning of fire and explosion: When the thermal camera detects a hot area exceeding the threshold, it activates the buzzer/light or sends a notification.

3.2 Limitations of the topic

- Focus only in industrial warehouses (indoor environments), flat floors, no steep slopes.
- Scale: Small robots, slow moving speed (0.2–0.5 m/s).
- Do not deploy outdoors or in harsh environments (rain, strong winds).
- Display and control interface can use LabVIEW or web dashboard.
- Only display video, robot status, temperature, warnings.
- No integration of SCADA or specialized warehouse management systems.

CHAPTER 2: HARDWARE ANALYSIS AND DESIGN FOR ROBOT

I. Overview

The topic "Autonomous temperature monitoring robot using thermal camera and SLAM positioning with LiDAR sensor" aims to design and manufacture an intelligent mobile robot system capable of automatically exploring the environment, building a 2D spatial map and monitoring the temperature in each area in that environment. The robot must be able to accurately locate in an unknown environment using the SLAM (Simultaneous Localization and Mapping) algorithm and integrate a thermal camera to detect and record abnormal temperature points. During operation, the robot will use the LiDAR sensor to collect distance data and build a real-time map and navigate to the locations that need to be checked through the path planning algorithm (Global Planner) and local control (DWA Local Planner). Thermal image data will be assigned specific spatial coordinates and transmitted to the control station for monitoring or storage. The system needs to operate stably, be able to operate autonomously in real environments, quickly detect areas with abnormal temperatures, and be easily controlled remotely. The implementation of the project requires coordination between robot hardware, sensor systems, a central processor capable of running ROS 2, and software for navigation, positioning, and thermal image processing.

No.	Component / Device Name	Function
1	Raspberry Pi 5 (or Jetson Nano)	Central processing unit, runs ROS 2 and controls the entire system
2	Thermal Camera (MLX90640, FLIR Lepton) and RGB Camera	Captures thermal images for temperature monitoring
3	LiDAR Sensor (RPLiDAR A1/A2 or YDLiDAR X4)	Collects environmental distance data for SLAM mapping
5	DC Motor	Provides movement for the robot
6	Motor Driver (BTS7960, L298N)	Controls motor speed and direction

7	Mobile Robot Chassis (2-wheel, 4-wheel, or RC car base)	Mechanical platform to mount components
8	Power Supply / UPS HAT	Powers the Raspberry Pi and motors, ensure stable operation
9	UART-to-USB Converter Module	Interfaces sensors with the processor
10	Arduino Uno	Communication, transmission of information between the central processor and motor
11	Jumper Wires, Connectors, Breadboard	Connects modules and components

Table 2: Evaluate solutions.

From the above requirements we get the overall block diagram of the system.

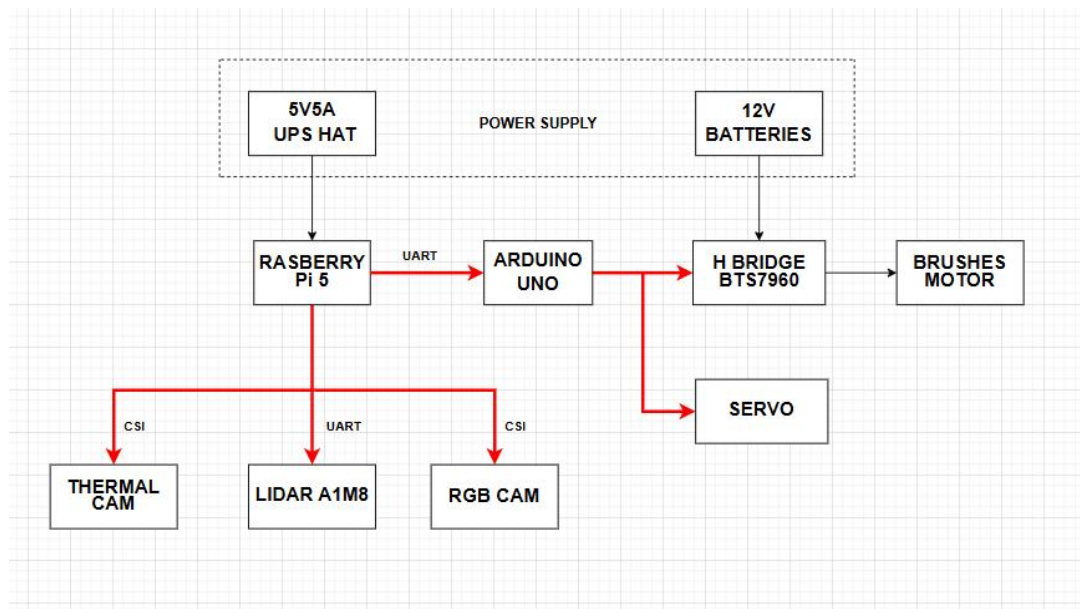


Figure 5 : Block diagram of system

Signal: 
Source: 

II. Tools and Equipment needed for implementation

1. Brushes Motor (Surpass Hobby 540)

The Surpass Hobby 540 brushed motor is a popular DC brushed motor, widely used in RC car applications, mini robots, and DIY projects. It is a 540-size motor, which means it has a standard size of 36mm x 50mm, compatible with many commercial motor mounts and ESCs.

Parameter	Typical value
Engine type	DC Brushed
Size	540 (36mm diameter x 50mm length)
Operating voltage	6V – 7.4V (12V short term max)
No-load speed	~20,000 – 28,000 rpm
Idle current	1 – 2A
Maximum operating current	30A (ESC)
Shaft diameter	3.175mm (1/8 inch)
Shaft head type	D-type or smooth round
Heat resistance	Medium – not suitable for continuous heavy loads

Table 3: specifications of motor



Figure 6 : Surpass Hobby 540 motor

<https://picclick.co.uk/SURPASSHOBBY-540-Brushed-Motor>

2. BTS7960 43A Motor Driver Circuit



Figure 7 : BTS7960 43A

<https://linhkienviet.vn/module-dieu-khien-dong-co-dc-bts7960-43a>

The BTS7960 motor driver can easily interface with a microcontroller and comes with a full set of protection and control features, including:

- Current sensing (using a shunt resistor for current measurement).
- Dead-time generation to prevent shoot-through.

Overtemperature, overvoltage, overcurrent, undervoltage, and short-circuit protection.

Specifications:

- Operating voltage: 6V ~ 27V

- Maximum load current: 43A
- Maximum control
- frequency: 25kHz
- Automatic shutdown at low voltage: The driver will turn off when the voltage drops below 5.5V and will resume operation once it exceeds 5.5V.
- Overtemperature protection: The BTS7960 has a built-in thermal sensor that automatically disables output when overheating is detected.

3. Arduino

Parameters	Arduino Uno	Arduino Mega 2560	Arduino Nano
ClockRate (MHz)	16	16	16
Voltage (V)	5	5	5
Flash Memory (KB)	32	256	32
RAM (KB)	2	8	2
WIFI	None	None	None
USB Port	USB-B	USB-B	Mini-USB
Power supply (mA)	30	70	40
Size(mm)	68.6 × 53.4	101.52 × 53.3	45 × 18
Cost (VND)	~150K	~300K	~120K

Table 4: Comparison of some popular Arduinos

- Arduino Uno – Stable, easy to program, full of features, cheap.
- Mega 2560 – Too many features, only needed if you need a lot of I/O or a lot of RAM.
- Nano – Compact, but harder to connect than Uno

-> We decided to choose Arduino Uno for the topic

Arduino Uno is an open source microcontroller board based on the Microchip ATmega328 microcontroller developed by the board equipped with Digital and Analog input/output pins that can communicate with various expansion boards.

4. Raspberry Pi 5

Highlights of Raspberry Pi 5 8GB:

- **More powerful CPU:** Cortex-A76 2.4GHz is significantly faster than previous versions.
- **LPDDR4X RAM:** Higher performance, lower power consumption.
- **Upgraded GPU:** VideoCore VII supports OpenGL ES 3.1 and Vulkan 1.2.
- **Integrated PCIe 2.0:** Expand connectivity with high-performance peripheral hardware.
- **Good support for AI and SLAM:** Faster data processing improves accuracy in robot navigation.

So we decided to choose Raspberry Pi5 8GB for this project



Figure 8: Raspberry Pi5 8GB

<https://rasberrypi.vn/san-pham/mach-may-tinh-raspberry-pi-5>

Raspberry Pi 5 is the latest generation of Raspberry Pi embedded computers, providing significantly higher performance than previous versions. The 8GB LPDDR4X version is designed for applications that require powerful processing such as autonomous robots, AI, and machine learning. Raspberry Pi 5 still retains the same 40 GPIO (General Purpose Input/Output) pins as previous versions, allowing connection to many different types of sensors, modules, and peripherals.

5. OV5647 camera



Figure 9: OV5647 camera

<https://www.ubuy.vn/vi/product/KM59VZCG-raspberry-pi-camera-module>

Specification	Value
Sensor	OV5647 CMOS
Maximum Resolution	5MP (2592 × 1944)
Interface	CSI (15-pin / 22-pin ribbon)
Frame Rate	30fps 1080p, 60fps 720p
Image Format	RAW, YUV, JPEG
Field of View	~54° (standard lens)
Auto White Balance	Supported
Auto Exposure Adjustment	Supported

Table 5: OV5647 Camera Specifications

The OV5647 camera module is a high-resolution 5MP CMOS image sensor developed by OmniVision, commonly used in embedded vision systems with the Raspberry Pi through the CSI (Camera Serial Interface). It supports resolutions up to 2592×1944 pixels and various video modes including 1080p at 30 frames per second and 720p at 60 frames per second. The module delivers images in RAW, YUV, or JPEG formats and features automatic white balance and auto-exposure adjustment, making it suitable for applications such as visual surveillance, object recognition, and real-time image processing. In this project, the OV5647 serves as the RGB imaging component to complement thermal data from the AMG8833 sensor. It provides visual context for thermal readings, enabling real-time identification of persons, hazardous behaviors like smoking, or early signs of fire using deep learning models such as YOLOv8. The camera transmits image data to the Raspberry Pi via CSI, where signal processing algorithms enhance image clarity through functions such as noise reduction and contrast adjustment. Its advantages include ease of integration with Raspberry Pi, good image quality relative to cost, and compatibility with open-source libraries like OpenCV. However, it has some limitations such as narrow field of view without additional lenses and reduced performance in low-light conditions without IR support. Overall, the OV5647 is a cost-effective and reliable solution for integrating RGB imaging into thermal surveillance systems, contributing to multi-modal analysis and improved decision-making.

6. Thermal camera

Based on the comparison table above, the team decided to use the AMG8833 because:

- Compact size: Suitable for autonomous robot design.
- Reasonable price: Significantly cheaper than FLIR Lepton or Hikvision.
- Wide viewing angle: AMG8833 has a viewing angle version of up to 60°, helping to scan a larger range.
- Easy integration with Raspberry Pi: Through the I2C protocol, no need for complex hardware.



Figure 10: Thermal camera AMG8833

<https://emin.vn/adafruit3538-gwc-temperature-sensor-amg8833-ir-thermal-camera>

7. LiDAR A1M8

LiDAR (Light Detection and Ranging) is a sensor technology that uses laser beams to measure distances and create a 3D map of the surrounding environment. It operates by emitting laser pulses and measuring the time-of-flight (ToF) for the pulses to return after reflecting off objects. Currently, there are various LiDAR sensors on the market, each with unique technical specifications and functionalities. The table below compares the features and specifications of common LiDAR models:

Type/S PEC	A1M8	A2M12	A3M1	S1	S2M1-R2L (18m)	S2M1-R2 (30m)	T1 (40m)	M2M2
Ranging range	0.15-12m	0.2-12m	0.2-25m	0.1-40m	0.05-18m	0.05-30m	0.05-40m	40m
Ranging method	triangulation	triangulation	triangulation	TOF	TOF	TOF	TOF	TOF
Scan angle	0-360°	0-360°	0-360°	0-360°	0-360°	0-360°	0-270°	0-360°
Pitch angle	±1.5°	±1.5°	±1.5°	/	0-1.5°	0-1.5°	/	±3°
measurement accuracy				±5cm	±5cm	±5cm		±5cm
Ranging resolution	<0.5mm 1% actual range	<0.5mm 1% actual range	/	30mm			13mm	Mapping resolution 5cm
Angular resolution	1°	0.225°	0.225°	0.391°	0.12°	0.12°	0.12°	0.391°
Sampling frequency (Hz)	8k	16K	16K	9.2K	32K	32K	60K	9.2K
Sweep frequency (Hz)	5.5	10 (5-15Hz)	10 (5-15Hz)	10 (8-15Hz)	10	10	20-40	10
Communication Interface	UARTserial port (115200 bps)	URATserial port (256000 bps)	URATserial port (256000 bps)	URATserial port (256000 bps)	URATSerial/Netw ork Port (1 M, 10/100M)	URATSerial/Netw ork Port (1 M, 10/100M)	Network Port (10/100M)	Network Port, wifi
drive mode	brushed motor	brushless motor	brushless motor	brushless motor	brushless motor	brushless motor	brushless motor	brushless motor
Protection class	/	/	/	/	IP65	IP65	IP67	/
Anti-ambient light	NO	NO	Yes	Yes	Yes	Yes	Yes (Outdoor 80000 LUX)	Yes
IEC60825 Class 1	√	√	√	√	√	√	√	√

Table 6: Comparison between Lidar sensors from manufacturers

Based on manufacturer specifications and market price analysis, we have selected the

LiDAR A1M8 according to the following criteria:

- Lowest cost (~\$100–\$150), making it suitable for prototyping and small mobile robots.
- Low power consumption (5V, 500mA), ideal for battery-powered systems.
- Sufficient accuracy for indoor SLAM, with an error margin of $\pm 1\%$ for distances under 3m.
- Seamless integration with Arduino, Raspberry Pi, and Jetson Nano via UART interface.
- 360° field of view, ensuring comprehensive data collection for SLAM applications.
- Energy efficiency, as it utilizes a brushed motor, making it suitable for mobile, battery-operated systems.



Figure 11: Thermal camera AMG8833

<https://www.indiamart.com/proddetail/rplidar-a1m12-laser-ranging-sensor>

8. MG996R RC Servo Motor

The MG996R is a high-torque, metal gear standard servo motor widely used in robotics, robotic arms, RC vehicles, and various mechatronic systems that require precise angular positioning. It is an improved version of the MG995, offering better accuracy, faster response, and enhanced durability.

Parameter	Value
Operating Voltage	4.8V – 7.2V
Maximum Torque	~9.4 kg·cm @ 4.8V~11 kg·cm @ 6.0V
Rotation Angle	0° – 180° (extendable with calibration)
Response Speed	~0.17s / 60° @ 6V
Control Signal Type	PWM (Pulse Width Modulation)
Weight	Approx. 55g
Gear Material	Metal (durable and wear-resistant)
Connector Interface	3-pin: VCC (red), GND (brown), PWM (orange)

Table 7: Value MG996R RC Servo Motor



Figure 12: MG996R RC Servo Motor

<https://forum.arduino.cc/t/powerfull-servo-board-for-16-high-end-servos/526837>

9. Power Supply

In autonomous mobile robotic systems, a reliable and properly regulated power supply is critical to ensure safe and stable operation of both high-current components

(such as motors) and sensitive processing units (like single-board computers). In this project, the power supply system is divided into two independent parts:

a. Power Supply for the Drive Motors – 18650 Li-ion Battery Pack

The drive motors are powered by a custom battery pack made from four 18650 Li-ion cells, configured in a 2S2P topology:

- Two pairs of batteries are connected in parallel to increase the overall capacity (Ah),
- These two parallel groups are then connected in series to raise the nominal voltage to approximately 7.4V (2S configuration).

The output voltage ranges from 8.4V (fully charged) to ~6.0V (fully discharged).

This configuration provides both sufficient voltage and extended runtime to drive brushed DC motors such as the Surpass Hobby 540. The battery pack is connected to the motor driver (e.g., BTS7960 H-bridge or brushed ESC), which enables bidirectional speed and direction control via PWM signals from the main controller.

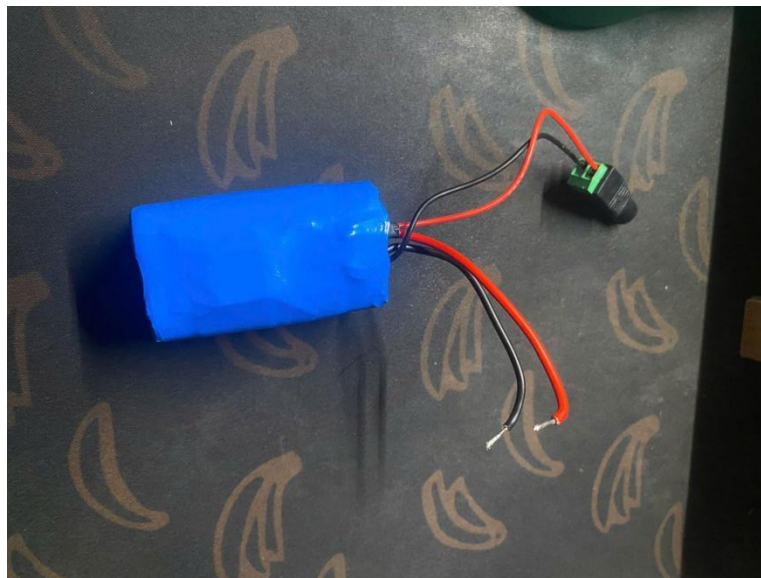


Figure 13: 18650 Li-ion Battery Pack

b. Power Supply for the Raspberry Pi 5 – UPS HAT E

The Raspberry Pi 5 serves as the central processing unit of the system, running the ROS 2 stack and processing data from various devices. To ensure stable and uninterrupted operation, especially in mobile and remote applications, the system uses a dedicated UPS HAT E module. This board provides:

- A stable 5V 5A power output via GPIO header or USB-C connection.

- Built-in rechargeable battery cells and an integrated battery management system (BMS) for charging, protection, and automatic switching.
- Uninterruptible Power Supply (UPS) capability – the Raspberry Pi can continue operating even if the external power source is disconnected.
- I2C communication interface, allowing the Raspberry Pi to monitor battery voltage, current, and remaining capacity in real time.



Figure 14: UPS HAT E

<https://pivietnam.com.vn/ups-hat-e-danh-cho-raspberry-pi-ho-tro-4-cell-pin-li-21700>

By isolating the power sources for the motors and the processing unit, the system prevents voltage dips, electrical noise, and unwanted resets caused by motor current spikes. This separation ensures stable operation and enhances the reliability of the robot during autonomous missions.

III. ROBOT kinematics

Kinematics modeling is the study of the motion of mechanical systems without considering the dynamic actions on the motion. The main purpose of the modeling is to express the relationship between the speed of the robot and the speed wheel along with the robot's learning parameters.

In order for the autonomous robot to be able to control accurately in a real environment, building a learning model is a crucial design step to describe the relationship between the robot's motion system and the control parameters. In this diagram, the robot is designed with a similar steering wheel structure - that is, using

the Ackerman configuration basis, which helps to limit the slip mode on the wheels and is suitable for adjusting work in narrow spaces.

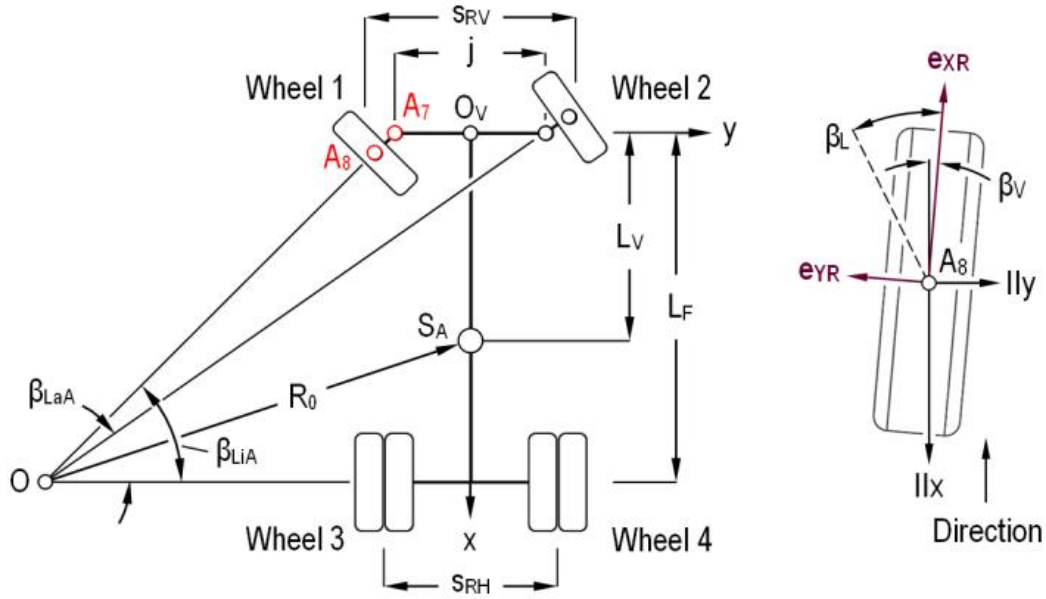


Figure 15: Ackerman kinematic model

The Ackerman kinematic model uses the following assumptions:

- The robot moves on a 2D plane.
- There is no slipping between the wheels and the road surface.
- The motion is controlled by linear velocity and steering angle.

We consider robot models with the following parameters:

- x, y : Coordinates of the center of the robot's rear axle on the plane.
- θ : The robot's orientation angle relative to the x-axis.
- v : Linear velocity.
- δ : Steering angle of the front wheels.
- L : Wheelbase - distance between the front and rear axles.

State variables of the system:

$$\mathbf{x} = \begin{bmatrix} x \\ y \\ \theta \end{bmatrix}$$

Control variable:

$$\mathbf{u} = \begin{bmatrix} v \\ \delta \end{bmatrix}$$

Based on the assumptions and geometry of the Ackerman mechanism, we obtain the following kinematic equation of the robot:

$$\dot{x} = v \cdot \cos(\theta)$$

$$\dot{y} = v \cdot \sin(\theta)$$

$$\dot{\theta} = \frac{v}{L} \cdot \tan(\delta)$$

The above equation shows that the direction of movement of the robot depends on the linear speed and steering angle, from which we can simulate or design a suitable controller for positioning and navigation.

In the case of applying numerical control or algorithmic simulation (e.g. in SLAM or MPC), the above continuous model can be discretized in time steps Δt :

$$x_{k+1} = x_k + v_k \cdot \cos(\theta_k) \cdot \Delta t$$

$$y_{k+1} = y_k + v_k \cdot \sin(\theta_k) \cdot \Delta t$$

$$\theta_{k+1} = \theta_k + \frac{v_k}{L} \cdot \tan(\delta_k) \cdot \Delta t$$

This discretized model will be used in the robot trajectory simulation steps or combined with the extended Kalman filter (EKF) to perform positioning in SLAM.

IV. Realistic model

After analyzing the functional requirements and selecting appropriate components, the real-world prototype of the autonomous temperature monitoring robot has been developed with a tightly integrated hardware–software architecture. The robot is built on a sturdy four-wheel chassis, equipped with DC motors. The central processing unit is a Raspberry Pi 5, functioning as the “brain” of the system. It handles sensor data collection and processing, motion control, and runs the ROS 2 operating system to perform navigation and SLAM algorithms.

The sensor system includes a LiDAR sensor mounted at the back of the robot to scan and collect environmental data for real-time mapping and localization. A thermal camera is mounted at the front of the chassis, with a wide field of view to capture surface temperature readings during movement. Thermal data is tagged with spatial coordinates, enabling users to identify the exact location of heat anomalies within the environment.

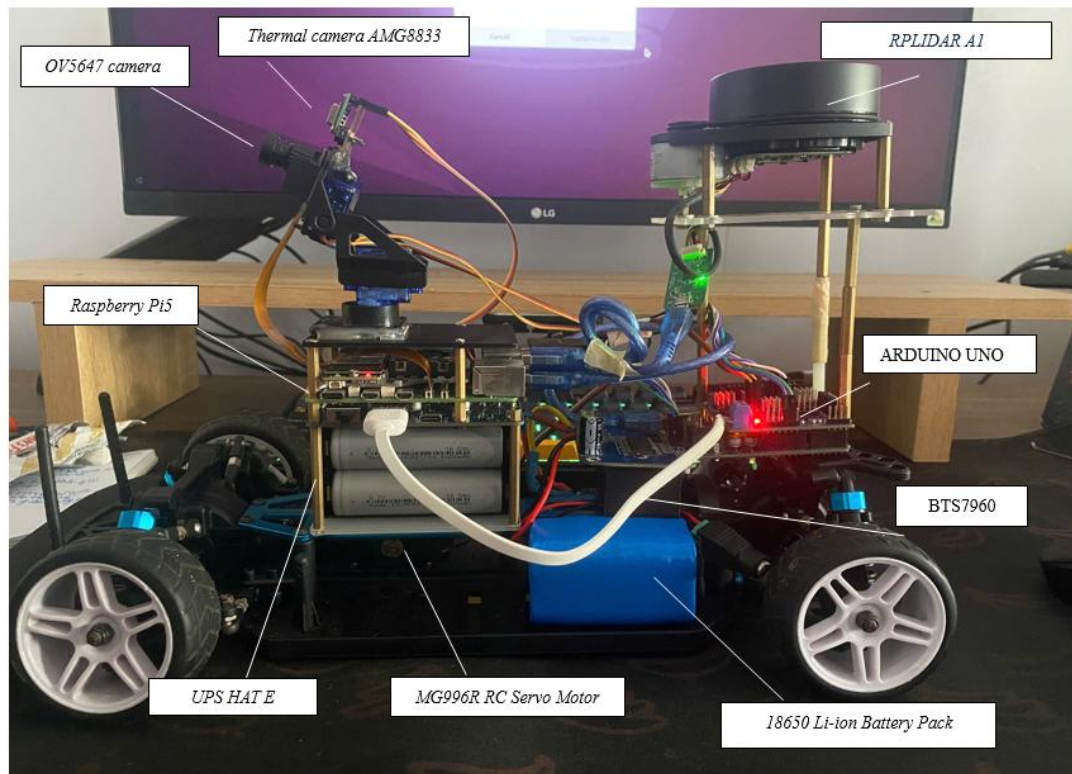


Figure 16: Realistic model

This physical model demonstrates a high level of automation: the robot can navigate, localize, collect temperature data, and transmit information to a control station in real time. As a result, operators can monitor the robot's activities and send it to any desired location on the generated map. This provides a robust foundation for applications in environmental monitoring, thermal anomaly detection, and autonomous safety patrol in temperature-critical areas.

CHAPTER 3: ROBOT OPERATING SYSTEM (ROS2) AND SLAM ALGORITHM IN AUTONOMOUS ROBOT

I. Robot Operating System (ROS 2):

1. Overview

Robot Operating System 2 (ROS 2) is an open source software platform for robots, developed to inherit and overcome the limitations of ROS 1. ROS 2 was born in 2017, using a new communication architecture based on DDS (Data Distribution Service), supporting real-time programming, cross-platform (Linux, Windows, macOS), improved security, and scalability suitable for distributed systems.[1]

ROS 2 allows building robot systems with the ability to locate, map, navigate, sense and interact in complex environments. With a strong development community, ROS 2 is increasingly becoming the standard in modern robot projects.

We choose ROS 2 (specifically the ROS 2 Jazzy version on Ubuntu 24.04) as the development platform for the autonomous robot system for the following outstanding reasons:

Modernity and long-term future:

ROS 2 is the version developed and maintained by the ROS community for a long time. ROS 1 has stopped official support since 2025, so developing based on ROS 2 is an inevitable trend.

Real-time support:

ROS 2 uses DDS (Data Distribution Service) to allow real-time data communication - which is very important in controlling autonomous robots that need to react quickly and accurately to the environment.

Scalability and distributed deployment:

ROS 2 can be deployed on many different devices and platforms (such as Raspberry Pi, industrial computers, microcontrollers) thanks to its powerful distributed architecture.

Higher security:

ROS 2 integrates security features (SROS2), allowing data encryption, access

authorization between nodes - suitable for the trend of robots operating in networked environments (IoT).

Good support for modern hardware:

ROS 2 can operate smoothly with new drivers, supporting camera systems, Lidar, IMU, GPS... with continuously updated drivers.

Strong development community and complete documentation:

ROS 2 is developed by many large companies and research institutes in the world (such as Open Robotics, Intel, Bosch, Amazon, Nvidia...). The user manual is provided in detail, easy to learn and easy to access.

Optimized for positioning and navigation software (SLAM, Navigation2):

ROS 2 provides modern tool packages such as slam_toolbox, nav2, lifecycle nodes, rviz2, ros2_control, with higher performance and accuracy than ROS1.

Key Features of ROS 2

- **DDS Middleware:** DDS standard communication enables reliable, real-time, flexible communication with QoS (Quality of Service) settings.
- **Multi-platform support:** ROS 2 runs on Linux, Windows, macOS and microcontrollers such as Raspberry Pi.
- **Node Lifecycle Management:** Allows control of the operating status of nodes (inactive, active, shutting down...).
- **Modern libraries:** Support for rclcpp (C++) and rclpy (Python), along with the colcon tool for building the system.
- **Security:** Support for SROS2 - a security framework for ROS 2 systems.

2. Main Nodes used in the this project

To control the robot to move along the desired trajectory and reach the destination, at each time it is necessary to determine information about the robot's position and direction. The structural diagram of the system on ROS2 is developed based on nodes, each node is created with different tasks and functions. The information exchanged between each node is communicated through messages, the information is transmitted and received according to predetermined protocols and has a common standard for communicating with external data packets. The structural diagram of the

system designed on ROS2 includes nodes and the data transmission and reception process is described as follows

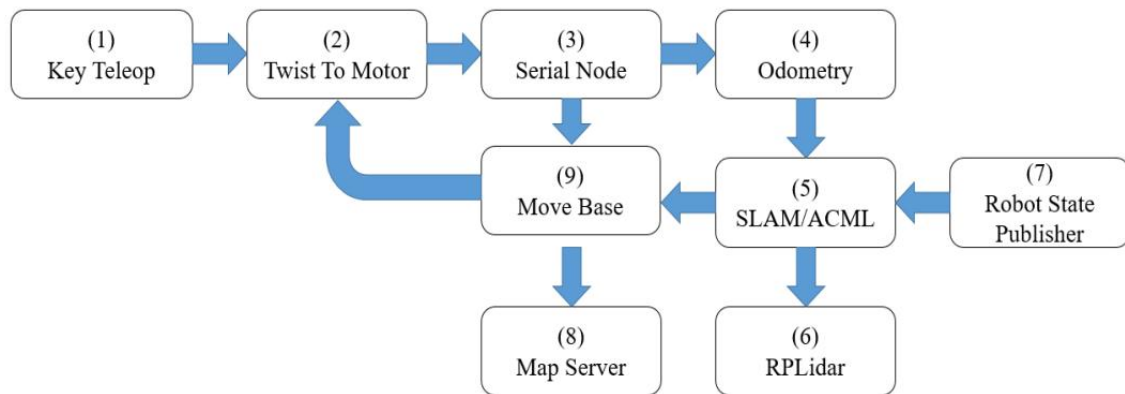


Figure 17: Nodes on ROS2

A typical ROS 2 robot system will have important nodes such as:

- `slam_toolbox / cartographer`: Used for SLAM mapping.
- `nav2_bt_navigator`: Used for automatic robot navigation.
- `amcl`: Locates the robot in an existing map.
- `nav2_controller`: Controls the output velocity for the robot.
- `nav2_map_server`: Saves and loads the map.
- `robot_state_publisher`: Exports transforms of robot parts from URDF.
- `lidar_node`: Reads data from the Lidar sensor and publishes it to the `/scan` topic.

3. COORDINATE TRANSFORMATION IN ROS2

In a mobile robot system using ROS, correctly defining the coordinate frames between the robot and its sensors is crucial to ensure that algorithms such as SLAM and navigation operate accurately. The coordinate frame attached to the robot's body, typically located at the robot's rotational center, is called *base_footprint*. All components mounted on the robot, including the LiDAR sensor, must have their own coordinate frames, and these frames need to be defined within the system in relation to *base_footprint*. Specifically, the LiDAR sensor has its own frame, commonly named *laser* or *laser_frame*, which should be declared as a child frame of *base_footprint*.

To allow ROS to understand the exact position of the LiDAR sensor relative to the robot's body, a transform from *base_footprint* to *laser* must be specified. This transform includes the position (x, y, z) of the sensor relative to the robot's rotational center, along with the orientation (roll, pitch, yaw) to define its direction. This transformation can be declared in a URDF file describing the robot model or, more simply, by using the *static_transform_publisher* node in ROS2. [2]

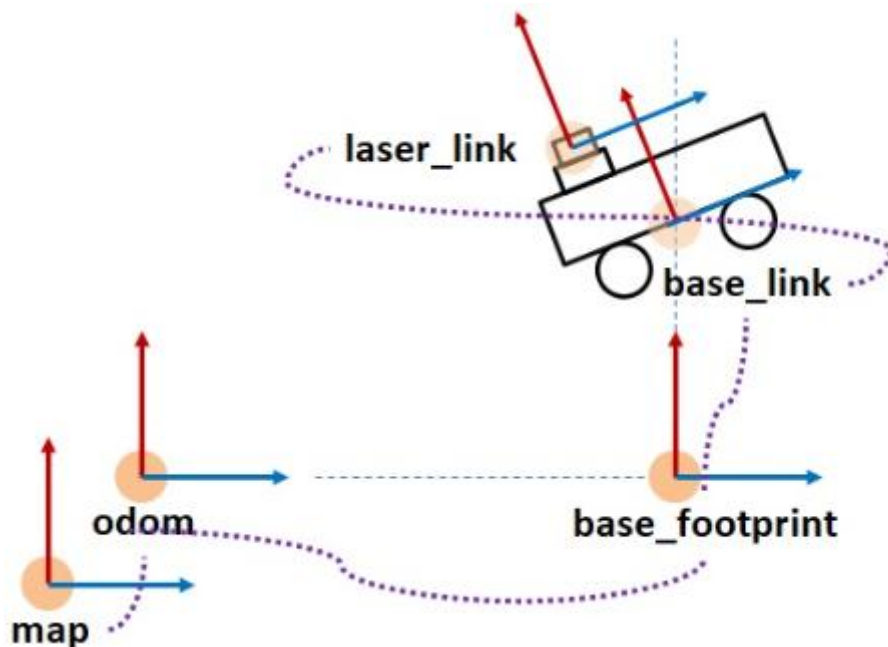


Figure 18: TF transformation

<https://automaticaddison.com/coordinate-frames-and-transforms-for-ros-based-mobile-robots>

II. Slam Algorithm

1. Overview

SLAM (Simultaneous localization and mapping) is a system that uses image information obtained from cameras, or Lidar or some other types of sensors to recreate the external environment by putting environmental information into a map (2D or 3D), from which the device (robot, camera, vehicle) can localize where it is, its state, and posture in the map to automatically establish a path (path planning) in the current environment.

Automatic robot control is divided into 3 main issues:

- Localization

- Environment reconstruction (mapping)
- Path planning

Initially, the two issues of localization and environment reconstruction were studied independently, however, after realizing that:

- Localization: it is necessary to determine the current position of the robot based on the reconstructed map.
- Map reconstruction: it is necessary to determine the location of the object in the map, to build the most accurate map with the least error.

Navigation Stack blocks used in robot positioning and planning:

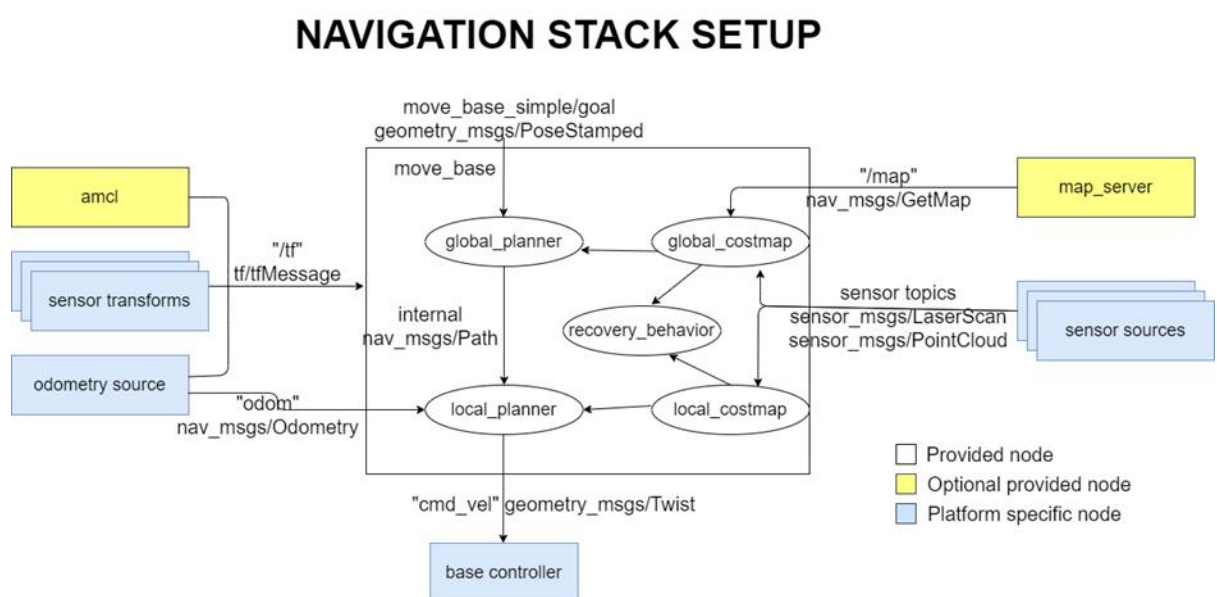


Figure 19 : Navigation diagram on ROS2

<https://wiki.ros.org/>

- **amcl:** is a probabilistic mapping system for a robot moving in 2D mode, using a particle filter based on data from laser sensors such as lidar to give an estimated position of the robot.
- **sensor transforms:** After receiving data from sensors such as ultrasound, encoder, ... through the transformation of TF coordinate systems to help locate the robot's position in the environment more accurately. In this topic, only 1 sensor is used for positioning, which is Lidar.

- **odometry source:** information about the robot's speed (read from the encoder) or IMU to estimate the robot's position compared to the previous position.
- **move_base_simple/goal:** Determine the end point (position) the robot needs to move to.
- **map_server:** contains data about the map that has been pre-established in the mapping section for the robot.
- **sensor source:** get sensor data from the nodes.
- **global costmap:** map system built from lidar and added some scale components to adjust the map system.
- **local costmap:** current map system when the robot is in the environment and compare with the global costmap system
- **global planner:** construction movement plan from global costmap..
- **local planner:** construction movement plan from local costmap.

2. Algorithms used:

A robot system with SLAM application usually has 3 main algorithms, related to 3 core problems of SLAM, which are mapping, localization and path planning.

3.1 Mapping algorithm:

In this project, we use several hardware components integrated in the ROS 2 framework that support manual control for mapping. For mapping, we use the RPLIDAR A1 sensor, a low-cost 360-degree LIDAR scanner ideal for indoor SLAM applications. It provides a scanning range of up to 12 meters and supports an adjustable scanning frequency between 2 Hz and 10 Hz, generating up to 360 samples per rotation. The LIDAR is connected via a serial interface and its driver publishes real-time data to the `/scan` topic using the `sensor_msgs/LaserScan` message type, which is then processed by the Cartographer SLAM algorithm to generate a 2D occupancy grid map.

For manual control of the robot, especially during mapping or initial testing, we use a PlayStation 3 (PS3) controller. Input from the gamepad is read by a custom ROS 2 Python node called `gamepad_controller.py`, which converts the joystick and button

inputs into velocity commands (`geometry_msgs/Twist`) and publishes them to the `/cmd_vel` topic. This allows the user to directly control the robot in real time.

The control signals generated from the gamepad are transmitted to the Arduino Uno via the UART serial interface, which is managed by a second script called `ros_serial_bridge.py`. This bridge node is responsible for encoding and forwarding the velocity commands to the BTS7960 for sending the PWM pulse to the motor.

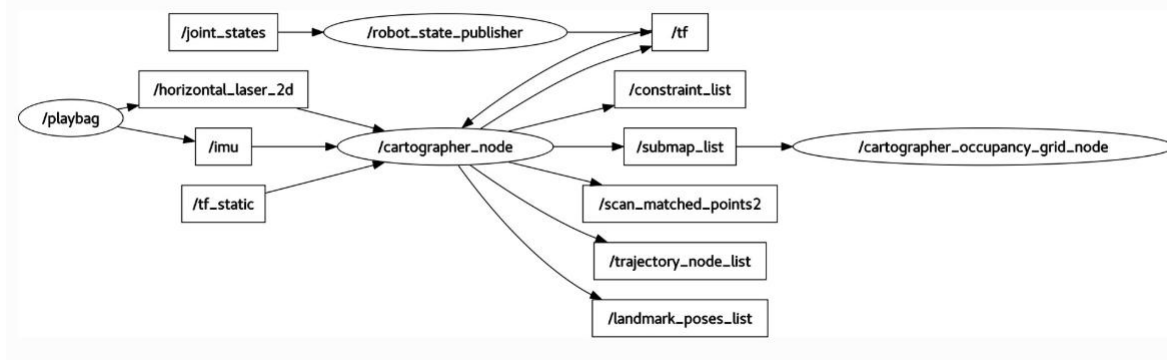


Figure 20 : Catographer node

<https://wiki.ros.org/>

Cartographer uses an algorithm known as Pose Graph SLAM, which consists of two main stages:

- **Local SLAM:** Using input from sensors (LaserScan or PointCloud) and an estimate of the current position (odometry), the algorithm calculates the relative positions between frames to incrementally build local submaps. This enables the robot to construct maps in real time.
- **Global SLAM (Loop Closure):** When the robot revisits a previously mapped area, Cartographer performs loop closure detection and readjusts the entire map using graph optimization algorithms. This process helps eliminate drift and distortion caused by accumulated odometry error.[3]

We utilize the `cartographer_ros` package in conjunction with the core cartographer library to perform real-time environment mapping. The implementation process includes:

- Installing required packages: `cartographer`, `cartographer_ros`, and `cartographer_ros_msgs`.

- Creating *.lua* configuration file to define scan frequency, sensor range, robot speed, and other parameters.
- Writing a launch file to run *cartographer_node* and visualize results in Rviz2.
- During operation, the robot publishes Lidar data to the `/scan` topic, and Cartographer processes this data to incrementally construct the map.

After completing the SLAM process, the map is saved as *.pgm* and *.yaml* image files to serve the following navigation steps such as Navigation Stack.

3.2 AMCL algorithm:

In a ROS 2-based autonomous robot system, localization is achieved using the Adaptive Monte Carlo Localization (AMCL) algorithm, implemented through the `nav2_amcl` package in the Navigation 2 Stack (Nav2). This is an upgraded and modular version of the AMCL node from ROS 1, designed to work efficiently with ROS 2's distributed and real-time architecture.

AMCL is a probabilistic localization algorithm based on a particle filter, which estimates the robot's pose (x, y, θ) in a known map (`/map`). The algorithm operates in three main phases:

- **Prediction (Motion Update):** Each particle updates its state based on the robot's motion model (typically derived from odometry or velocity commands).
- **Correction (Sensor Update):** Incoming sensor data (e.g., Lidar data on `/scan`) is used to evaluate how likely each particle's estimated position is.
- **Resampling:** Particles with higher likelihoods are retained or duplicated, while unlikely particles are discarded. AMCL applies adaptive resampling, dynamically adjusting the number of particles based on localization uncertainty.

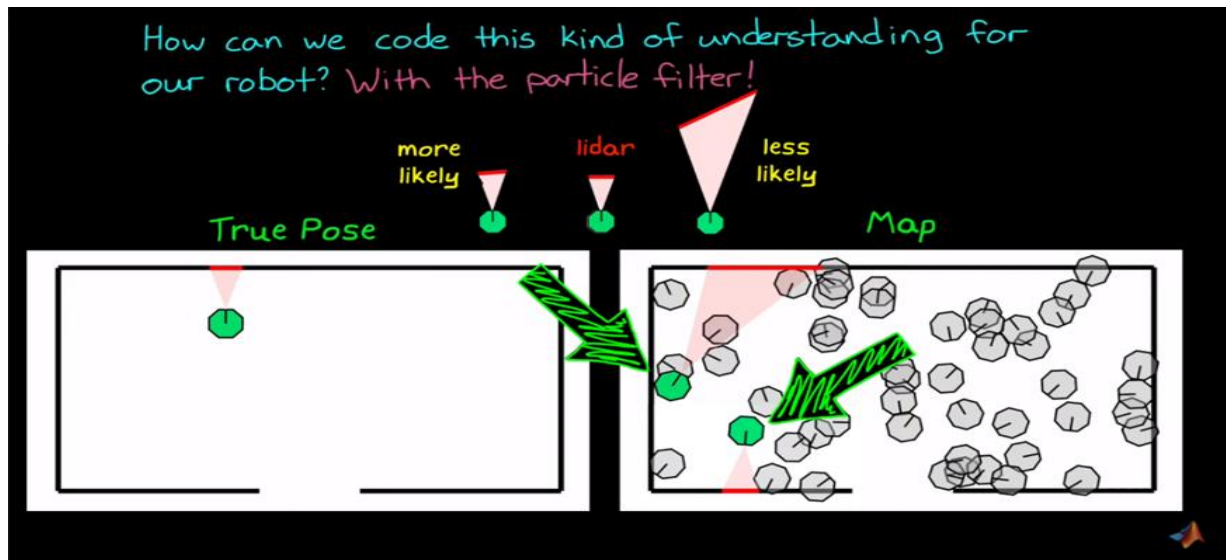


Figure 21 : Describe the operating principle of AMCL

<https://maitreyakv.medium.com/robot-localization-and-the-particle-filter>

In this project, the robot performs localization using a digital map, specifically an occupancy grid map that has been previously generated through SLAM (e.g., using Cartographer or SLAM Toolbox). This map is essential for determining the robot's position within a known environment.

AMCL subscribes to the digital map from the *map_server* node, typically published on the */map* topic as a message of type *nav_msgs/msg/OccupancyGrid*. This occupancy grid map contains probabilistic information about free space, obstacles, and unknown regions in the environment, encoded in a 2D grid. It serves as a reference for comparing real-time sensor observations from the robot to estimate its current pose.

The robot also provides real-time Lidar data via the */scan* topic (*sensor_msgs/msg/LaserScan*). AMCL uses this data to compare current observations against the stored map, calculating the likelihood of different poses and selecting the most probable one using a particle filter approach. If configured, AMCL may also incorporate odometry data (*/odom*) to improve the prediction step of the particle filter.

Once the robot's pose is estimated, AMCL publishes it to the */amcl_pose* topic as a *geometry_msgs/msg/PoseWithCovarianceStamped* message. This pose includes both

the estimated position and orientation of the robot as well as the covariance matrix representing uncertainty.

Mathematical Representation

Assuming x_t is the state (position and orientation) of the robot at time t , z_t is the observation from the sensor, u_t is the control (movement command), then the localization problem is described by the posterior

$$p(x_t | z_{1:t}, u_{1:t}) \approx \sum_{i=1}^N w_t^{[i]} \delta(x_t - x_t^{[i]})$$

probability:

Where:

- $x_t[i]$: position of the i -th particle
- $w_t[i]$: weight corresponding to the degree of agreement with the observation
- δ : Dirac delta function

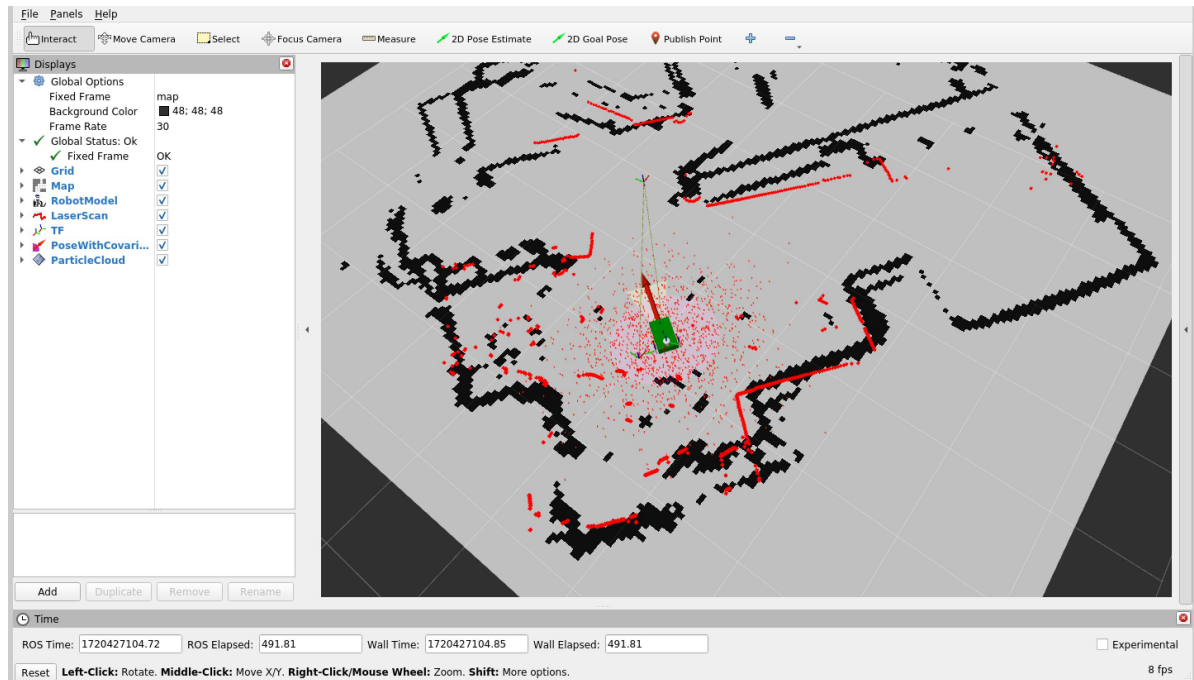


Figure 22 : Localization algorithm AMCL

By using a pre-built digital map and the AMCL algorithm, the robot can localize itself accurately in real time within the environment. This process is critical for enabling autonomous navigation, allowing the robot to understand where it is and plan safe paths to target destinations.

3.3 Robot navigation algorithm

3.3.1 Costmap 2D

In robot navigation, the weight map is used to calculate the obstacle area, the potential collision area, and the area where the robot can move. Depending on the navigation type, the weight map can be divided into two parts.

- ◆ One is the `global_costmap`, which sets the movement plan for robot navigation in the entire space of the map.
- ◆ The other is the `local costmap`, which is used for the movement plan and obstacle avoidance in the limited space around the robot.
- ◆ The weight map represents values from 0 to 255. Each value determines where the robot can move or collide with obstacles:
 - 000: the open space where the robot can move.
 - 001 - 127: the area with a low probability of obstacles appearing.
 - 28 - 252: the area with a high probability of obstacles appearing.
 - 253 - 254: area with obstacles. 255: area occupied, robot cannot move

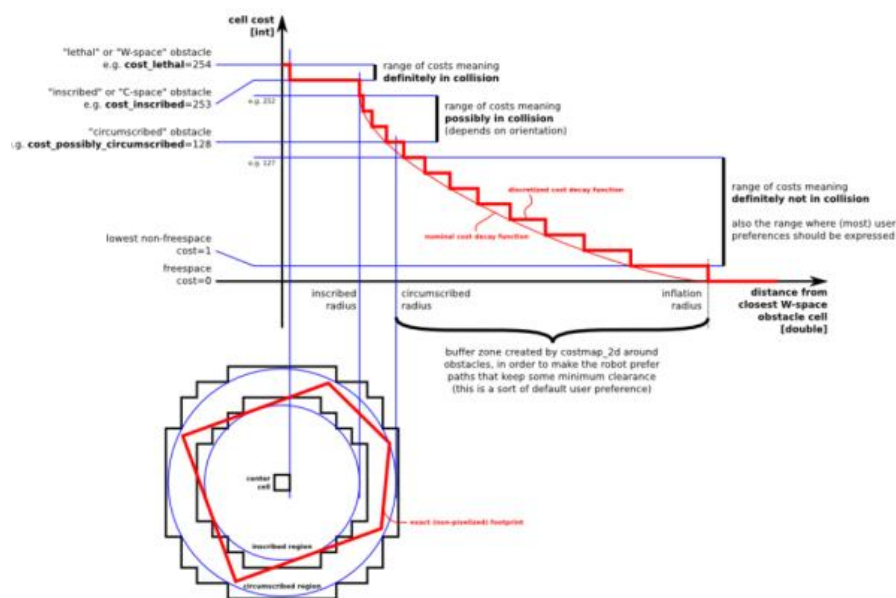


Figure 23: Weight map values influence robot behavior

https://wiki.ros.org/costmap_2d

- ◆ **Cost_lethal:** the value of a cell containing an obstacle. If the robot's center of gravity enters this area, the robot will collide
- ◆ **Cost_inscribed:** the value of a cell from the obstacle to the robot's inscribed

radius. If the robot's center of gravity is in this area, the robot will collide with the obstacle.

- ◆ **Cost_possibly_circumscribed:** the value of costmap, which is like cost_inscribed but is the value from the obstacle to the robot's circumscribed radius. If the robot's center of gravity enters this area, the possibility of the robot colliding with the obstacle depends on the direction of the robot
- ◆ **Cost_freespace:** Has a value of 0, indicating that there are no obstacles here, the robot can move freely in this area.
- ◆ **Cost_unknown:** Cells that do not have information about the value of costmap.

3.3.2 Dijkstra's algorithm for global planner

In navigating a robot to reach a desired location, we must plan a path from the starting point (the current position of the robot) to the ending point (the desired destination). In order to optimize the path, we need to choose the shortest path between the two points. With this requirement, Dijkstra's algorithm is one of the simple algorithms to find the most optimal path.

The problem for Dijkstra's algorithm is to have a graph $G = (V, E)$ (where V is the set of vertices and E is the undirected or directed edges), each edge will have a weight always positive and a source vertex s . We need to calculate the shortest path from the source vertex to each vertex on the graph. To understand better, let's look at the following figure:

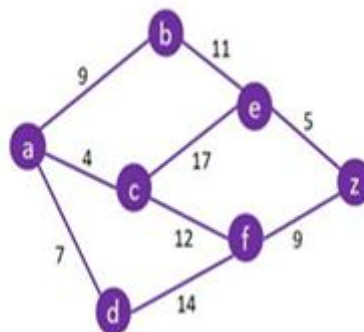


Figure 24 : System structure diagram on ROS2

Step 1: Select node A as the source node, calculate the distance from node A to

adjacent nodes.

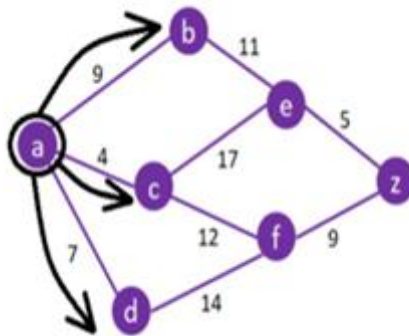


Figure 25: Select node

Step 2: The distance (A –C) is the shortest. Consider node C, calculate the sum of the distances from node C to adjacent nodes.

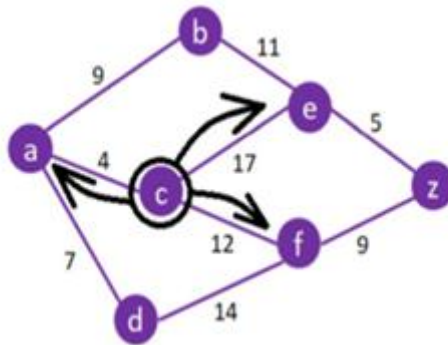


Figure 26: Shortest distance

Step 3: The distance from A to D is the smallest. Consider node D and calculate the distance from node D to adjacent nodes. We calculate that the path from node (A –D –F) = $7+14=21 > (A -C -F)=16$. The shortest path from (A – F) should be kept as (A –C –F).

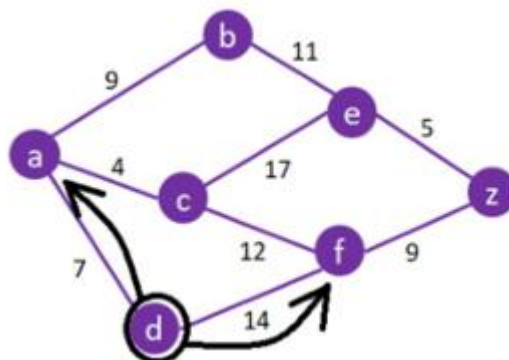


Figure 27: Minimum distance

Step 4: At node B, the distance from A to E has two paths. The distance from (A –C –E) = $4+17 = 21 > (A-B-E) = 9+11=20$. So, we choose the shortest path from (A –E) is (A –B –E).

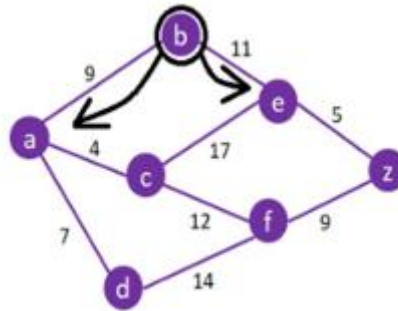


Figure 28: Distance traveled

Step 5: Then, we continue to consider node F similar to the above process.

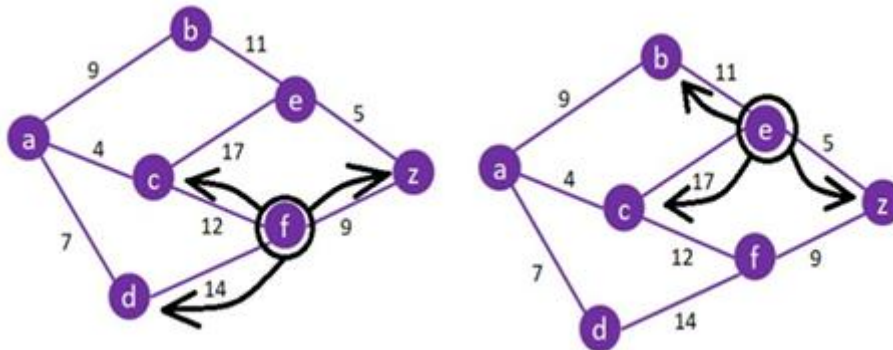


Figure 29: Consider node F next.

Step 6: At node E, the distance from A – Z is (A –C –E–Z) = $4+17+5 = 26 > (A –C –F–Z) = 4+12+9 = 25$. So we choose the smallest path from A – Z is (A –C –F–Z).

Plan	Source Node	Nodes					
		B	C	D	E	F	Z
A (0)	A	9 (A-B)	4 (A-C)	7 (A-D)	∞	∞	∞
A-C (4)	C	9 (A-B)	✓	7 (A-D)	21 (A-C-E)	16 (A-C-F)	∞
A-D (7)	D	9 (A-B)	✓	✓	21 (A-C-E)	16 (A-C-F)	∞
A-B (9)	B	✓	✓	✓	20 (A-B-E)	16 (A-C-F)	∞
A-C-F (16)	F	✓	✓	✓	20 (A-B-E)	✓	25 (A-C-F-Z)
A-B-E (20)	E	✓	✓	✓	✓	✓	25 (A-C-F-Z)

Table 8: Algorithm result

Through the above algorithm we can easily find the shortest path from the starting point to the destination point.

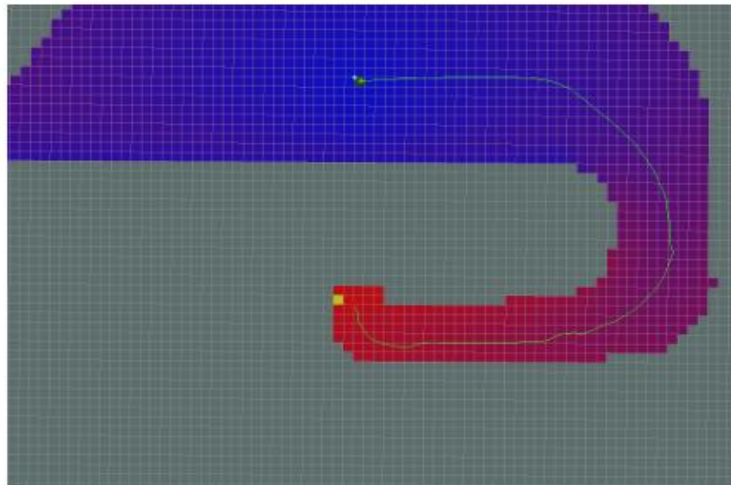


Figure 30 : Path planning using Dijkstra's algorithm.

3.3.3 Pure Pursuit trajectory tracking algorithm

The Pure Pursuit algorithm is one of the simple and effective trajectories tracking control algorithms for autonomous robots with Ackerman-type steering mechanisms. The main idea of the algorithm is: the robot always steers the rudder to a target point (called the lookahead point) located on the desired trajectory, a predetermined distance from the robot (lookahead distance). The goal is to adjust the steering angle so that the robot gradually approaches this point in a stable manner.

Key variables and parameters

- (x,y) : Current position of the robot in the global coordinate system.
- θ : The robot's current orientation angle relative to the global x-axis.
- L : Robot wheelbase (distance between front and rear axles).
- l_d : Lookahead distance (adjustable parameter).
- (x_t,y_t) : Coordinates of the lookahead point on the target trajectory.

Geometric Analysis

The algorithm assumes that the robot will move in a circular arc to reach the lookahead point. Then, we can calculate the circular arc radius R geometrically:

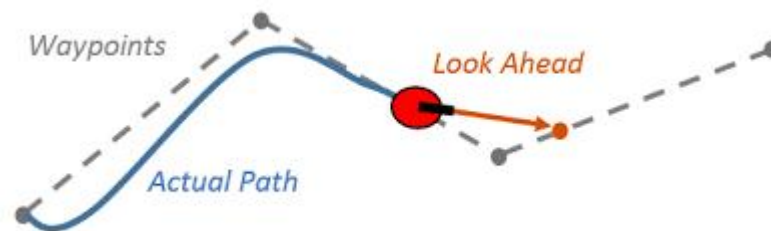


Figure 31 : Pure Pursuit trajectory tracking

<https://www.researchgate.net/figure/illustration-of-Pure-Pursuit>

From the triangle created by the robot and the lookahead point, we have:

$$\alpha = \arctan 2(y_t - y, x_t - x) - \theta$$

This is the angle between the robot's direction and the line connecting to the lookahead point.

The ideal arc radius is given by:

$$R = \frac{l_d}{2 \sin(\alpha)}$$

From there, the required steering angle of the front wheel δ (steering angle) is calculated according to the Ackerman formula as follows:

$$\delta = \arctan \left(\frac{2L \cdot \sin(\alpha)}{l_d} \right)$$

III. Obtained result:

After the process of building the Mobile robot model, setting up the ROS2 operating system environment on the embedded board, configuring and programming the necessary nodes for the process in the system, conduct experimental verification to calibrate the parameters and find out the advantages and disadvantages of the system.

1. TEST ENVIRONMENT

To evaluate the accuracy and performance of the proposed solution, the positioning system is integrated into the robot as shown in Figure 4. The test is conducted in a closed indoor environment, tiled floor, suitable terrain for the designed robot and ensures that external factors such as operating time, travel distance, etc. do not affect the test results.

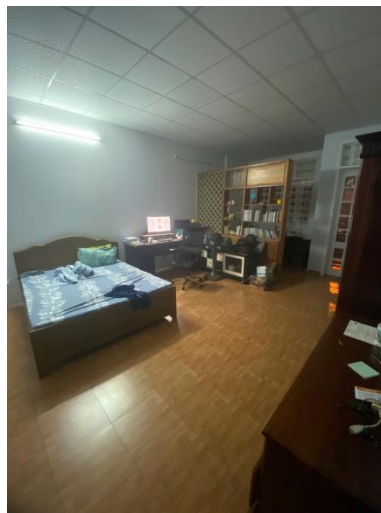


Figure 32 : Test environment

2. Mapping process and map results

Mapping will be done by sending the robot carrying lidar to scan the room, places where the robot needs to go through. In places where the space is not yet defined, we can remotely control the robot to map.

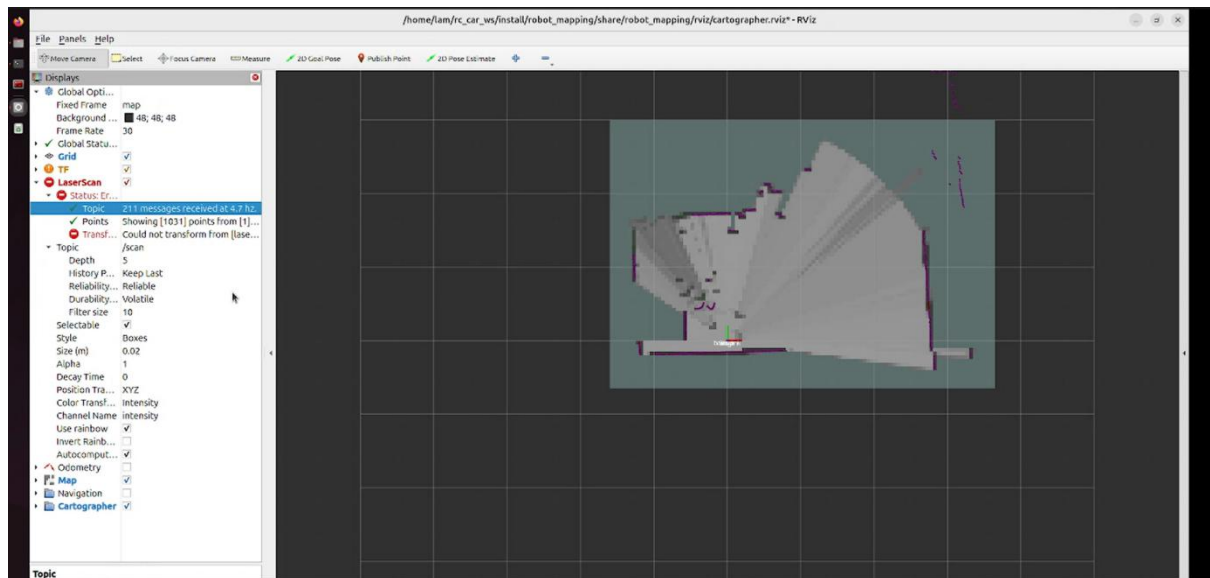


Figure 33 : Robot location and initial map

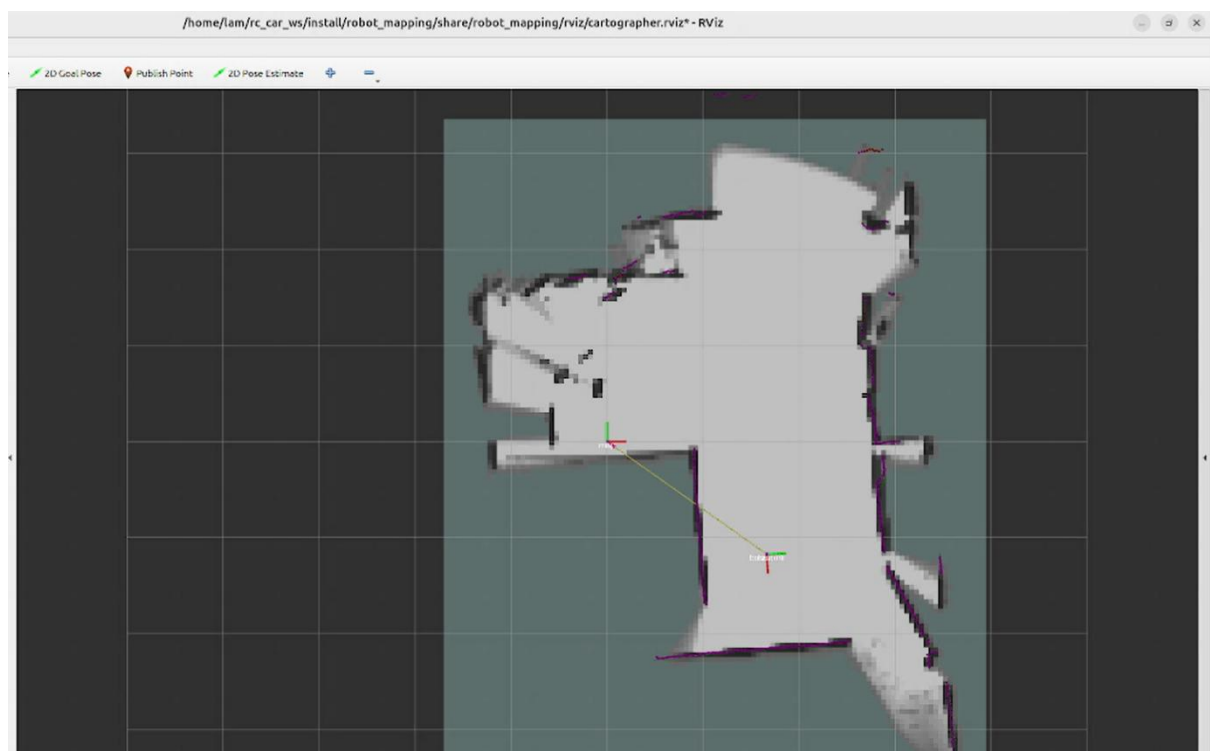


Figure 34 : Mapping process

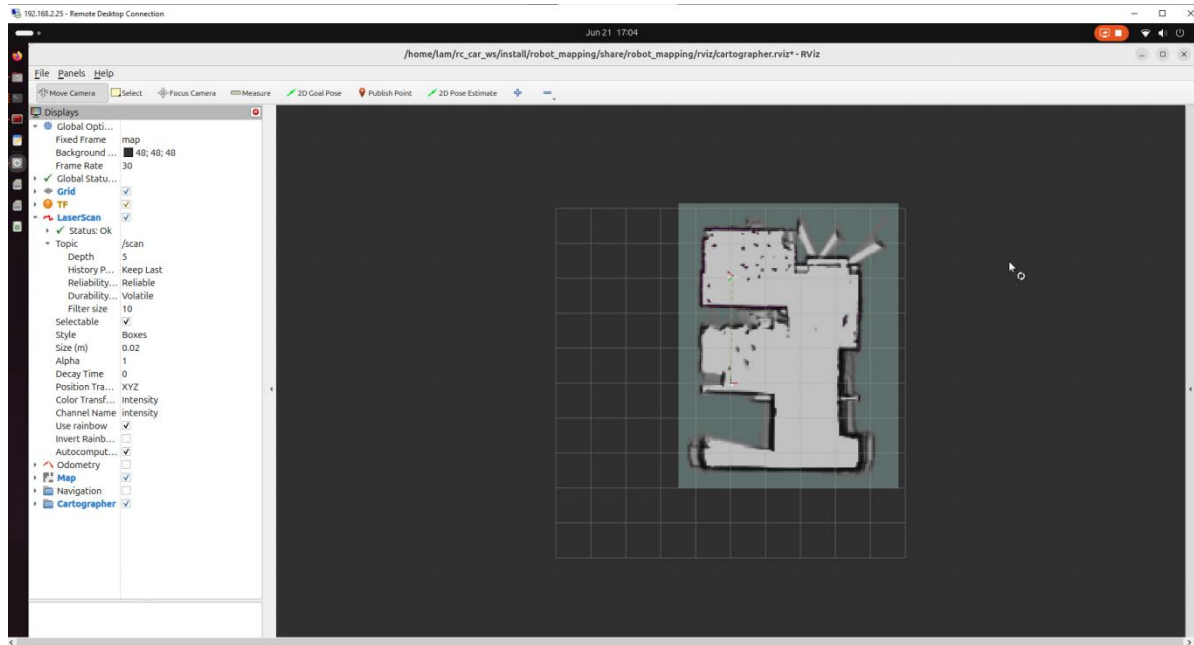


Figure 35 : Final map

The map after being built will integrate Costmap 2D before being put into use to plan the path for the robot. Costmap 2D is a 2D or 3D map of the space blocked by obstacles or walls in the environment and spread out in the map based on the grid map and the width of the spread area. The figure below shows the map integrated with Ros2's Costmap_2D software package. The spread parts which cannot be moved to ensure that the robot does not have an accident when getting too close to the obstacle or wall. The dark black area represents the obstacle or wall of the costmap map, the light pink area surrounding the black area represents the spread of the obstacle or wall, its width depends on the radius of the robot, and the green polygon represents the robot's perimeter or footprint. To avoid collision, the robot's footprint must not touch the black area, and the robot's center must not touch the pink area.

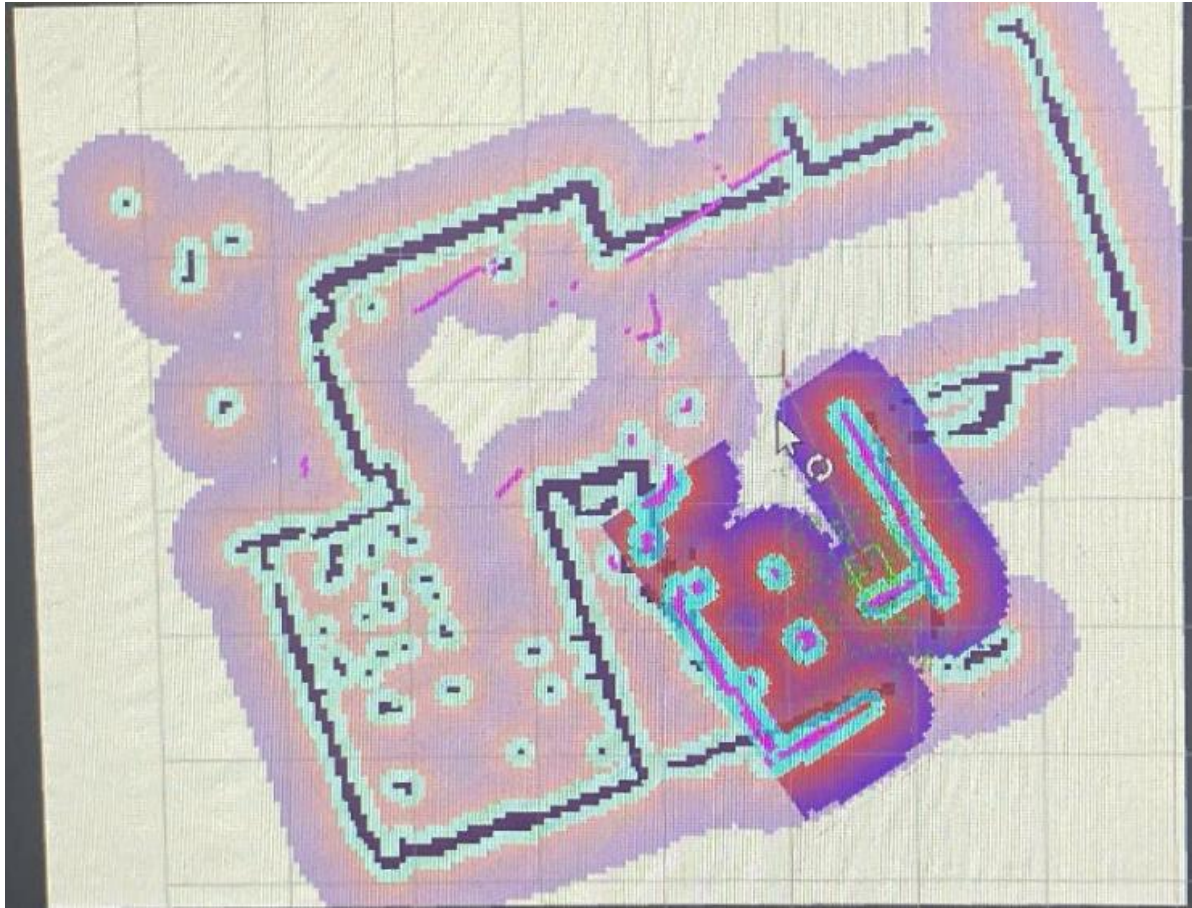


Figure 36 : Costmap 2D

3. Locate the robot on the map

AMCL takes in an existing laser-based map that has been done in the mapping, laser scanning and message transformation part, and the output results provide estimates.

At startup, AMCL initializes its particle filter according to the provided parameters.

Note that, by default, if no parameters are set, the initial filter state will be a moderately sized particle cloud centered around (0,0,0). The particles after being generated from the sensor data, will be assigned a certain weight, after resampling, the particles with higher weights will be retained, the particles with lower weights will be excluded, from which the particle with the largest weight will be assigned as the current position of the robot. This process is continuous every time there is data returned from the sensor, the robot both positions itself and updates the map to the database at the end of the process. The figure below shows the particle sampling process when the robot is in the process of establishing the map, as in the figure, the particles are red arrows after the resampling process, they tend to converge to the

robot's position in reality, in this case, the longer the robot rotates in place the more the particles will converge, thereby giving the most accurate coordinates. The purple lines show the lidar laser lines that match fixed obstacles, which coincide with what was created on the map after mapping process.

7

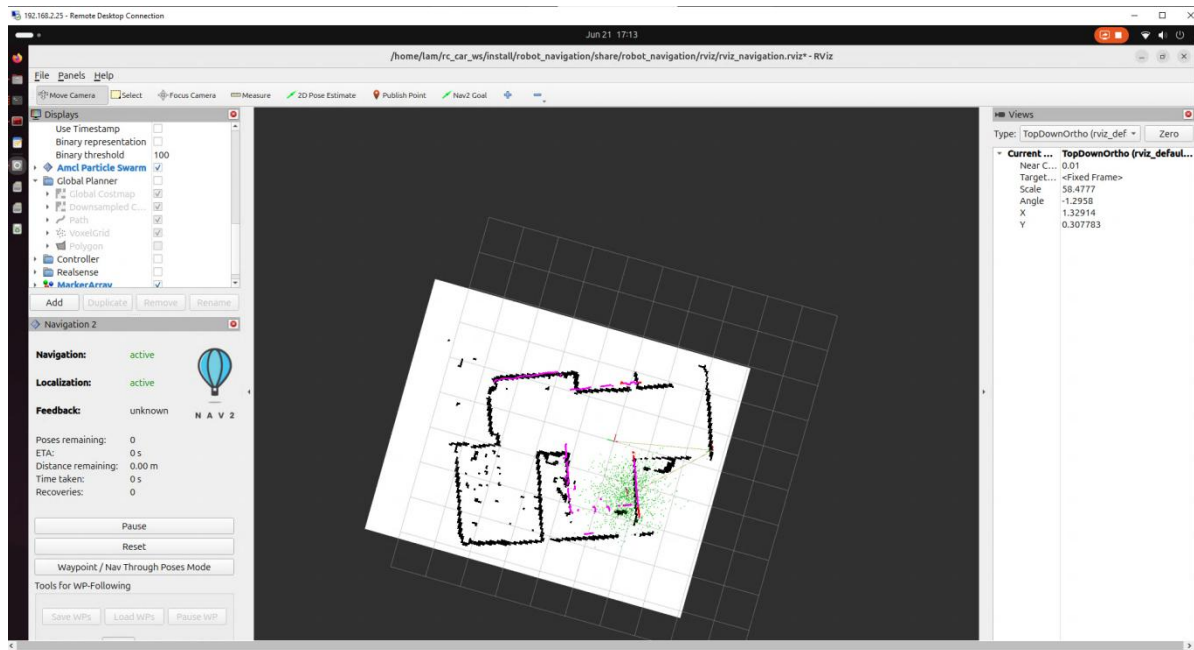


Figure 37 : AMCL algorithm

3. Navigation for robots

Plan a path to the destination for the robot and start moving.

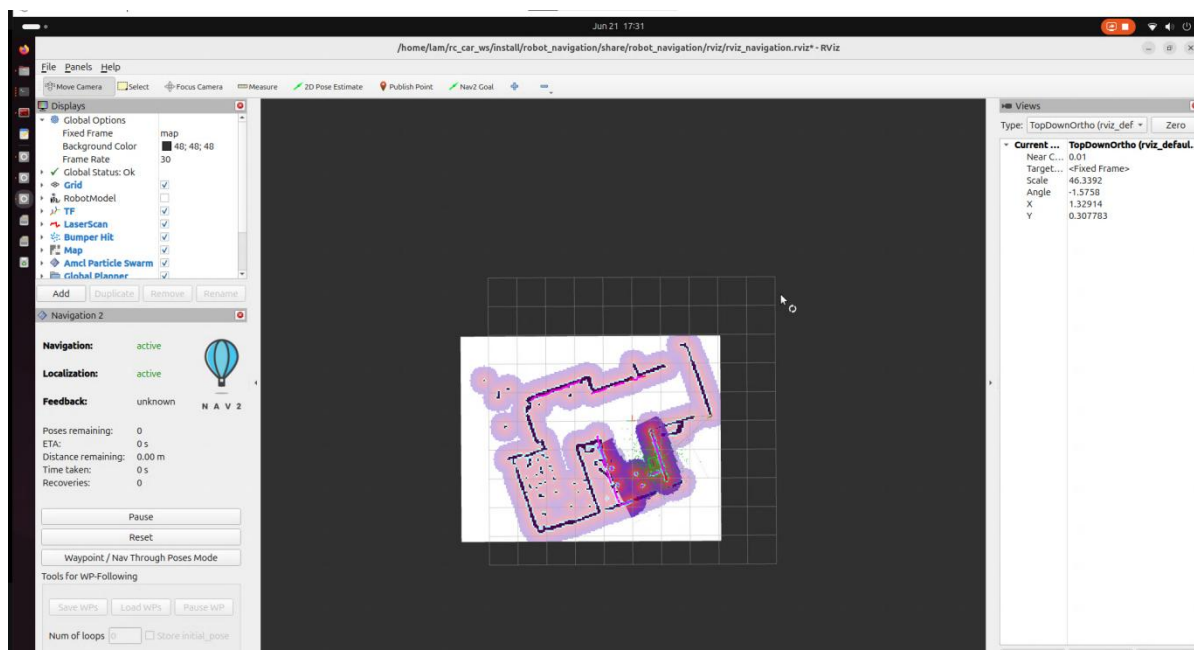


Figure 38 : Initial position

Move to the destination according to the previously planned route.

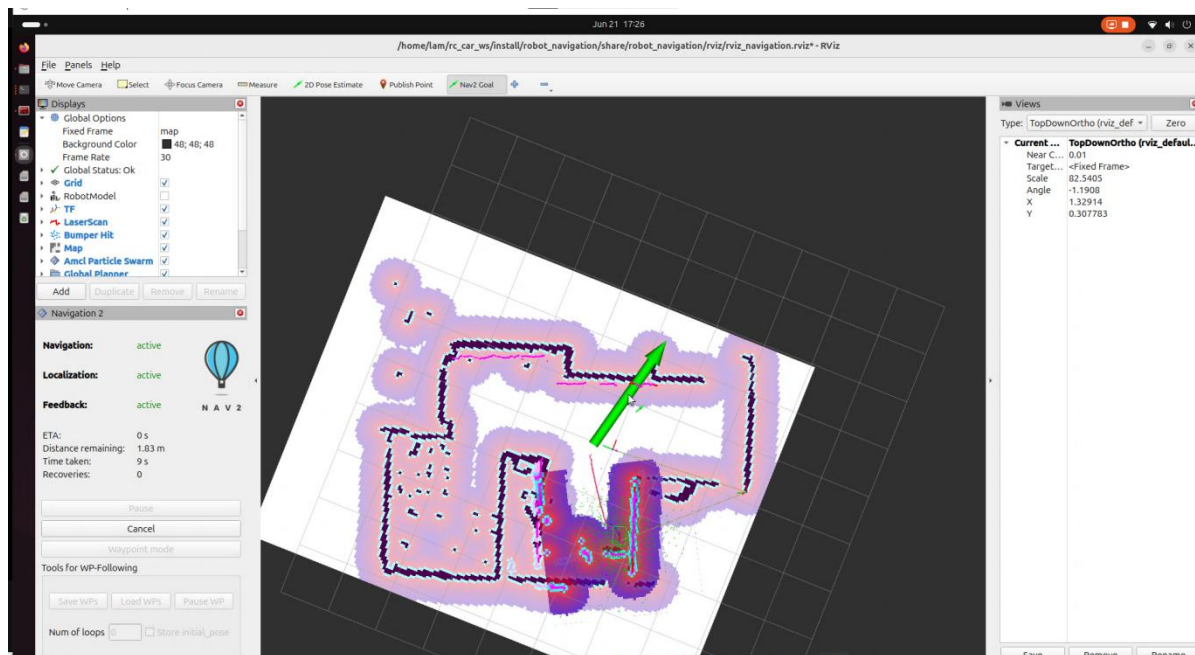


Figure 39: Navigation and path planning

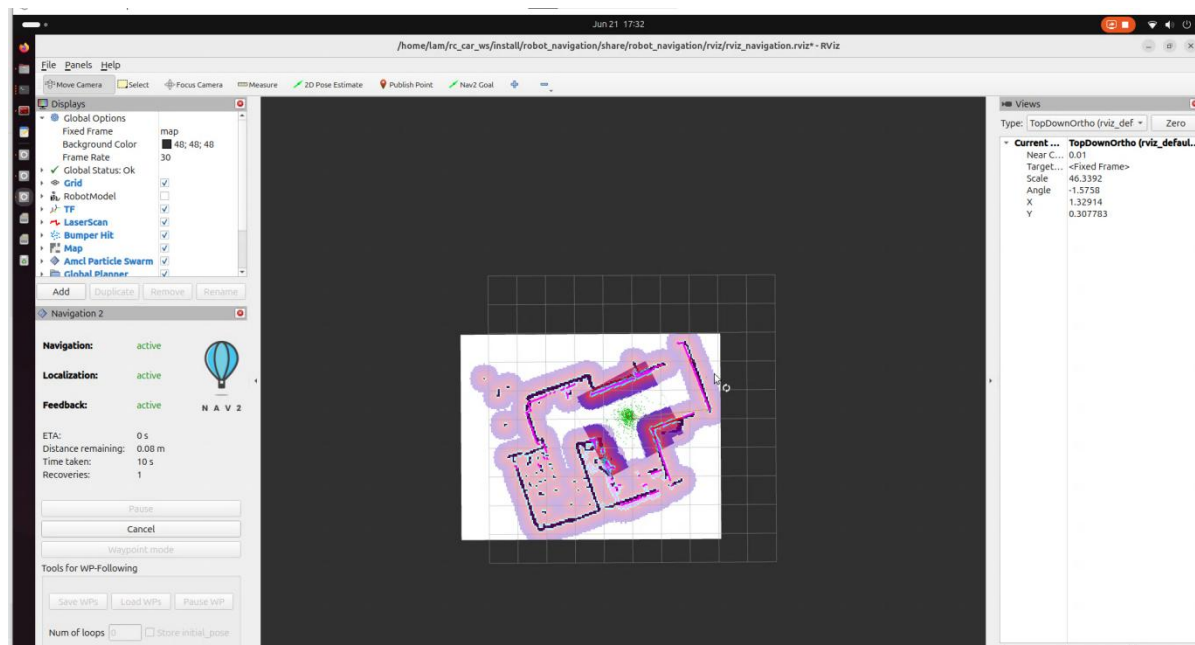


Figure 40: New location after moving

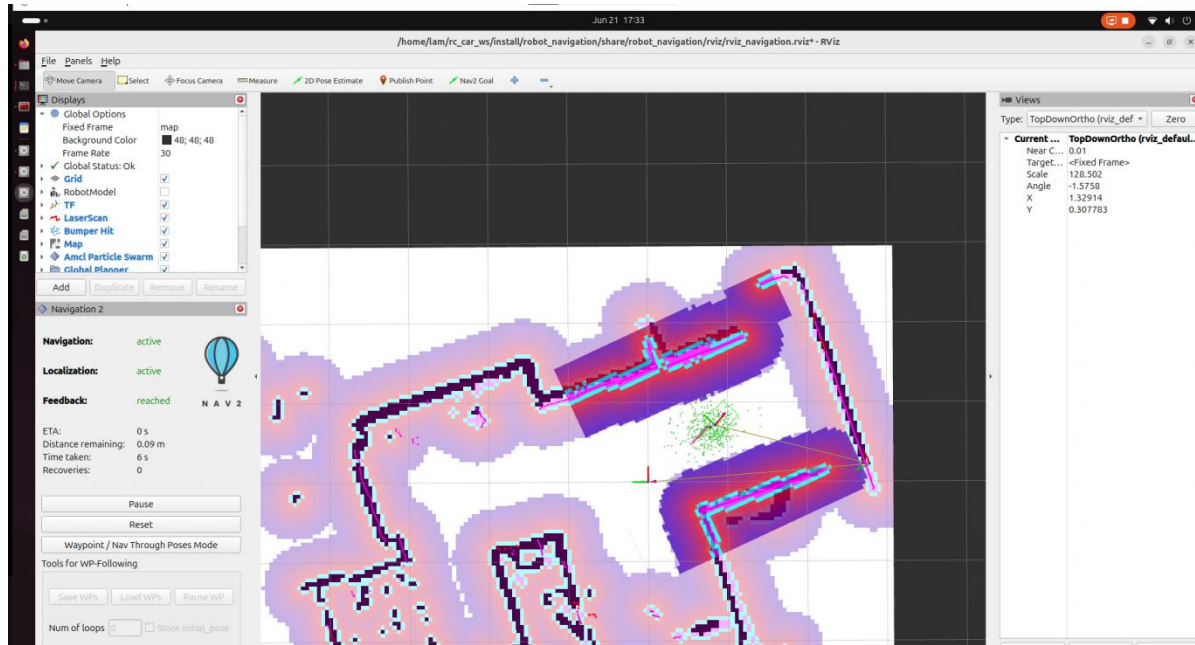


Figure 41: Changing location

The robot has moved close to the desired destination according to the given path plan. However, there is still a small difference between the robot's position and the destination and the robot's movement is not stable during the movement.

4. Conclusion:

After the process of design, construction, and implementation, the robot has successfully completed its core functions, including building an environmental map using the SLAM algorithm, localizing its current position, and autonomously navigating to specified target locations on the map. Data from the LiDAR sensor is processed through a SLAM algorithm to generate a real-time 2D spatial map and determine the robot's relative position within the environment.

The navigation system has been effectively deployed by integrating both Global, enabling the robot to autonomously plan its path and reach specified destinations in a stable and safe manner. During operation, the robot continuously updates the map and adjusts its trajectory to adapt to environmental changes.

However, there are still some limitations. The robot currently performs best in simple environments with few obstacles and limited area. Its performance in narrow or dynamic spaces has not been thoroughly tested. Furthermore, there is still a noticeable deviation between the robot's actual final position and the intended target location after the navigation process is completed, especially in uneven environments or when sensor data becomes unstable. The system also lacks a recovery mechanism in case of localization loss, and it does not yet support map reuse in subsequent sessions.

Overall, the robot model has successfully fulfilled the basic requirements of mapping, localization, and autonomous navigation, providing a solid foundation for further development and application in more complex real-world scenarios.

CHAPTER 4: INTEGRATED OV5647 CAMERA FOR VISUAL IMAGE MONITORING

I. Ideas and approaches

1. RGB image surveillance system concept using OV5647 camera

1.1. System Concept for RGB Image Monitoring using OV5647 Camera

In the context of increasing demands for environmental safety, early hazard detection, and autonomous surveillance, the integration of computer vision into autonomous robots becomes vital. Particularly in areas with high risk of fire or human negligence, early identification of activities such as smoking or the presence of fire and people can play a crucial role in triggering timely alerts and thermal inspections.

This system incorporates the OV5647 RGB camera, directly interfaced with a Raspberry Pi, to capture real-time visual images of the environment. These images are analyzed using a deep learning-based object detection model — **YOLOv8n**, which is specifically trained and optimized to detect the following classes: **Person, Smoking, Fire**

The detection results are visualized on-screen with bounding boxes and labels, and the system can optionally trigger the **thermal camera** or **send alerts** when a high-risk object is detected.

1.2. Overall Methodology and Design Strategy

To build a lightweight yet effective vision system suitable for edge deployment (on Raspberry Pi 5), the approach follows several well-defined stages:

- Acquiring RGB Image Data from OV5647 Camera
- The **OV5647 camera module** communicates with the Raspberry Pi through the **CSI (Camera Serial Interface)** for high-speed data transmission.
- Resolution is configured at **640×480** pixels to balance between detection accuracy and computational load.
- Deploying the YOLOv8n Object Detection Model

- The **YOLOv8n (Nano)** variant is chosen due to its small model size (~6 MB) and fast inference speed, making it ideal for real-time edge processing.
- The model is trained on a custom dataset consisting of domain-specific classes: **people, fire, and smoke**.
- The model is deployed using the **Ultralytics Python SDK** offering better inference speed on resource-constrained hardware.
- Image Processing and Output Visualization
- The captured image is passed through the YOLOv8n model for prediction.
- Detected bounding boxes and class labels are rendered using **OpenCV**, allowing real-time visualization on an HDMI display or via remote desktop.
- Performance Optimization for Real-Time Processing

To achieve **maximum frames per second (FPS)** and minimize inference delay, the following strategies are applied:

- **Resolution reduction:** Processing is done at 640×480 resolution, which is sufficient for detection and suitable for the YOLOv8n model.
- **Model optimization:** If required, the YOLOv8n model for faster inference.
- **Minimal post-processing:** Only essential results are rendered to save rendering time.

II. Theoretical Background

1. OV5647 Image Sensor

The OV5647 is a 5-megapixel CMOS image sensor developed by OmniVision, widely used in Raspberry Pi camera modules. It provides a balance between high-resolution image capture and power efficiency, making it ideal for embedded vision applications, including real-time surveillance and object detection.

- **CMOS Sensor Architecture[4]**

CMOS sensor is an electronic device with a structure consisting of many extremely small pixels (photoelectric cells) placed close together to form an array (matrix). The surface is covered with a Bayer Pattern coating including red, green, blue dots and placed in a ratio of 2 green, 1 red and 1 blue.

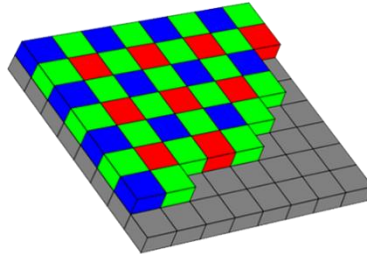


Figure 42 : CMOS_Bayer_Array.(<https://thietbisohoa.vn/cong-nghe-cam-bien-cmos>)

Since the Bayer filter uses 50% Green, 25% Red, and 25% Blue, it can also be referred to as RGBG, GRGB, or RGGB. Bayer uses twice as much Green as the other two colors because the human eye is most sensitive to Green.

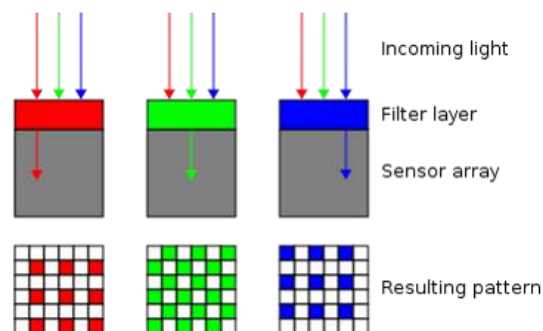


Figure 43: CMOS Sensor Pixel Filtering with Bayer Pattern.(<https://thietbisohoa.vn/cong-nghe-cam-bien-cmos>)

Each Pixel only filters and records one of the three colors because Pixels cannot determine their own red, green, and blue values. The array sensor will take a picture to record the image, then to get a full color image, the Demosaicing algorithm will be used to interpolate the color values for each Pixel.

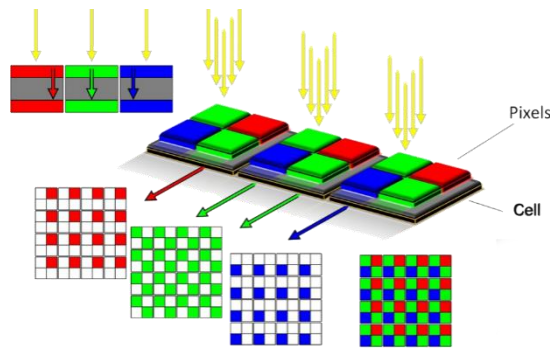


Figure 44: CMOS Bayer Filter Mosaic Structure. (<https://thietbisohoa.vn/cong-nghe-cam-bien-cmos>)

In fact, some companies have now designed array sensors with 3 layers of red, green, blue filters placed on top of each other. With this method, the image signals after being captured will also have all 3-color information. Simulating the Foveon X3 sensor line compared to conventional array sensors.

- **MIPI CSI-2 Interface**

The sensor communicates via the MIPI CSI-2 (Camera Serial Interface 2) standard, which supports high-speed, low-latency transmission of image data from the camera to the processor. This makes it suitable for real-time video streaming applications such as object tracking, motion detection, and AI-based vision.

- **Working Principle:**

OV5647 uses a Bayer filter to capture light intensity through an array of red, green, and blue filters. The image data is transmitted to the GPU/CPU using the CSI interface and processed using libcamera or OpenCV for image enhancement, capture, or object detection tasks.

2. YOLOv8n Object Detection Model

2.1 Overview of Computer Vision and YOLOv8

- **Computer Vision**

Computer Vision (CV) is a subfield of Artificial Intelligence (AI) and computer science that focuses on enabling machines to “see,” interpret, and understand visual data (such as images and videos) in a way similar to how humans perceive the world. The ultimate goal of computer vision is to extract meaningful information from visual inputs and use it to perform tasks such as:

- Object detection
- Image classification
- Image segmentation
- Object tracking
- Depth estimation and 3D reconstruction

Computer vision plays a critical role in various real-world applications, including autonomous vehicles, intelligent robotics, medical diagnostics, surveillance systems, smart agriculture, and more.

In recent years, advancements in deep learning — particularly Convolutional Neural Networks (CNNs) — have significantly enhanced the performance and accuracy of computer vision systems, enabling them to achieve near-human or even superhuman capabilities in certain tasks.

- **Introduction to YOLOv8**

YOLOv8 is the newest state-of-the-art YOLO model that can be used for object detection, image classification, and instance segmentation tasks. YOLOv8 was developed by Ultralytics, who also created the influential and industry-defining YOLOv5 model. YOLOv8 includes numerous architectural and developer experience changes and improvements over YOLOv5.

YOLO has been nurtured by the computer vision community since its first launch in 2015 by Joseph Redmond. In the early days (versions 1-4), YOLO was maintained in C code in a custom deep learning framework written by Redmond called Darknet.

Here are a few main reasons why i should consider using YOLOv8

- **State-of-the-Art Accuracy**

YOLOv8 delivers impressive performance across widely recognized benchmarks. On the MS COCO dataset, the medium-sized YOLOv8m model achieves a remarkable 50.2% mAP, outperforming many previous YOLO versions. On the Roboflow 100, a benchmark designed to test real-world, domain-specific use cases, YOLOv8 consistently surpasses the performance of YOLOv5 — making it one of the most robust models available today.

- **Developer-Friendly Design**

YOLOv8 is built with developers in mind. Unlike older versions that required juggling multiple scripts and configurations, YOLOv8 provides:

- A streamlined Command-Line Interface (CLI) for tasks like training, validation, and inference.
- A modern Python API that simplifies integration into custom pipelines and applications.
- This makes the entire development workflow more intuitive, efficient, and scalable.

2.2 YOLOv8 Architecture: A Deep Dive

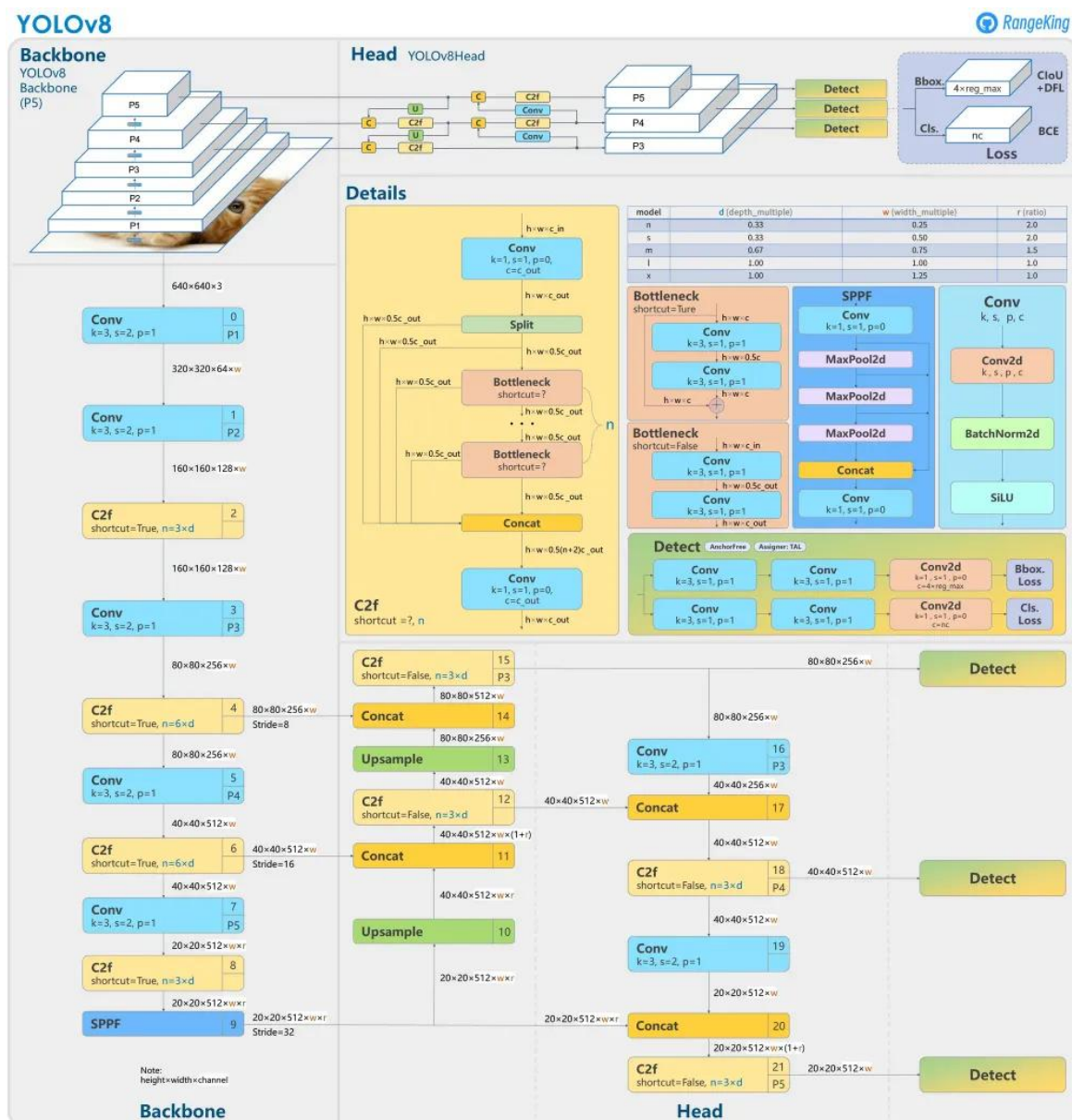


Figure 45 : YOLOv8 Architecture, visualisation made by GitHub user RangeKing. (<https://blog.roboflow.com/what-is-yolov8/>)

The architecture of YOLOv8 is organized into three core components:

- **Backbone** – Responsible for feature extraction from the input image.
- **Neck** – Enhances and fuses features across multiple scales.
- **Head** – Outputs the final detection predictions (bounding boxes, object confidence, and class scores).

This modular structure enables YOLOv8 to balance computational efficiency and representational power.

- **Backbone – Feature Extraction[5]**

The backbone extracts hierarchical features from input images. YOLOv8 introduces the C2f (Cross-Stage Partial Fusion) module as a replacement for the CSP (Cross Stage Partial) blocks used in YOLOv5. This allows for more efficient gradient flow, reduced parameter count, and enhanced learning dynamics.

- Convolutional Layers (Conv)

Standard 2D convolutions are applied with kernel sizes of 3×3 , strides of 2 or 1, and appropriate padding.

Early layers reduce the spatial resolution (e.g., from $640 \times 640 \rightarrow 320 \times 320$) while increasing channel depth.

These layers capture low-level patterns like edges and textures.

- **C2f Modules**
 - The C2f module splits input features into multiple branches.
 - Some branches are passed through lightweight Bottleneck blocks (with optional shortcut connections), while others bypass them.
 - Features are concatenated and passed through a final Conv layer to unify the output.
 - This design improves feature reuse and reduces computational cost while maintaining expressiveness.
- **SPPF – Spatial Pyramid Pooling (Fast)**
 - Located at the end of the backbone, the SPPF module aggregates multi-scale contextual information.

- It performs max pooling with different kernel sizes (5×5 , 9×9 , 13×13), then concatenates the outputs.
- This design enlarges the receptive field without increasing the feature map size, allowing better object localization.
- **Neck – Multi-scale Feature Fusion**

The Neck serves to combine features from different stages (P3, P4, P5), allowing the model to detect objects of varying sizes by fusing high-resolution, low-semantic features with low-resolution, high-semantic features.

- Upsample
- Higher-level features (e.g., from P5) are upsampled (usually via bilinear interpolation) to match the spatial resolution of features from earlier layers (e.g., P4).
- This step ensures proper feature alignment for concatenation.
- Concat
- Features from different levels (after upsampling) are **concatenated along the channel dimension**.
- This enables the network to access both fine-grained and abstract information.
- Re-convolution and C2f
- After concatenation, a set of convolutional layers or another C2f block is applied to **refine the fused features**.
- This ensures that the joint representation is both compact and semantically rich.

This top-down and bottom-up fusion approach is similar to PANet but optimized for simplicity and speed.

- Head – Prediction

YOLOv8 uses a **decoupled detection head**, consisting of three output branches corresponding to three spatial resolutions:

Output Scale	Feature Map Size	Target Objects
P3	80×80	Small objects
P4	40×40	Medium-sized
P5	20×20	Large objects

Table 9: Multi-Scale Feature Maps and Target Object Sizes..

Each head contains:

Conv2D layers to generate three types of outputs:

Bounding box regression (x, y, w, h)

Objectness score

Class probabilities

The final output tensor has the format:

[batch, num_anchors, (x, y, w, h, obj_conf, class_probs...)]

This allows the model to simultaneously detect multiple classes and localize their bounding boxes in real-time.

- **Loss Function**
- YOLOv8 employs an advanced loss function tailored for object detection:
- Bounding Box Loss: Uses CIoU Loss (Complete IoU) to account for distance, overlap area, and aspect ratio alignment.
- Objectness Loss: Binary Cross-Entropy (BCE) loss for determining if an object exists in a cell.
- Classification Loss: BCE or Focal Loss (configurable) to address class imbalance.
- The total loss is a weighted combination of these components, optimized jointly during training.

2.3 Introduction to Supporting Tools for Model Training – Roboflow and Google Colab

2.3.1 Roboflow – A Platform for Dataset Management and Computer Vision Acceleration

In the domain of Artificial Intelligence (AI), particularly in computer vision applications, the quality and structure of the dataset play a fundamental role in determining the accuracy, robustness, and generalization capabilities of deep learning models. However, the process of collecting, labeling, preprocessing, and managing image data is often time-consuming, error-prone, and technically challenging especially without the aid of dedicated tools.

Roboflow is a web-based platform designed to streamline the end-to-end workflow of managing computer vision datasets. It assists users in image annotation, preprocessing, data augmentation, format conversion, and even deployment of trained models. Supporting over 30 standard data formats such as YOLOv5/YOLOv8, TensorFlow TFRecord, Pascal VOC XML, and COCO JSON, Roboflow ensures compatibility with popular deep learning frameworks and tools.

- Intuitive data labeling interface: Users can annotate images with bounding boxes, segmentation masks, or classification labels through a user-friendly graphical interface.
- Automated data augmentation: The platform provides a wide array of augmentation techniques including rotation, flipping, cropping, blur, and brightness adjustment to enhance model generalization.
- Smart dataset splitting: Roboflow enables randomized splitting of the dataset into training, validation, and test sets (e.g., 70-20-10) to ensure balanced data distribution.
- Version control for datasets: Every modification to a dataset is tracked and logged, enabling users to manage and revert to previous versions if needed.

Cloud-based model training and deployment: Roboflow also supports training and deploying models directly in the cloud and provides API endpoints to integrate the model into real-world applications.

2.3.2 Google Colaboratory (Colab) – A Cloud-based Environment for AI Model Training

While Roboflow addresses the data preparation phase, the actual training of deep learning models requires access to significant computational resources, particularly GPUs or TPUs. These resources are often expensive and inaccessible to students or small research teams. To bridge this gap, Google Colaboratory, commonly known as Google Colab, offers a powerful, cost-effective solution that brings cloud computing to AI development.

Google Colab is a cloud-based integrated development environment (IDE) provided by Google. It is built on top of the Jupyter Notebook architecture and offers free access to GPU and TPU resources through the browser.

Notable advantages of Google Colab include:

- Free access to GPU/TPU: Colab provides high-performance hardware such as NVIDIA Tesla T4, P100, and Google's custom TPU chips to accelerate model training.
- Integration with Google Drive: Users can store and retrieve datasets, model checkpoints, logs, and scripts seamlessly through cloud storage.
- Support for Python and popular AI libraries: Google Colab supports deep learning libraries such as TensorFlow, PyTorch, OpenCV, Keras, and Scikit-learn out of the box.
- Real-time collaboration: Similar to Google Docs, Colab allows multiple users to work on the same notebook simultaneously, making it ideal for team-based projects.

- Flexible environment setup: Users can install custom packages using pip install, making Colab highly customizable.

In this project, Google Colab was employed as the main environment for training the YOLOv8n object detection model, with the following workflow:

- Download the dataset from Roboflow using API access.
- Install necessary libraries such as ultralytics, opencv-python, matplotlib, and others.
- Configure and train the YOLOv8n model using the annotated dataset.
- Export the trained model in .pt and .onnx format for deployment on a Raspberry Pi 5 device for real-time object detection.

III. System Integration Workflow

1 Model Preparation and Training with Roboflow and Google Colab

1.1 Dataset Preparation using Roboflow

In any object detection system, dataset quality is critical to the model's accuracy and generalization capability. For this project, Roboflow was employed as the primary platform for dataset annotation, preprocessing, augmentation, and management due to its intuitive interface and advanced pipeline automation.

Dataset Overview:

More than 6,000 images were collected from real cameras and Internet sources, then manually labeled into three object classes: person (humans), smoking (people holding or using cigarettes), fire (visible flames).

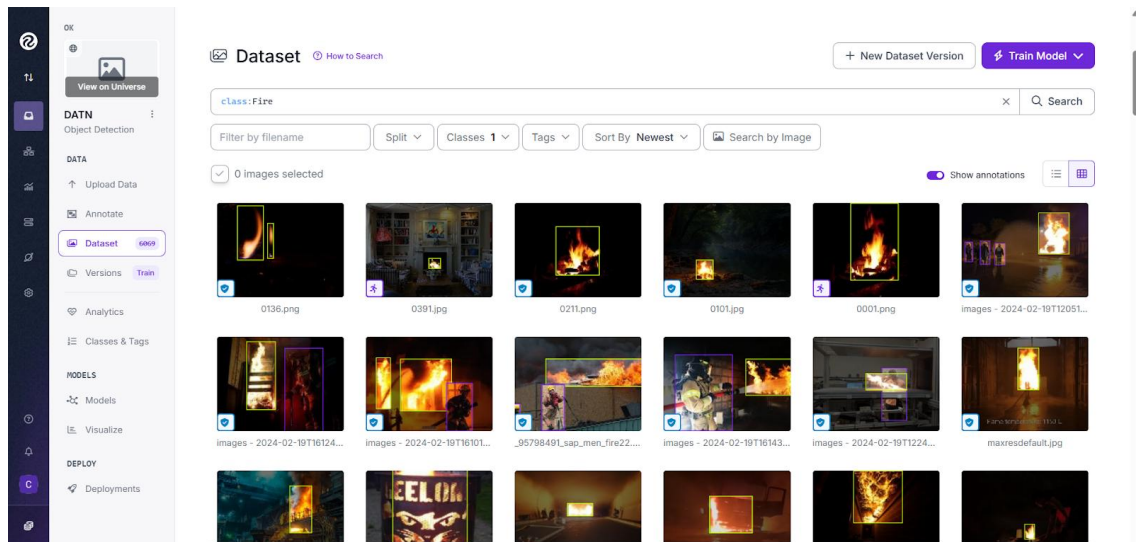


Figure 46: Fire class dataset.

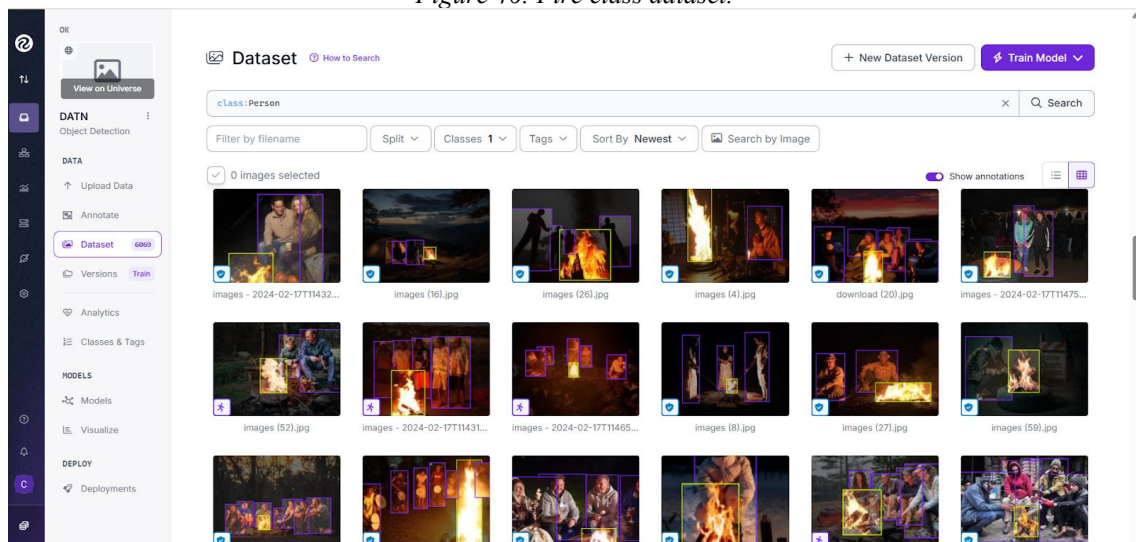


Figure 47: Person class dataset.

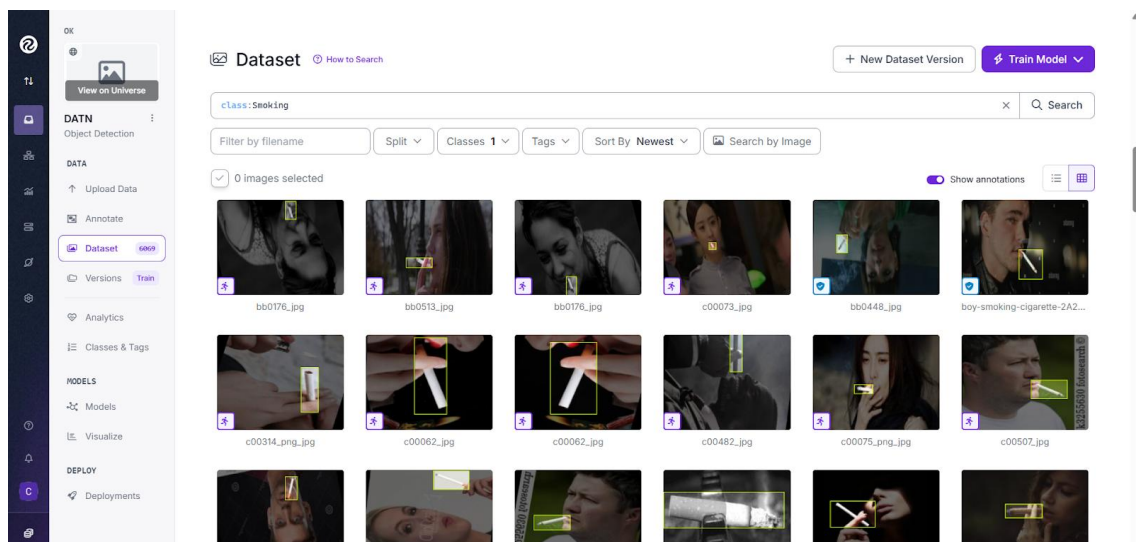


Figure 48: Smoking class dataset.

Processing Steps in Roboflow:

- Manual annotation: All images were manually labeled using Roboflow's built-in bounding box tools to ensure high labeling precision.
- Image resizing: Every image was resized to 640×480 pixels to maintain consistency with the YOLOv8 input size and to optimize real-time inference.
- Annotation format conversion: Roboflow automatically converted bounding box annotations to YOLOv8 TXT format, ensuring compatibility with the training pipeline.
- Automatic data augmentation: To improve model robustness, Roboflow applied several augmentation techniques, including:
 - Horizontal flipping
 - Rotation
 - Brightness and contrast adjustments
 - Exposure simulation.

Versioning: Roboflow supports version control for datasets. In this project, version 9 of the dataset (project ID: datn-tdzab) was used, which provided the best balance between generalization and performance.

1.2 Training with Google Colab

The training process was carried out on Google Colab, a cloud-based Jupyter notebook environment that offers access to free NVIDIA GPUs, which significantly accelerate deep learning tasks. This setup eliminates the need for high-end local hardware while still achieving professional-grade training results.

Model and Training Configuration:

Model Architecture: YOLOv8n (YOLOv8 Nano) – chosen for its small size and suitability for real-time inference on embedded systems.

- Epochs: 80
- Batch size: 16
- Input image size: 640x480
- Training framework: Ultralytics YOLOv8 Python package

Epoch	GPU_mem	box_loss	cls_loss	dfl_loss	Instances	Size			
1/80	2.04G	1.675	2.886	1.669	40	640: 100%		600/600	[02:44<00:00, 3.64it/s]
	Class	Images	Instances	Box(P	R	mAP50	mAP50-95): 100%		31/31 [00:09<00:00, 3.43it/s]
	all	961	1557	0.618	0.522	0.56	0.252		
Epoch	GPU_mem	box_loss	cls_loss	dfl_loss	Instances	Size			
2/80	2.26G	1.672	2.289	1.664	31	640: 100%		600/600	[02:40<00:00, 3.74it/s]
	Class	Images	Instances	Box(P	R	mAP50	mAP50-95): 100%		31/31 [00:07<00:00, 4.11it/s]
	all	961	1557	0.495	0.489	0.469	0.211		
Epoch	GPU_mem	box_loss	cls_loss	dfl_loss	Instances	Size			
3/80	2.27G	1.781	2.274	1.769	36	640: 100%		600/600	[02:35<00:00, 3.85it/s]
	Class	Images	Instances	Box(P	R	mAP50	mAP50-95): 100%		31/31 [00:07<00:00, 4.14it/s]
	all	961	1557	0.468	0.412	0.385	0.171		
Epoch	GPU_mem	box_loss	cls_loss	dfl_loss	Instances	Size			
4/80	2.29G	1.858	2.299	1.852	17	640: 100%		600/600	[02:40<00:00, 3.75it/s]
	Class	Images	Instances	Box(P	R	mAP50	mAP50-95): 100%		31/31 [00:06<00:00, 4.49it/s]
	all	961	1557	0.505	0.48	0.464	0.203		

Figure 49 : Results when training model.

Optimization: The training process automatically handled learning rate scheduling, loss function minimization (bounding box loss, objectness loss, classification loss), and checkpoint saving.

Throughout training, metrics such as mAP@0.5:0.95, Precision, Recall, and Loss curves were monitored to assess convergence and model quality.

1.3 Exporting to ONNX for Embedded Deployment

To deploy the trained model on a low-power embedded device such as the Raspberry Pi 5, the final YOLOv8 model was exported to ONNX format (Open Neural Network Exchange), which enables cross-platform, hardware-agnostic inference.

```
[ ] model.export(format='onnx')
Ultralytics 8.3.156 Python-3.11.13 torch-2.6.0+cu124 CPU (Intel Xeon 2.00GHz)
PyTorch: starting from '/content/yolo_train/yolov8n_custom2/weights/best.pt' with input shape (1, 3, 640, 640) BCHW and output shape(s) (1, 7, 8400) (6.0 MB)
ONNX: starting export with onnx 1.17.0 opset 19...
ONNX: slimming with onnxslim 0.1.57...
ONNX: export success 1.8s, saved as '/content/yolo_train/yolov8n_custom2/weights/best.onnx' (11.7 MB)
Export complete (2.1s)
Results saved to /content/yolo_train/yolov8n_custom2/weights
Predict:      yolo predict task=detect model=/content/yolo_train/yolov8n_custom2/weights/best.onnx imgsz=640
Validate:     yolo val task=detect model=/content/yolo_train/yolov8n_custom2/weights/best.onnx imgsz=640 data=/content/DATN-9/data.yaml
Visualize:    https://netron.app
'/content/yolo_train/yolov8n_custom2/weights/best.onnx'
```

Figure 50 : Results when exporting to ONNX

The exported ONNX model is optimized for inference using ONNX Runtime, which supports CPU execution with high performance and low memory overhead – making it ideal for edge deployment.

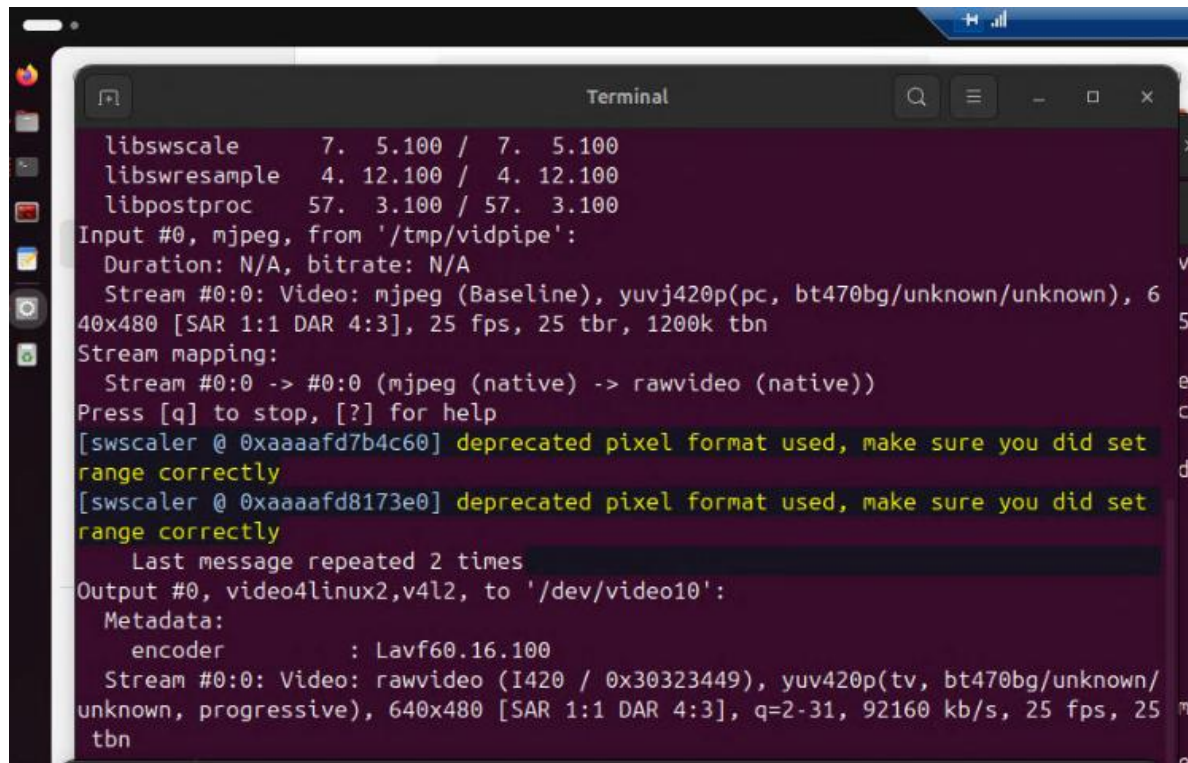
2. Hardware Integration and Video Streaming Configuration

2.1 OV5647 CSI Camera Setup on Raspberry Pi 5

The Raspberry Pi 5 was used as the edge inference device. A CSI camera module (OV5647) was connected via the MIPI interface and accessed using the libcamera stack – the modern alternative to legacy raspistill.

To begin capturing video, the following terminal was used:

```
./apps/rpicam-vid -t 0 --width 640 --height 480 --framerate 30 --codec mjpeg --  
nopreview -o /tmp/vidpipe
```

A terminal window titled "Terminal" with a dark background and light text. It shows the output of an ffmpeg command. The output includes library versions (libswscale, libswresample, libpostproc), input details (Input #0, mjpeg, from '/tmp/vidpipe'), stream information (Stream #0:0: Video: mjpeg (Baseline), yuvj420p(pc, bt470bg/unknown/unknown), 640x480 [SAR 1:1 DAR 4:3], 25 fps, 25 tbr, 1200k tbn), stream mapping (Stream #0:0 -> #0:0 (mjpeg (native) -> rawvideo (native))), and output details (Output #0, video4linux2,v4l2, to '/dev/video10'). It also shows metadata for the encoder (Lavf60.16.100) and the output stream (Stream #0:0: Video: rawvideo (I420 / 0x30323449), yuv420p(tv, bt470bg/unknown/unknown, progressive), 640x480 [SAR 1:1 DAR 4:3], q=2-31, 92160 kb/s, 25 fps, 25 tbn). There are two deprecation warnings about pixel format used, each repeated 2 times.

```
libswscale      7.  5.100 /  7.  5.100
libswresample   4. 12.100 /  4. 12.100
libpostproc    57.  3.100 / 57.  3.100
Input #0, mjpeg, from '/tmp/vidpipe':
  Duration: N/A, bitrate: N/A
    Stream #0:0: Video: mjpeg (Baseline), yuvj420p(pc, bt470bg/unknown/unknown), 640x480 [SAR 1:1 DAR 4:3], 25 fps, 25 tbr, 1200k tbn
  Stream mapping:
    Stream #0:0 -> #0:0 (mjpeg (native) -> rawvideo (native))
Press [q] to stop, [?] for help
[swscale @ 0xaaaaafd7b4c60] deprecated pixel format used, make sure you did set range correctly
[swscale @ 0xaaaaafd8173e0] deprecated pixel format used, make sure you did set range correctly
Last message repeated 2 times
Output #0, video4linux2,v4l2, to '/dev/video10':
  Metadata:
    encoder      : Lavf60.16.100
    Stream #0:0: Video: rawvideo (I420 / 0x30323449), yuv420p(tv, bt470bg/unknown/unknown, progressive), 640x480 [SAR 1:1 DAR 4:3], q=2-31, 92160 kb/s, 25 fps, 25 tbn
```

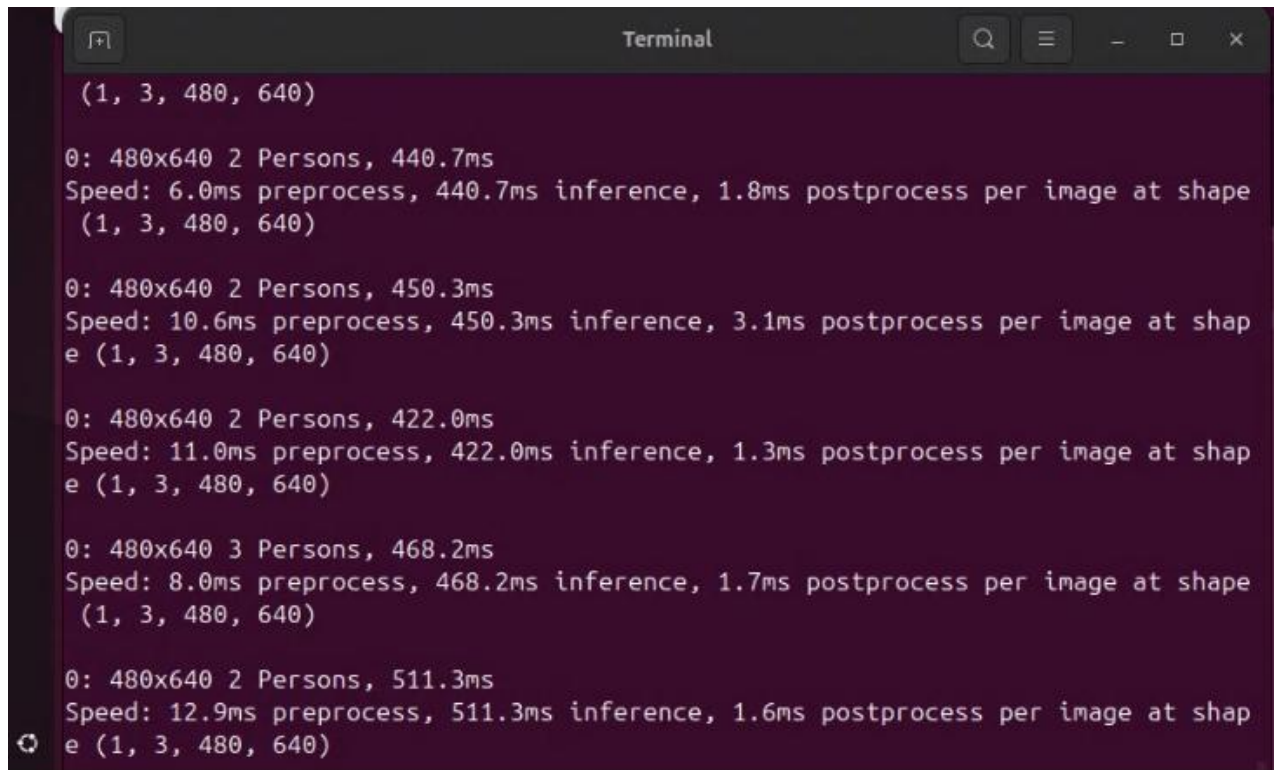
Figure 51 : Results when streaming MJPEG video frames

This streams MJPEG video frames to a named pipe (/tmp/vidpipe), which acts as an intermediary for video redirection.

2.2 Streaming to a Virtual Webcam (/dev/video10)

Since OpenCV requires access to a standard /dev/videoX device, FFmpeg was used to redirect the MJPEG stream into a virtual webcam via v4l2loopback.

ffmpeg -f mjpeg -i /tmp/vidpipe -f v4l2 -vcodec rawvideo -pix_fmt yuv420p /dev/video10

A terminal window with a dark background and light text. The title bar says "Terminal". The output shows several lines of inference results for a video stream. Each line starts with a frame number (0), followed by the resolution (480x640), the number of persons detected, and the total inference time in milliseconds. Below each total time, a breakdown of speeds for preprocess, inference, and postprocess steps is provided for the same frame. The shape of the input is consistently (1, 3, 480, 640).

```
(1, 3, 480, 640)

0: 480x640 2 Persons, 440.7ms
Speed: 6.0ms preprocess, 440.7ms inference, 1.8ms postprocess per image at shape
(1, 3, 480, 640)

0: 480x640 2 Persons, 450.3ms
Speed: 10.6ms preprocess, 450.3ms inference, 3.1ms postprocess per image at shap
e (1, 3, 480, 640)

0: 480x640 2 Persons, 422.0ms
Speed: 11.0ms preprocess, 422.0ms inference, 1.3ms postprocess per image at shap
e (1, 3, 480, 640)

0: 480x640 3 Persons, 468.2ms
Speed: 8.0ms preprocess, 468.2ms inference, 1.7ms postprocess per image at shape
(1, 3, 480, 640)

0: 480x640 2 Persons, 511.3ms
Speed: 12.9ms preprocess, 511.3ms inference, 1.6ms postprocess per image at shap
e (1, 3, 480, 640)
```

Figure 52 : Results when streaming to a Virtual Webcam

With this setup, the inference script can access the live video using OpenCV API:

```
cv2.VideoCapture("/dev/video10")
```

This method ensures minimal latency and full compatibility with Python-based image processing pipelines.

3. ONNX Inference and Real-Time GUI Deployment

3.1 Running the ONNX Model with ONNXRuntime

After capturing frames from the virtual webcam, each frame is preprocessed and passed into the ONNX model using ONNXRuntime:

```
import onnxruntime as ort
session=ort.InferenceSession("best.onnx",providers=["CPUExecutionProvider"])

input_name = session.get_inputs()[0].name
```

Preprocessing steps:

- Resize the image to 640x480
- Normalize pixel values to range [0, 1]
- Convert the image to a tensor shape (1, 3, 640, 640) in BGR format

Once the inference is complete, post-processing is performed to extract bounding boxes, class labels, and confidence scores. This information is overlaid onto the original frame for real-time visualization.

3.2 GUI Development with Tkinter

To display the detection results interactively, a lightweight Tkinter GUI was developed. It is optimized for low-resource devices, providing a smooth user experience without compromising performance.

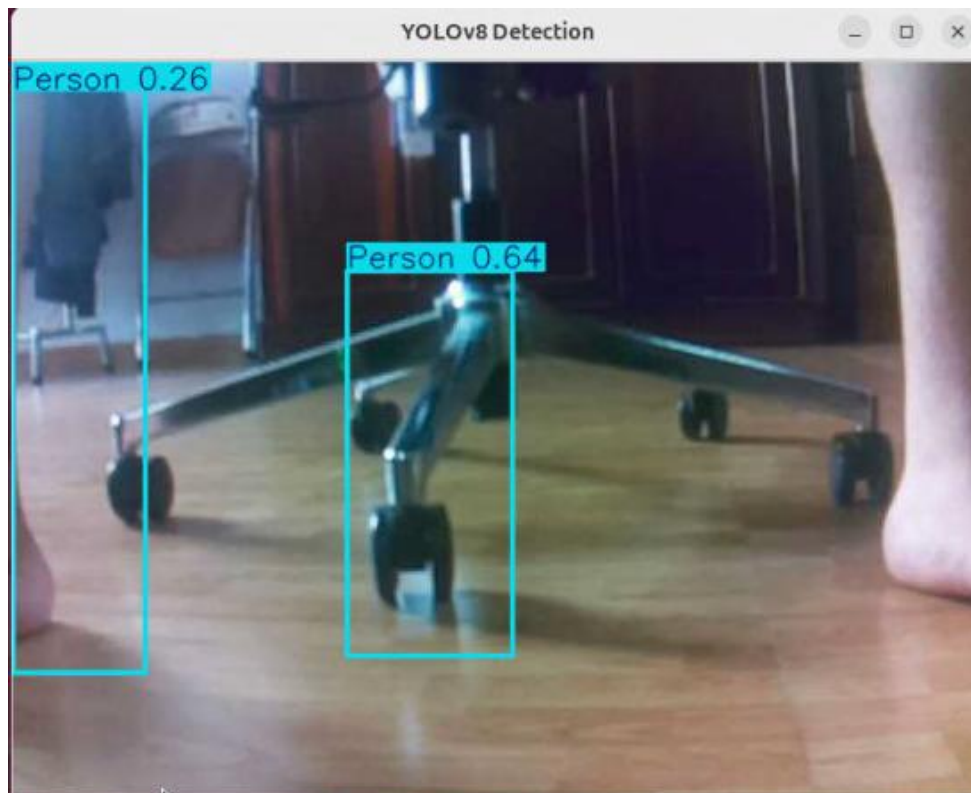


Figure 53 : Results displayed with Tkinter gui

- Live video feed with bounding boxes and class names drawn in real-time

- FPS display in the terminal for performance monitoring
- Responsive design with `window.after(1, update_frame)` loop ensuring low-latency updates
- Easily extendable for alert mechanisms (e.g., sending warning when smoke or fire is detected)

IV. Results

1. Training and Evaluation Metrics

To assess the performance of the YOLOv8n model trained for visual monitoring of three critical object classes: **Fire**, **Person**, and **Smoking**, a comprehensive set of evaluation metrics was used. Key performance metrics include **Precision**, **Recall**, **mean Average Precision (mAP)**, and **F1-score**, evaluated across multiple IoU thresholds.

1.1. Confidence Curve Analysis

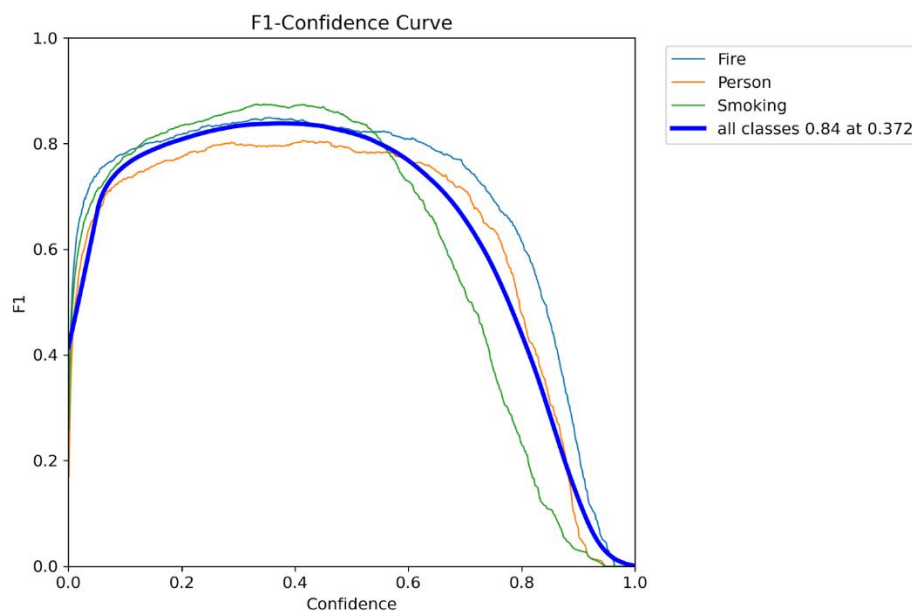


Figure 54 : F1 Score vs Confidence Threshold Curve.

The F1-Confidence curve reflects the balance between precision and recall over different confidence thresholds. As depicted in the graph:

- The model achieved its optimal F1-score of 0.84 at a confidence threshold of 0.372.
- All three classes show stable F1-scores above 0.8 within the 0.3 to 0.6 confidence range.
- Notably, the "Fire" class exhibited the most stable F1 curve, indicating robustness in its detection, even under variable lighting conditions.

This threshold was selected as the operating point for deployment to balance sensitivity and specificity in real-time inference scenarios.

1.2 Class-wise Precision, Recall, and mAP

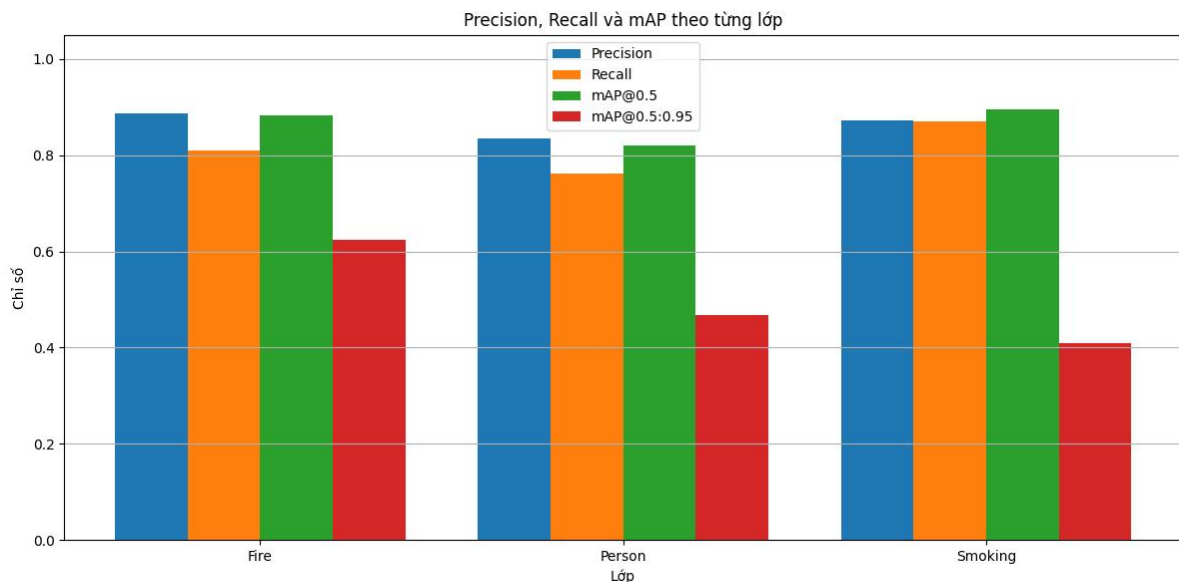


Figure 55 : Precision, Recall, and mAP per Class.

The bar chart illustrates individual class performance metrics:

Class	Precision	Recall	mAP@0.5	mAP@0.5:0.95
Fire	0.89	0.81	0.89	0.63
Person	0.84	0.76	0.82	0.47
Smoking	0.87	0.87	0.90	0.41

Table 10: Model parameters after training

- The **"Fire"** class exhibited excellent detection performance, particularly in controlled indoor environments, with high precision and mAP.

- **"Person"** class achieved reliable detection but slightly lower recall, possibly due to occlusion and varied poses.
- The **"Smoking"** class had the highest mAP@0.5, but lower mAP@0.5:0.95, indicating difficulty in precise localization. This is expected due to the small object size and visual similarity with background or hands.

1.3 Confusion Matrix Interpretation

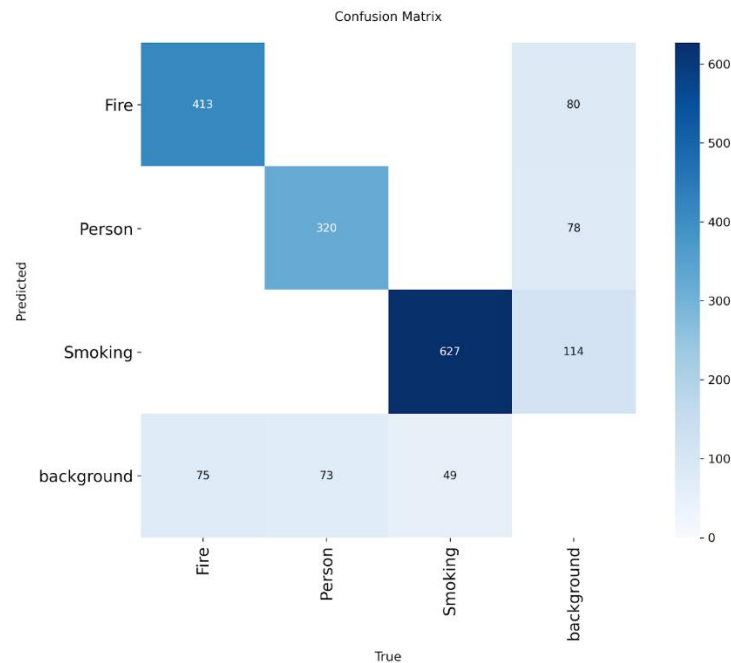


Figure 56 : Confusion Matrix of Detection Results Across Classes.

The confusion matrix offers a deeper understanding of misclassification patterns:

- The **Smoking** class is most prone to confusion, particularly with **Fire** and **background**, contributing to 114 misclassified instances.
- **Person** is misidentified as background in 78 cases, reflecting limitations in low-resolution or occluded scenarios.
- **Fire** is the most confidently predicted class, with 413 correct detections and 80 misclassified as background.

These results emphasize the need for data augmentation or additional training samples for borderline or small-object classes such as Smoking, which often share similar visual features with hands or lighters.

2. Qualitative Results on Real-time Input

Live tests were conducted using a Raspberry Pi 5 connected with the OV5647 camera module. Images captured in real time were passed through the YOLOv8n model deployed via Ultralytics library. Below are representative outputs:

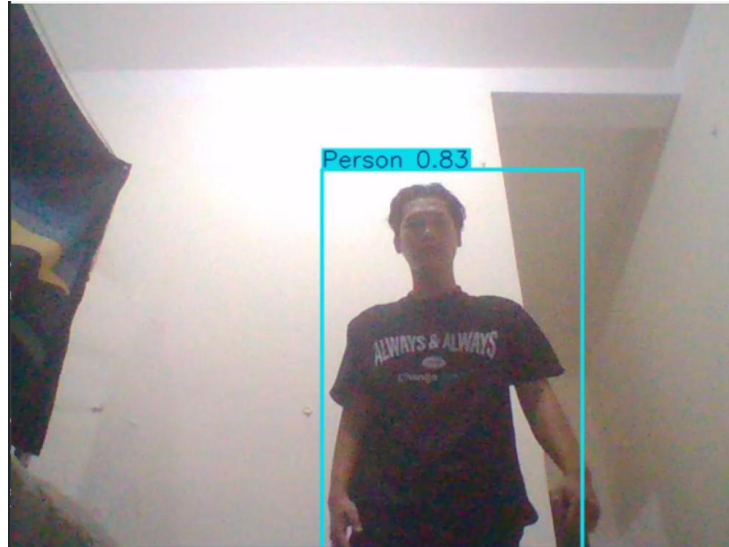


Figure 57 : YOLOv8n Detection of 'Person' Class.

The model successfully detected a human subject at the center of the frame with **confidence 0.83**. Despite moderate background clutter and a relatively small resolution (640×480), the bounding box accurately encompassed the subject, demonstrating YOLOv8n's efficiency at detecting medium-scale objects.



Figure 58 : YOLOv8n Detection of 'Fire' Class.

The system correctly identified a visible flame with **confidence 0.87**. Notably, the fire occupied a limited portion of the image but was correctly localized without false positives. This proves the model's ability to capture fine, high-frequency features (e.g., fire edges, color gradients) even under low contrast or dim indoor lighting.

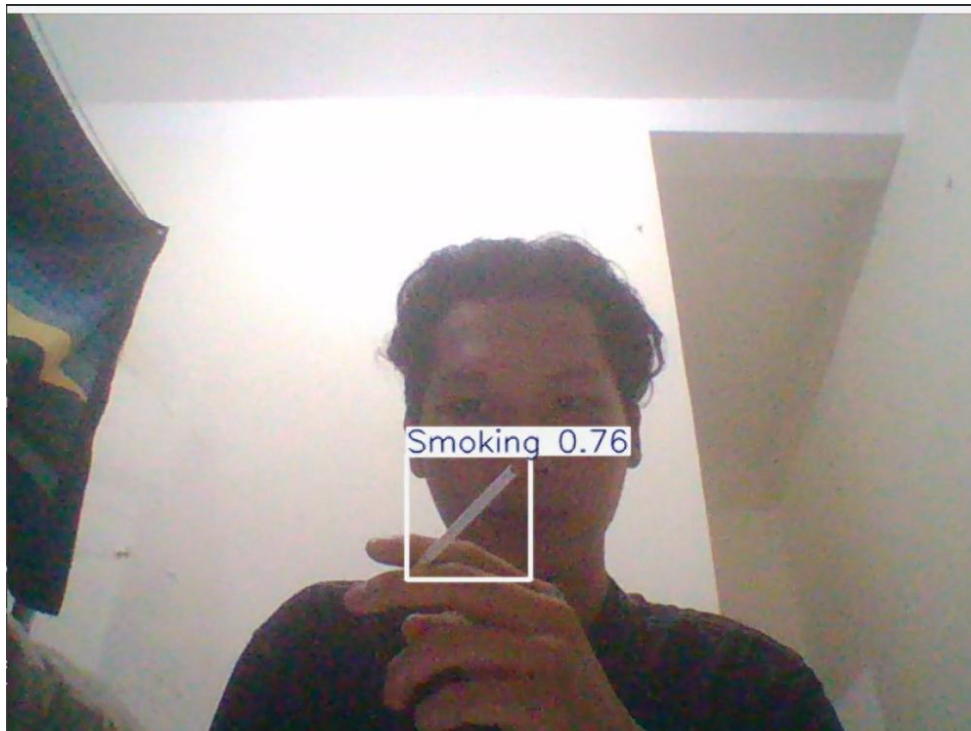


Figure 59 : YOLOv8n Detection of 'Smoking' Class.

A challenging scenario was presented where a cigarette was partially occluded by the subject's hand. The model succeeded in recognizing the **Smoking** class with a **confidence score of 0.76**, though with a slightly offset bounding box. This result reflects the model's high sensitivity to small object patterns, despite being trained with limited pixel resolution and data diversity.

- Scalability and System Integration
- The system is designed to be scalable for additional features such as **object counting, human behavior analysis, or cloud-based monitoring**.
- Detected results can be synchronized with a thermal imaging module (e.g., AMG8833) or integrated into a **navigation subsystem** for automated thermal inspection.
- Alerts can be sent to a remote server or mobile application for real-time response.

The model's response to more complex scenes also provides valuable insight:

- **Overlapping Classes:** When both Person and Smoking co-occur in the frame, the model occasionally predicted Smoking within the person's bounding box. This behavior, while not entirely erroneous, could be improved through fine-tuned bounding box regression or contextual multi-object modeling.
- **False Negatives in Low Light:** In some dimly lit environments, Smoking instances were missed or detected with low confidence. This highlights a potential weakness in the dataset's illumination variance and the model's sensitivity to grayscale gradients, especially for subtle objects like smoke trails.
- **Temporal Inconsistencies:** In live stream operation, when using per-frame detection without temporal filtering, the model sometimes fluctuated between detections due to slight movement or blur.

Inference speed: The system maintains an average processing speed of 15–18 FPS, which translates to approximately 55–65 milliseconds per frame — effectively achieving real-time responsiveness for most visual surveillance applications.

Thermal stability and uptime: Throughout extended continuous operation, the system remains thermally stable with no signs of overheating or memory errors. CPU

temperatures stay within safe operating thresholds without requiring additional cooling systems.

While the frame rate may not match that of GPU-powered inference systems (≥ 30 FPS), achieving 15–18 FPS on an edge device like the Raspberry Pi 5 is highly acceptable — especially for critical use cases such as fire detection, smoke monitoring, or smoking behavior recognition. In such scenarios, the system's response time is sufficiently fast to trigger real-time alerts or activate safety mechanisms, making it suitable for deployment in industrial zones, smart buildings, and public safety infrastructures.

CHAPTER 5: THERMAL IMAGING MONITORING USING AMG8833 SENSOR

I. Ideas and Approaches

The system is designed to visualize thermal data acquired from the AMG8833 infrared sensor through a real-time graphical user interface. Displaying temperature information in the form of thermal images not only enhances observation capabilities but also enables several practical applications, such as hotspot detection and integration into autonomous robotic systems for thermal area monitoring and tracking in real time.

To realize the above concept, the system is developed through the following steps:

- **Collect thermal data** from the AMG8833 sensor in the form of an 8×8 pixel matrix.
- **Apply interpolation techniques** to increase the image resolution, thereby improving visualization and analysis capabilities.
- **Render thermal images** using intuitive colormaps to clearly distinguish different temperature zones.
- **Integrate a temperature warning mechanism**, which alerts the user when any pixel exceeds a pre-configured threshold.
- **Design a user-friendly interface** that allows flexible adjustment of various parameters such as alert thresholds, colormap styles, and data update frequency.

II. Theoretical Overview

1. AMG8833 Sensor

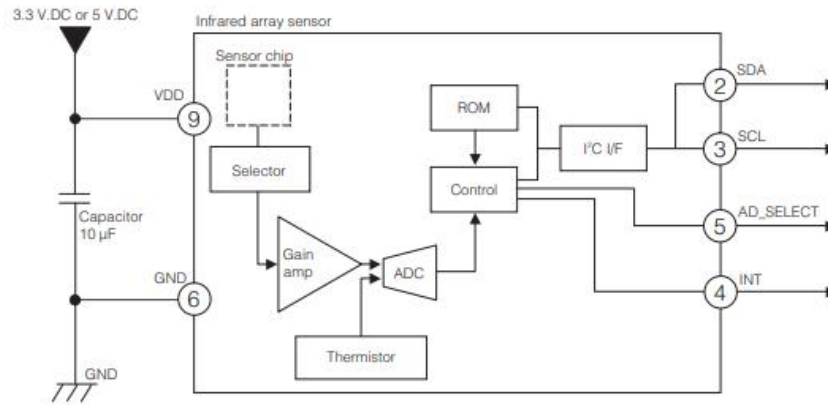


Figure 60 : Circuit diagram of AMG8833

The internal architecture of the AMG8833 thermal infrared sensor is composed of several key functional blocks that enable real-time temperature detection and digital communication with external systems.[6]

The sensor operates at a supply voltage of 3.3VDC or 5VDC, with a 10 µF decoupling capacitor placed between the VDD and GND to stabilize power and suppress noise.

At the heart of the device is an 8×8 infrared array sensor chip that detects thermal radiation from the environment. The analog signals from the pixel array are first processed by a selector circuit, which scans through each pixel. These signals are then fed into a gain amplifier, which enhances the weak IR signals, and subsequently passed to an Analog-to-Digital Converter (ADC) for digital conversion.

To compensate for environmental temperature drift, a thermistor is included to provide reference temperature data for internal calibration.

The digital output is then handled by a control logic unit, which manages data flow and synchronization. A built-in ROM stores the device firmware and calibration data. The processed thermal data is made accessible via an I²C communication interface, allowing integration with microcontrollers or embedded systems through the SDA and SCL lines.

The AD_SELECT pin allows users to configure the I²C address, enabling multiple

sensors to coexist on the same I²C bus.

Furthermore, the sensor features an interrupt pin , which remains at the VDD level during normal operation and transitions to GND (0V) to signal an interrupt event, such as rapid temperature changes or movement detection, allowing efficient event-driven data acquisition.

Operating Principle

The AMG8833 sensor operates based on the principle of **infrared thermal radiation** emitted by objects. Any object with a temperature above absolute zero (0 Kelvin) emits infrared radiation. The intensity of this radiation increases with surface temperature.

The AMG8833 contains an 8×8 microbolometer array, where each pixel captures the infrared radiation from a specific position in the field of view. The analog signals from the pixel array are:

- Sequentially selected by a multiplexer circuit (Selector),
- Amplified using a gain amplifier,
- Digitized by an Analog-to-Digital Converter (ADC),
- Calibrated using data from the built-in thermistor,
- Processed and controlled by the internal logic (Control),
- Output via an I²C communication interface.

The system can generate alerts when it detects a temperature exceeding a pre-defined threshold—useful for human detection or hotspot monitoring in equipment.

Interfacing AMG8833 with Raspberry Pi

The AMG8833 communicates with the Raspberry Pi using the **I²C protocol**, a widely used serial communication standard in embedded systems.

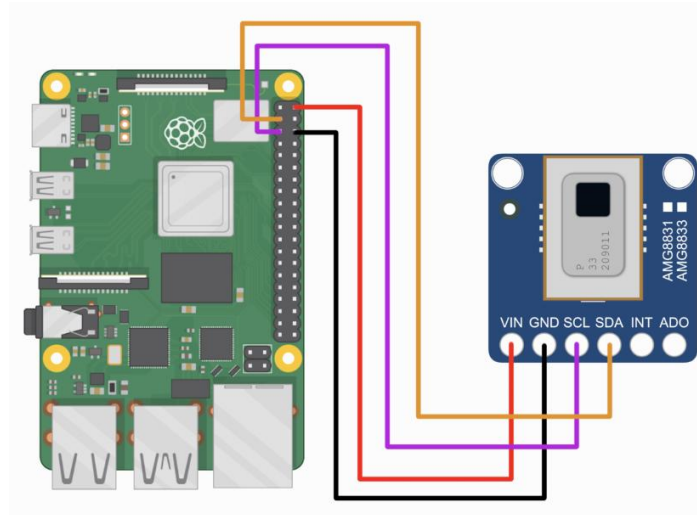


Figure 61 : AMG8833 and Raspberry Pi Connection

AMG8833 Pin	Function	Raspberry Pi Connection
VDD	Power Supply (3.3V/5V)	Pin 1 (3.3V) or Pin 2 (5V)
GND	Ground	Pin 6 (GND)
SDA (Pin 2)	I2C Data	Pin 3 (GPIO2 - SDA)
SCL (Pin 3)	I2C Clock	Pin 5 (GPIO3 - SCL)
AD_SELECT	I2C Address Select	GND or VDD depending on usage
INT (Pin 4)	Interrupt Output	Connect to any GPIO if used

Table 11: Connection between AMG8833 and Raspbberry.

2. Image interpolation - resize and distort example[7]

Image interpolation is the process of estimating unknown pixel values between known pixels. It is widely used in image processing, especially for scaling up low-resolution images. In this project, interpolation is used to enhance thermal images obtained from the AMG8833 sensor, which provides only an 8×8 thermal matrix.

Given an original image matrix $Z \in \mathbb{R}^{(m \times n)}$ the goal is to create a new matrix $Z' \in \mathbb{R}^{(M \times N)}$ where $M > m$ and $N > n$. Each interpolated pixel value z'_{ij} is estimated from the surrounding known values in Z .

Even if the same image resize or remap is performed, the results can vary significantly depending on the interpolation algorithm. It is only an approximation, therefore an image will always lose some quality each time interpolation is performed. This tutorial aims to provide a better understanding of how the results may vary — helping you to minimize any interpolation-induced losses in image quality.

IMAGE RESIZE EXAMPLE

Image interpolation works in two directions, and tries to achieve a best approximation of a pixel's color and intensity based on the values at surrounding pixels. The following example illustrates how resizing / enlargement works:

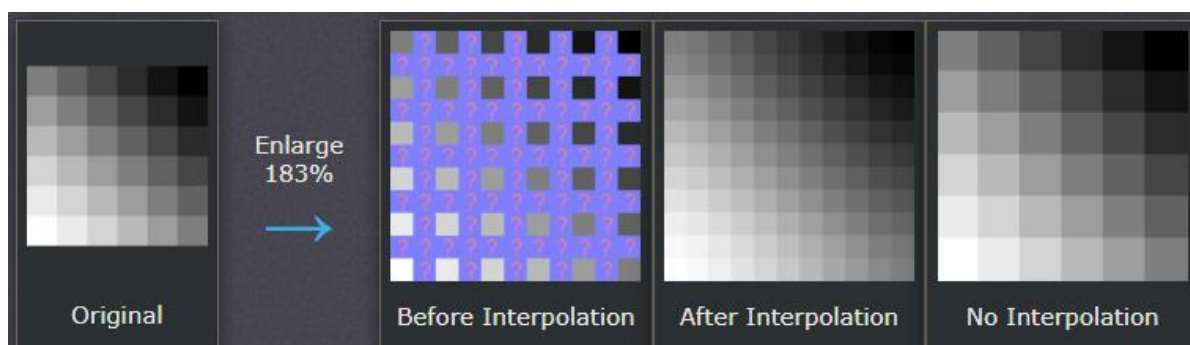


Figure 62 : Interpolation example.

Unlike air temperature fluctuations and the ideal gradient above, pixel values can change far more abruptly from one location to the next. As with the temperature example, the more you know about the surrounding pixels, the better the interpolation will become. Therefore results quickly deteriorate the more you stretch an image, and interpolation can never add detail to your image which is not already present.

Types of interpolation algorithms

- Nearest Neighbor Interpolation

Nearest neighbor is the most basic and requires the least processing time of all the interpolation algorithms because it only considers one pixel — the closest one to the interpolated point. This has the effect of simply making each pixel bigger.

- Bilinear Interpolation

Bilinear interpolation considers the closest 2x2 neighborhood of known pixel values surrounding the unknown pixel. It then takes a weighted average of these 4 pixels to arrive at its final interpolated value. This results in much smoother looking images than the nearest neighbor.

The diagram to the left is for a case when all known pixel distances are equal, so the interpolated value is simply their sum divided by four.

- Bicubic Interpolation

Bicubic interpolation estimates a value $Z(x,y)$ using 16 nearby pixels (4x4 neighborhood):

$$z(x, y) = \sum_{i=-1}^2 \sum_{j=-1}^2 w_{ij}(x, y) \cdot z_{i,j}$$

$W_{ij}(x,y)$ a weight function depending on the distance to the point (x,y) ,

Z_{ij} are known pixel values around the interpolation point.

The weights are calculated using cubic polynomials such as spline or Hermite functions. This method ensures smooth gradients and continuous transitions between pixel values.

Bicubic goes one step beyond bilinear by considering the closest 4x4 neighborhood of known pixels — for a total of 16 pixels. Since these are at various distances from the unknown pixel, closer pixels are given a higher weighting in the calculation. Bicubic produces noticeably sharper images than the previous two methods, and is perhaps the ideal combination of processing time and output quality. For this reason it is a standard in many image editing programs, printer drivers and in-camera interpolation.

III. Program Design for AMG8833 Thermal Image Processing

1. Software Environment Configuration

- OS: unbutu 24.04
- Python: Version 3.12.3
- AMG8833 Library: open-source driver

Open TERMINAL  and enter the following commands:

Update system: **sudo apt update && sudo apt upgrade -y**

Enable I2C Interface: **sudo raspi-config # → Interface Options → I2C → Enable**

Install Required Libraries: **sudo apt install python3-pip python3-tk i2c-tools -y**

pip3 install numpy matplotlib scipy

Check I2C detection: **i2cdetect -y 1**

Clone or install AMG8833 driver: **git clone https://github.com/your-source/amg8833_lib.git**

2. Program implementation

The infrared thermal imaging system utilizing the AMG8833 sensor is designed to monitor temperature distribution in real time. The system leverages the capabilities of the Raspberry Pi 5 and the AMG8833 IR thermal camera module to acquire, process, interpolate, and visualize thermal data. The final output is rendered via a custom graphical user interface (GUI) developed in Python using the Tkinter and Matplotlib libraries.

The system provides a lightweight, standalone, and interactive thermal monitoring solution suitable for embedded applications such as robotics, surveillance, and early fire detection.

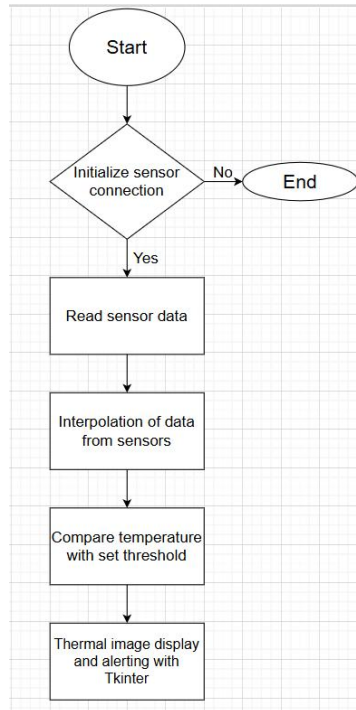


Figure 63 : Program Flowchart AMG8833.

Main Functionalities:

- Establishes communication with the AMG8833 sensor via the I2C interface
- Continuously reads 8×8 thermal matrix data (64 temperature points)
- Applies interpolation techniques (bilinear, bicubic) to improve thermal resolution
- Visualizes thermal data using customizable colormaps
- Enables real-time threshold-based temperature alerts
- Offers user interaction through GUI for adjusting temperature range, colormap, rotation, and capturing snapshots

AMG8833 Sensor Initialization and Communication

The AMG8833 thermal sensor communicates with the Raspberry Pi via the I2C protocol. It supports two common I2C addresses: 0x68 and 0x69, depending on the hardware configuration.

Upon program startup, the following initialization steps occur:

- Importing Dependencies: Essential Python libraries are loaded, including:
 - ❖ numpy for array operations
 - ❖ matplotlib and scipy.interpolate for visualization and interpolation

- ❖ tkinter for GUI elements
- Appending Custom Driver Path: A custom AMG8833 driver is loaded dynamically into the Python environment.
- I2C Communication Attempt: The system attempts to connect to the sensor at address 0x69. If unsuccessful, it retries with address 0x68. If both attempts fail, the system halts and informs the user.

Thermal Data Acquisition and Interpolation

The AMG8833 sensor produces an 8×8 matrix of temperature values, each representing the temperature in °C at a given pixel. Although compact and efficient, this resolution is insufficient for detailed thermal imaging. To address this, the system applies 2D interpolation to increase image clarity.

Mode	Output Resolution	Interpolation Method
raw8x8	8×8	No interpolation
interp32x32	32×32	Bilinear interpolation
2		
interp64x64	64×64	Bicubic interpolation
4		

Table 12 : Supported Interpolation Modes

Interpolation is implemented using SciPy's `interpolate.interp2d()` function. For example:

```
f = interpolate.interp2d(xx, yy, z, kind='cubic')
interpolated = f(grid_x, grid_y)
```

This generates a smooth, high-resolution thermal grid based on the original 8x8 data, enhancing hotspot visibility and image quality.

GUI Design and Implementation

The system GUI is implemented using Python's Tkinter module and consists of two primary sections:

Left Panel – Control Interface:

- Temperature Threshold Input: User-defined alert limit (e.g., 40°C)
- Min/Max Temperature Scale: Adjusts the display range to highlight thermal variations
- Colormap Selection: Dynamically changes color rendering (e.g., hot, jet, inferno, rainbow)
- Image Controls:
 - Rotate Button: Rotates the thermal image 90° clockwise
 - Freeze Mode: Locks the current image frame for analysis
 - Save Button: Saves the current thermal image as a .png snapshot
- Live Status Display:
 - Current maximum temperature
 - Alert banner (e.g., "⚠ TOO HIGH") in colored font

Right Panel – Image Display:

- Matplotlib Imshow: Plots interpolated thermal matrix in real time
- Colorbar: Indicates color-temperature scale
- No Axis Display: Axes are removed for a cleaner visualization

This modular structure separates user interaction from visualization logic, enhancing maintainability.

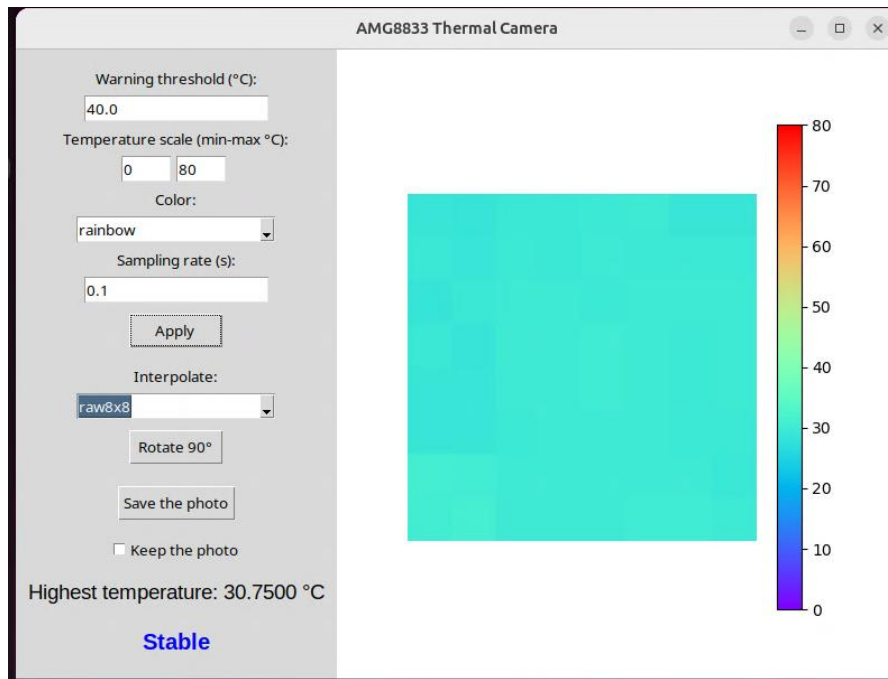


Figure 64 : GUI snapshot during real-time operation

Real-Time Processing Loop

The core of the system's real-time behavior is governed by the `update_plot()` method, which is recursively scheduled using Tkinter's `after()` function with a refresh interval (e.g., 100 ms).

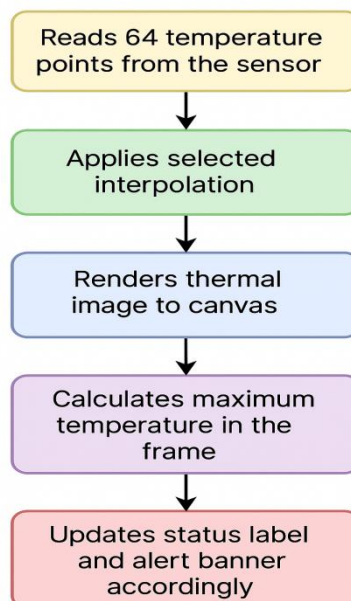


Figure 65: Real-Time Workflow

```
if max_temp >= self.alert_threshold:
```

```
    self.alert_label.config(text="TOO HIGH!", foreground="red")
```

This loop enables responsive user feedback and dynamic visualization without freezing the UI.



Figure 66 : GUI with visual overheat warning

IV. Result

After completing the hardware–software integration, the thermal monitoring system proved to function reliably and met the design objectives: non-contact temperature detection, real-time thermal visualization, and overheat alerting via an intuitive user interface. Below are two representative test cases:

Case 1: Stable Ambient Temperature (No Heat Source)

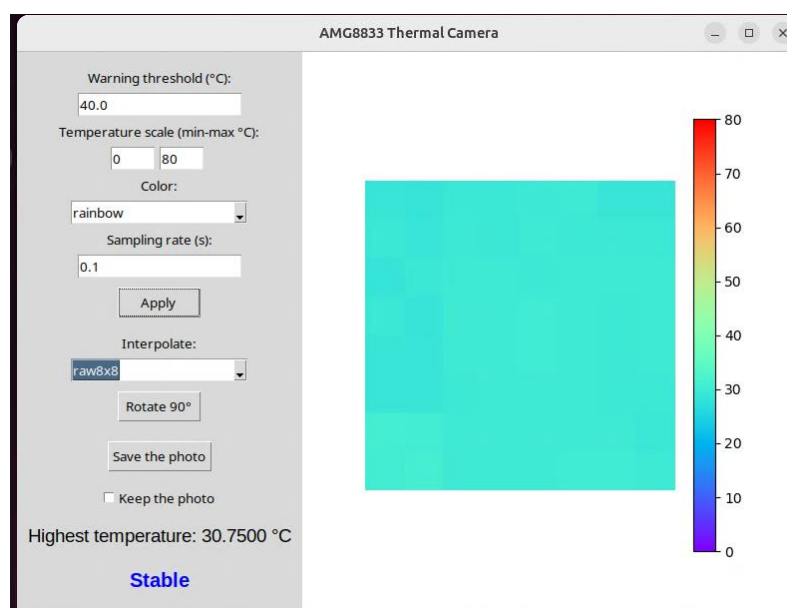


Figure 67 : Measure non-overheated objects with 8x8 native resolution.

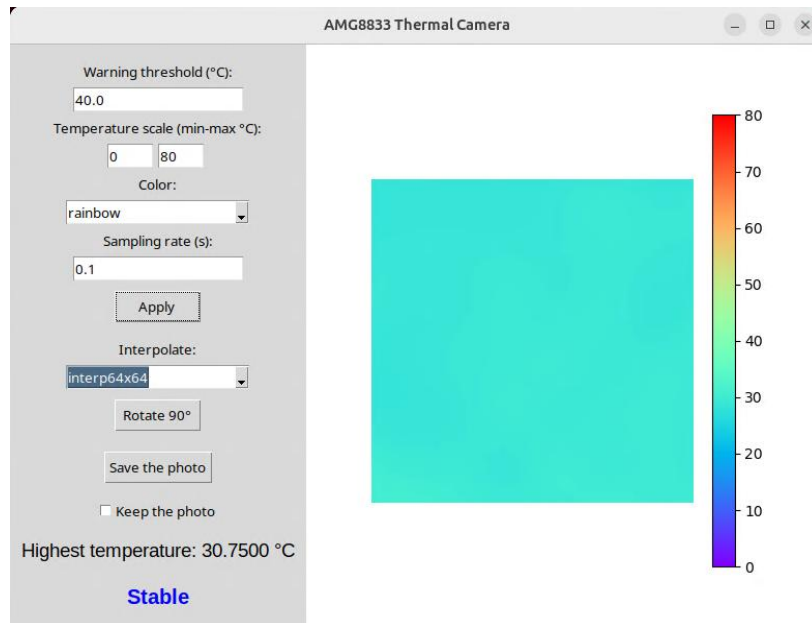


Figure 68 : Measure non-overheated objects with 64x64 interpolation.

The system operated under normal room conditions without any significant heat source in the sensor's field of view. The thermal image displayed mostly blue to cyan hues, corresponding to low temperatures. The highest recorded value was 30.75 °C and 31.00°C, which is significantly below the user-defined alert threshold of 40.0 °C.

- The interface immediately displayed the status: "Stable" in blue text.
- The heatmap was smooth and noise-free, rendered using real-time interpolation.
- Users were able to adjust the colormap (e.g., *rainbow*, *jet*), interpolation method (raw8x8, interp32x32), and temperature scale in real time.
- The frame update rate was set to 0.1 s per frame, allowing fast, responsive visualization.

This scenario represents the system's idle state, ideal for continuous thermal surveillance in environments where early fire detection or equipment monitoring is required.

Case 2: Overheat Detection (Above Threshold)

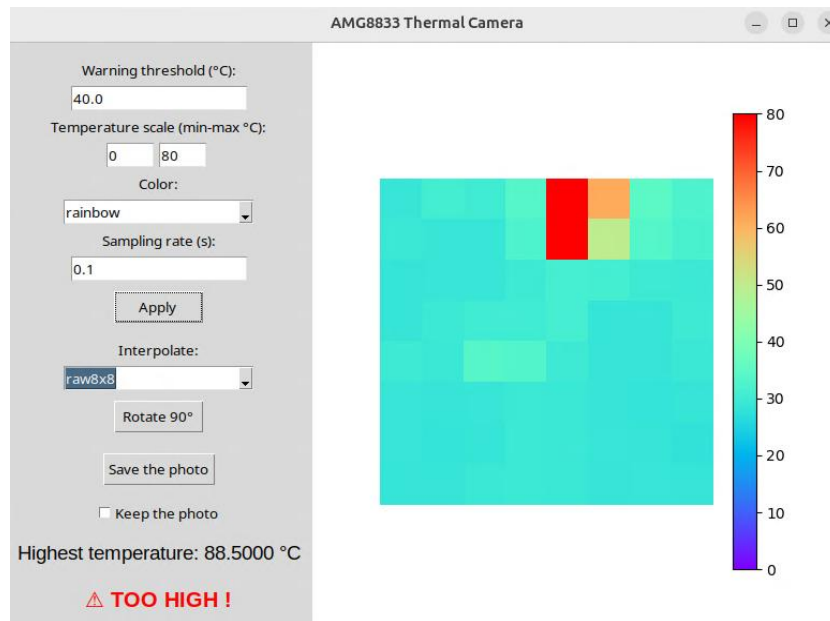


Figure 69 : Measure overheated objects with 8x8 native resolution.

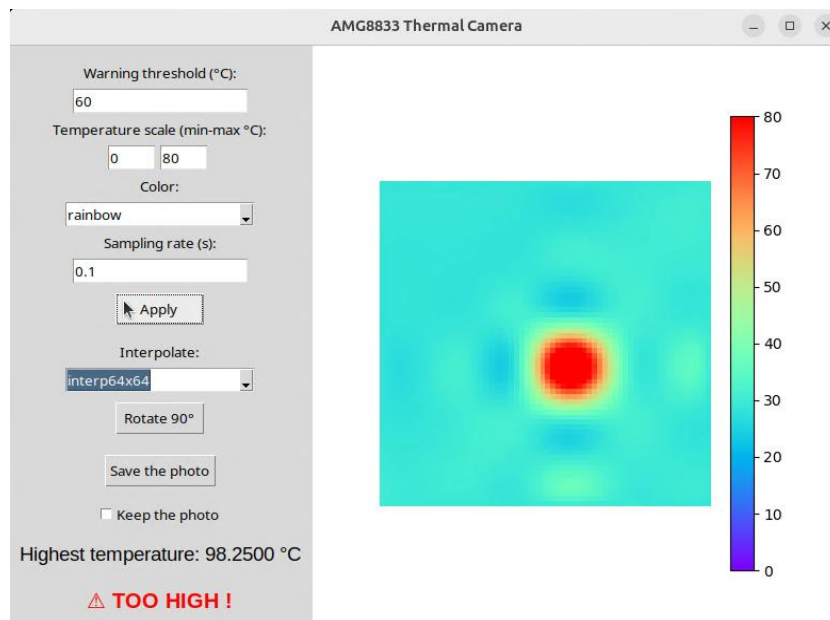


Figure 70 : Measuring Overheated Objects with 64x64 Interpolation.

Demonstrates a scenario where a localized hot spot was introduced. The AMG8833 sensor successfully detected a thermal anomaly, with a maximum temperature of 0 °C and 80°C—clearly exceeding the defined alert threshold of 40.0 °C.

- The hotspot was visually highlighted in red-orange on the heatmap.
- The system status changed to "TOO HIGH!" in bold red, accompanied by a warning icon.

- The updated maximum temperature value was displayed below the image in real time.

This illustrates the system's core function: to detect abnormal temperature increases early and provide clear, unambiguous alerts for operators or automation systems to act upon.

General Evaluation of the Interpolation Method in Thermal Image Processing with the AMG8833 Sensor

The interpolation method serves as an essential technique in enhancing thermal image quality when using the AMG8833 infrared sensor, which inherently provides low-resolution output (8×8 pixels). Due to this limitation, raw thermal frames often appear blocky and lack sufficient detail for effective interpretation. Interpolation addresses this challenge by estimating additional data points between the original pixels to produce higher-resolution visual outputs, such as 32×32 or 64×64 pixel grids. This results in smoother, more continuous temperature gradients, improving visual clarity and aiding users in identifying heat patterns, anomalies, or hot spots more effectively.

This technique brings notable advantages, especially in GUI-based applications, where improved visual feedback enhances usability and allows more intuitive monitoring. It enables better representation of gradual thermal changes, making it easier to analyze scenes or surfaces with subtle temperature differences. However, interpolation also introduces certain limitations. Since it is a mathematical estimation rather than actual sensor measurement, it does not increase the precision of thermal readings. In fact, excessive reliance on interpolation may mislead users into perceiving artificial detail or obscure critical but small hot zones due to smoothing effects.

In conclusion, interpolation is a valuable visualization tool that significantly improves the readability of thermal images captured by the AMG8833. Nevertheless, it should be applied with an understanding of its role as an enhancement—not as a substitute for higher-resolution hardware or more precise sensing. For accurate thermal diagnostics, interpolation is best used in conjunction with complementary

strategies such as sensor fusion, image stitching, or hardware upgrades

Technical Analysis and Performance Evaluation

Metric	Evaluation
Response Time	Real-time updates every 0.1 seconds depending on processing load and GUI rendering.
Sensor Sensitivity	AMG8833 detects temperature changes as small as $\pm 0.25^{\circ}\text{C}$, suitable for anomaly detection.
GUI Usability	User-friendly, supports color maps, rotation, interpolation, and image saving.
Alerting Mechanism	Clear, visible status text with color and icon cues when threshold exceeded.
System Stability	Continuous operation with no crashes or I2C communication loss over extended sessions.
Extensibility	Modular design enables easy integration with actuators, remote systems, or higher-level AI functions.

Table 7: Technical Analysis and Performance Evaluation

Limitations

Despite its success in basic operation, the system has several technical limitations that must be addressed for deployment in more demanding environments:

- **Low Resolution (8×8):** The AMG8833 only offers 64 temperature data points, limiting spatial detail. While upscaling via interpolation (32×32, 64×64) improves visual smoothness, it does **not** enhance true resolution.
- **Narrow Field of View:** With a horizontal FoV of approximately 60°, the sensor cannot cover wide areas without mechanical assistance (e.g., a pan-tilt module).

- **No Object or Motion Recognition:** Unlike RGB cameras, the AMG8833 cannot detect or track moving targets or identify objects.
- **Basic Alerting:** Alerts are currently only shown on the GUI. There is no automated action such as sound alarms, email notifications, or actuator triggers.

Future Development Directions

To improve performance and extend the application scope, the following directions are proposed:

Improving Thermal Image Resolution

- **Method 1: Motorized Scanning with Pan-Tilt Unit**
Mounting the AMG8833 on a pan-tilt gimbal to capture thermal data from multiple angles and stitch them into a higher-resolution composite image.
- **Method 2: AI-Based Super-Resolution**
Applying deep learning models like SRCNN or EDSR to synthetically increase the resolution of the thermal image for better visualization.

Autonomous Overheat Reaction

- Integration with systems that send automatic alerts via email, messaging (e.g., Telegram bots), or activate external alarms when temperature thresholds are exceeded.
- Logging temperature data over time for time-series analysis, trend prediction, and fault diagnosis.

Smart Object-Aware Monitoring

- Fusion with RGB camera and object detection (e.g., YOLO) to detect persons, machinery, or specific zones. The system can then command the thermal camera to automatically scan those regions.
- Classify thermal anomalies using machine learning to distinguish between harmless heat sources and dangerous events (e.g., humans vs overheating machines vs open flame).

Real-World Deployment

- Install the system on a mobile robotic platform for autonomous thermal patrol in industrial environments.
- Applications include data center monitoring, battery rooms, transformer stations, or EV charging hubs for early fire or overload detection.

REFERENCES

- [1] Shan He, Tao Song, Pengcheng Wang, et al. “An Enhanced Adaptive Monte Carlo Localization for Service Robots in Dynamic and Featureless Environments.” *Journal of Intelligent & Robotic Systems*, vol. 108, article 6, 2023.
- [2] Frank Dellaert, Dieter Fox, Wolfram Burgard, Sebastian Thrun. “Monte Carlo Localization for Mobile Robots.” In: *IEEE International Conference on Robotics and Automation (ICRA)*, 1999.
- [3] Wallace Pereira Neves dos Reis, Guilherme José da Silva, Orides Morandin Junior, Kelen Cristiane Teixeira Vivaldini. “An Extended Analysis on Tuning the Parameters of Adaptive Monte Carlo Localization ROS Package in an Automated Guided Vehicle.” 2021.
- [4] Công nghệ cảm biến CMOS là gì? (<https://thietbisohoa.vn/tin-tuc/cong-nghe-cam-bien-cmos-la-gi>)
- [5] The Art and Science of Interpolation (<https://medium.com/thedeephub/the-art-and-science-of-interpolation-b12b99f2e053>)
- [6] What is YOLOv8? A Complete Guide (<https://blog.roboflow.com/what-is-yolov8/>)
- [7] Adafruit, “Adafruit AMG8833 8×8 Thermal Camera Sensor,” PDF, 2024. <https://cdn-learn.adafruit.com/.../adafruit-amg8833-8x8-thermal-camera-sensor.pdf>
- Localization And Mapping: Towards the Robust-Perception Age.”

