Development of a Real-Time Gesture Recognition System for Human-Robot Interaction

BY

Ong Niam Chi 20ACB05969

A REPORT

SUBMITTED TO

Universiti Tunku Abdul Rahman

in partial fulfillment of the requirements

for the degree of

BACHELOR OF INFORMATION TECHNOLOGY (HONOURS) COMPUTER ENGINEERING

Faculty of Information and Communication Technology

(Kampar Campus)

JUNE 2024

UNIVERSITI TUNKU ABDUL RAHMAN

REPORT STATUS DECLARATION FORM

Title: Development of a Real-Time Gesture Recognition System for Human-Robot Interaction

Academic Session: JUNE 2024

I ONG NIAM CHI
(CAPITAL LETTER)

declare that I allow this Final Year Project Report to be kept in
Universiti Tunku Abdul Rahman Library subject to the regulations as follows:

1. The dissertation is a property of the Library.
2. The Library is allowed to make copies of this dissertation for academic purposes.

Verified by,

(Author's signature)

(Supervisor's signature)

Address:
No.1, Jalan Kurau Indah 1,
Taman Kurau Indah 34350
Kuala Kurau , Perak.

Dr. Teoh Shen Khang
Supervisor's name

Date: 8/9/2024

Date: 13 September 2024

FACULTY OF INFORMATION AND COMMUNICATION TECHNOLOGY

UNIVERSITI TUNKU ABDUL RAHMAN

Date: 8/9/2024

SUBMISSION OF FINAL YEAR PROJECT/DISSERTATION/THESIS

It is hereby certified that ***Ong Niam Chi*** (ID No: ***2005969***) has completed this final year project entitled "*Development of a Real-Time Gesture Recognition System for Human-Robot Interaction*" under the supervision of Dr Teoh Shen Khang (Supervisor) from the Department of Computer and Communication Technology, Faculty of Information and Communication Technology.

I understand that University will upload softcopy of my final year project in pdf format into UTAR Institutional Repository, which may be made accessible to UTAR community and public.

Yours truly,

(*Ong Niam Chi*)

DECLARATION OF ORIGINALITY

I declare that this report entitled "**Development of a Real-Time Gesture Recognition System for Human-Robot Interaction**" is my own work except as cited in the references. The report has not been accepted for any degree and is not being submitted concurrently in candidature for any degree or other award.

Signature : 

Name : Ong Niam Chi

Date : 13/9/2024

ACKNOWLEDGEMENTS

I would like to express my sincere thanks and appreciation to my supervisor, Dr Teoh Shen Khang who has given me this golden opportunity to develop a partner robot project. It is my first step to establish a career in partner robot development field. When I was facing problems in this project, the advice and guidance from Dr Teoh always assists me in overcoming the problems. A million thanks to you.

Finally, I must say thanks to my parents and my family for their love, support, and continuous encouragement throughout the course.

ABSTRACT

In the quickly developing field of robotics and human-robot interaction (HRI), it is crucial for robots enable to recognize and react to human gestures in real-time. This study describes the creation and application of a real-time hand gesture detection system intended using TurtleBot3 Burger in Humble version to improve HRI's effectiveness and naturalness by leveraging recent developments in Robot Operating System (ROS2), computer vision, sensing, machine/deep learning, and Internet of Things (IoT). The system supports navigation and delivery tasks, monitors environmental temperature and humidity, captures images or records videos for surveillance and security, and integrates with Telegram for remote monitoring and alerts.

To capture the finer details of hand gestures and ensure its supported functionality, the suggested system uses a multi-modal method that integrates data from laptop and raspberry pi cameras, and sensors such as LiDAR and DHT22. Robots can now understand a variety of gestures by detecting the number and sequence of open and closed fingers, thanks to the system's robust and accurate gesture detection, which is made possible by a carefully curated dataset and cutting-edge deep neural networks. Low latency between gesture input and robot reaction is made possible by effective model optimization and parallel processing, which gives the system its real-time characteristics. For fluid and interactive HRI situations including collaborative activities, assistive robotics, and entertainment applications, this real-time capacity is essential. The design architecture of the system, data pretreatment methods, and deep learning models used are discussed in the study, with an emphasis on the model's adaptation to various robot platforms and situations. Robots will be able to respond to human cues more contextually if natural language processing (NLP) techniques are incorporated to improve the contextual comprehension of gestures [2]. The system's great accuracy and robustness have been demonstrated through thorough testing in a variety of HRI settings. It has prospective applications in fields including home services, education, manufacturing, business, and entertainment where human-robot interaction must be natural and intuitive.

In summary, the created real-time hand gesture detection system is a significant development in the field of HRI, allowing efficient and smooth communication between people and robots to bridge the gap between them. Its versatility and precision enable a wide range of real-world applications, potentially transforming how humans and robots collaborate and interact.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS

AI	Artificial intelligence
API	Application Programming Interface
AR	Augmented Reality
CPU	Central Processing Unit
CRNN	Convolutional Recurrent Neural Network
IDE	Integrated Development Environment
EMG	Electromyography
GPIO	General Purpose Input Output
GPU	Graphics Processing Unit
GRU	Gated Recurrent Unit
GUI	Graphical User Interface
HCI	Human-Computer Interaction
HRI	Human-Robot Interaction
IMU	Inertial Measurement Unit
IOT	Internet of Things
LCD	Liquid Crystal Display
LDS	LiDAR Sensor
NLP	Natural Language Processing
NN	Neural Network
RAM	Random Access Memory
RGB	Red, Green, Blue
ROS	Robot Operating System
TPU	Tensor Processing Unit
USB	Universal Serial Bus
VR	Virtual Reality

Chapter 1
Introduction

New opportunities for human-robot interaction (HRI) have emerged because of the integration of robots into a variety of facets of our life, from manufacturing, business, and education to entertainment and daily help. But to fully take advantage of these interactions, it is crucial to create clear and effective channels of communication between humans and robots. In order to meet this demand, real-time gesture recognition systems have emerged as a potential option, allowing people to communicate with robots through simple hand gestures. The goal of this project is to create a real-time gesture recognition system for human-robot interaction using cutting-edge robotics, computer vision, IoT, and deep learning techniques. The system intends to improve human-robot communication across a variety of disciplines in terms of simplicity, security, and effectiveness.

1.1 Problem Statement and Motivation

Problem Statement

The issue at hand is on the necessity to develop a reliable and effective real-time gesture detection system that can be easily included into scenarios of human-robot interaction.

The main difficulties and issues with this project are as below:

1. **Recognition of gestures Accuracy**

   Creating and putting into practice a hand gesture recognition algorithm or model that can reliably and accurately identify a variety of user-performed gestures. This covers hand movements produced in various lighting environments, hand orientations, and hand shapes.
2. **Real-Time Performance**

   Making sure the gesture recognition system runs quickly and in real-time. For genuine and responsive interactions between humans and robots, real-time performance is essential.

3. **User Interface and Feedback**

   Creating an intuitive user interface for gestural communication with the robot. Giving people feedback to verify that the system comprehends their motions and the robot's activities.

4. **Gesture-to-Action Mapping**

   The creation of a sophisticated mapping system that converts recognized gestures into useful robot actions or responses such as navigation and delivery tasks, monitoring environmental temperature and humidity, capturing images or recording videos for surveillance and security, and integrating with Telegram for remote monitoring and alerts. This entails specifying the necessary responses for each motion recognized and making sure the system is responsive.

Motivation

Several important variables and objectives are what spur the development of a real-time gesture recognition system for human-robot interaction. Systems for human robot interaction try to make human-robot interaction feel as natural and intuitive as feasible. Humans communicate primarily through gestures and incorporating them into interactions with robots enables users to express their intentions and directions more organically.

The user experience can be improved through gesture-based interaction since it makes controlling and interacting with robots more enjoyable and user-friendly. Increased user happiness and robotic technology acceptance may result from this. Robots can be more usable for people who have physical limitations or have trouble utilizing conventional input methods (such keyboards or touchscreens) thanks to gesture recognition. It makes it possible for those who might have trouble speaking or moving around to communicate with robots efficiently.

Real-time gesture recognition can increase safety in situations where humans and robots interact closely, like in home services or manufacturing, business, and education facilities. Accident risk can be decreased by users being able to quickly issue commands or warnings to robots. Particularly in circumstances when verbal communication is difficult or impracticable, gestures can be an effective means of communicating instructions or giving robots feedback. This may result in increased task productivity and efficiency.

Gesture recognition offers a level of immersion and interaction to entertainment and gaming apps. In virtual settings, players can manipulate items or characters by using their own hands or bodies. With applications in computer vision, deep learning, robotics, IoT, and

human-computer interaction, gesture recognition technology is a vibrant area of research and innovation. Real-time system development fosters development in these areas.

1.2 Research Objectives

The project's main objective is **to develop a real-time gesture recognition system for human-robot interaction (HRI)**, which enables intuitive and effective communication between humans and robots using natural hand signals.

The project can be divided into several sub-objectives, each addressing a specific aspect of the development and deployment of the gesture recognition system, helping to achieve the main objective:

Accuracy of Gesture Recognition:

1. To develop and implement a deep learning-based gesture recognition model that can properly identify a variety of hand gestures.
2. To collect and prepare a broad collection of gestures for training and validation.

Real-Time Performance:

3. To reduce processing and response times by optimizing the gesture recognition system for real-time performance.

User Interface and Feedback:

4. To create and build a user-friendly interface that enables people to communicate with the robot using gestures.
5. To implement feedback methods to give users confirmation and feedback on their gestures and the activities the robot takes.

Gesture-to-Action Mapping:

6. To develop an intelligent mapping system that translates recognized gestures into useful robot actions or reactions.
7. To define and put into practice the relevant actions for each gesture that has been identified.

1.3 Project Scope and Direction

This project's goal is to design, develop, and implement a real-time gesture detection system for human-robot interaction (HRI) employing cutting-edge robotics, computer vision, IoT, and deep learning techniques. The system attempts to make it possible for people and robots to communicate naturally and effectively through hand and body gestures. The following crucial elements are included in the project:

1. **Gesture Recognition**

   The precise identification of a wide range of hand gestures and signals. The system ought to be able to recognize gestures reliably in a variety of locations and lighting situations.

2. **Real-Time Performance**

   Ensure responsive interactions between users and robots, the gesture recognition system will run in real-time with little latency.

3. **User Interface and Feedback**

   Enable interaction with the robot through gestures, a user-friendly interface will be created. Users will receive feedback from the system to confirm that the robot has understood their motions and activities.

4. **Gesture-to-Action Mapping**

   Convert recognized gestures into useful robot actions or reactions, a sophisticated mapping system will be created. This entails determining the proper course of action and making sure the system is responsive.

1.4 Contributions

Our investigation and evaluation validate the viability of the proposed methodologies and features for real-time gesture recognition in human-robot interaction (HRI). Firstly, the integration of multi-modal data sources, including laptop and raspberry pi cameras, and sensors such as LiDAR and DHT22, proves effective in capturing nuanced hand gestures, laying a foundation for robust gesture recognition system that supports navigation and delivery tasks, monitors environmental temperature and humidity, captures images or records videos for surveillance and security, and integrates with Telegram for remote monitoring and alerts. Secondly, the implementation of cutting-edge deep learning models enables accurate and real-time gesture detection, enhancing the responsiveness and effectiveness of HRI systems. Thirdly, the development of an intuitive user interface and feedback mechanisms fosters

seamless communication between humans and robots, further enhancing the user experience and usability of the system.

The experiment and analysis confirm the improvements in Human-Robot Interaction. It improved connection between humans and robots through the use of instinctive and natural gestures. By improving the entire user experience by making it more accessible and user-friendly, particularly in situations where users might lack technical expertise. Secondly, it expanded robotic versatility. By allowing robots to recognize gestures and understand and respond to non-verbal cues, the variety of jobs and applications where robots can be used efficiently is being increased. Thirdly, its inclusivity and accessibility are enhancing how people with physical limitations or those unable to communicate using conventional methods connect with robotics and technology. Fourthly, its efficiency and security. By enabling real-time gesture-based instructions and cautions to be sent from humans to robots in collaborative contexts, safety is increased. It is also enhancing the effectiveness of collaboration between humans and robots in fields like manufacturing, business, and education. Fifthly, it contributed to games and entertainment too. It is enhancing interactive entertainment and gaming by giving users the ability to manipulate real-world or virtual robots with gestures. It is also designing engrossing virtual reality (VR) and augmented reality (AR) experiences. Lastly, developmental and Research. Supporting study in the fields of robotics, deep learning, computer vision, IoT, and human-computer interaction (HCI) and giving algorithms and models for gesture recognition useful datasets and benchmarking opportunities.

1.5 Report Organization

The details of this research are shown in the following chapters. The introduction segment offers a broad perspective of the project's aims, outlining the objectives and scope in chapter 1. Following this, Chapter 2 reviews some related backgrounds delve into existing research and technologies relevant to gesture recognition systems, providing a foundation for the proposed approaches. Then, In Chapter 3, the proposed methodology and approach are detailed, encompassing project phases and the utilization of the Embedded Development Life Cycle (EDLC) model. System design, architecture, and various diagrammatic representations like use case and activity diagrams are also discussed. Furthermore, Chapter 4 presents the detailed design of the gesture recognition system, including system block diagrams, and the design of circuits and their interactions. Specifications of hardware such as TurtleBot3 Burger, raspberry pi, and OpenCR, firmware like Ubuntu and Robot Operating System 2 (ROS2), and software

components such as visual studio codes, OpenCV, and Nav2 Simple Commander are also outlined. Chapter 5 describes the practical setup and integration of the system, including hardware and software setup, connections between the PC and TurtleBot3, SLAM, navigation, and system operations, along with challenges faced during implementation. Chapter 6 evaluates the system's performance through testing, discusses the challenges encountered, and assesses whether the project's objectives have been achieved. Lastly, Chapter 7 is concluding the report which delineates the development process, including data pre-processing, model training architecture building, and data training phases. It also synthesizes the project's findings, proposes future directions, and offers closing remarks. Through this structured approach, the report aims to provide a clear understanding of the project's progression, methodologies employed, and outcomes achieved.

Chapter 2
Literature Review

2.1 Previous Works on Introduction to Gesture Recognition

In recent times, the utilization of human movements, especially hand gestures, has become increasingly prominent in the realm of Human-Computer Interaction (HCI). This advancement has spurred extensive research efforts focused on modelling, analyzing, and identifying hand gestures. The techniques developed within HCI have wide-ranging applications in areas such as surveillance, robot control, and teleconferencing.

Detecting gestures is a complex undertaking involving various aspects like motion modelling, motion analysis, pattern recognition, and machine learning. It even incorporates studies from the field of psycholinguistics. Within the discipline of studying and interpreting human motion, there are already numerous comprehensive survey papers accessible. In this literature review, there will be some developed projects being reviewed. The first developed project is a degree project "**Hand Gesture Detection and Recognition System [19]**" which completed by a Master student of computer engineering, Muhammd Inayat Ullah Khan and the second project "**Hand Gesture Recognition [20]**" also completed by students which are Jerald Siby and Hilwa Kader. The third project is "**EMG-Based Dynamic Hand Gesture Recognition Using Edge AI for Human–Robot Interaction [17]**" by E. Kim, J. Shin, Y. Kwon, and B. Park. While the fourth project is "**Virtual Reality-Based Interface for Advanced Assisted Mobile Robot Teleoperation [18]**" by J. E. Solanes, A. Muñoz, L. Gracia, and J. Tornero.

2.1.1 Review of equipment

Review of hardware platform

In the project of Muhammd Inayat Ullah Khan, the hardware used is a webcam system and a 2.8 GHz processor Computer system.

As a summary for the webcam system, the specifications required for this project are listed below:

Table 2-1 Specifications required for the webcam system.

Specifications	Descriptions
Resolution	640 x 480
Video frame rate	30fps @ 640 x 480
Pixel depth	Minimum 1,3 Mega pixels
Connection port	USB

The webcam was attached to the computer through a USB connection and was set to continually capture frames. Users just needed to pick the correct algorithm technique button on the interface to record a certain frame, and the system would then recognize the hand in that specific frame. The webcam collected color photos, which were then converted to grayscale format. The decision to work in grayscale was inspired largely by the increased computing load associated with processing color photographs.

In the project of E. Kim, J. Shin, Y. Kwon, and B. Park, the hardware used is a critical component of the system designed for dynamic hand gesture recognition in human-robot interaction (HRI).

As a summary for the system designed, the specifications required for this project are listed below:

Table 2-2 Specifications required for third project.

Specifications	Descriptions
Industrial Robot Arm (UR3)	Six-axis joint robot arm, Maximum payload as 3 kg and the maximum working radius as 500 mm.
Gripper (Robotiq 2F-140)	Two fingers to grasp and manipulate objects, each 140 mm wide.
EMG Sensor (Myo Armband)	Eight surface EMG electrodes, Nine-axis inertial measurement unit (IMU) composed of a three-axis accelerometer, three-axis gyroscope, and three-axis magnetometer.

Embedded Environment (NVIDIA Jetson Nano)	Designed for edge AI applications and capable of running Ubuntu 18.04, Have GPU acceleration capabilities, which are beneficial for deep learning and AI tasks.

The Universal Robot UR3, a small robotic arm created for collaborative purposes, is used in the investigation. The selection of this hardware demonstrates how useful the system is for actual industrial applications. A commercially accessible EMG sensor called the Myo armband is used to record surface electromyography signals from the user's forearm. The Myo armband supports the ROS package, which makes system integration easier, and is lightweight, making it suited for wearable applications. The embedded environment for the suggested system is the NVIDIA Jetson Nano developer kit. The selection of the Jetson Nano fits with the paper's focus on edge AI for in-the-moment EMG signal processing.

Overall, the hardware platform used for the research article is appropriate for the study's objectives. It combines an edge AI platform, a wearable EMG sensor, and a powerful industrial robot arm and gripper. Data integration is made simpler by the Myo armband's interoperability with the ROS package, while the NVIDIA Jetson Nano offers the necessary processing power for real-time gesture detection.

In order to design a system for dynamic hand gesture detection in human-robot interaction, practical and functional considerations were taken into account when choosing the hardware. It guarantees that the system can be put to use in actual situations, especially in industrial settings where accuracy, efficiency, and real-time control are crucial.

In the project of J. E. Solanes, A. Muñoz, L. Gracia, and J. Tornero, the following hardware specifications and components are mentioned or implied as being required for the teleoperation system.

As a summary for the teleoperation system, the specifications required for this project are listed below:

Table 2-3 Specifications required for the teleoperation system.

Specifications	Descriptions
Oculus Quest 2 VR Headset	6 GB of RAM, a Quadcomm Snapdragon XR2 processor, an LCD screen with a resolution of 1832 × 1920 pixels per eye, and standalone capabilities.
Xbox Wireless Controller (Gamepad)	Utilized for virtual and actual robot control. The VR headset and gamepad pair together to create Bluetooth connection.
Turtlebot3 Burger (Mobile Robot)	Featuring two servos. The 360º LiDAR sensor (LDS) for environmental sensing, the Raspberry Pi-3 for high-level control, the OpenCR (32-bit ARM Cortex-M7) embedded controller for robot control, and the Dynamixel XL430-W250-T for the wheels.

While these are the hardware elements specifically listed, other hardware elements, such as power supplies, communication cables, and any required adapters to assure compatibility between the components, can also be needed for the configuration. Additionally, the precise versions of the VR headgear (Oculus Quest 2) and gamepad (Xbox Wireless Controller) are specified, so it's crucial to use these models—or suitable substitutes—for the system to function as intended.

2.1.2 Review of programming language

In the project of Muhammd Inayat Ullah Khan, the programming language used is C++. C++ was used to develop this project is because the given the time limits and complexities associated with creating a system in C++ will be shorter. Besides, This prototype prioritized detection performance optimization. The system was built to accept a broad variety of inputs

with varied sizes and picture resolutions. The emphasis was on building a rigorously programmed and extensively documented system to ease future growth.

C++ is a flexible middle-level programming language that was established in 1979 by Danish computer scientist Bjarne Stroustrup while working at Bell Laboratories USA. A middle-level programming language, for those unfamiliar, incorporates characteristics of both low-level and high-level programming languages. C++'s origins may be traced back to the C programming language, which was widely used in system programming and Unix-based operating systems. C++ has developed as one of the most widely used programming languages during the course of its existence. Its applications are many, spanning gaming, robotics, finance, and scientific computing [4].

In the project of Jerald Siby and Hilwa Kader, the algorithm was implemented in MATLAB programming language [5]. MATLAB is a high-level programming language for engineers and scientists that allows them to easily express matrix and array mathematics. MATLAB may be used for every aspect, from basic interactive instructions to constructing large-scale systems. Benefits using MATLAB develop project are vectorized operations, user friendly, expandable functionality, simple math annotations [6].

The system's implementation language is not stated directly in the research article titled “EMG-Based Dynamic Hand Gesture Recognition Using Edge AI for Human-Robot Interaction”. However, it is likely that the authors utilized a combination of the following programming languages and libraries given the context and accepted procedures in the fields of machine learning, robotics, and edge AI. Python is a popular programming language for applications involving deep learning and machine learning. It provides a huge ecosystem of tools and packages, including scikit-learn, pytorch, and tensorflow, which are frequently used to build neural networks and analyze EMG data. Google created the open-source deep learning framework known as TensorFlow. It is a popular alternative for implementing deep learning models, particularly convolutional recurrent neural networks (CRNNs), because it offers tools for creating and training neural networks.

Robot Operating System (ROS) is a versatile platform for developing robot software. Although not a programming language in and of itself, developing robot control and communication modules frequently entails using Python and C++. Python and C++ were probably used for ROS-related programming, as the publication references ROS for controlling the robot. NVIDIA GPUs may have been utilized for deep learning projects if GPU acceleration was used (which is typical for computationally heavy workloads). CUDA is a parallel

computing platform and API. Linux Shell scripts, such as Bash scripts, are frequently used when working with embedded systems like the NVIDIA Jetson Nano for setting up and configuring the system, managing dependencies, and automating activities [1].

The system's implementation language is not stated directly in the research article titled “Virtual Reality-Based Interface for Advanced Assisted Mobile Robot Teleoperation”. Typically, in robotics and virtual reality applications, a combination of programming languages and frameworks is used to achieve the desired functionality. Thus, the programming languages used in this project will be quite similar to the previous projects which are Python, C++ and ROS. However, there are some different programming languages should be used such as Unity3D and Unreal Engine, Java, and Web Technologies. The creation of VR worlds and simulations frequently makes use of these game development engines. They offer scripting and development support for C# (Unity3D) and C++ (Unreal Engine). Due to its prominence as the primary programming language for Android app development, Java may be utilized for VR applications that run on Android. HTML5, JavaScript, and WebGL can be used to build web-based VR applications that enable users to interact with VR content in web browsers [3].

2.1.3 Review of algorithms and techniques

In the project of Muhammd Inayat Ullah Khan, the hand gesture recognition system was divided into 3 modules which are Preprocessing, Feature extraction of the processed image and Real time classification. Preprocessing mainly consists of 4 steps which are Skin Modelling, Removal of Background, Conversion of RGB to binary and Hand Detection. After these steps were prepared, then diagonal Sum and other algorithms only can be calculated.

RGB: The basic colors of the RGB color paradigm are represented by RGB, which stands for red, green, and blue. This model works on an additive concept, combining various quantities of red, green, and blue light to produce a broad range of colors. It is critical to understand in RGB image processing that an RGB image is basically a composite of three independent grayscale images, each corresponding to the intensity of red, green, and blue light [7].

After Preprocessing, the algorithm implemented in Muhammd Inayat Ullah Khan’s project is:

Table 2-4 Algorithm implemented in Muhammd Inayat Ullah Khan's project.

Names	Description
Row vector algorithm	A row vector is defined as a single row of numerical values with a resolution of 1*Y, where Y is the total number of columns in the picture matrix. Each element in this row vector represents the sum of the corresponding column elements in the matrix. It essentially condenses the data from each column into a one-dimensional representation.
Edging and row vector passing	After converting a picture from RGB to grayscale format, where each pixel is represented by a brightness or darkness value. There are two techniques for defining pixel brightness: the "Double" class, which uses floating-point values between 0 and 1, and the "unit8" class, which uses integers from 0 to 255. The "unit8" class uses less storage space. The text mentions edge extraction with a threshold of 0.5 to reduce noise after grayscale conversion. The bordered picture generates a row vector, which is utilized as input for training a neural network (NN) to categorize movements. To express this algorithm in mathematics, Input to NN= Row vector [Edge (Grayscale image)]

Mean and standard deviation of edged image	The mean, denoted as μ (mu), is calculated by summing up all the pixel values within a matrix and then dividing this sum by the total number of values in the matrix. Mathematically, this can be expressed as: $$Xm = \sum_{i=1}^{u} Xi/n$$ While Standard Deviation is: $$s = \sqrt{\frac{\sum_{i=1}^{n}(xi - xm)^2}{n}}$$ Mathematically, the input provided to the neural network is defined as follows: Input to NN= Mean (Edge (Binary image)) + S.D (Edge (Binary Image))
Diagonal sum algorithm	The binary picture format saves image data as a matrix but only permits two unique colors: black and white, with no intermediate shades. It assigns the value 0 to black pixels and the value 1 to white pixels. The system then computes the sum of all the elements in the matrix along each diagonal in the following stage. The real-time input delivered to the system may be described mathematically as follows: $$Xi = \sum_{i=1}^{u} Diagonals$$

In the project of Jerald Siby and Hilwa Kader, coding approaches was used to read, display, convert the image. The image was captured using the Image Processing Toolbox in MATLAB. Before begin programming, must first acquire information about the attached

camera. The 'imaqhwinfo' command (which stands for image acquisition hardware information) is used to do this. It should be noted that one adapter may be connected to several devices. To correctly recognize the camera, the device's ID must be obtained. In the MATLAB command window, use 'imaqhwinfo('winvideo')' to do this. The programme defines the number of frames per trigger and the frame capture intervals. When the specified frame is obtained, the image is recorded using the webcam. Some examples of basic commands used was listed below:

Table 2-5 Basic commands used in the project of Jerald Siby and Hilwa Kader.

Commands	Description
Imread	Read an image
Imshow	Display an image
Imaqreset	Reset the camera
rgb2gray	Convert a color image to
gray2rgb	Convert a gray scale image to color image

Electromyography (EMG) signals and edge AI approaches are proposed in the research paper titled "EMG-Based Dynamic Hand Gesture Recognition Using Edge AI for Human-Robot Interaction" to create a system for dynamic hand gesture identification. In this study, the Convolutional Recurrent Neural Network (CRNN) is the main approach employed. Convolutional Recurrent Neural Networks (CRNNs) are a class of deep learning models that combine the sequential modelling capabilities of RNNs with the feature extraction capabilities of CNNs. In this study, hand gesture recognition using EMG signal processing using CRNNs. Convolutional layers, activation functions, pooling layers, fully linked layers, a gated recurrent unit (GRU) layer, and a Softmax output layer make up the CRNN architecture.

The Myo armband's EMG data are pre-processed before being sent into the CRNN model. The EMG data are normalized using min-max normalization, according to the paper, which adjusts the data to a range between 0 and 1. Following that, the EMG data are reorganized into a two-dimensional format with dimensions of 50 x 8. The CRNN model is trained using this pre-processed data as input features. The research focuses on edge AI, which employs an embedded board (NVIDIA Jetson Nano) to process AI rather than using standalone personal

computers. An embedded system is the best option when need processing that is lightweight, reliable, and quick, especially in the context of smart factories and real-time robot control.

In summary, for dynamic hand gesture detection utilizing EMG signals, the paper uses cutting-edge deep learning techniques, notably CRNN. The authors use examples of live human-robot interaction to show the efficacy of this method. They also stress the significance of edge AI for applications in smart manufacturing and robotics since it enables lightweight, real-time processing. The transferability of the trained model to new users is also highlighted in the research, demonstrating its potential for expanded application in human-robot interaction systems.

The journal article “Virtual Reality-Based Interface for Advanced Assisted Mobile Robot Teleoperation” does mention the use of a potential field-based navigation method for mobile robot control, which is a significant algorithmic component in this context. It is renowned for being both straightforward and successful at avoiding obstacles. It might, however, be constrained in complicated situations that contain tight spaces or moving obstructions. More information regarding the precise settings and tweaking applied to this algorithm in their system may have been supplied in the article. Since the article's main focus is on the VR interface and usability study, it doesn't provide a thorough examination of cutting-edge robotics or VR-related algorithms or methodologies. The particular application and the hardware capabilities of the robot are frequently taken into consideration while choosing an algorithm.

2.1.4 Review of application of Gesture Recognition

1. Hand gesture recognition technology is used to operate robotic arms and other gear, allowing for natural human-machine interaction.
2. This technology may be linked into security systems, where unique hand gestures can be used to access lockers or safes, so improving security.
3. Hand gesture-controlled phones, such as dumb assistance phones, can be deployed in public areas and offices, allowing both people with disabilities and the general public to use them.
4. The system's capabilities can be expanded to help deaf people communicate. Voice-to-text translation allows a deaf or mute person to connect with those who can hear and speak, bridging communication gaps.

As explained in the research paper "EMG-Based Dynamic Hand Gesture Recognition Using Edge AI for Human-Robot Interaction," the use of gesture recognition is an essential part of human-robot interaction (HRI) technology in a variety of fields, particularly in industrial settings and smart factories. For examples, Smart Factories and Industrial Automation, Robotic Control, Gripper Control, Human Work Augmentation, Rehabilitation and Assistance, Edge AI Implementation, and Universal Gesture Classification. Gesture recognition has several uses in scenarios including human-robot interaction, smart manufacturing, and industrial automation. Control systems are made easier to use, human-robot interaction is improved, and manufacturing processes are more productive and safer as a result. Additionally, the adoption of edge AI for gesture detection fits with the need for quick, reliable, and portable solutions in robotics and industrial applications.

2.2 Review of challenges, issues, and limitations

The research paper "EMG-Based Dynamic Hand Gesture Recognition Using Edge AI for Human–Robot Interaction" presents an innovative approach to gesture recognition, but it also acknowledges several challenges, issues, and limitations. Even for the same person doing the identical motion, EMG signals might differ greatly from person to person. The development of a reliable and comprehensive system for gesture recognition is complicated by this variability. The paper uses a single Myo armband sensor, which limits the number of muscles and data points that can be monitored. Using multiple sensors could enhance the system's ability to capture complex gestures accurately.

The one of the limitations for the journal article discuss the development of a virtual reality (VR)-based interface for the teleoperation of mobile robots is the usage of a LiDAR sensor on the robot for obstacle detection. The fact that the quality of the teleoperation experience is highly dependent on the accuracy and field of view of the sensors presents one of the obstacles in VR-based teleoperation. LiDAR sensors might not be able to detect transparent or reflecting objects in some settings.

The common challenges, issues, and limitations are Gesture Recognition Reliability, Latency and Real-Time Control, Hardware Requirements and Limited Real-World Testing. Although it is not covered in detail in the article, if gesture recognition were to be added to the VR interface, it would run into problems with accurately recognizing user gestures. Accurate recognition can be impacted by a variety of factors, including lighting, occlusions, and human

variation in gesture execution. Real-time processing and control are essential for industrial and robotic applications. Safety and efficiency issues could arise from slow gesture detection. Although appropriate for lightweight processing, edge AI platforms may have hardware restrictions that slow down and impair identification performance. The choice and optimization of hardware are crucial. The performance assessment of the recognition model in a carefully monitored experimental setting is the main emphasis of the research. Real-world industrial environments could present extra complexities and difficulties.

2.3 Proposed Solutions

This project aims to propose solutions for the above challenges, issues, and limitations. Future study could investigate the integration of numerous sensors, possibly positioned on various areas of the arm, to capture a more thorough set of muscle actions to alleviate the constraints associated with a single EMG sensor. The robot's perception abilities can be improved by combining data from various sensors, including depth sensors, cameras, and LiDAR. By enabling a more thorough awareness of the surroundings, sensor fusion approaches can enhance navigation and obstacle identification in challenging situations.

To improve the reliability of gesture detection, machine learning models can be trained on a wide dataset of user gestures, accounting for differences in lighting conditions and user behaviors. Accuracy can also be increased by using more sophisticated computer vision algorithms, including 3D gesture recognition. To accomplish real-time processing of EMG signals and reduce processing times for gesture identification, optimize the edge AI platform and algorithms. The advantage is in industrial environments, lower latency guarantees that the system reacts quickly to user gestures, increasing safety and effectiveness.

It is possible to work on creating VR accessories and headsets that are more affordable to address the pricing issue. Additionally, users' hardware requirements may be reduced via cloud-based VR solutions that delegate certain processing to distant servers. A more thorough evaluation of the system's viability would result from extensive testing in actual industrial settings with a range of environmental factors and user demographics. Conduct thorough system testing and validation in actual industrial settings. The Advantage is Real-world testing ensures the system's viability by identifying and addressing problems unique to industrial applications.

2.3 Autonomous Navigation Evaluation

Autonomous navigation is an essential component of mobile robotics, particularly in applications that require precision and agility in dynamic surroundings. “Development of an automated benchmark for the analysis of Nav2 controllers” by Federica Schena gives a detailed analysis of the performance of various Nav2 controllers in the context of smart wheelchair navigation using the ROS 2 framework. This work is especially important to projects requiring mobile robot navigation and path planning since it compares the performance of four distinct controllers in a range of real-world scenarios [8].

The study focuses on the Dynamic Window Approach (DWB), Regulated Pure Pursuit (RPP), Time Elastic Band (TEB), and Model Predictive Path Integral (MPPI) controllers. Each controller has various benefits for particular navigation assignments, such as basic environment traverse, sophisticated obstacle avoidance, and sharp turning. The primary technical focus of this thesis is on simulation-based evaluation using ROS 2's Gazebo environment, which measures parameters such as kinematic performance, route consistency, and avoiding collisions across five different scenarios: empty surroundings, stationary obstacles, single obstacles, and restricted spaces [8].

The findings of this test reveal that, while each controller shines in specific situations, none provides a universally superior answer. For example, the DWB controller performed well in smooth path following but suffered in intricate obstacle avoidance. In contrast, MPPI and TEB displayed more adaptability in dynamic situations, particularly in scenarios featuring rapid turns or thick obstacle fields. However, these controllers tend to demand more computing resources, which may be an issue for real-time systems with limited processing capacity. The overall results are shown in figure below [8]:

Performance indexes	DWB	RPP	TEB	MPPI
Success rate	100.00%	100.00%	100.00%	100.00%
Avg linear speed (m/s)	0.466/0.5	0.5/0.5	0.476/0.5	0.459/0.5
Avg path length (m)	7.238	7.231	7.247	7.236
Avg time taken (s)	15.498	14.512	15.14	15.66
Avg integrated x jerk (m^2/s^6)	1.811	0.393	1.445	0.834
Max velocity (m/s)	100.00%	100.00%	100.00%	100.00%
Max local planned path error (m)	0.062	0.063	0.051	0.061
Min local planned path error (m)	0.00	0.00	0.00	0.00
Avg local planned path error (m)	0.023	0.017	0.017	0.017
Avg heading error $(°)$	0.495	0.373	0.358	0.349
Std heading error $(°)$	0.005	0.001	0.023	0.025
Energy Consumption (J)	3.509	3.647	3.55	3.475

Figure 2-1 Sample of overall results for autonomous navigation evaluation [8]

Schena's advice to investigate a hybrid system that combines the capabilities of each controller is very notable. This is consistent with the necessity for adaptable solutions in real-world robotic systems, where navigation issues vary greatly depending on ambient complexity and robot characteristics. The thesis also emphasizes the need of creating modular systems with ROS 2 and Nav2, which enable the easy integration of additional controllers or modifications to current ones based on unique use cases [8].

Chapter 3

Proposed Method/Approach

The processes of the project were categorized into different phases in the development, which were project pre-development, data pre-processing, model training architecture building and data training, and prediction on test dataset. The design methodology used in this project is prototyping model of embedded development life cycle (EDLC) approaches. Prototyping model is developed in multiple cycles. Produces a more refined prototype at the end of each cycle instead of functionality.

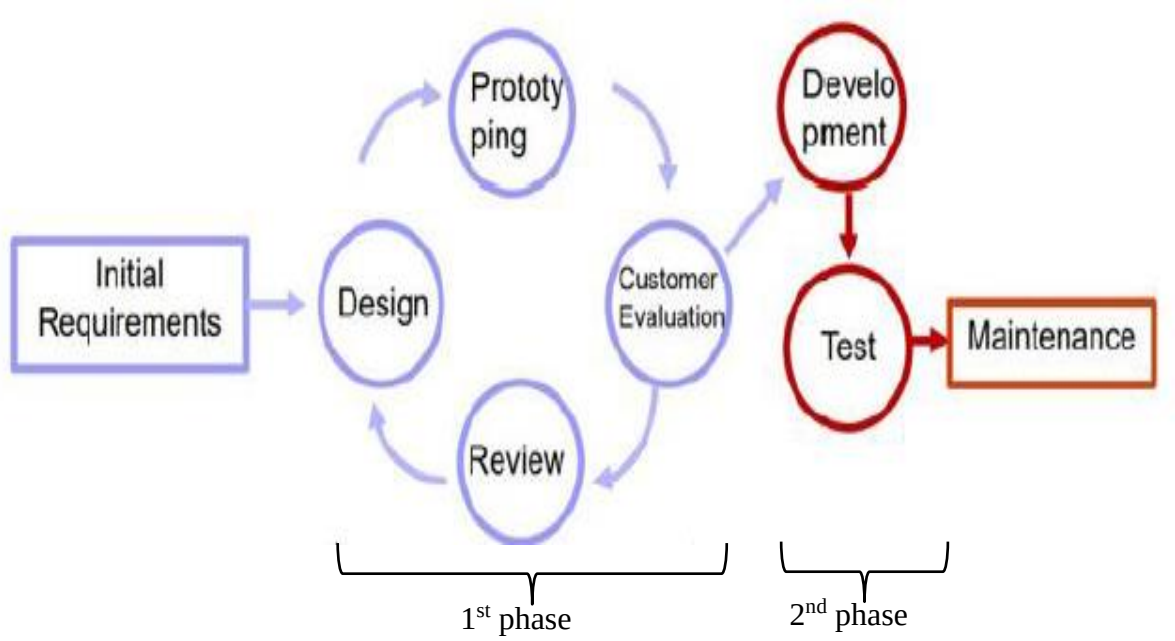


Figure 3-1 Embedded Development Life Cycle (EDLC)

In previous FYP1, the progress is still at the 1st phase of the Embedded Development Life Cycle (EDLC). While FYP2 is progressing the 2nd approaches phase of the Embedded Development Life Cycle (EDLC). The maintenance will be continued in the future when the system product is launched in the market.

3.1 System Design Diagram/Equation

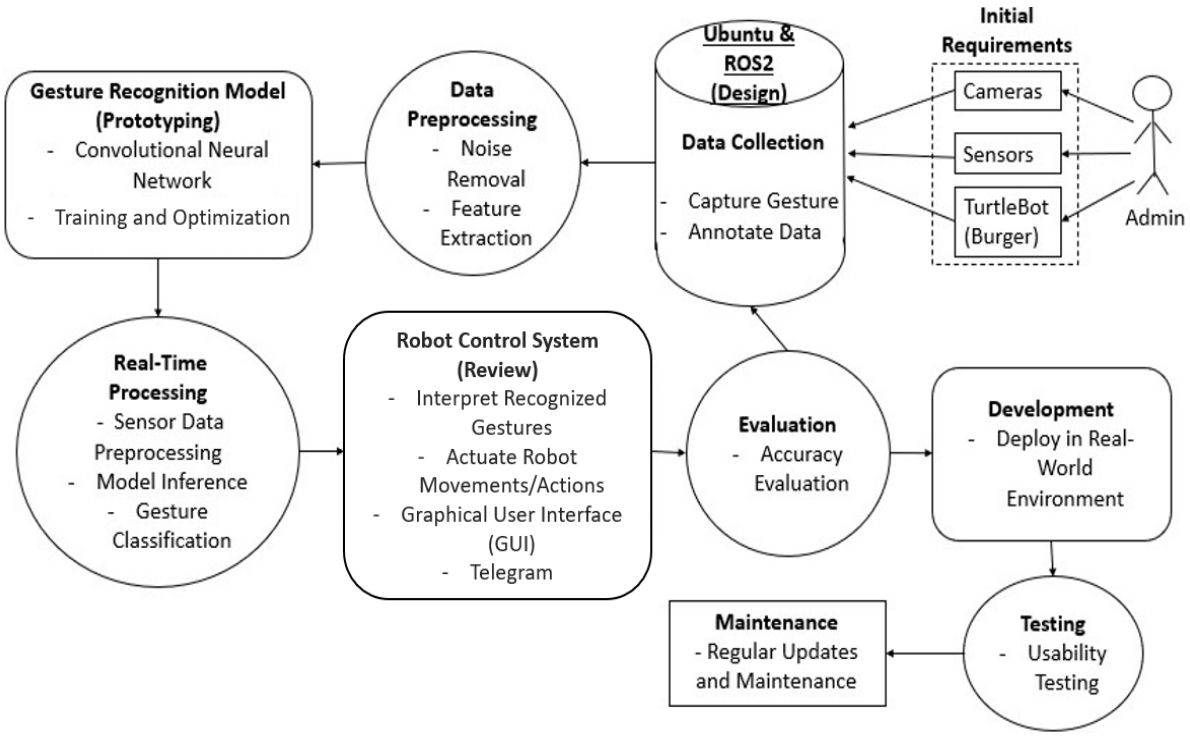


Figure 3-2 System Design for Development of a Real-Time Gesture Recognition System for Human-Robot Interaction

The following are the suggested techniques and technologies for attaining the main and sub-objectives of the real-time gesture detection system for human-robot interaction:

1. **Gesture Recognition Model:**

Technique/Technology: Deep Learning (Convolutional Neural Networks).

Justification: Deep learning models, particularly CNNs, have shown exceptional success in identifying the presence and location of hands within a video frame and determining the positions of key points (e.g., fingertips, joints) on the detected hands, which qualifies them for the recognition of intricate gestures. They can accurately recognize gestures by learning hierarchical features from gesture data.

2. **Real-time performance improvement:**

Technique/Technology: Model quantization and parallel processing are all optimization strategies.

Justification: To attain real-time performance, shorten inference times, and ensure rapid user-robot interactions, optimization strategies are essential.

3. **User Interface and Feedback:**

Technique/Technology: Designing a graphical user interface (GUI) for gesture interaction and feedback mechanisms (such as visual cues).

Justification: A user-friendly interface makes it easier for users to interact, and feedback mechanisms confirm robot operations and gesture detection, improving the user experience.

4. **Gesture-to-Action Mapping:**

Technique/Technology: The development of a mapping algorithm or rule-based system to convert recognized gestures into robot actions.

Justification: By defining the robot's behavior based on recognized gestures, mapping algorithms make sure the system responds appropriately to user commands.

The equation represents the mathematical model or algorithm used within the system for gesture recognition or data processing. A general placeholder equation:

Gesture Recognition Algorithm: G(x) = f(input data)

Where:

- G(x) represents the recognized gesture.
- f() is the function representing the gesture recognition algorithm.
- "input data" refers to the data collected from input devices, such as images or sensor readings.

This equation outlines the basic structure of the gesture recognition algorithm within the system. Specific details about the algorithm and its implementation would be provided in further sections or documentation.

3.2 System Architecture Diagram

The system architecture diagram illustrates the flow of data and interaction between various components of the real-time gesture recognition system for human-robot interaction.

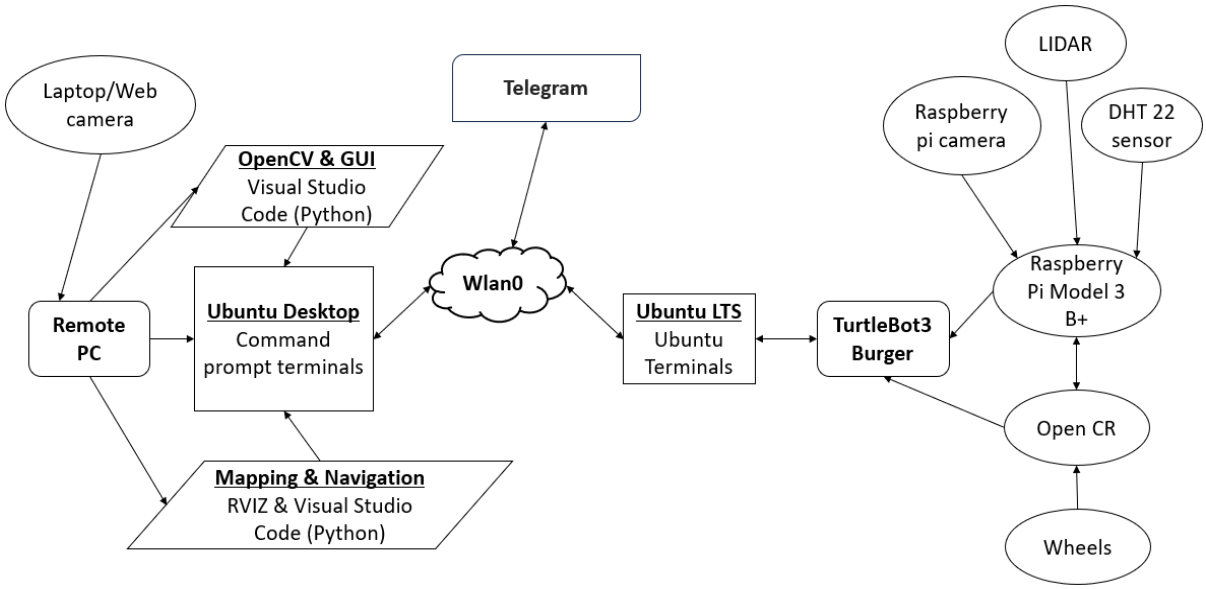


Figure 3-3 System Architecture Diagram for Development of a Real-Time Gesture Recognition System for Human-Robot Interaction

In the diagram, the components are represented as follows:

Gesture Recognition Model: This component comprises deep learning devices such as Convolutional Neural Networks (CNNs) which analyze input data to recognize gestures.

Cloud Functions: These functions act as subscribers to receive data from the deep learning devices. They may perform additional processing or trigger actions based on the recognized gestures.

Input Devices: These devices capture input data, such as images or sensor readings, which are then processed by the gesture recognition model.

Robot Controller: The controller receives instructions or commands based on the recognized gestures and controls the actions of the robot accordingly.

Feedback Mechanisms: These mechanisms provide feedback to users, confirming gesture recognition or indicating the robot's response.

IoT: Telegram is integrated for IoT monitoring and alerting purpose. Telegram will receive input data, such as images or sensor readings and send output data to control some functions of the robot system through the Internet.

The arrows indicate the flow of data and control signals between the components, illustrating the system's operation in real-time human-robot interaction scenarios.

3.3 Timeline

3.3.1 FYP 1 timeline

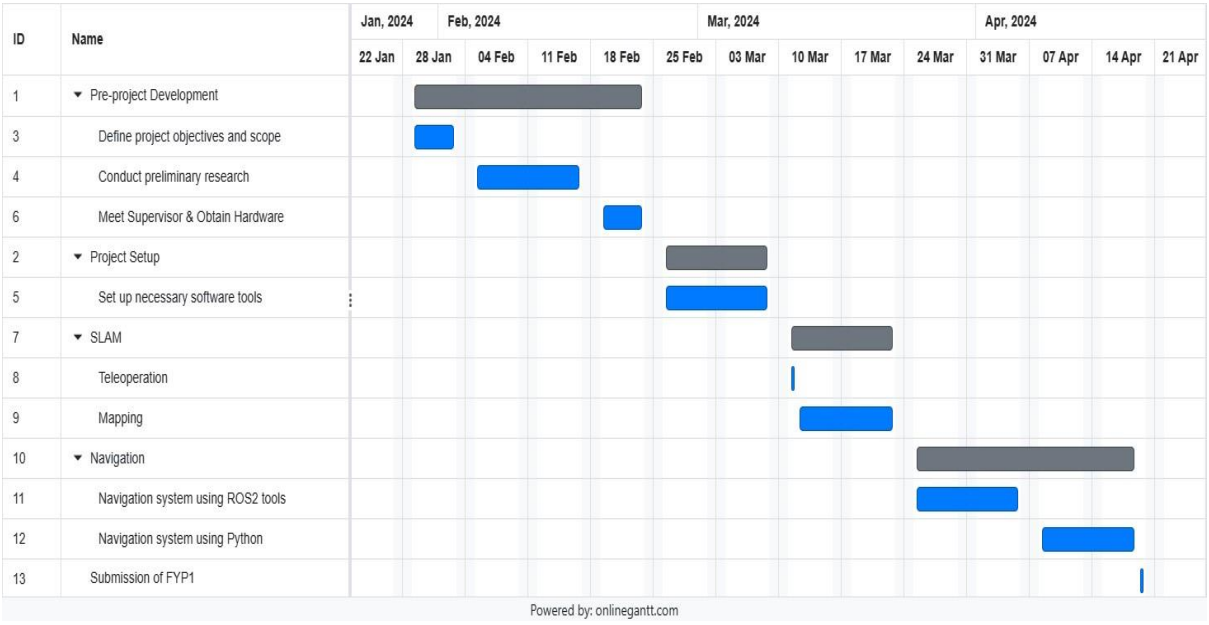


Figure 3-4 timeline of FYP1

3.3.2 FYP 2 timeline

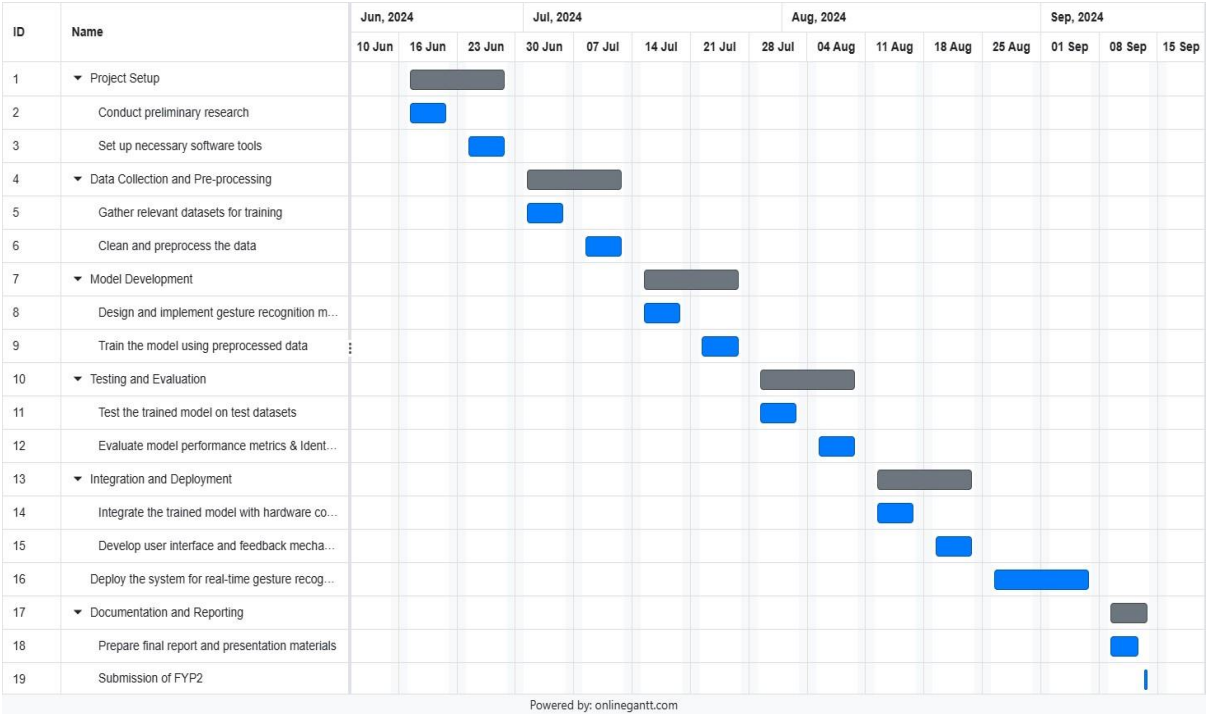


Figure 3-5 timeline of FYP2

Chapter 4

SYSTEM DESIGN

4.1 System Block Diagram

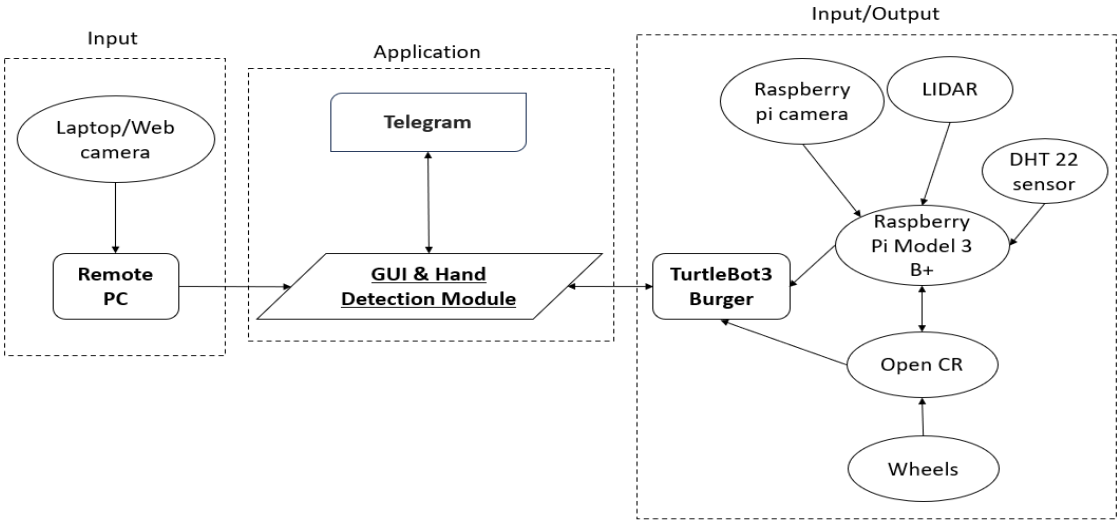


Figure 4-1 System Block Diagram

1. Input Section

   Laptop/Web Camera captures live video, sent to the **Remote PC** for processing.

2. Application Section

   GUI & Hand Detection Module: Detects hand gestures from the video feed and provides a user interface. It can also send control commands or notifications via Telegram.

3. Input/Output Section

   TurtleBot3 Burger: Receives control commands and integrates sensor data.

   Raspberry Pi Model 3 B+: Central processing unit for handling data from sensors like the **Raspberry Pi Camera**, **LIDAR** (for obstacle detection), and **DHT22 Sensor** (for temperature and humidity).

   Open CR: Control board that drives the robot's wheels.

In summary, the Camera captures video, which the Remote PC uses for gesture recognition. Then, the recognized gestures are sent as commands to the TurtleBot3 for action. Thus, the raspberry pi processes sensor data and sends instructions to the Open CR for movement control. The system enables hand gesture-based control of the TurtleBot3 robot, integrating camera input, gesture recognition, and various sensors for interactive operation.

4.2 System Components Specifications

4.2.1 Hardware

The hardware involved in this project is TurtleBot (Burger). A TurtleBot is equipped with appropriate sensors, such as a camera (e.g., Intel RealSense) and a LiDAR sensor.

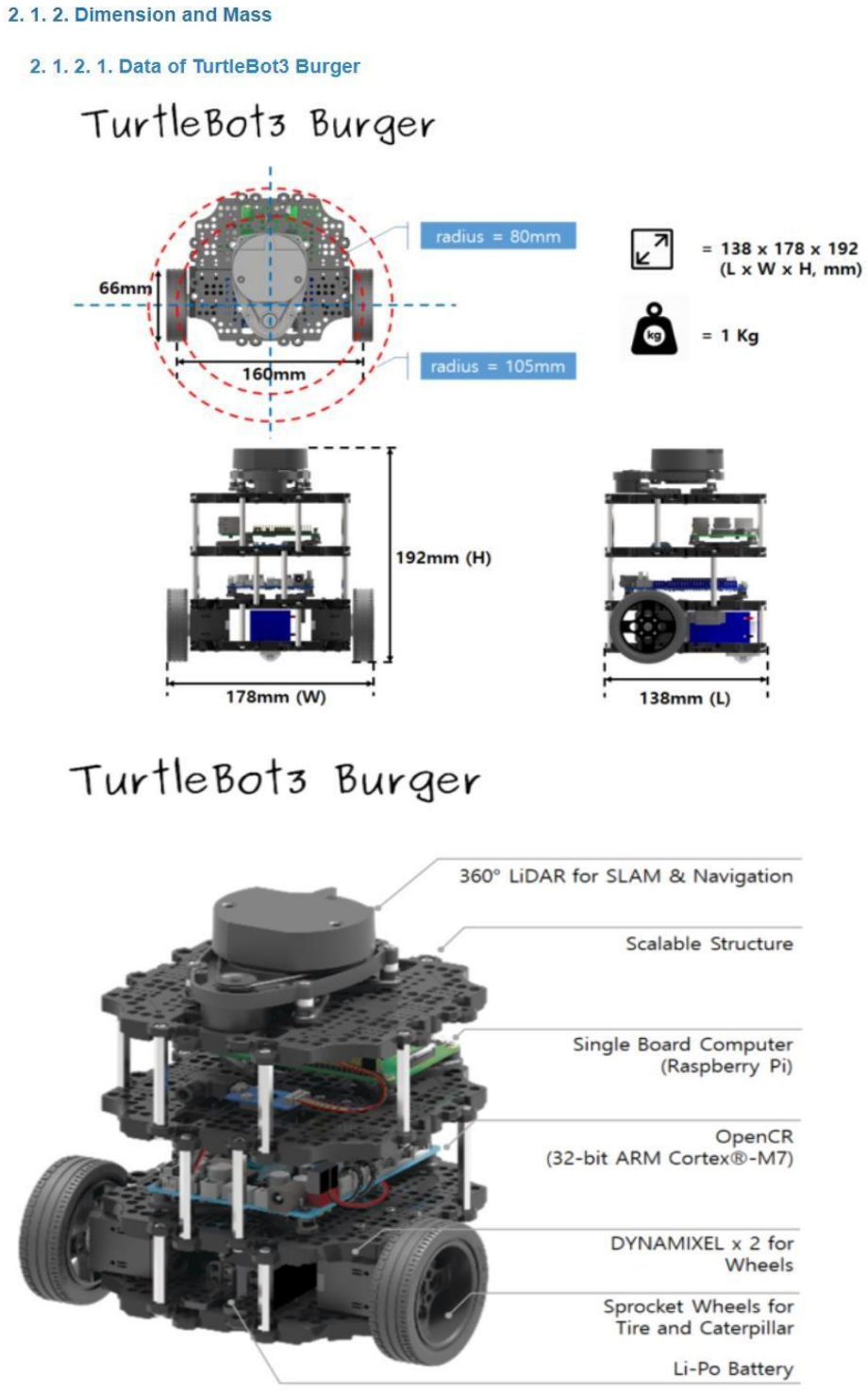


Figure 4-1 TurtleBot3 Burger [9]

Table 4-1 Specifications of TurtleBot3 Burger [9].

Description	Specifications
Maximum translational velocity	0.22 m/s
Maximum rotational velocity	2.84 rad/s (162.72 deg/s)
Maximum payload	15kg
Size (L x W x H)	138mm x 178mm x 192mm
Weight (+ SBC + Battery + Sensors)	1kg
Threshold of climbing	10 mm or lower
Expected operating time	2h 30m
Expected charging time	2h 30m
SBC (Single Board Computers)	Raspberry Pi
MCU	32-bit ARM Cortex®-M7 with FPU (216 MHz, 462 DMIPS)
Remote Controller	-
Actuator	XL430-W250
LDS (Laser Distance Sensor)	360 Laser Distance Sensor LDS-01 or LDS-02
Camera	-
IMU	Gyroscope 3 Axis Accelerometer 3 Axis
Power connectors	3.3V / 800mA 5V / 4A 12V / 1A
Expansion pins	GPIO 18 pins Arduino 32 pin
Peripheral	UART x3, CAN x1, SPI x1, I2C x1, ADC x5, 5pin OLLO x4
DYNAMIXEL ports	RS485 x 3, TTL x 3
Audio	Several programmable beep sequences
Programmable LEDs	User LED x 4

Status LEDs	Board status LED x 1 Arduino LED x 1 Power LED x 1
Buttons and Switches	Push buttons x 2, Reset button x 1, Dip switch x 2
Battery	Lithium polymer 11.1V 1800mAh / 19.98Wh 5C
PC connection	USB
Firmware upgrade	via USB / via JTAG
Power adapter (SMPS)	Input: 100-240V, AC 50/60Hz, 1.5A @max Output: 12V DC, 5A

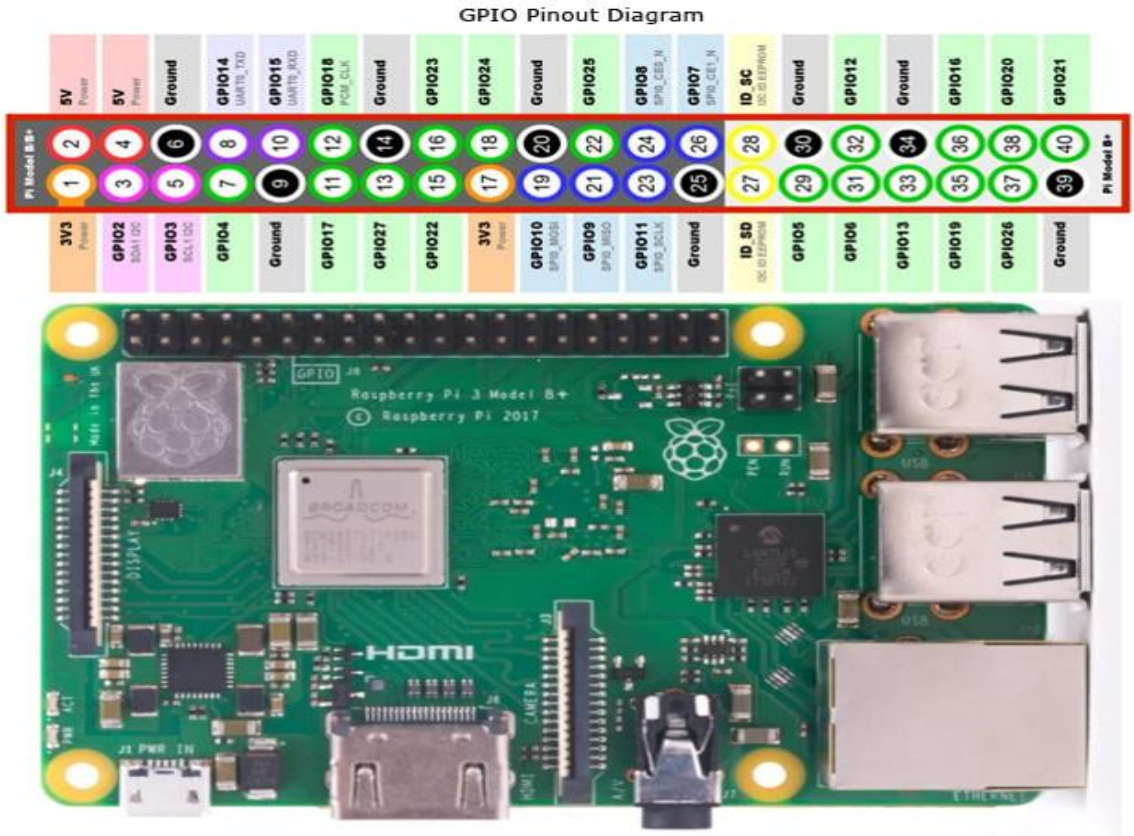


Figure 4-3 Raspberry Pi [10]

The Raspberry Pi single-board computer line is made by the UK nonprofit Raspberry Pi Foundation, whose goals are to increase computer literacy and make computer science education more accessible. Numerous iterations and versions of the Raspberry Pi have been released since its launch in 2012. The first Pi model has a single-core 700MHz CPU and just 256MB RAM, whereas the most recent Pi model had a quad-core CPU clocked at over 1.5GHz and 4GB RAM. The Raspberry Pi Zero is the most economical variant, costing only $5, while the others have traditionally been priced at roughly $100 (typically $35 USD) [10].

Worldwide, people use the Raspberry Pi to learn to programme, build hardware projects, automate their homes, use Edge computing and Kubernetes clusters, and even for industrial purposes. The Raspberry Pi is a reasonably priced Linux computer with a set of GPIO (general purpose input/output) pins that enable control over electronic components for physical computing and exploration of the Internet of Things [10]. Raspberry pi 3 model B+ is used for this project.

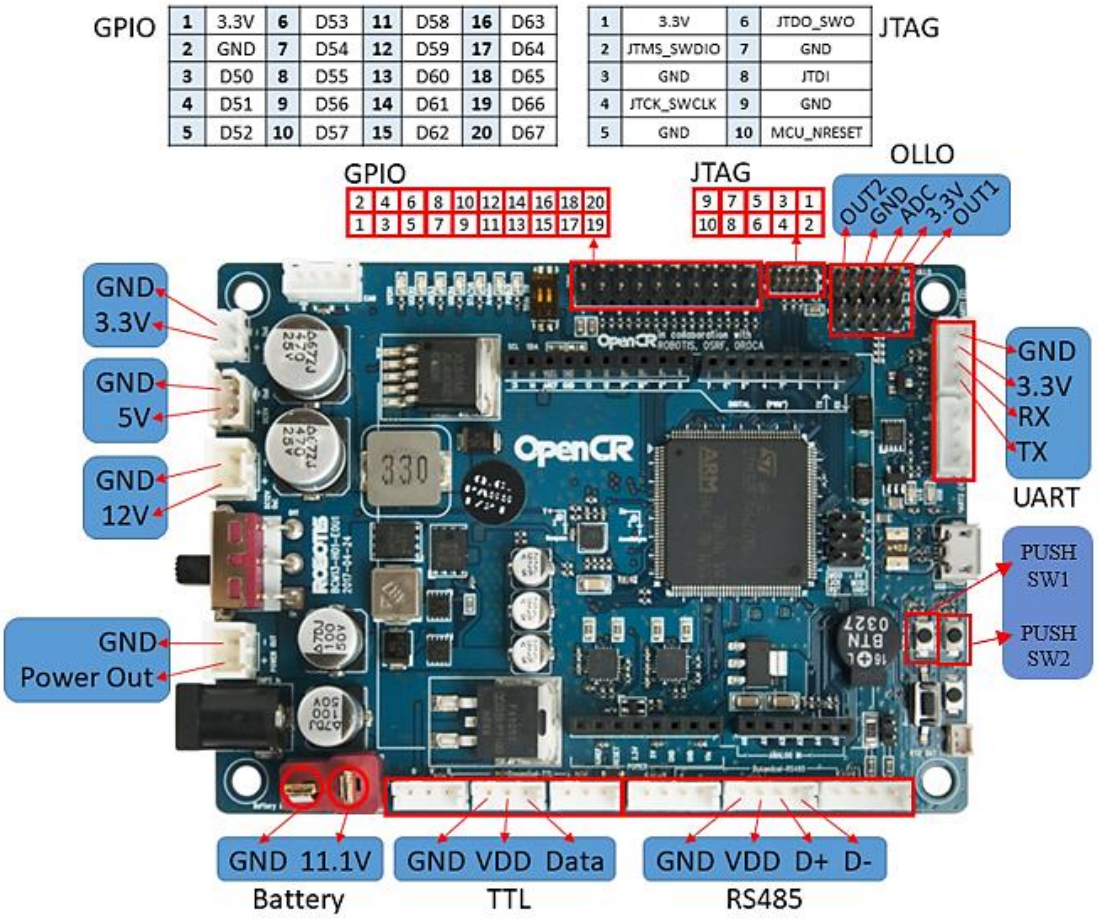


Figure 4-4 OpenCR 1.0 [21]

For ROS embedded devices, OpenCR1.0 is being developed to offer fully open-source hardware and software. The schematics, PCB Gerber, BOM, and firmware source code for the TurtleBot3 and OP3 are all freely distributable to users and the ROS community under open-source licenses. The OpenCR1.0 board's STM32F7 series CPU is built around an ARM Cortex-M7 with a floating-point unit, which is an extremely potent processor. OpenCR1.0's development environment is quite flexible, allowing experts to create traditional firmware or use the Arduino IDE and Scratch for younger pupils [21].

Figure 4-5 Raspberry pi camera with flex cable

This is a high-quality 5-megapixel Raspberry Pi Camera Module, specifically the Jectse OV5647 model, which comes with a flexible cable. To capture images and high-definition video of the surrounding environment, the end of the flex cable should be inserted into the connector marked "CAMERA" on the Raspberry Pi attached to the TurtleBot3 Burger [11]. The specifications of this Raspberry Pi camera module are as follows:

Table 4-2 Specifications of Raspberry pi camera Module Jectse OV5647 [11].

Description	**Specifications**
Name	camera module
Brand	Jectse
Chip	OV5647
Video capture resolution	2592P x 1944P
Screen Size	1 Inches
Lens type	Wide Angle
Angle	175°
Weight	0.02 Kilograms
Applications	Fully compatible with Raspberry Pi B 3/2

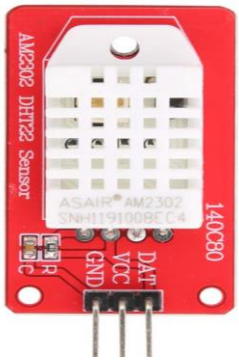


Figure 4-6 DHT22

The DHT22 is an affordable digital sensor designed to measure temperature and humidity. It is connected to GPIO pin 4, 5V pin and Ground pin on the Raspberry Pi, which is mounted on the TurtleBot3 Burger. The sensor uses a thermistor to measure air temperature and a capacitive humidity sensor to determine the surrounding moisture levels, outputting a digital signal on its data pin, which eliminates the need for analog input pins. While the DHT22 is straightforward to use, precise timing is essential for accurate data capture. The specifications of the DHT22 are listed below:

Table 4-3 Specifications of DHT22 [12].

Description	**Specifications**
Model	DHT22
Pins	Output pin (DAT), Power pin (VCC), Ground pin (GND)
Power supply	3.3-6V DC
Output signal	digital signal via single bus
Sensing element	Polymer capacitor
Operating range	humidity 0-100%RH; temperature -40~80Celsius
Accuracy	humidity +-2%RH (Max +-5%RH); temperature <+-0.5Celsius
Resolution or sensitivity	humidity 0.1%RH; temperature 0.1Celsius
Repeatability	humidity +-1%RH; temperature +-0.2Celsius
Humidity hysteresis	+-0.3%RH
Long-term Stability	+-0.5%RH/year
Sensing period	Average: 2s
Interchangeability	fully interchangeable
Dimensions	small size 14*18*5.5mm; big size 22*28*5mm

4.2.2 Firmware

The software involved in this project is **Ubuntu** and Robot Operating System 2 **(ROS2)**. Ubuntu is a popular free, open-source Linux-based operating system that can use on computer or virtual private server. While a collection of software libraries and tools called the Robot Operating System (ROS) are used to create robot applications [13].

Figure 4-7 Ubuntu [11]

In 2004, the British company Canonical unveiled Ubuntu. It was built on Debian, a popular distribution at the time that required a lot of installation work. Ubuntu was consequently suggested as a user-friendly substitute. The advantages of using Ubuntu are its user-friendliness, strong security, more software options, enhanced privacy, lightweight performance and free of charge. The lists of Ubuntu library packages [14]:

- focal (20.04LTS)
- focal-updates
- focal-backports
- jammy (22.04LTS)
- jammy-updates
- jammy-backports
- kinetic (22.10)
- kinetic-updates
- kinetic-backports
- lunar (23.04)
- lunar-updates
- lunar-backports
- mantic

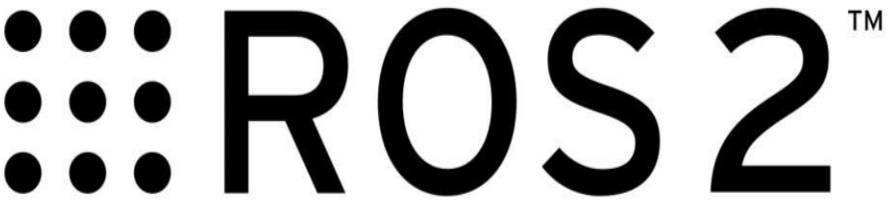


Figure 4-8 ROS 2 [12]

ROS contains the open-source resources needed for upcoming robotics project, including drivers, cutting-edge algorithms, and robust development tools. The robotics and ROS communities have undergone significant development since the founding of ROS in 2007. By utilizing what is excellent about ROS 1 and enhancing what isn't, the ROS 2 project hopes to adjust to these developments [15]. There are two supported client libraries which are the C++ client library (rclcpp) and the Python client library (rclpy). Both client libraries utilize common functionality in rcl [16]. Several ROS2 topics are published or subscribed to control the robot system.

4.2.3 Software/API

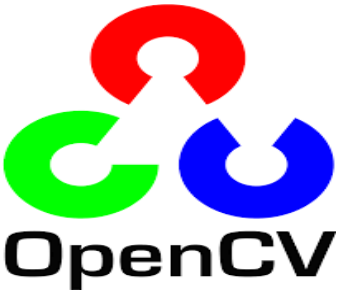


Figure 4-9 Open CV [22]

OpenCV is a comprehensive open-source library for computer vision, machine learning, and image processing. It has become a significant part in real-time operation, which is critical in modern systems. It allows to process photos and movies to recognize items, faces, and even human handwriting.

Python, when combined with other libraries such as NumPy, can parse the OpenCV array structure for analysis purposes. Vector space and apply mathematical operations were applied to identify an image pattern and its many aspects with the help of Python. OpenCV offers users to do numerous operations on an image. Let take an example of the image below:

Figure 4-10 Example of the image to do numerous operations [22]

Descriptions below are the operation done by OpenCV.

1. Read the Image: OpenCV allows users to retrieve images from a file or straight from a camera, making them available for further processing.
2. Image Enhancement: Users can enhance an image by altering its brightness, sharpness, or contraction. This helps to visualize the image's quality.
3. Object detection: As shown in the image above, objects may also be detected using OpenCV. Bracelet, watch, patterns, and faces can be identified. This can include recognizing people, shapes, or even objects.
4. Image Filtering: Users can adjust the image by using filters like blurring or sharpening.

   Draw the Image: OpenCV lets users to draw text, lines, and other forms in images.
5. Saving Changed photos: After processing, users can store the photos that have been edited for future study [19].

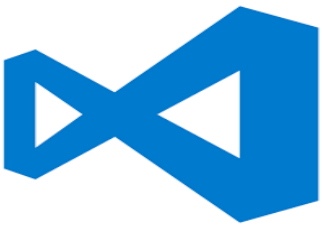


Figure 4-11 Visual Studio Code [23]

Visual Studio Code (also known as VS Code) is a free open-source text editor developed by Microsoft. VS Code is accessible for Windows, Linux, and Mac. Although the editor is very portable, it contains some impressive features that have helped VS Code become one of the most popular software development tools in recent years.

VS Code offers a wide range of programming languages, including Java, C++, Python, CSS, Go, and Dockerfiles. Furthermore, VS Code lets users to add and even create additional extensions, such as code linters, debuggers, and cloud and web development capabilities.

VS Code's layout enables a lot of customization than other text editors. To enhance the user experience, VS Code is organized into five major regions which are Activity Bar, Side Bar, Editor's groups, Panel and Status Bar as shown in figure below [23].

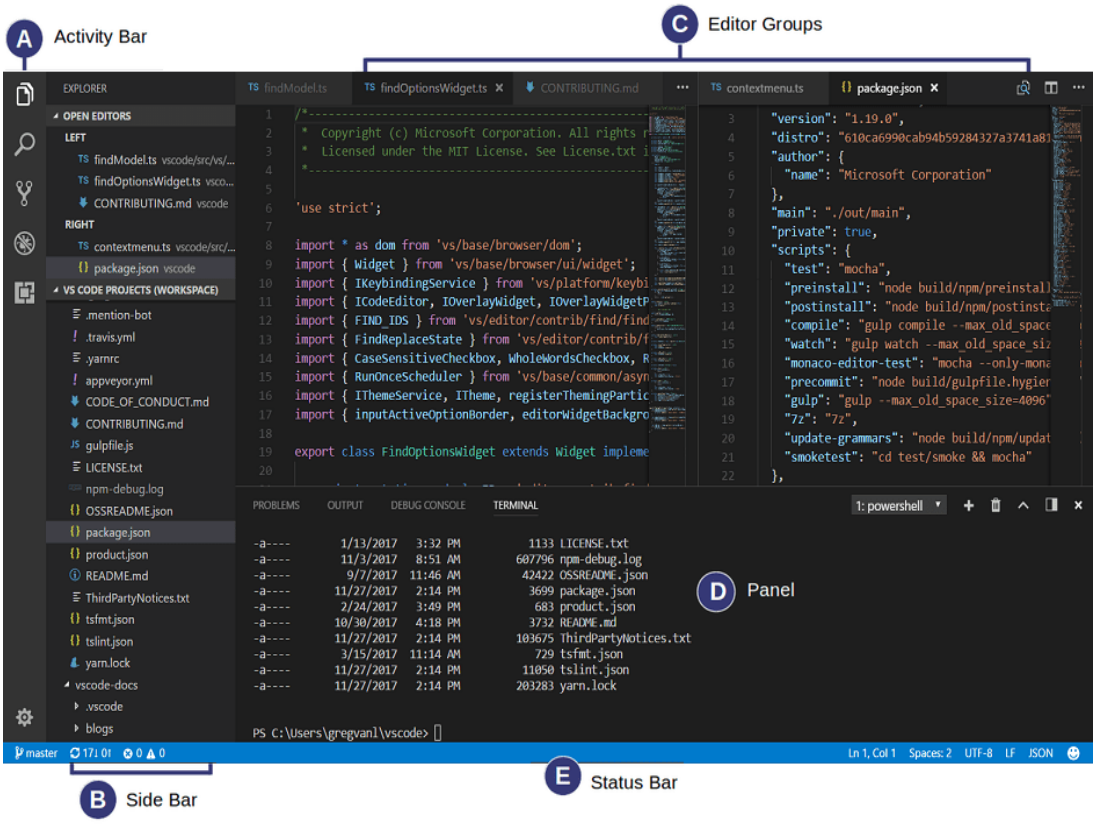


Figure 4-12 User interface of Visual Studio Code [23]

Figure 4-13 Hand Tracking Module

The hand tracking module allows for real-time detection and tracking of hands. OpenCV will be used to develop or import a hand tracking module capable of detecting and tracking landmarks of up to two hands within the camera's view. To install the module, run the command “pip install HandTrackingModule” in the terminal. In Visual Studio Code, the module can be imported using the Python statement “from cvzone.HandTrackingModule import HandDetector”.

Figure 4-14 Simple Commander API [24]

The Nav2 Simple Commander provides Python-based ways for interacting with the Navigation 2 system. The Nav2 Simple (Python3) Commander aims to give Python3 users with "navigation as a library" capabilities. Nav2 Simple Commander provide an API that handles all ROS 2 and Action Server chores for users, allowing users to focus on developing an application that takes advantage of Nav2's features.

A simple illustration of the main structure is provided below. Note that goToPose(), goThroughPoses(), followWaypoints(), and related methods are non-blocking, allowing users to obtain and handle feedback in a single threaded program. As a result, while waiting for an assignment to be completed, the while not nav.isTaskComplete() design is required to poll for adjustments to navigating completion and, if necessary, carry out certain duties of interest to user's application (such as analysing feedback, performing something with the data collected by the robot, or verifying for errors) [24].

```
from nav2_simple_commander.robot_navigator import BasicNavigator
import rclpy

rclpy.init()
nav = BasicNavigator()

# ...

nav.setInitialPose(init_pose)
nav.waitUntilNav2Active() # if autostarted, else use lifecycleStartup()

# ...

path = nav.getPath(init_pose, goal_pose)
smoothed_path = nav.smoothPath(path)

# ...

nav.goToPose(goal_pose)
while not nav.isTaskComplete():
  feedback = nav.getFeedback()
  if feedback.navigation_duration > 600:
    nav.cancelTask()

# ...

result = nav.getResult()
if result == TaskResult.SUCCEEDED:
    print('Goal succeeded!')
elif result == TaskResult.CANCELED:
    print('Goal was canceled!')
elif result == TaskResult.FAILED:
    print('Goal failed!')
```

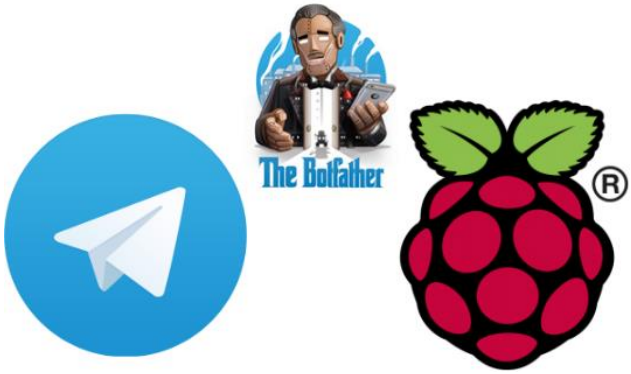


Figure 4-15 Telegram Bot API [26]

Telegram released the Bot API on June 24, 2015, allowing robots to communicate with Telegram. This innovation allowed not only humans but also machines to communicate via Telegram. Telegram bots can be configured for IoT applications, such as sending messages to a Raspberry Pi or receiving data from a Raspberry Pi back to Telegram [26].

4.3 Circuits and Components Design

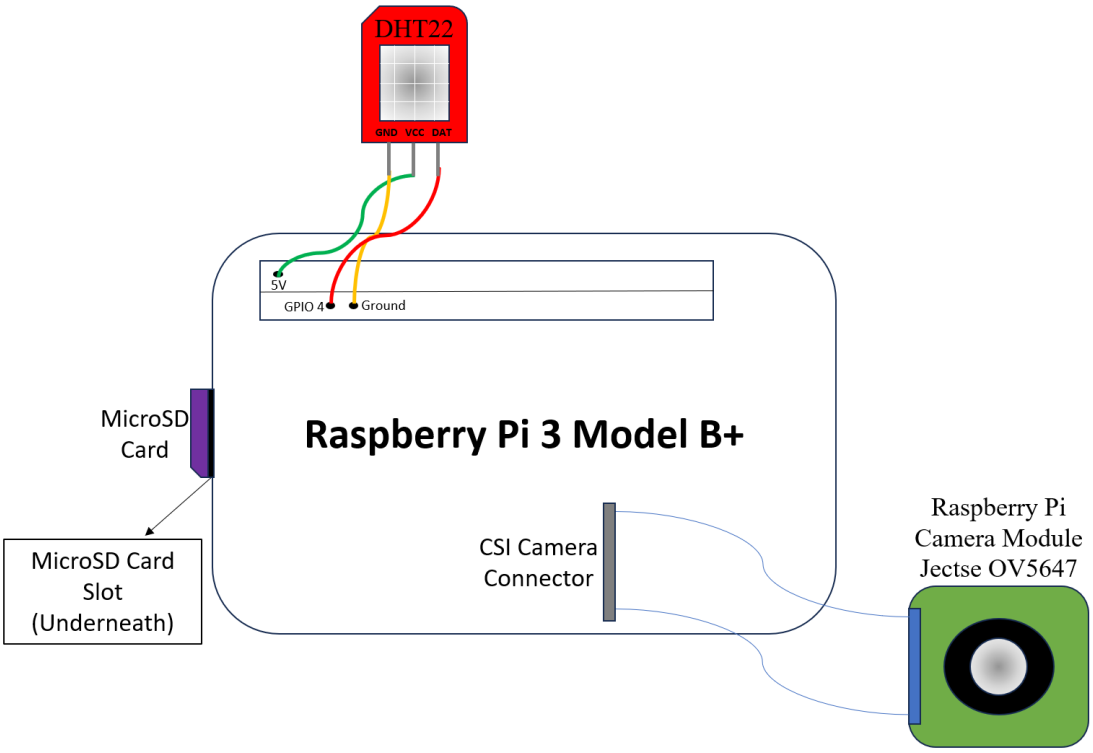


Figure 4-16 Block Diagram of Circuits and Components Design

The block diagram above illustrates the circuit and component design for this project. The central component is the Raspberry Pi 3 Model B+ attached to the TurtleBor3, to which three subcomponents are connected: the DHT22 sensor, the Raspberry Pi camera, and the microSD card. Together, these elements form the overall circuit design.

4.4 System Components Interaction Operations (Flowcharts)

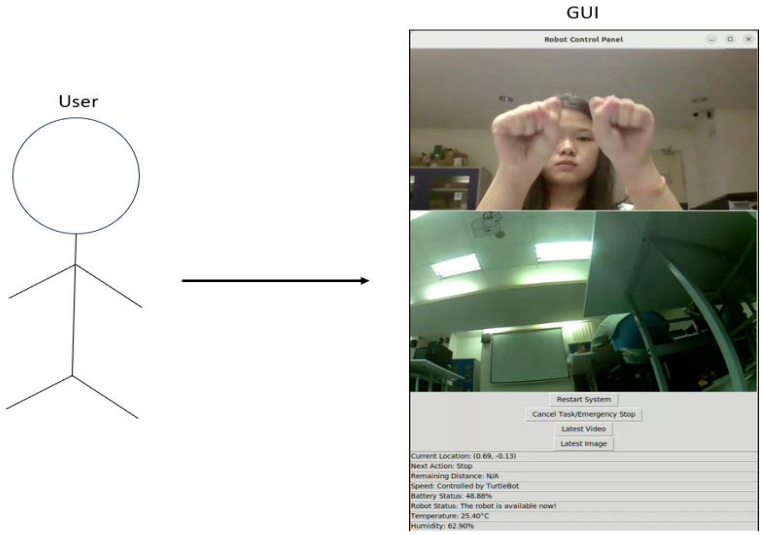


Figure 4-17 System Components Interaction Operations

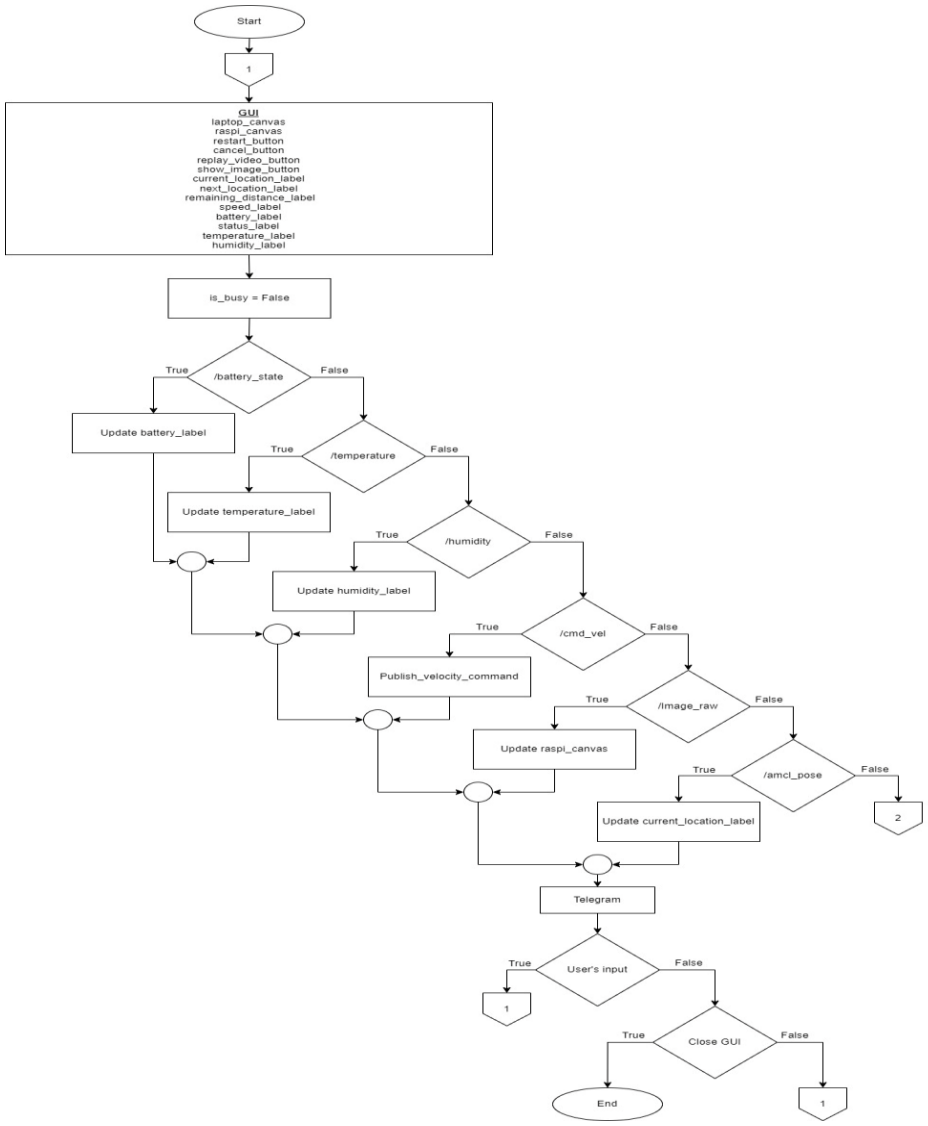


Figure 4-18 Flowchart1 for System Components Interaction Operations

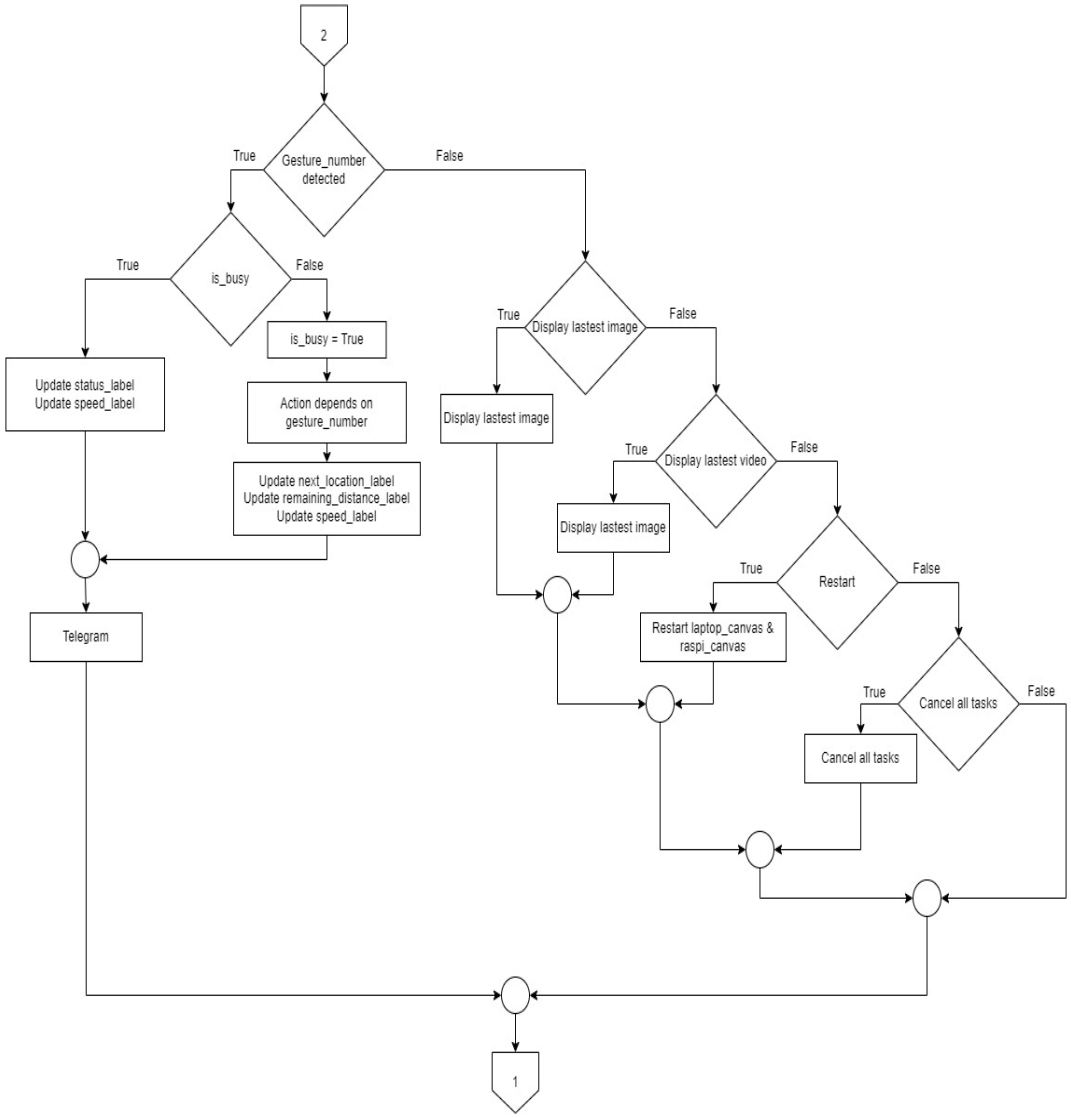


Figure 4-19 Flowchart2 for System Components Interaction Operations

The figure 4-17 shows the system components interaction operations between user and GUI of the project system. The flowcharts outline a gesture recognition system that monitors and controls a TurtleBot3 robot. It starts by initializing the GUI components and monitoring the system's status (battery, temperature, and humidity). The system continuously detects gestures and based on recognition results, updates the GUI then sends commands to the robot, or sends notifications to/via Telegram. The hand gesture numbers, and their corresponding actions can be viewed in the results section of Chapter 6. It also manages user interactions, such as displaying images or videos and handling input to restart or cancel tasks. The process concludes by closing the program when the user decides to exit. Overall, the system integrates real-time gesture detection, robot control, and user interaction.

Chapter 5

SYSTEM IMPLEMENTATION

The System Implementation conducted in this project aimed to lay the groundwork for the implementation of the proposed method outlined in Chapter 3. This involved setting up the necessary software and conducting initial experiments to access the feasibility of the approach. The preliminary work can be divided into two main sections: software setup and visualization/segmentation.

5.1 Setting up

5.1.1 Software

Before starting to develop this project, there are two OS needed to be installed and downloaded into PC and SD card for TurtleBot respectively:

1. Ubuntu 22.04 LTS Desktop (64-bit)
2. Ubuntu Server 22.04.5 LTS (64-bit)

Before delving into the development process, it was essential to set up the required software tools. Visual Studio Code was identified as the primary Integrated Development Environment (IDE) for coding and project management. Additionally, other software dependencies, such as TurtleBot3 via Debian Packages, libraries, nav 2 simple commander API and frameworks, were installed to support the development environment. Some dependent ROS 2 Packages installed such as Gazebo, Cartographer, Navigation 2 for mapping and navigation used. This setup process ensured that the project had a stable and efficient coding environment to facilitate smooth development.

5.1.2 Connection between PC and TurtleBot 3

The connection between PC and TurtleBot 3 is via the same Wi-Fi by edit the WIFI_SSID and WIFI_PASSWORD in the network configuration file. Then get the IP address of WLAN0 of the TurtleBot. With that, work can be done from the Remote PC using SSH. This SSH is configured with the Cmd_Vel (PC terminal). After that, Cmd_Vel will turn into Ubuntu terminal and can continue work for installation inside TurtleBot3. Besides, ROS Domain ID

Setting in ROS2 DDS communication, ROS_DOMAIN_ID must be matched between Remote PC and TurtleBot3 for communication under the same network environment. A default ID of TurtleBot3 is 30 [25].

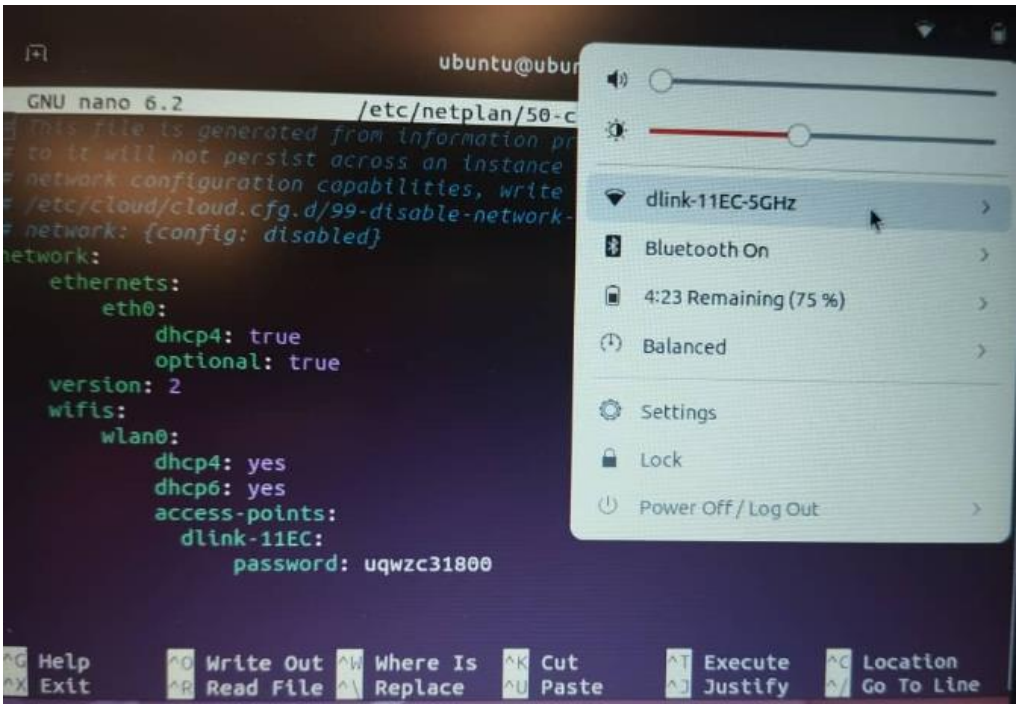


Figure 5-1 Connection between PC and TurtleBot 3 is via the same Wi-Fi

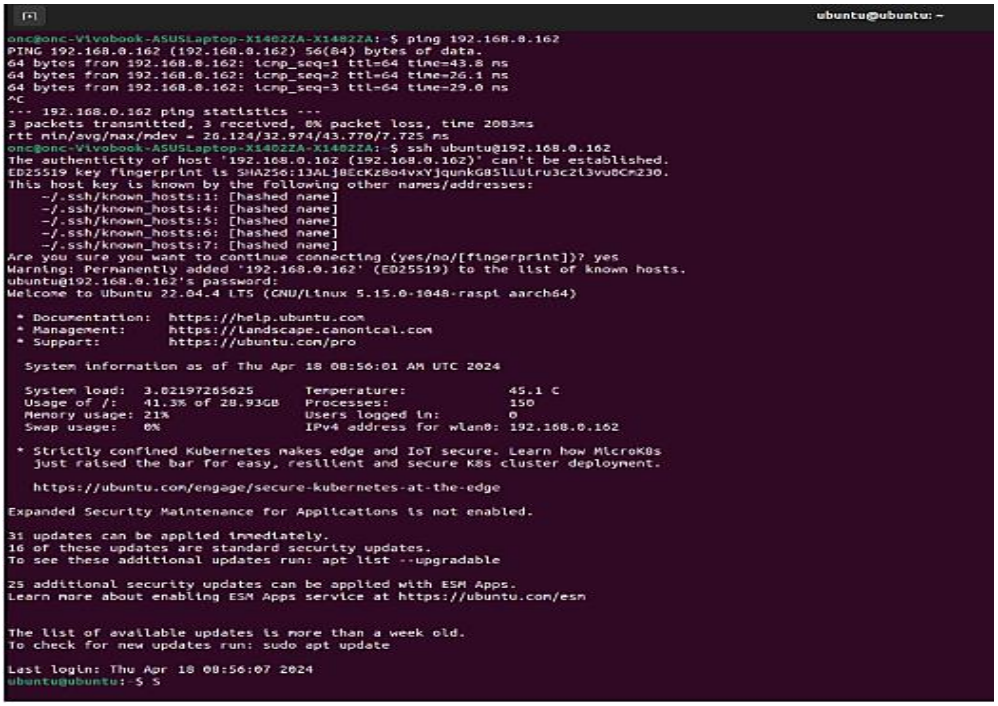


Figure 5-2 SSH is configured with the Cmd_Vel

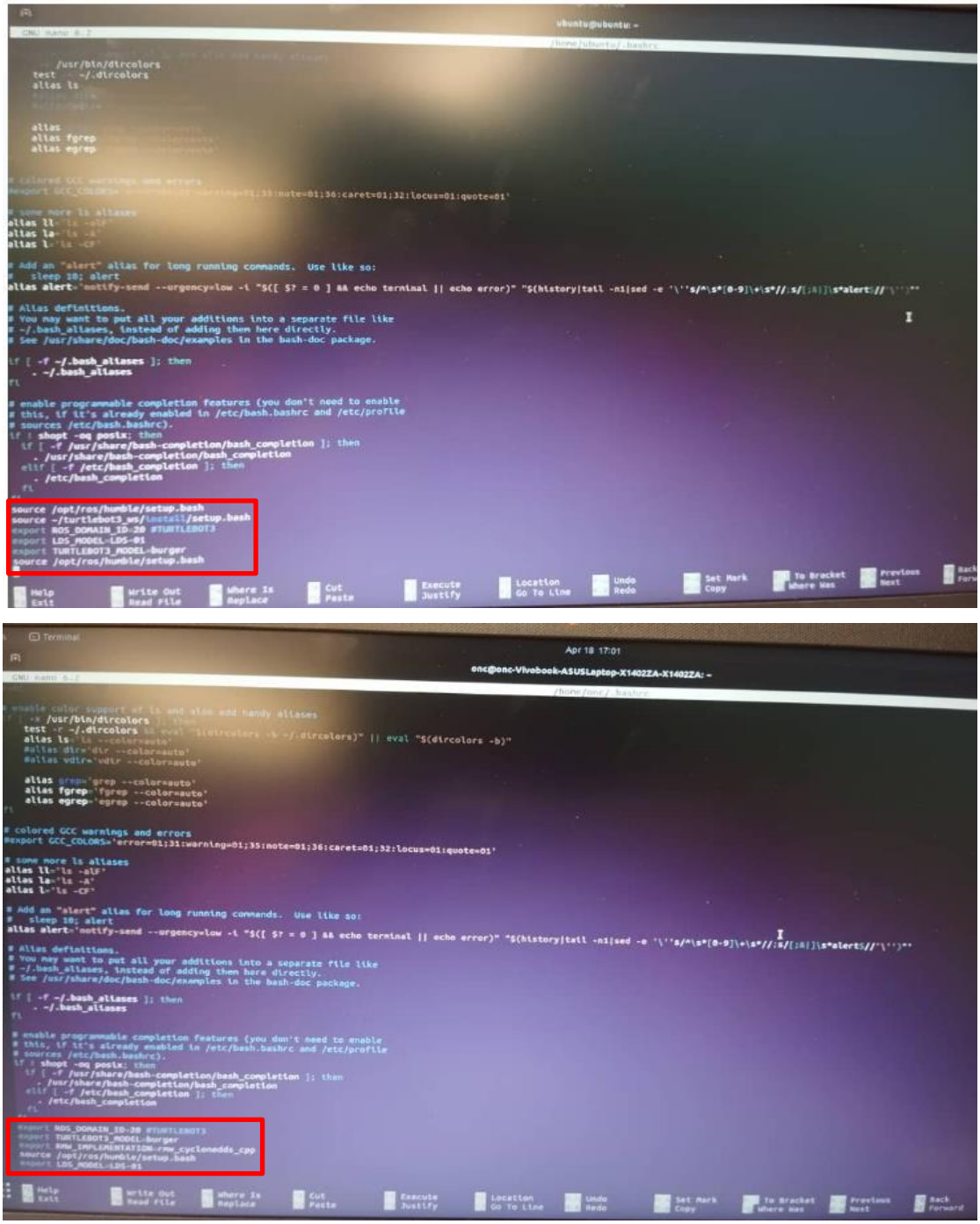


Figure 5-3 ROS Domain ID matched between Remote PC and TurtleBot3

5.1.3 Hardware Configuration /Firmware Setup

1. LDS Configuration

   Add LDS model inside the '~/.bashrc' file. With the commands below:

```
$ echo 'export LDS_MODEL=LDS-02' >> ~/.bashrc
$ source ~/.bashrc
```

   The LDS model for this project is LDS-01.

2. OpenCR Setup

   Connect the OpenCR to the Raspberri Pi using micro-USB cable. Install required Packages on Raspberry Pi to upload the OpenCR firmware. Add the OPENCR_MODEL which is 'Burger'. Download the fiemware and loader then upload it to the OpenCR [25]. A successful firmware upload for TurtleBot3 Burger will look like below:

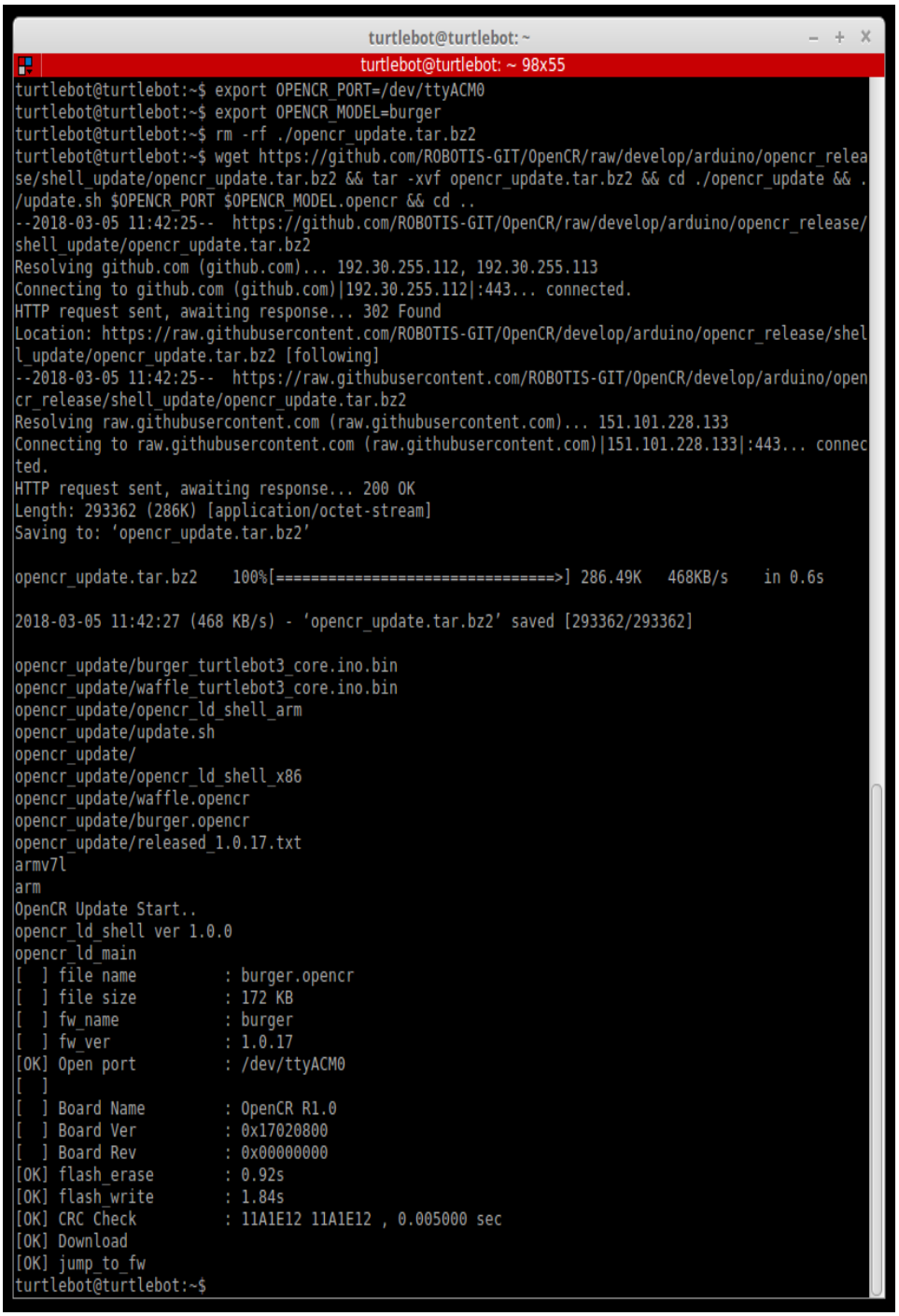


Figure 5-4 Output when successful firmware upload

5.2 Bring Up & Teleoperation

1. Bring Up TurtleBot3

   Connect to Raspberry Pi with its IP address using terminal from PC. Bring up basic packages to start TurtleBot3 applications with correct parameter which is 'Burger' [25]. The terminal output will look like below:

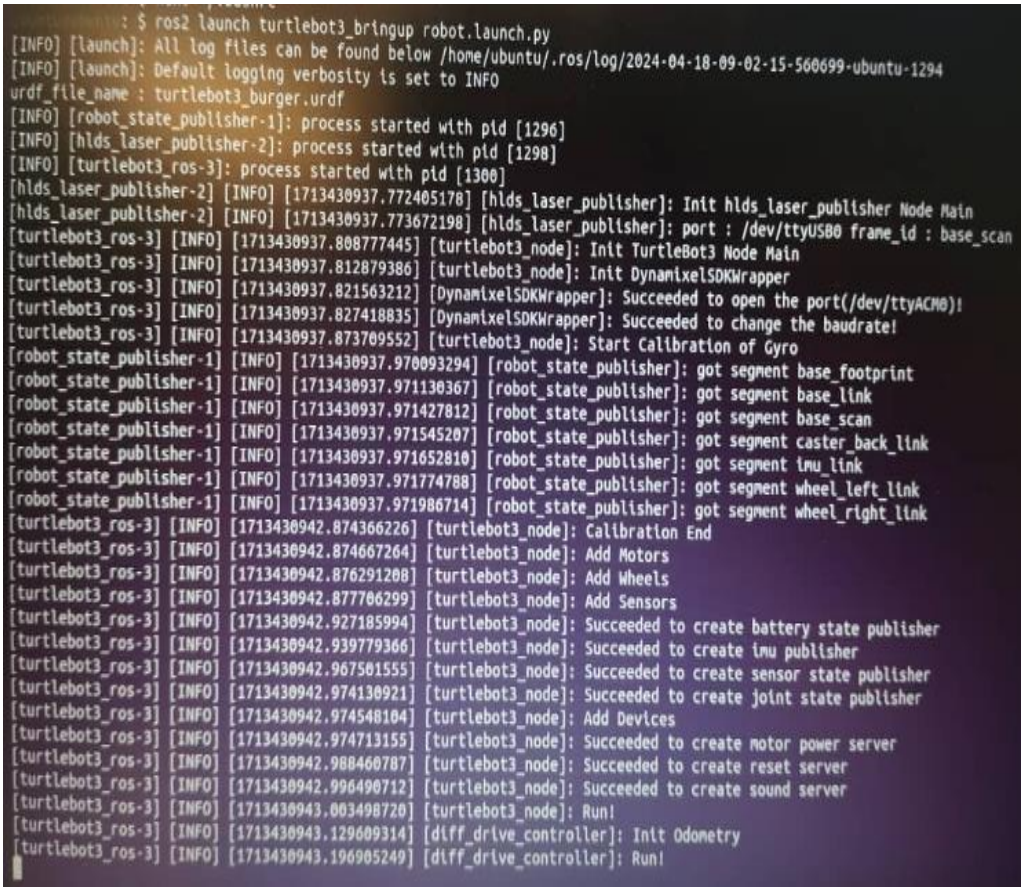


Figure 5-5 Output for Bring up command

Topic list with commands below:

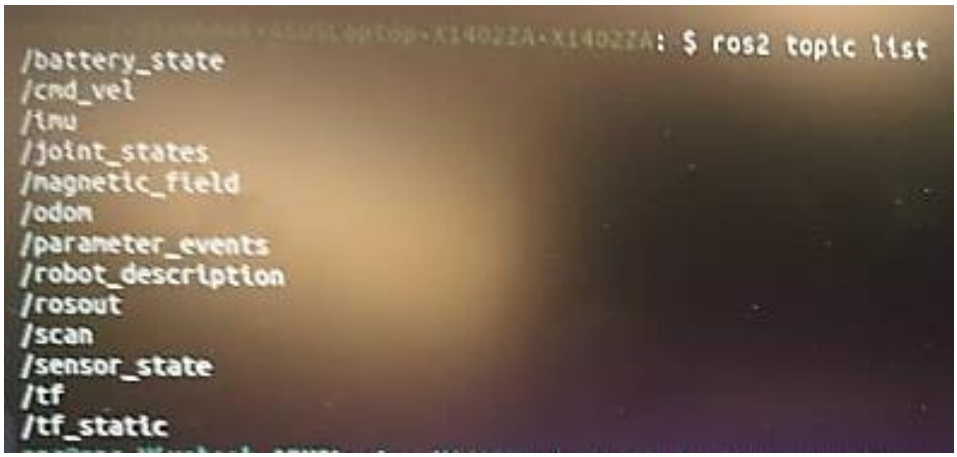


Figure 5-6 Output for ROS2 topic

Service list with commands below:

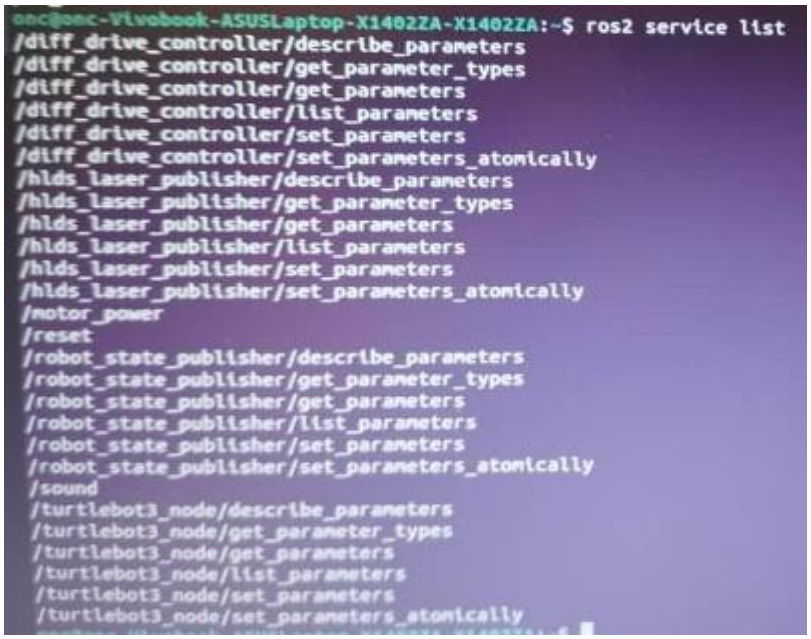


Figure 5-7 Output for service list

1. Teleoperation

   Connect to Raspberry Pi with its IP address using terminal from PC [25]. Run teleoperation node with command below:

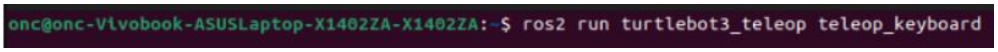


Figure 5-8 Output for Teleoperation command

If node successfully launched, the instruction appeared to the terminal window will be as below:

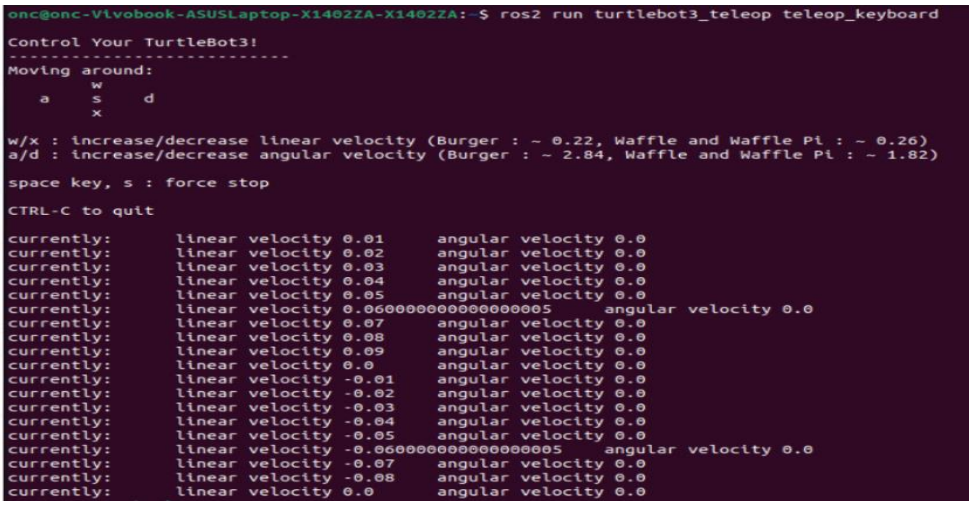


Figure 5-9 Output for Teleoperation when node successfully launched

Thus, the TurtleBot3 can now be controlled using keyboard of PC.

5.3 SLAM & Navigation

5.3.1 Run SLAM node

Firstly, run the bring up with command. Then, open a new terminal from Remote PC and launch the SLAM node using Cartographer as default SLAM method as command below:

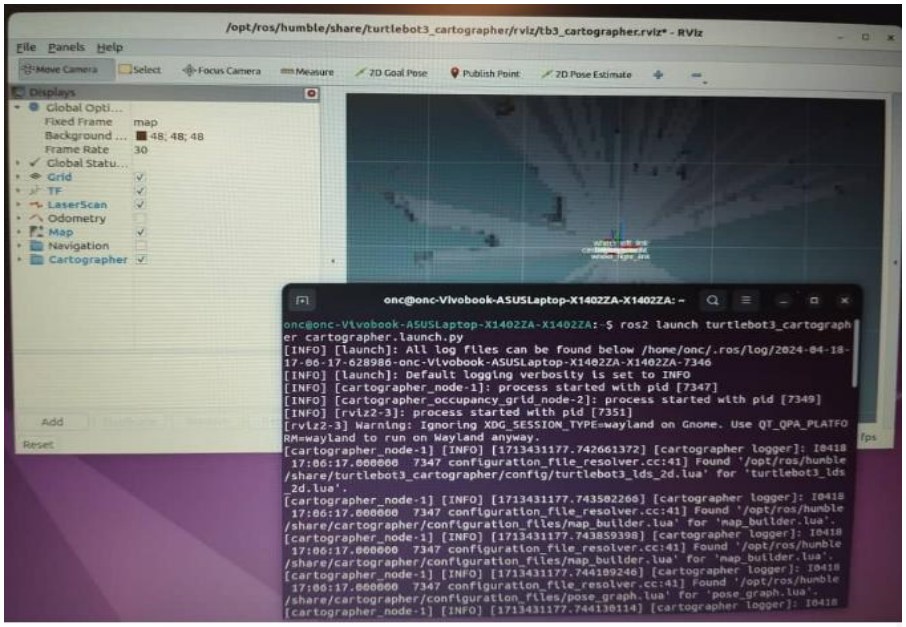


Figure 5-10 Cartographer

Next, run teleoperation node so that the TurtleBot3 can be controlled by keyboard to start exploring and drawing the map [22].

Figure 5-11 Map of FYP lab

Lastly, the map drawn need to be saved using the command below:

```
$ ros2 run nav2_map_server map_saver_cli -f ~/map
```

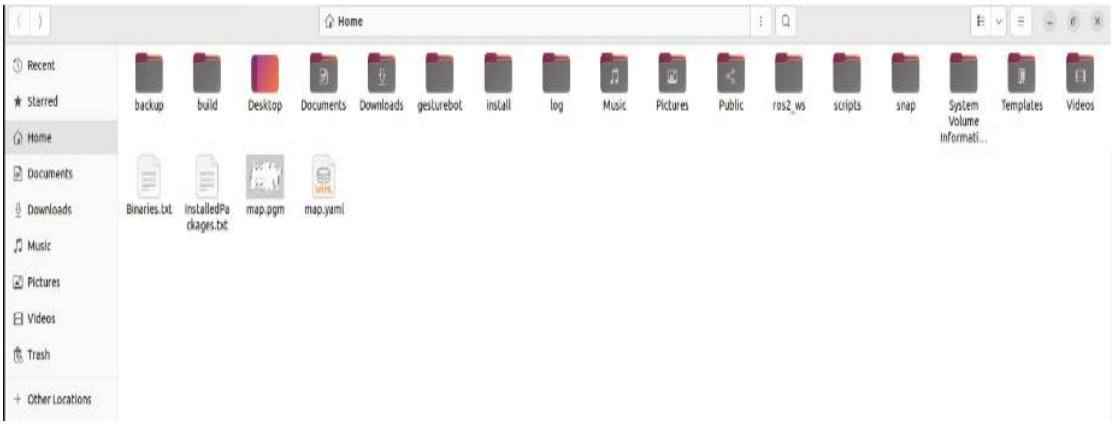


Figure 5-12 Directory of saved map

5.3.2 Navigation using ROS2 tools

Firstly, run the bring up with command. Then, open a new terminal from Remote PC and launch the Navigation node. ROS2 uses Navigation2. Initial Pose Estimation must be performed by clicking the '2D Pose Estimate' button in the RVIZ2 menu before running the Navigation as this process initializes the AMCL parameters that are critical in Navigation. TurtleBot3 must be correctly located on the map with LDS sensor data that neatly overlaps the displayed map [22].

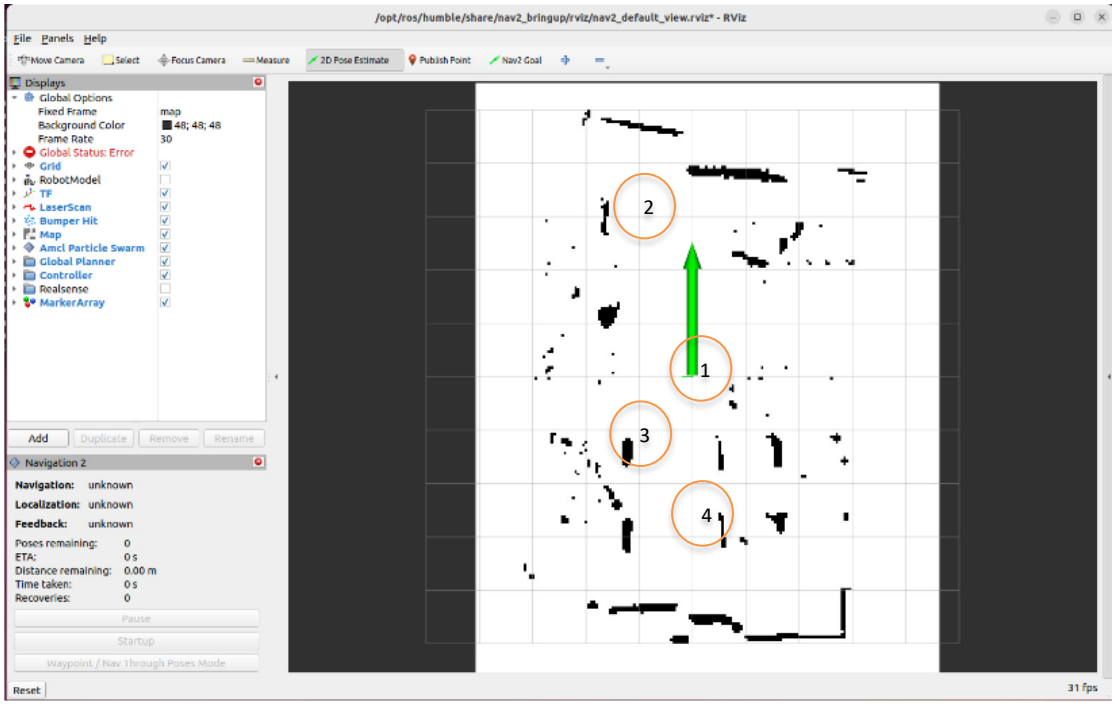


Figure 5-13 Initial Pose Estimation

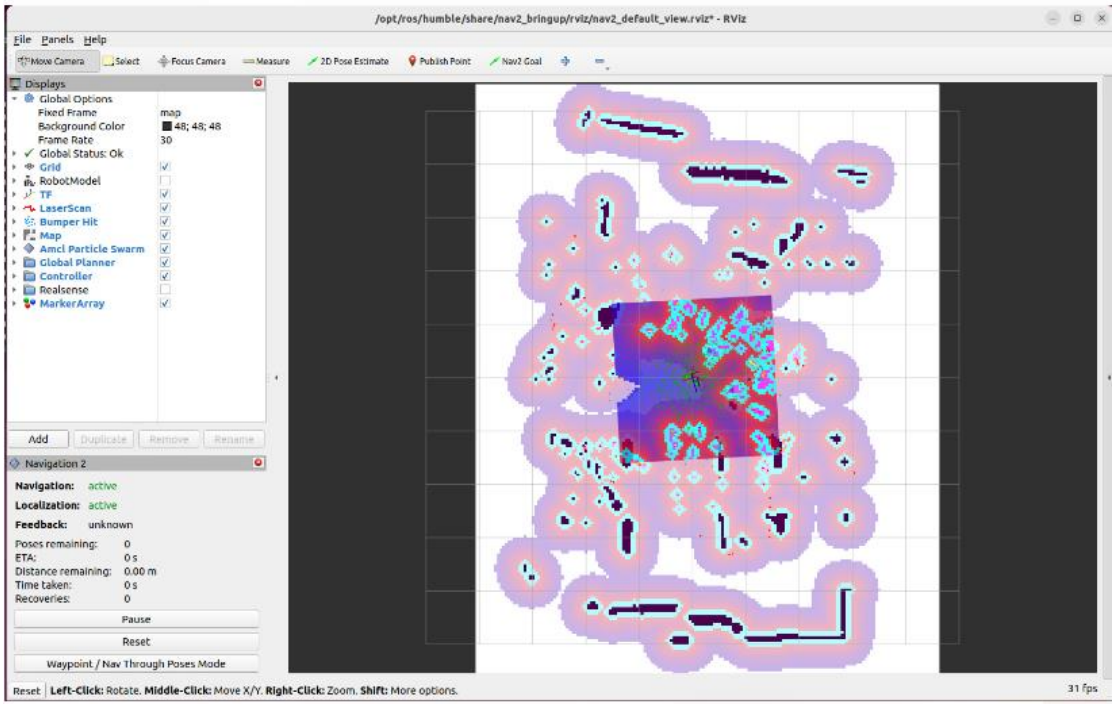


Figure 5-14 LDS sensor data

Launch keyboard teleoperation node to precisely locate the robot on the map. Lastly, click the 'Navigation2 Goal' button in the RVIZ2 menu to set Navigation Goal. Click on the map to set the destination of the robot and drag the green arrow toward the direction where the robot will be facing [22].

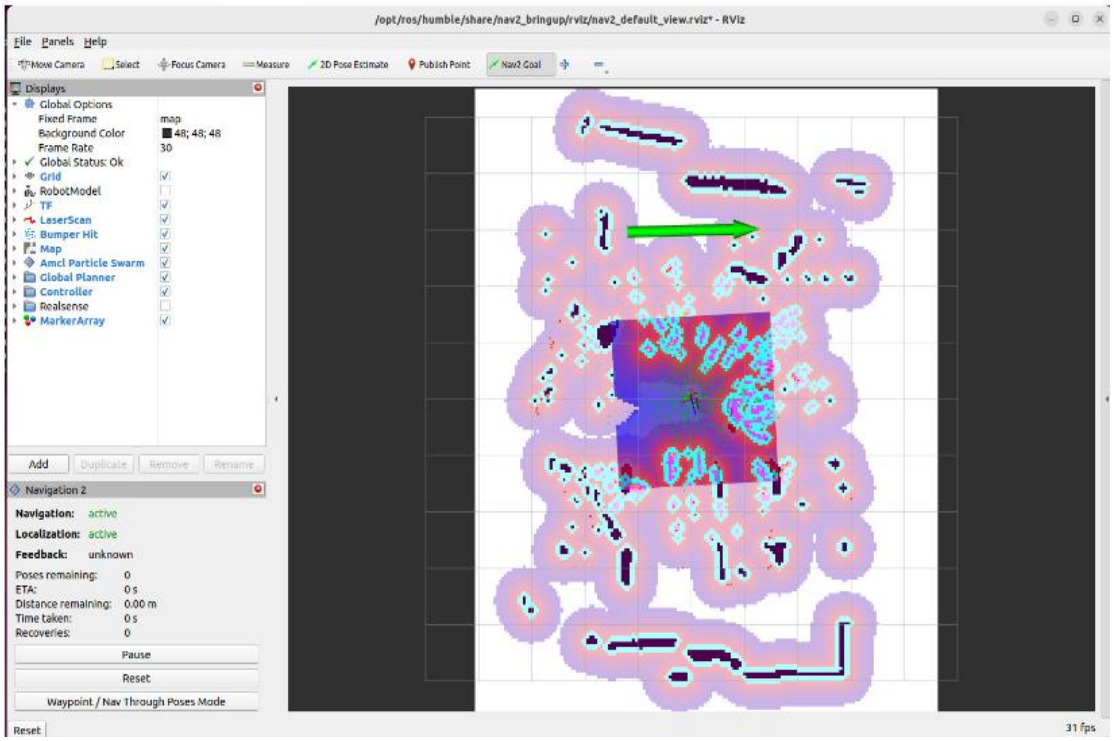


Figure 5-15 Navigation2 Goal

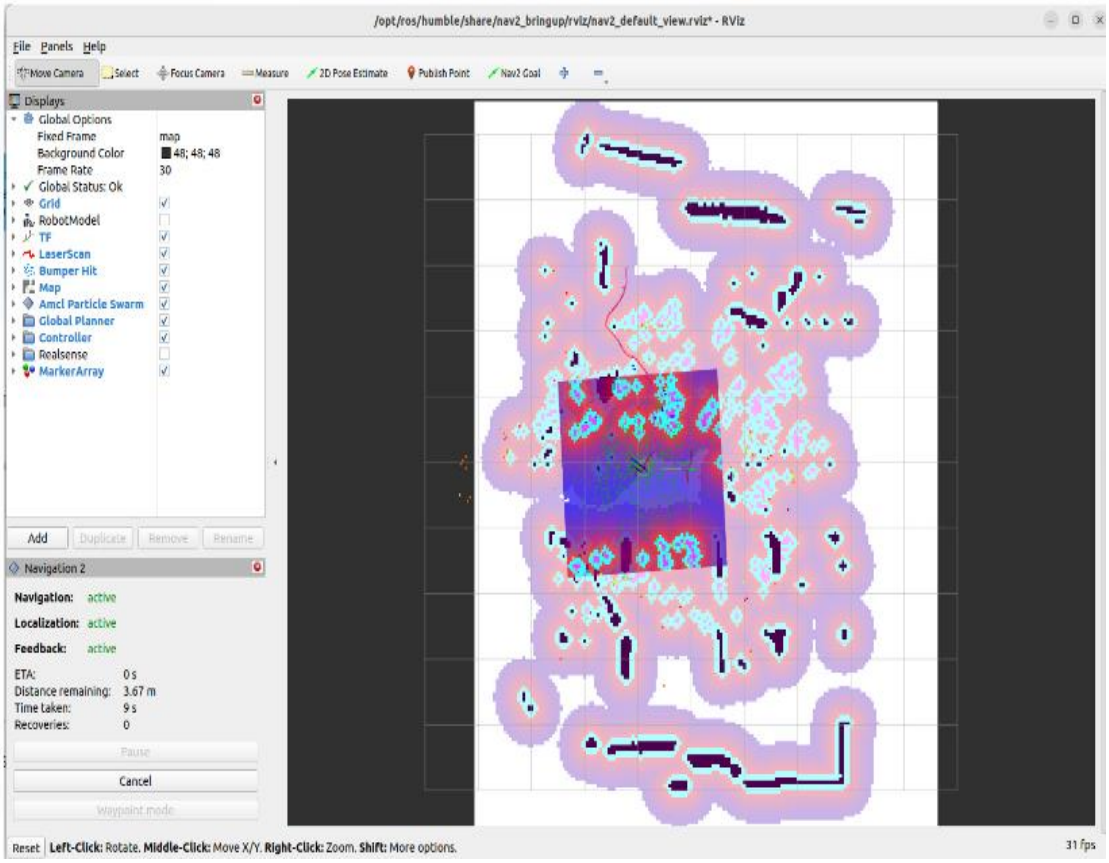


Figure 5-16 Navigation 2

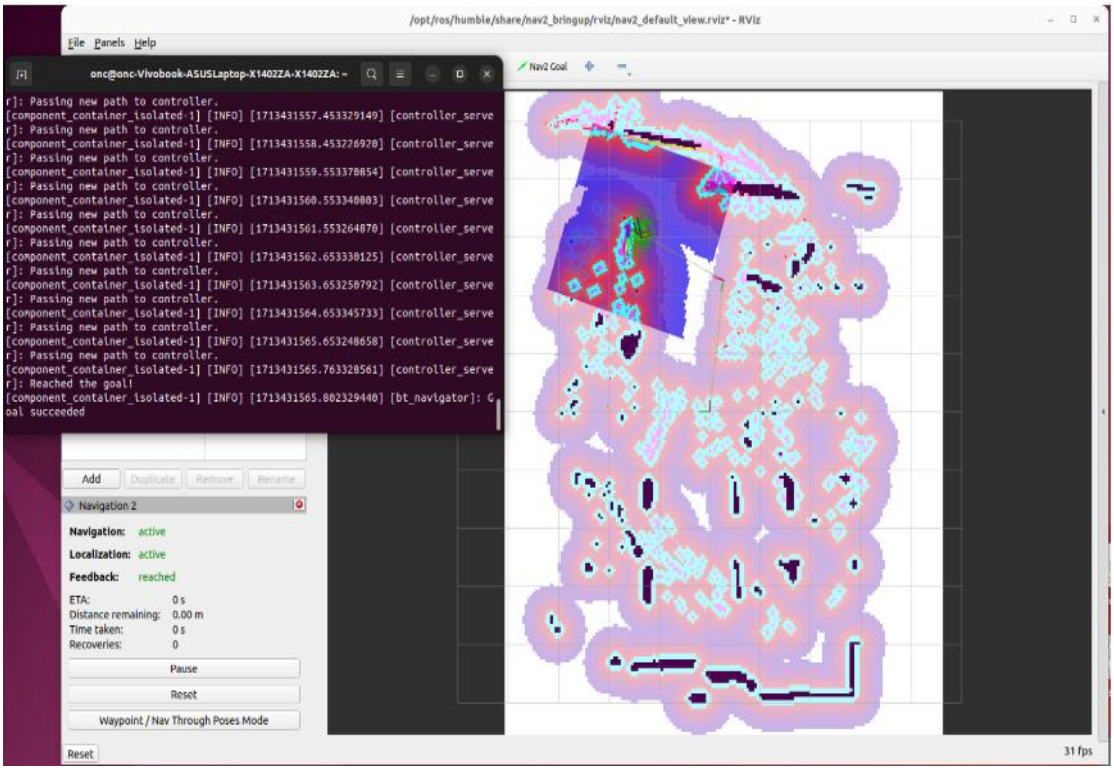


Figure 5-17 Navigation Goal

5.3.2 Navigation using Python with Visual Studio Code

Firstly, run the bring up with command. Then, open a new terminal from Remote PC and launch the Navigation node. ROS2 uses Navigation2. Next, open a new terminal again to Navigate to my_robot_controlled package with Nav2 Simple Commander API. Install the node that make it run in “ros 2 run” Modify the “setup.py” then navigate to ros_ws folder to build the source code and source new built package [21]. Lastly, run the executable as the command below:

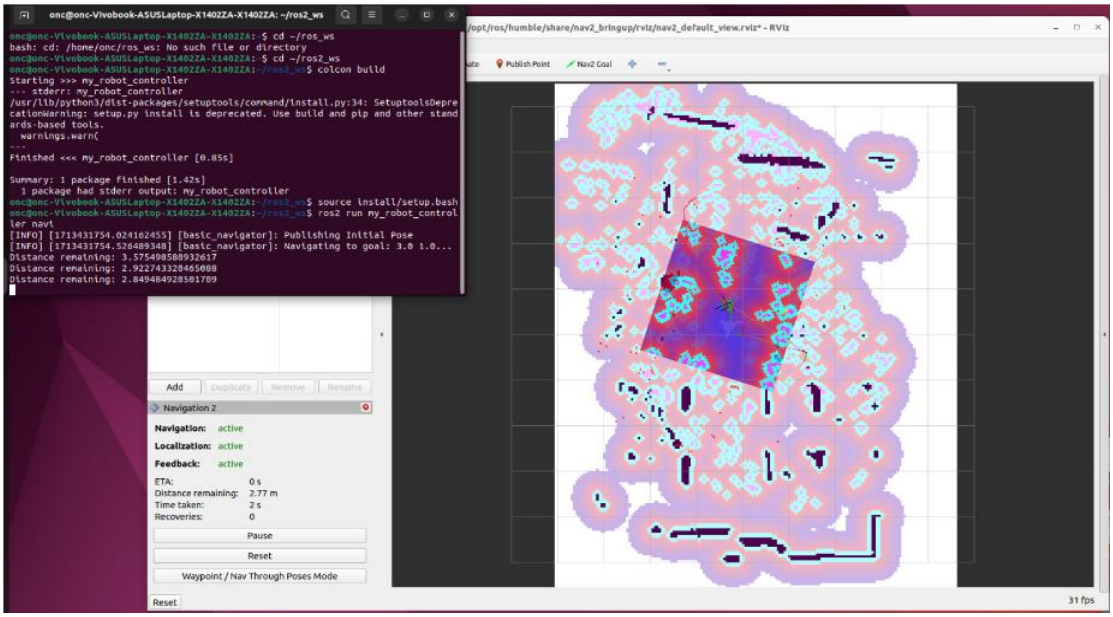


Figure 5-18 Command for Navigation using Python

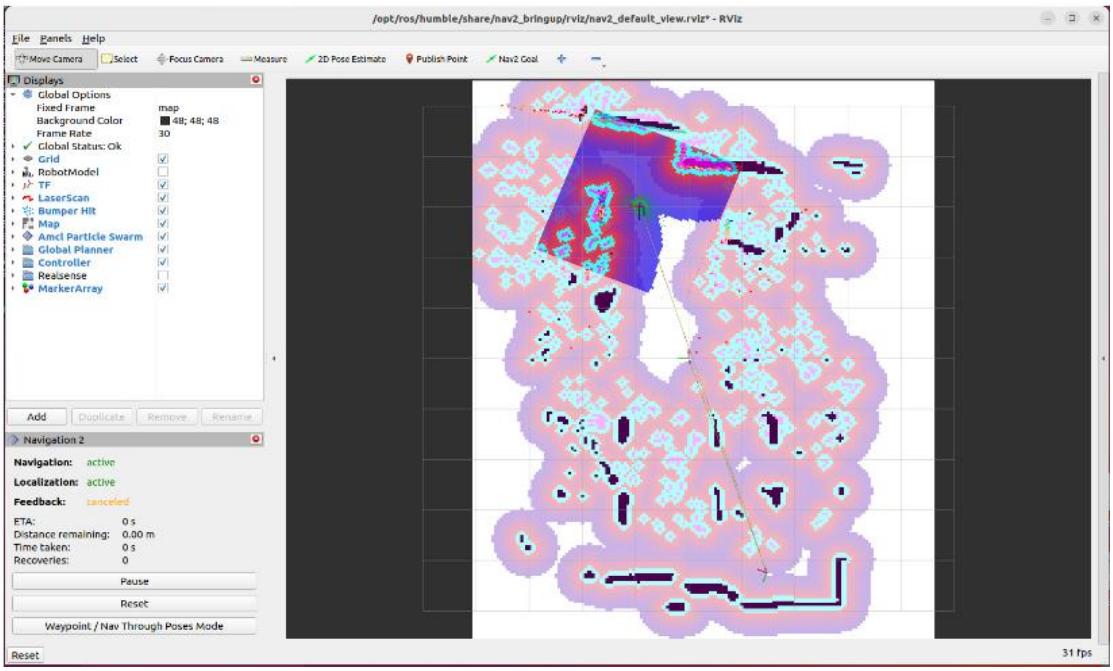


Figure 5-19 Result for Navigation using Python

5.4 Raspberry Pi Camera Setup

Firstly, connect raspberry pi camera to the camera connector at raspberry pi attached to the TurtleBot3. Next, configure and setup the camera via Ubuntu terminal.

```
ubuntu@ubuntu:~$ sudo apt install libraspberrypi-bin v4l-utils ros-humble-v4l2-camera
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
v4l-utils is already the newest version (1.22.1-2build1).
libraspberrypi-bin is already the newest version (0~20220324+gitc4fd1b8-0ubuntu1~22.04.1).
ros-humble-v4l2-camera is already the newest version (0.6.2-1jammy.20230920.005439).

ubuntu@ubuntu:~$ groups
ubuntu adm dialout cdrom floppy sudo audio dip video plugdev netdev lxd
ubuntu@ubuntu:~$ sudo apt-get install raspi-config
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
raspi-config is already the newest version (20220506-0ubuntu1~22.04.1).
0 upgraded, 0 newly installed, 0 to remove and 0 not upgraded.
ubuntu@ubuntu:~$ sudo raspi-config
```

Figure 5-20 Installations for raspberry pi configuration [27]

Install the necessary packages which are libraspberrypi-bin, ros-humble-v4l2-camera and raspi-config to configure the raspberry pi. Then the Ubuntu terminal will turn into the raspberry pi software configuration tool. After that, choose Interface options and enter then select legacy camera to enable it by following the figures below.

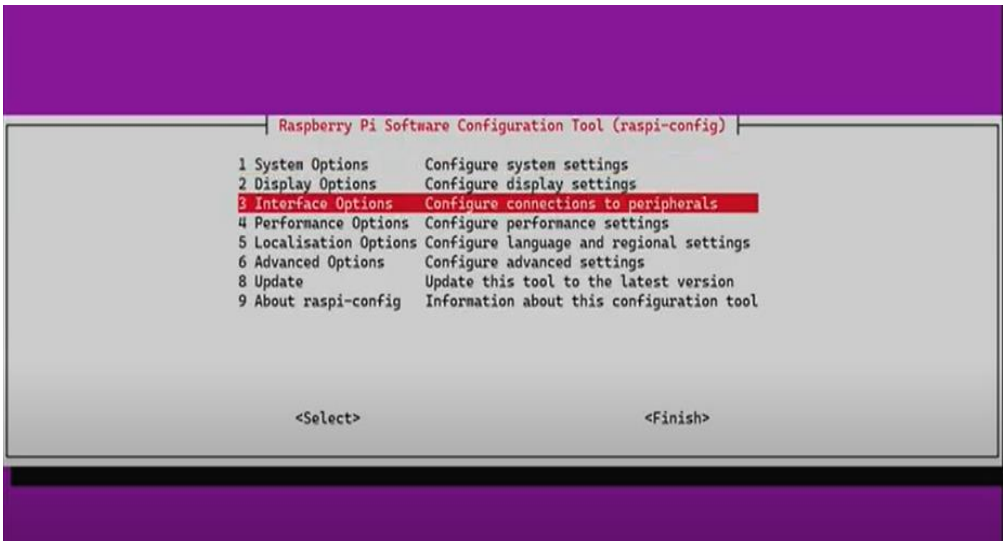


Figure 5-21 Raspberry pi software configuration tool

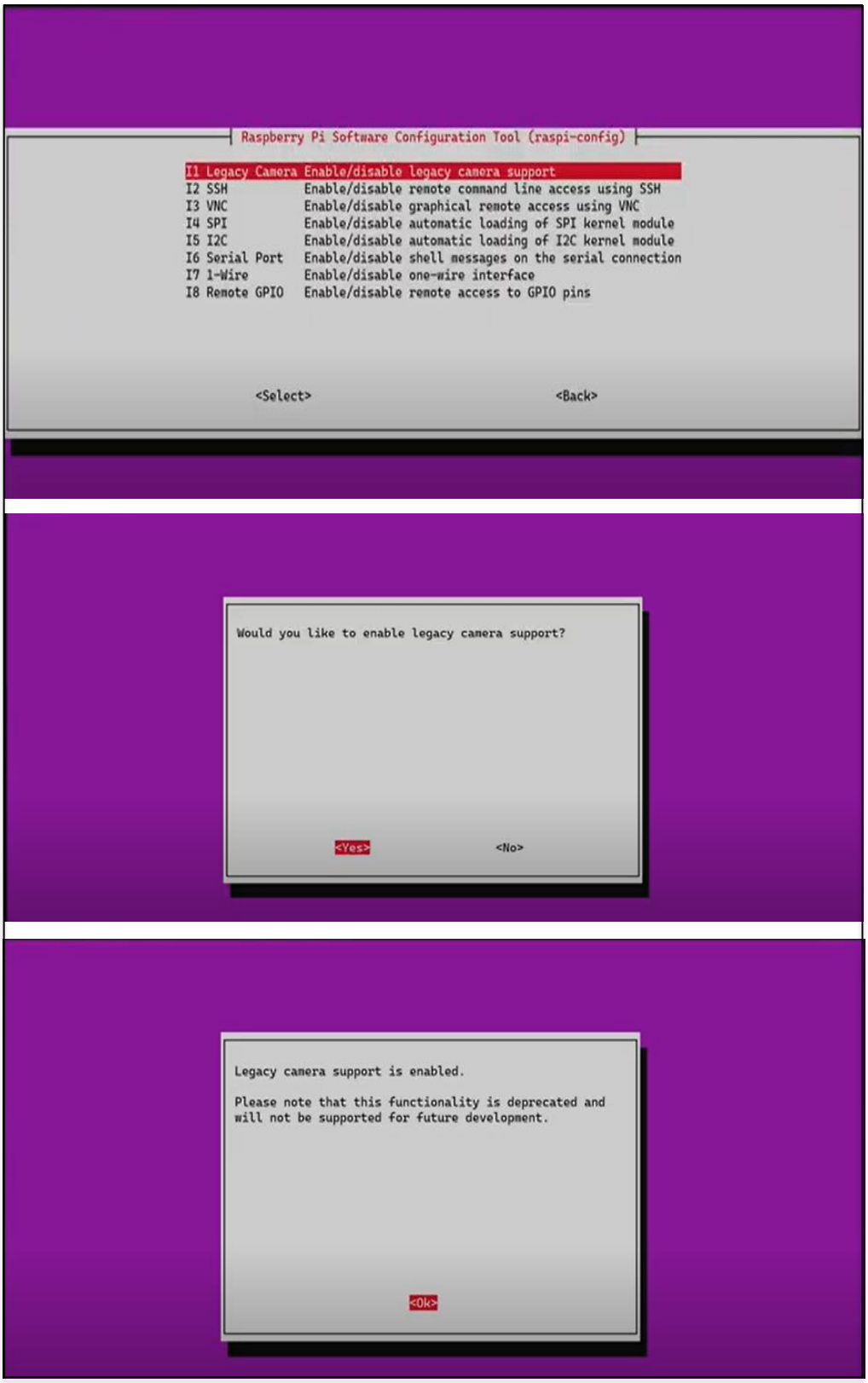


Figure 5-22 Steps to enable raspberry pi camera

Furthermore, enable automatic loading of SPI and I2C kernel modules through the same steps as above. Lastly, enter the finish button to exit the raspi-config tool.

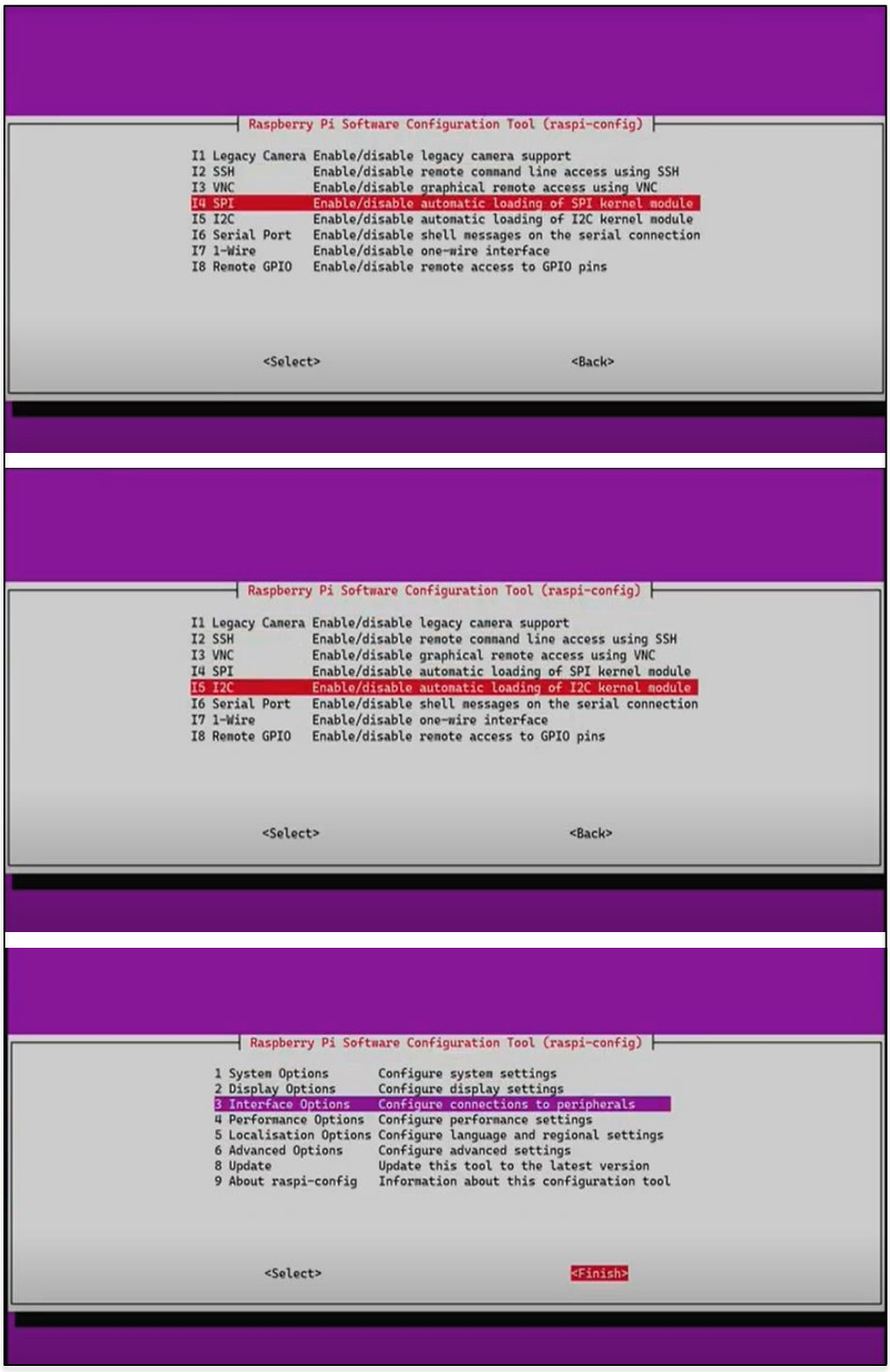


Figure 5-23 Steps to enable automatic loading of SPI and I2C kernel modules

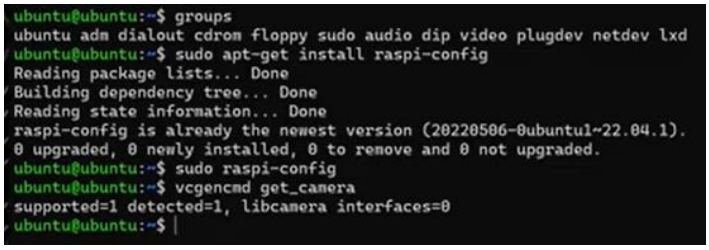


Figure 5-24 Command to check the availability of the raspberry pi camera

The availability of the raspberry pi camera can be checked with the command "vcgencnd get_camera". If supported and detected camera are equal to 1 means that there is one available camera to be used. Thus, "tmux" as a terminal multiplexer is used to enable other terminals that can run a separate program, to be created, accessed, and controlled from a single screen. Finally, run v4l2_camera_node to run the raspberry pi camera as "/image_raw" [28] in the ros2 topic list.

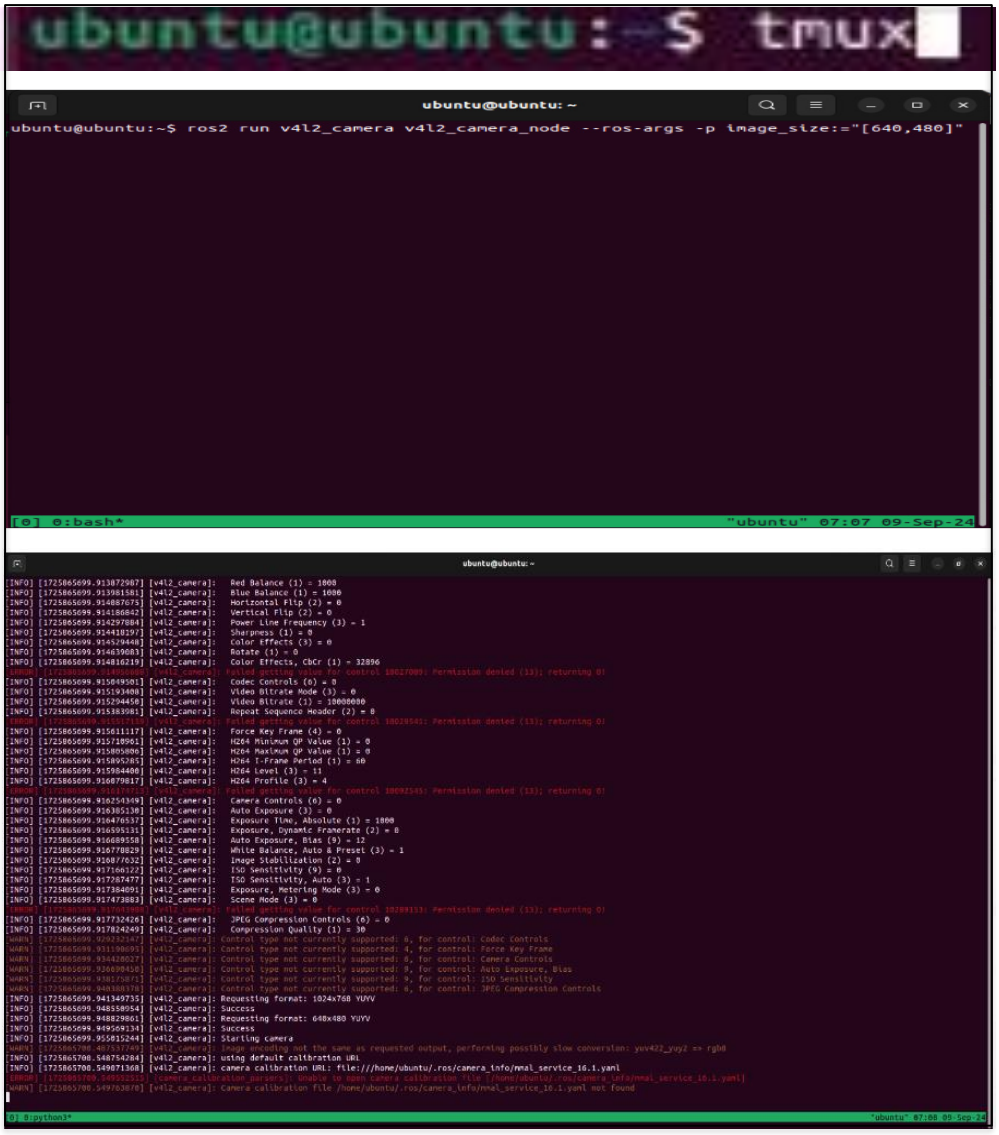


Figure 5-25 Results of the raspberry pi setup

5.5 DHT22 Sensor Setup

Firstly, connect DHT22 to GPIO 4, 5V and Ground pins of the raspberry pi attached to TurtleBot3. Then set up the DHT22 sensor by using Ubuntu terminal.

1. Create a package named “my_robot_controller” in /ros2_ws/src folder:
    - cd ~/ros2_ws/src
    - ros2 pkg create my_robot_controller --build-type ament_Python –dependencies rclpy
2. Navigate to the “my_robot_controller” package:
    - cd ~/ros2_ws/src/ my_robot_controller / my_robot_controller
3. Create and write a new Python file named as tempHumid.py with provided Python codes to the “my_robot_controller” folder:
    - nano tempHumid.py

Figure 5-26 Creation of my_robot_controller package and tempHumid.py

```python
import rclpy
from rclpy.node import Node
from sensor_msgs.msg import Temperature, RelativeHumidity
import adafruit_dht
import board
import time

class DHT22Publisher(Node):
    def __init__(self):
        super().__init__('dht22_publisher')

        # Setup DHT22 sensor
        try:
            self.dht_device = adafruit_dht.DHT22(board.D4)  # Use the actual GPIO pin
        except Exception as e:
            self.get_logger().error(f'Failed to initialize DHT22 sensor: {e}')
            raise

        # Create publishers for temperature and humidity
        self.temp_publisher = self.create_publisher(Temperature, 'temperature', 10)
        self.humid_publisher = self.create_publisher(RelativeHumidity, 'humidity', 10)

        # Create a timer to periodically read the sensor
        self.timer = self.create_timer(2.0, self.publish_sensor_data)  # 2 seconds interval

    def publish_sensor_data(self):
        try:
            # Read data from DHT22 sensor
            time.sleep(5.0)  # Ensure a delay between readings to avoid incomplete data
            temperature = self.dht_device.temperature
            humidity = self.dht_device.humidity

            if temperature is not None and humidity is not None:
                # Create and publish temperature message
                temp_msg = Temperature()
                temp_msg.temperature = float(temperature)  # Ensure temperature is a float
                temp_msg.header.stamp = self.get_clock().now().to_msg()
                self.temp_publisher.publish(temp_msg)

                # Create and publish humidity message
                humid_msg = RelativeHumidity()
                humid_msg.relative_humidity = float(humidity)  # Ensure humidity is a float
                humid_msg.header.stamp = self.get_clock().now().to_msg()
                self.humid_publisher.publish(humid_msg)

                self.get_logger().info(f'Temperature: {temperature} C, Humidity: {humidity} %')
            else:
                self.get_logger().warn('Failed to read from DHT22 sensor.')
        except Exception as e:
            self.get_logger().error(f'Error reading sensor: {e}')

def main(args=None):
    rclpy.init(args=args)
    node = DHT22Publisher()
    try:
        rclpy.spin(node)
    except KeyboardInterrupt:
        pass
    finally:
        node.destroy_node()
        rclpy.shutdown()

if __name__ == '__main__':
    main()
```

Figure 2-27 Codes for DHT22 sensor setup

After saving the tempHumid Python file, use colcon build and source the install/setup.bash. Then run the script as navi1. Finally, "/temperature" and "/humidity" topics is published in the ros2 topic list, and the values of temperature and humidity will be shown every 5 seconds as the figure below.

```
ubuntu@ubuntu:~$ cd ~/ros2_ws
ubuntu@ubuntu:~/ros2_ws$ ls
build  install  log  src
ubuntu@ubuntu:~/ros2_ws$ cd src
ubuntu@ubuntu:~/ros2_ws/src$ ls
my_robot_controller
ubuntu@ubuntu:~/ros2_ws/src$ cd my_robot_controller
ubuntu@ubuntu:~/ros2_ws/src/my_robot_controller$ ls
my_robot_controller  package.xml  resource  setup.cfg  setup.py  test
ubuntu@ubuntu:~/ros2_ws/src/my_robot_controller$ cd my_robot_controller
ubuntu@ubuntu:~/ros2_ws/src/my_robot_controller/my_robot_controller$ ls
__init__.py  tempHumid.py
ubuntu@ubuntu:~/ros2_ws/src/my_robot_controller/my_robot_controller$ nano tempHumid.py
ubuntu@ubuntu:~/ros2_ws/src/my_robot_controller/my_robot_controller$ cd ~/ros2_ws
ubuntu@ubuntu:~/ros2_ws$ colcon build
Starting >>> my_robot_controller
Finished <<< my_robot_controller [24.8s]

Summary: 1 package finished [33.3s]
ubuntu@ubuntu:~/ros2_ws$ source install/setup.bash
ubuntu@ubuntu:~/ros2_ws$ ros2 run my_robot_controller navi1
[INFO] [1725866307.350420887] [dht22_publisher]: Temperature: 25.0 C, Humidity: 66.3 %
```

Figure 5-28 Results of temperature and humidity collected by DHT22 sensor

5.6 Telegram Setup

Firstly, create a new Telegram bot via BotFather of the Telegram and get an API token. Then set a list of commands so that users can know the Telegram bot's functionalities on receiving data and controlling the TurtleBot3 of the robot system. The steps and results of Telegram setup are shown as below:

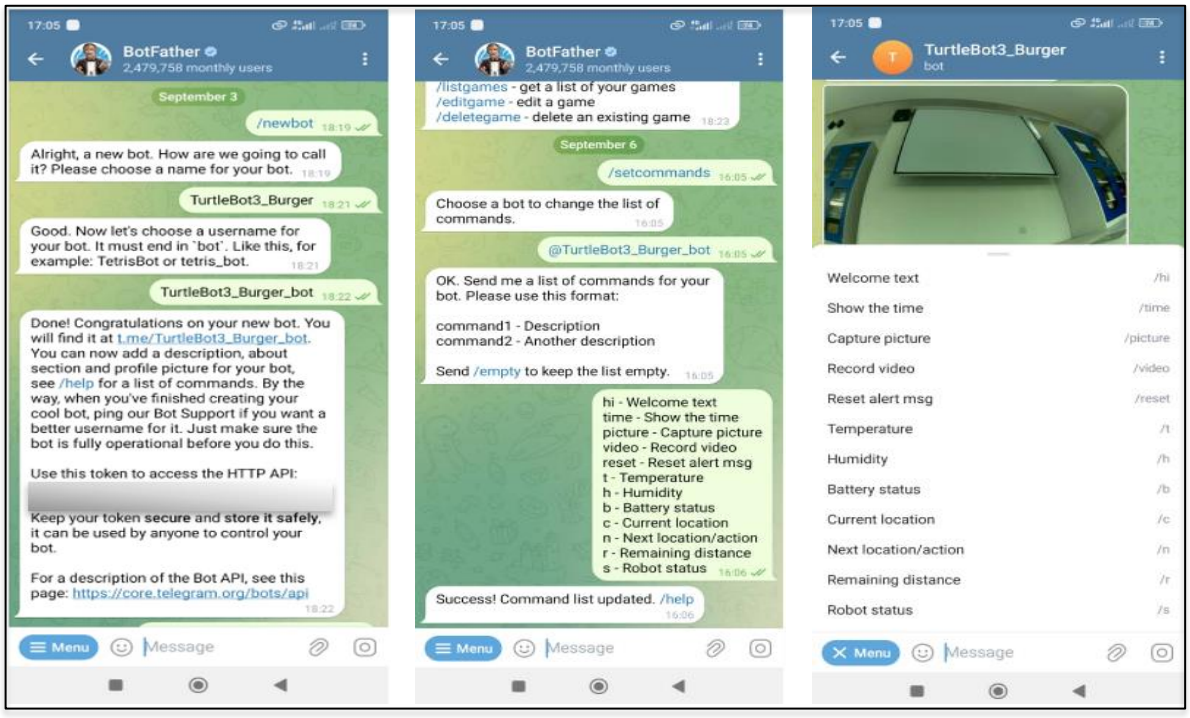


Figure 5-29 Steps and results of Telegram setup

5.7 Overall System Setup

Hardware setup: Raspberry pi camera and DHT22 are connected to the raspberry pi attached to the TurtleBot3 Burger. Battery is also be connected to supply power to the TurtleBot3.

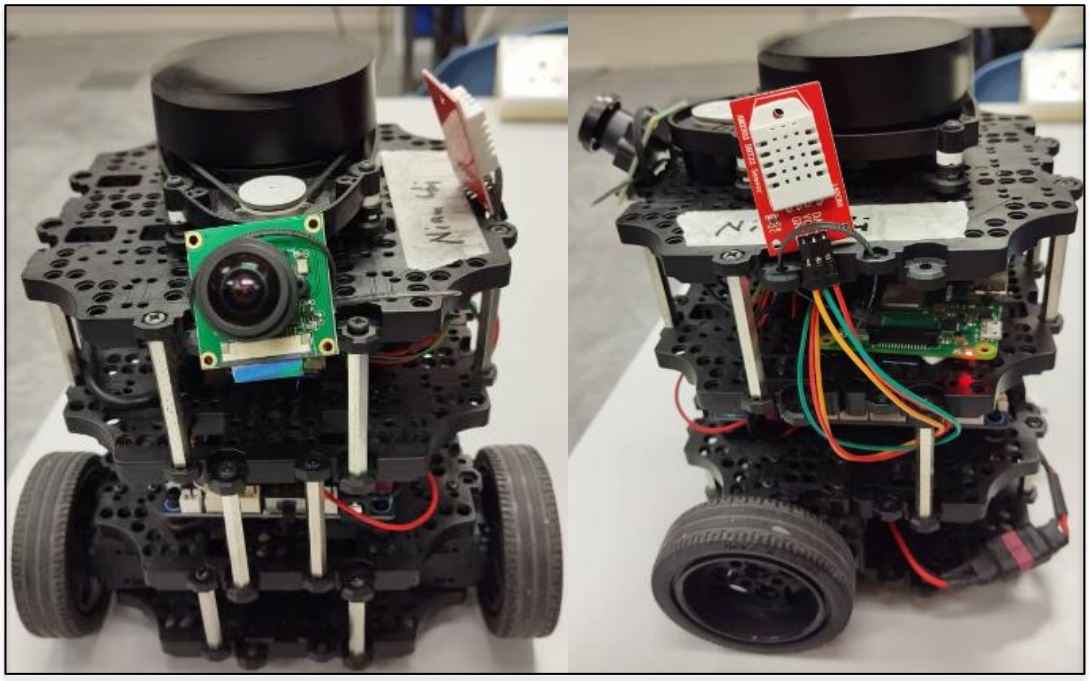


Figure 5-30 Hardware setup

Software setup: Firstly, Open the drawn map with RVIZ2 and configure the TurtleBot3's initial pose estimation by clicking the '2D Pose Estimate' button in the RVIZ2 menu to enable the navigation function. Next, a GUI is created and the ros2 topics such as /battery_state, /temperature, /humidity, /cmd_vel, /image_raw and /amcl_pose from the ros2 topic list are subscribed or published by writing in the Python file using visual studio code to access with the TurtleBot3. This enables the GUI to receive and show the values of battery percentage, temperature, humidity, current position of the TurtleBot3 and the stream video captured by the raspberry pi camera frame by frame. Besides, the /cmd_vel topic enables users to control the speed and movement of the TurtleBot3. Other than that, Telegram bot is also accessed by writing the API token for the in the Python file to enable remote interaction. After the Python file with the fully functional Python codes is created, use colcon build and source the install/setup.bash then run the script as navi. Finally, a GUI will launch and pop up. The system is now ready for users to control the TurtleBot3 using hand gestures.

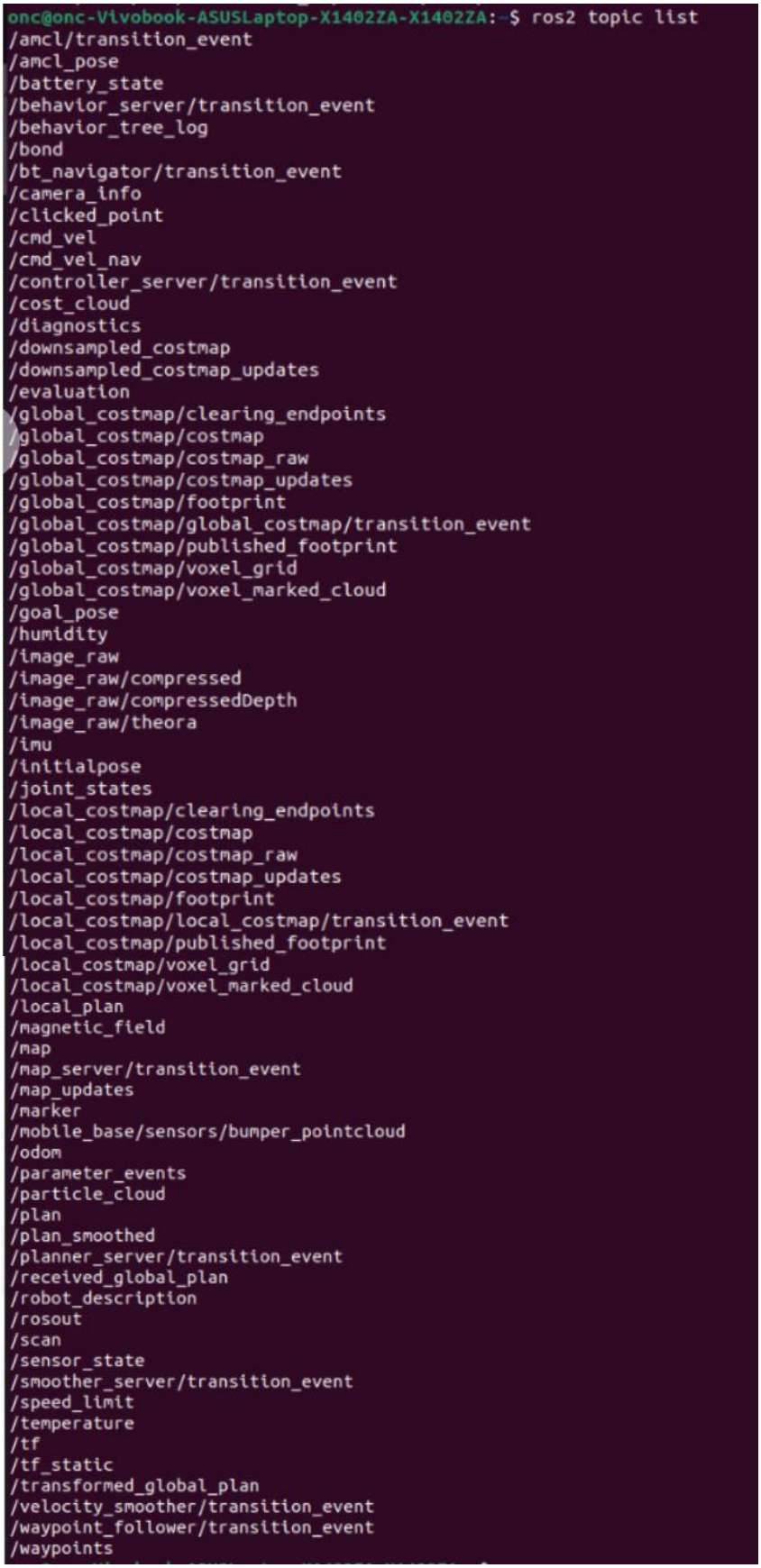


Figure 5-31 ROS2 Topic List (Latest)

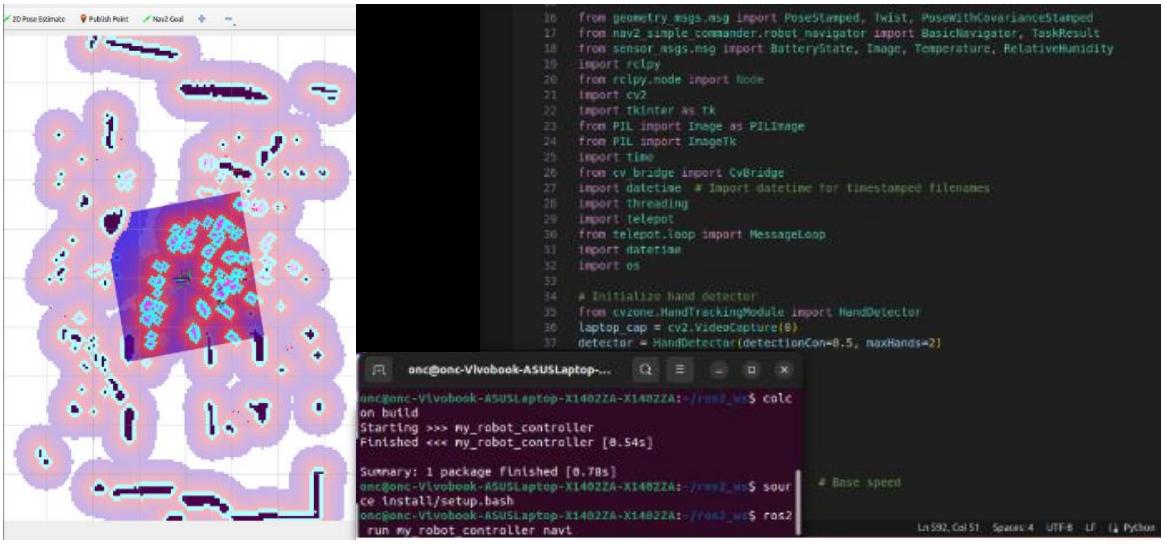


Figure 5-32 RVIZ and PC terminal for running the Python file

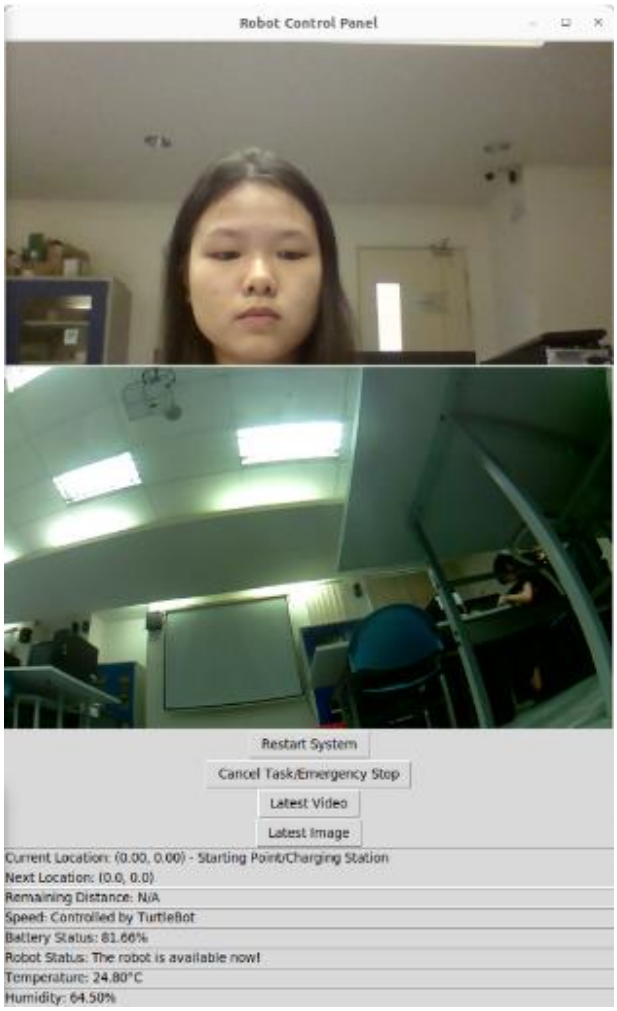


Figure 5-33 GUI of the robot system

5.8 Implementation Issues and Challenges

Several difficulties and problems arose during the project's execution that needed to be carefully considered and resolved. One major obstacle was integrating various hardware parts, like the TurtleBot and sensors, with the ROS2 and Ubuntu software environments. Technical challenges had to be overcome to guarantee these components' smooth compatibility and communication. Another issue was to optimize deep learning model performance for real-time gesture recognition on platforms with limited resources. To obtain sufficient performance in real-world circumstances, it was imperative to strike a balance between computational efficiency and precision.

Despite these difficulties, the project also included several cutting-edge elements that enhanced its originality. A fresh approach to human-robot interaction was represented using deep learning techniques, specifically Convolutional Neural Networks (CNNs), for gesture detection. These methods made it possible for the robot to reliably comprehend complicated motions, enabling smooth communication between people and robots. Additionally, the use of GUI and IoT in this project also let users feel convenient when using the robot system. In summary, resolving implementation obstacles and utilizing cutting-edge technologies were critical to the project's success.

5.9 Concluding Remark

In conclusion, there are 3 Ubuntu terminals and 2 PC terminals required for the robot system implementation. Three Ubuntu terminals are used to bring up the TurtleBot3, run the v4l2_camera_node and execute the tempHumid.py script (navi1) for the DHT22 sensor. Two PC terminals are used to open map in RVIZ2 for navigation and run the main Python script (navi) to launch the system. All the steps for all the setups are stated above.

Furthermore, there are still some implementation issues and challenges. For examples, the imperfection of the hardware and software. However, these implementation obstacles can be addressed by leveraging cutting-edge technologies.

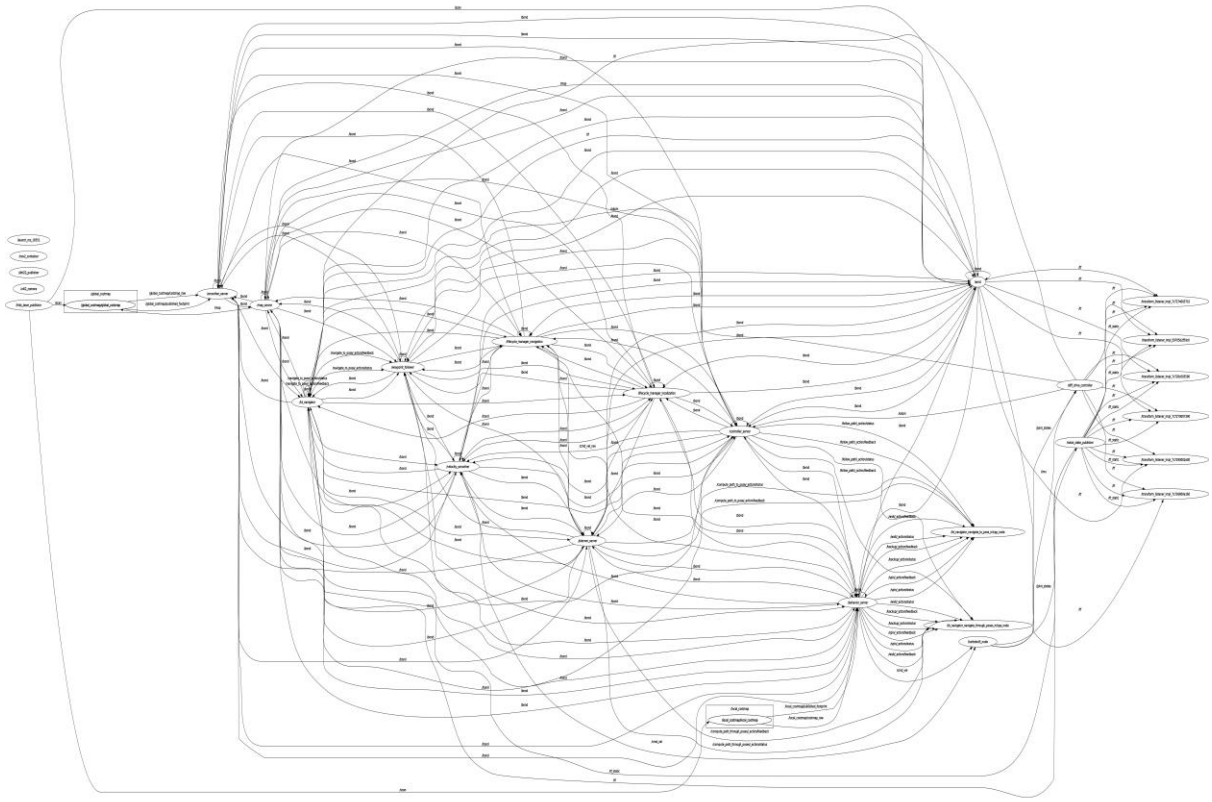


Figure 5-34 Node graph from rqt

Chapter 6

SYSTEM EVALUATION AND DISCUSSION

6.1 System Testing and Performance Metrics

Evaluation of the accuracy of the hand tracking module

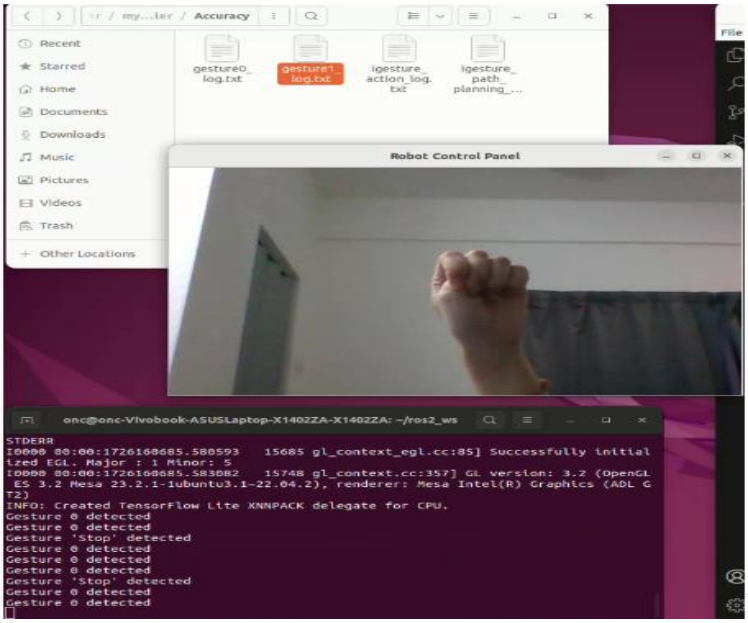
Pseudocode:

```
import necessary libraries
Path to the log file
Open the log file in append mode

gesture_number = {
   (0, 0, 0, 0, 0): 0, (0, 1, 0, 0, 0): 1, (0, 1, 1, 0, 0): 2, (0, 1, 1, 1, 0): 3, (0, 0, 1, 1, 1): "OK", (0, 1, 1, 1, 1): 4,
   (1, 1, 1, 1, 1): 5, (1, 0, 0, 0, 1): 6, (1, 1, 0, 0, 0): 7, (1, 1, 1, 0, 0): 8, (1, 1, 1, 1, 0): 9, (1, 0, 0, 0, 0): 10,
   (0, 0, 0, 0, 1): 11, (0, 0, 0, 1, 1): 12, (1, 0, 0, 1, 1): 13, (1, 1, 0, 1, 1): 14
}

Set gesture_count to 0
Define MAX_GESTURES as 20

Function detect_hand_gesture:
   If gesture_count >= MAX_GESTURES:
      Print "Max gestures reached. Stopping system."
      Call close_system
      Return
   Read frame from video capture
   If frame is read successfully:
      Display image on canvas
      Detect hands in the frame
      If hands are detected:
         If two hands are detected:
            Get finger positions for both hands
           # Determine detected gesture based on finger positions
            if fingers1 == [1, 0, 0, 0, 0] and fingers2 == [1, 0, 0, 0, 0]:
                  detected_gesture1 = "Move forward"
           elif fingers1 == [0, 0, 0, 0, 1] and fingers2 == [0, 0, 0, 0, 1]:
                  detected_gesture1 = "Move backward"
            elif fingers1 == [0, 1, 0, 0, 0] and fingers2 == [0, 0, 0, 0, 0]:
                  detected_gesture1 = "Turn left"
```

```
            elif fingers1 == [0, 0, 0, 0, 0] and fingers2 == [0, 1, 0, 0, 0]:
                detected_gesture1 = "Turn right"
            elif fingers1 == [0, 0, 0, 0, 0] and fingers2 == [0, 0, 0, 0, 0]:
                detected_gesture1 = "Stop"
            elif fingers1 == [0, 0, 0, 0, 1] and fingers2 == [0, 0, 0, 0, 0]:
                detected_gesture1 = "Speed up"
            elif fingers1 == [0, 0, 0, 0, 0] and fingers2 == [0, 0, 0, 0, 1]:
                detected_gesture1 = "Speed down"
            If a gesture is detected:
              Increment gesture_count
              Print detected gesture
              Call log_detected_gesture with fingers and detected gesture
        Else:
            Get finger positions for the single hand
            Check if the finger positions match any gesture in gesture_number
            If a gesture is matched:
              Increment gesture_count
              Print detected gesture number
              Call log_detected_gesture with finger positions and detected gesture number
    Set a timer to call detect_hand_gesture again in 10 milliseconds

Function log_detected_gesture(fingers, detected_gesture):
    Get current timestamp
    Write timestamp, fingers, and detected gesture to log file

Function close_system:
    Close log file
    Release video capture
    Destroy all OpenCV windows
    Quit the Tkinter application

Function main:
    Initialize Tkinter
    Create a canvas for video display
    Call detect_hand_gesture to start gesture detection
    Start Tkinter main loop

If script is run directly:
    Call main
```

The percentage of accurate predictions generated based on the test data is known as accuracy. By dividing the total number of guesses by the number of accurate predictions, it is simple to calculate [29].

$$\text{Accuracy} = \frac{Correct\ predictions}{All\ predictions}$$



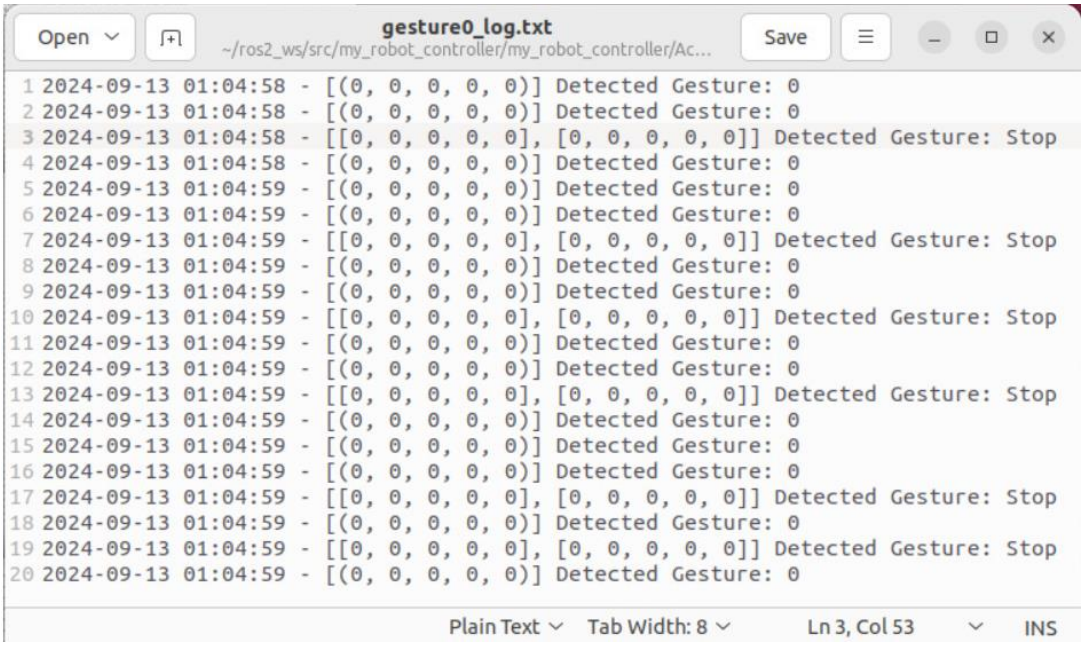
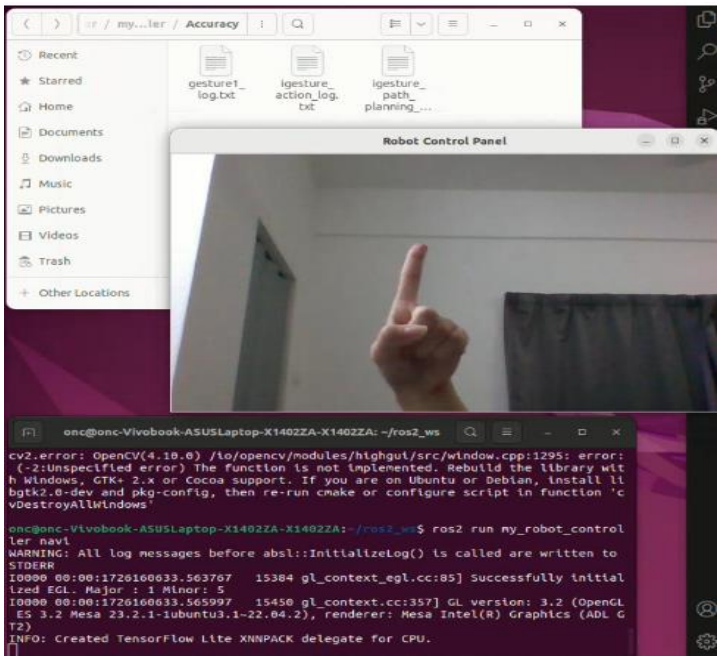


Figure 6-1 Results of the accuracy of the hand tracking module in "0"

$$\text{Accuracy} = \frac{14}{20} \times 100\%$$

$$= 70\%$$



Open ~/ros2_ws/src/my_robot_controller/my_robot_controller/Ac... gesture1_log.txt Save

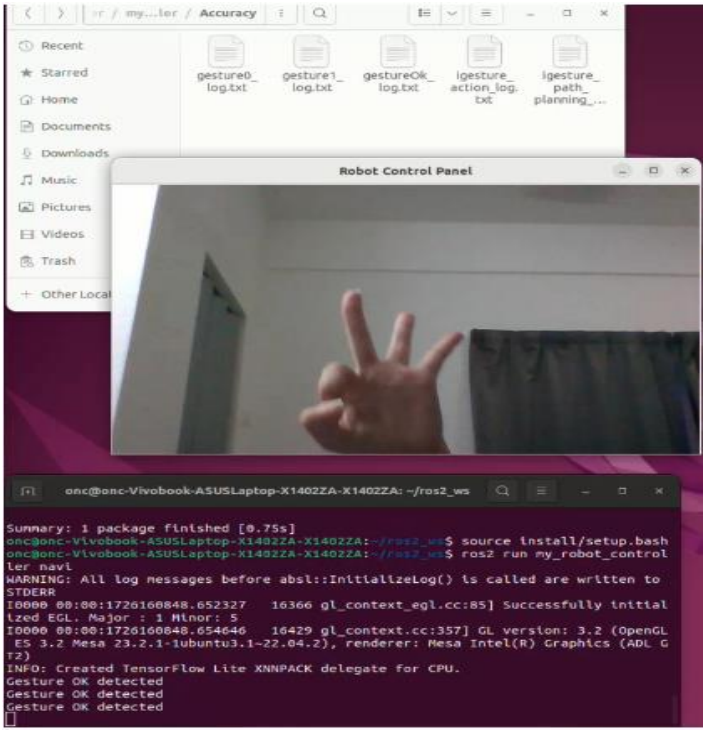
```
1 2024-09-13 01:04:03 - [(0, 1, 0, 0, 0)] Detected Gesture: 1
2 2024-09-13 01:04:03 - [(0, 1, 0, 0, 0)] Detected Gesture: 1
3 2024-09-13 01:04:03 - [(0, 1, 0, 0, 0)] Detected Gesture: 1
4 2024-09-13 01:04:03 - [(0, 1, 0, 0, 0)] Detected Gesture: 1
5 2024-09-13 01:04:03 - [(0, 1, 0, 0, 0)] Detected Gesture: 1
6 2024-09-13 01:04:03 - [(0, 1, 0, 0, 0)] Detected Gesture: 1
7 2024-09-13 01:04:03 - [(0, 1, 0, 0, 0)] Detected Gesture: 1
8 2024-09-13 01:04:03 - [(0, 1, 0, 0, 0)] Detected Gesture: 1
9 2024-09-13 01:04:04 - [(0, 1, 0, 0, 0)] Detected Gesture: 1
10 2024-09-13 01:04:04 - [(0, 1, 0, 0, 0)] Detected Gesture: 1
11 2024-09-13 01:04:04 - [(0, 1, 0, 0, 0)] Detected Gesture: 1
12 2024-09-13 01:04:04 - [(0, 1, 0, 0, 0)] Detected Gesture: 1
13 2024-09-13 01:04:04 - [(0, 1, 0, 0, 0)] Detected Gesture: 1
14 2024-09-13 01:04:04 - [(0, 1, 0, 0, 0)] Detected Gesture: 1
15 2024-09-13 01:04:04 - [(0, 1, 0, 0, 0)] Detected Gesture: 1
16 2024-09-13 01:04:04 - [(0, 1, 0, 0, 0)] Detected Gesture: 1
17 2024-09-13 01:04:04 - [(0, 1, 0, 0, 0)] Detected Gesture: 1
18 2024-09-13 01:04:04 - [(0, 1, 0, 0, 0)] Detected Gesture: 1
19 2024-09-13 01:04:04 - [(0, 1, 0, 0, 0)] Detected Gesture: 1
20 2024-09-13 01:04:04 - [(0, 1, 0, 0, 0)] Detected Gesture: 1
```

Loading file "/home/onc/ros2_ws/src/m... Plain Text Tab Width: 8 Ln 1, Col 1 INS

Figure 6-2 Results of the accuracy of the hand tracking module in "1"

$$\text{Accuracy} = \frac{20}{20} \times 100\%$$

$$= 100\%$$



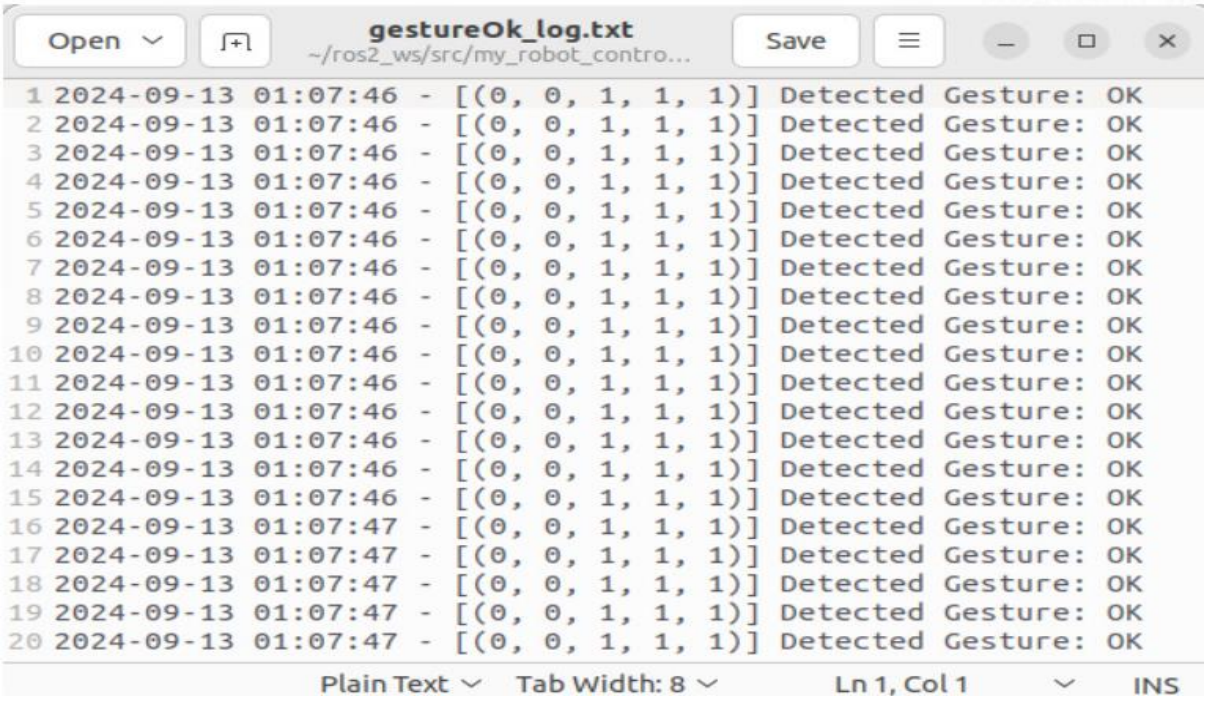
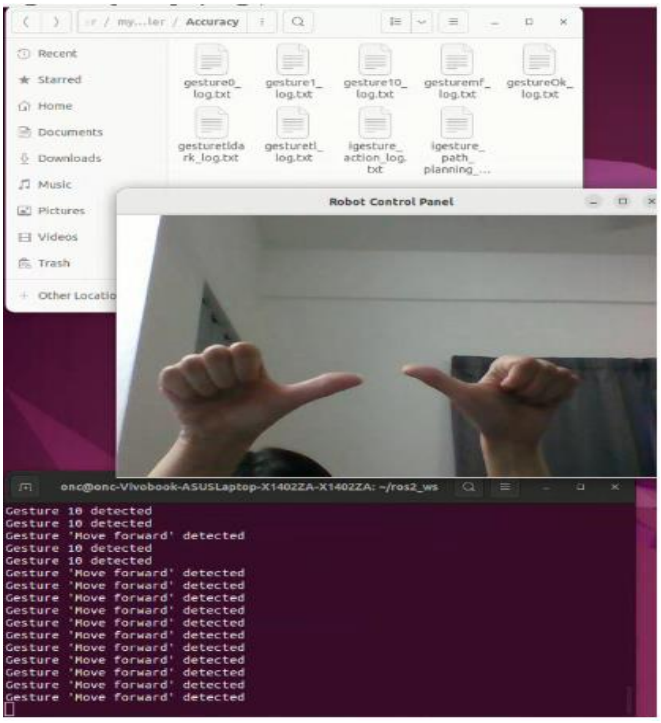


Figure 6-3 Results of the accuracy of the hand tracking module in "OK"

$$\text{Accuracy} = \frac{20}{20} \times 100\%$$
$$= 100\%$$



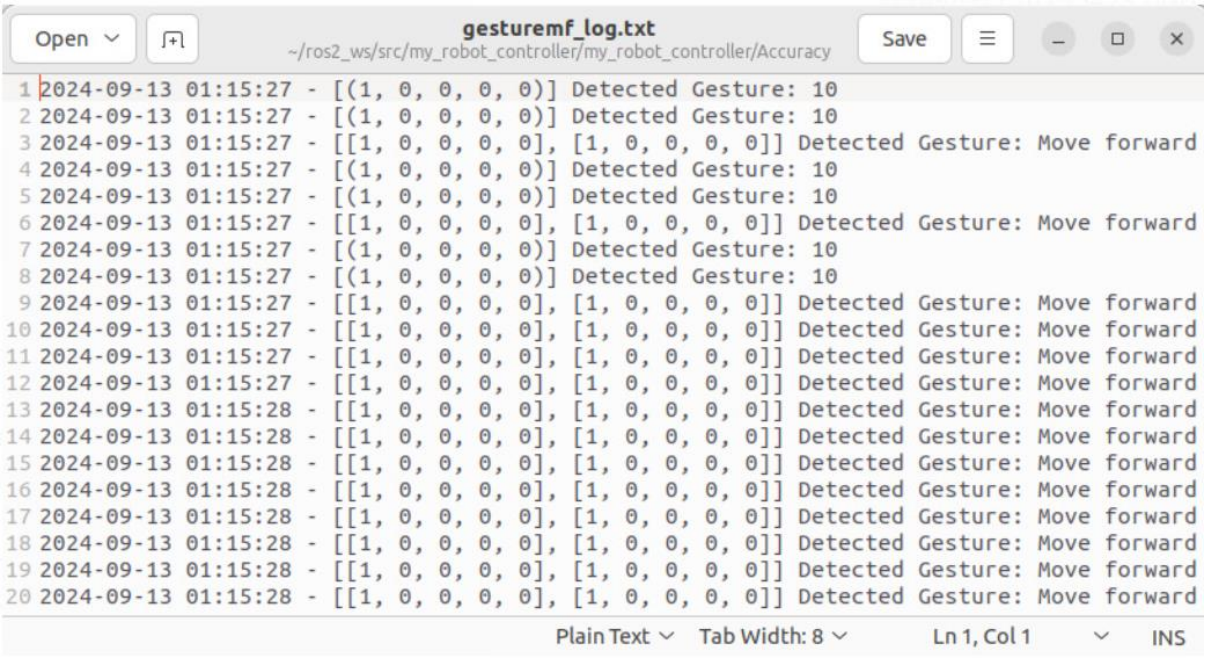


Figure 6-4 Results of the accuracy of the hand tracking module in "Move forward"

$$\text{Accuracy} = \frac{14}{20} \times 100\%$$

$$= 70\%$$

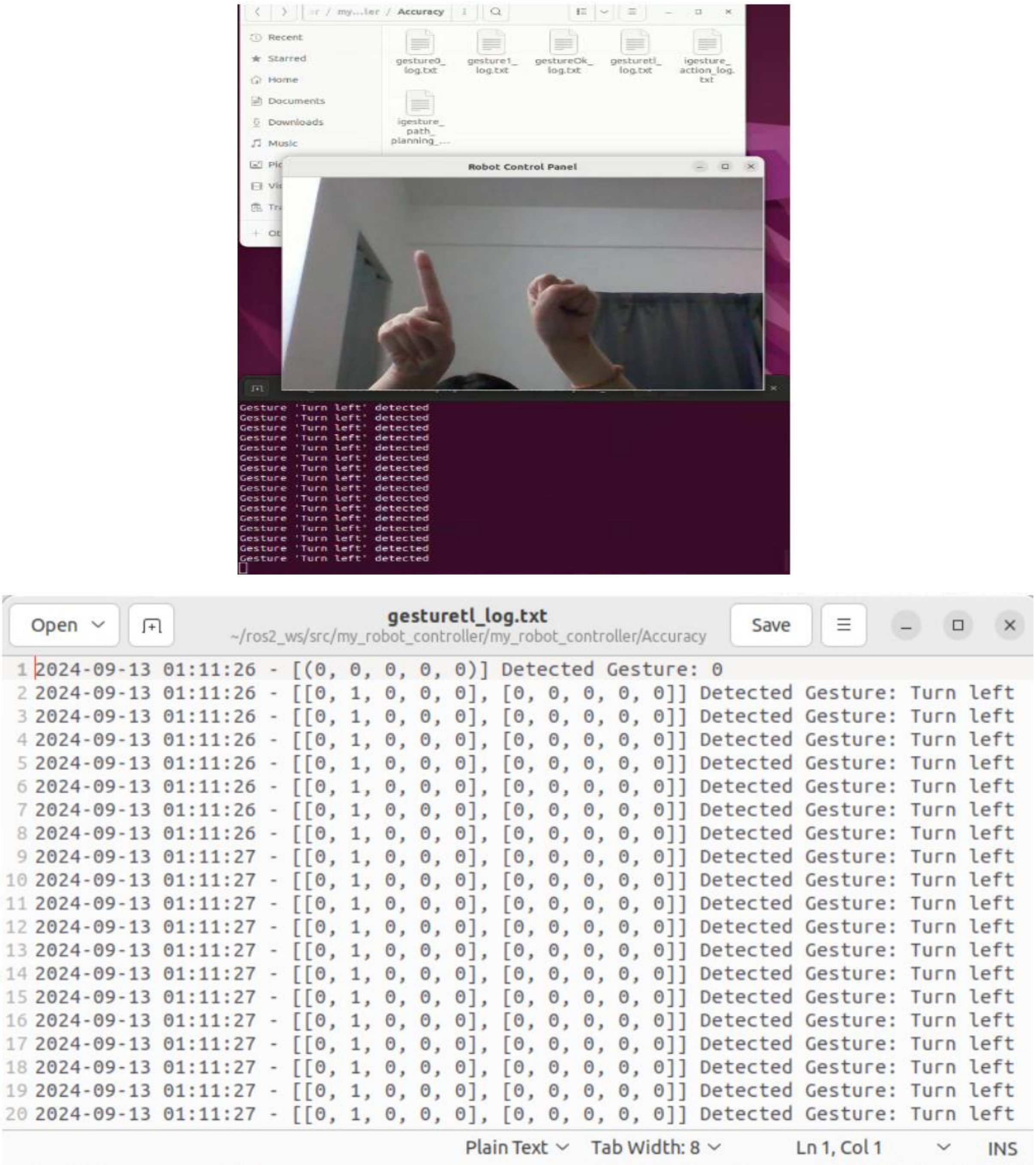
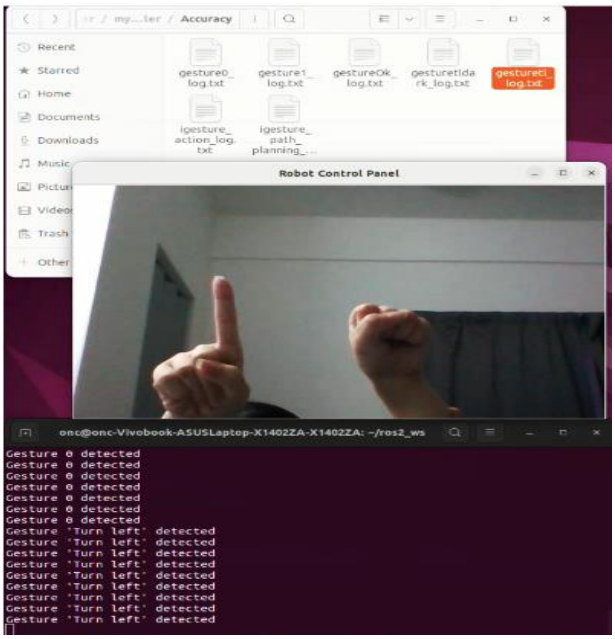


Figure 6-5 Results of the accuracy of the hand tracking module in “Turn left” at light environment

$$\text{Accuracy} = \frac{19}{20} \times 100\%$$

$$= 95\%$$



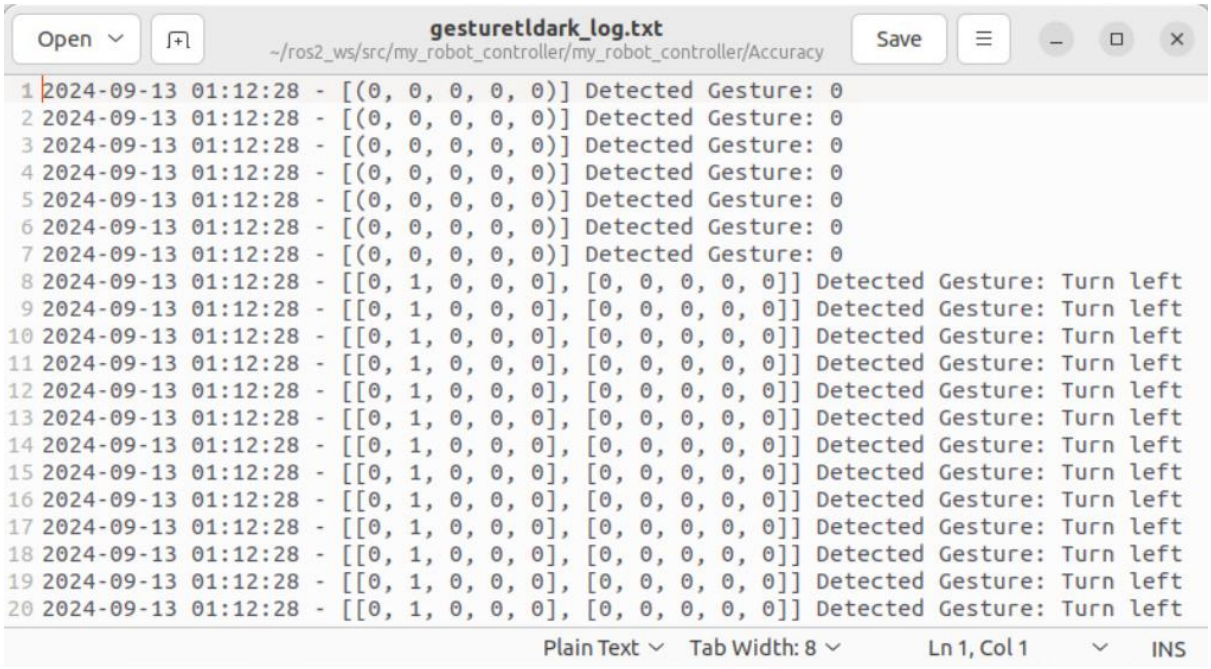


Figure 6-6 Results of the accuracy of the hand tracking module in "Turn left" at darker environment

$$\text{Accuracy} = \frac{13}{20} \times 100\%$$
$$= 65\%$$

Hand gesture recognition achieves 100% accuracy when each pattern for one hand is uniquely defined. Accuracy decreases to 95% when unique patterns involve both hands, and further drops to 70% when hand patterns are not distinct, such as recognizing "0" with both hands versus one hand. In darker environments, the accuracy reduces to 65%.

Evaluation of the real time between hand detection and navigation

Pseudocode:

```
Import necessary libraries
Path to the log file
Open the log file in append mode
.
.
.
Define gesture_number as a dictionary mapping finger tuples to gestures
gesture_number = { (0, 1, 0, 0, 0): 1, (0, 1, 1, 0, 0): 2, (0, 0, 0, 0, 0): 0, (0, 0, 1, 1, 1): 3, (0, 1, 1, 1, 1): 4 }
Define goal_positions as a dictionary mapping gestures to goal coordinates
goal_positions = { 1: (0.0, 0.0), # Goal for gesture 1 2: (3.00, 1.00), # Goal for gesture 2 }

Define test_gestures as a list of 5 sample gestures
Define true_labels as the same list as test_gestures

Function handle_gesture(gesture):
    Record current time as detection_time
    Record current time as action_time
    Compute latency as the difference between action_time and detection_time
    Log the detected gesture, action, and latency
    Return gesture, gesture_number, latency

Function evaluate_model_performance(true_labels, predicted_labels):
    Compute accuracy, precision, recall, and F1-score
    Print performance metrics
    Compute and display confusion matrix using true and predicted labels

Function main():
    Initialize ROS (Robot Operating System) components
    Initialize GUI window with labels for sensor data and video feeds
    Define detect_hand_gesture function:
        Capture a frame from the camera
        Convert the frame to RGB
        Display the frame on the GUI canvas
        Detect hands in the frame
        If hands are detected:
            Extract finger states and convert to a tuple
            If the finger tuple is in gesture_number:
```

```
            Record start time
            Identify gesture from gesture_number
            If the gesture has a goal position:
               Update remaining distance label and navigate to goal
            Otherwise, handle movement based on gesture where gesture = 0 is stop, 3 is moving forward and 4 is moving backward
            Record end time and log time taken
        Schedule the next frame capture

    Define update_status function:
        Spin ROS node to receive updates
        If battery status is available:
            Update battery status label
            If battery is low, navigate to the charging station
            Monitor task completion and update remaining distance label
            Check task result and handle accordingly
        Schedule the next status update

    Start GUI event loop and background threads for sensor and camera updates
    In the main function:
        Simulate gesture detection and handle gestures
        Evaluate model performance based on simulated results
        Analyze gesture action log for latency and navigation success

    Define analyze_logs function:
        Read and parse the gesture action log file
        Extract and compute latency metrics
        Calculate and print navigation success rate
        Optionally print the full log DataFrame

Execute main function
```

Based on the results below, the calculation:

$$\text{Average time taken} = \frac{11(0.03)+2(0.02)+104(0.00)}{117}$$

$$= 0.003162 \text{ seconds}$$

The robot system operates in near real-time, with an average response time of 0.003162 seconds from hand gesture recognition to the TurtleBot2's response. This extremely short delay is nearly imperceptible, making the system highly responsive.

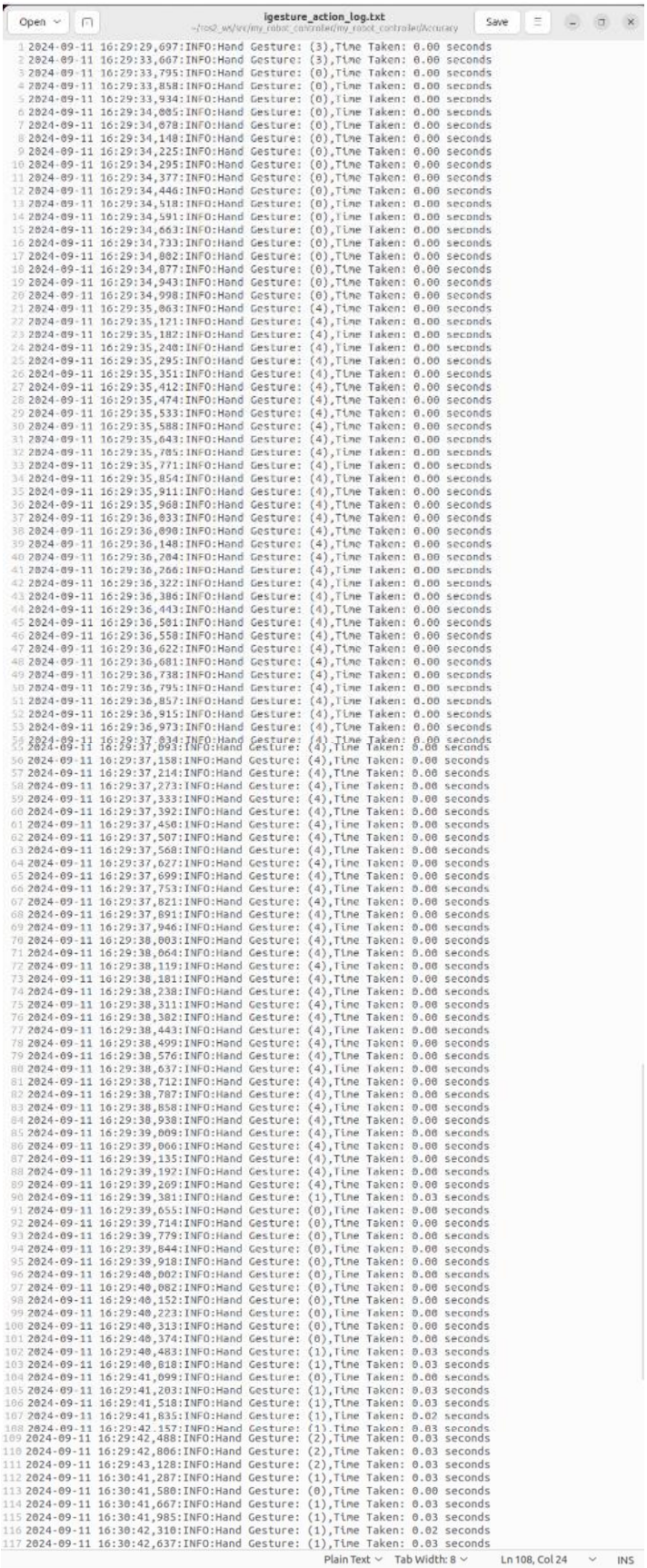


Figure 6-7 Results of the evaluation for the real time between hand detection and navigation

Evaluation of the deviation and time taken for path planning of navigation

Pseudocode:

```
Import necessary libraries
Path to the log file
Open the log file in append mode
…
DEFINE gesture_number AS {
   (0, 1, 0, 0, 0): 1, (0, 1, 1, 0, 0): 2, (0, 0, 0, 0, 0): 0, (0, 0, 1, 1, 1): 3, (0, 1, 1, 1, 1): 4
}
DEFINE goal_positions AS {
   1: (0.0, 0.0),       # Goal for gesture 1
   2: (3.00, 1.00),     # Goal for gesture 2
}
DEFINE test_gestures AS [1, 2 ]
DEFINE true_labels AS test_gestures

FUNCTION handle_gesture(gesture):
   DETERMINE detection_time AS current time
   DETERMINE action_time AS current time
   CALCULATE latency AS action_time - detection_time
   LOG gesture, gesture_number, latency
   RETURN gesture, gesture_number, latency

FUNCTION evaluate_model_performance(true_labels, predicted_labels):
   CALCULATE accuracy, precision, recall, f1 from true_labels and predicted_labels
   PRINT accuracy, precision, recall, f1
   COMPUTE confusion_matrix from true_labels and predicted_labels
   DISPLAY confusion_matrix as heatmap

FUNCTION main():
   INITIALIZE ROS nodes and navigator
   SETUP GUI with labels and canvases
   FUNCTION detect_hand_gesture():
      READ frame from camera
      CONVERT frame to RGB
      DISPLAY frame on laptop_canvas
      DETECT hands in frame
      IF hands detected:
         GET fingers' state
```

```
            CREATE tuple from fingers' state
            IF tuple in gesture_number:
                SET gesture_number1 from gesture_number
                IF gesture_number1 in goal_positions:
                    UPDATE remaining_distance_label
                    SET speed to "Controlled by TurtleBot"
                    GET goal position (x, y) from goal_positions
                    NAVIGATE to goal position (x, y)
                    RESET camera
        CALL detect_hand_gesture() again in 10 milliseconds

    FUNCTION update_status():
        SPIN ROS node
        IF battery_status is available:
            PRINT battery level
            IF battery level < 30%:
                CANCEL any current task
                STOP robot
                SET goal_pose to (0.0, 0.0)
                NAVIGATE to goal_pose
                MONITOR task completion and UPDATE remaining_distance_label
                HANDLE task result (SUCCEEDED, CANCELED, FAILED)
        CALL update_status() again in 1 second
    CALL detect_hand_gesture() to start gesture detection
    CALL update_status() to start status updates

    FUNCTION analyze_logs():
        READ log file
        EXTRACT and CALCULATE latency from log
        PRINT average, minimum, maximum latency
        EXTRACT and CALCULATE navigation success rate from log
        PRINT success rate
    EXECUTE main()
    SIMULATE gesture detection
    EVALUATE performance with true_labels and predicted_labels
    ANALYZE logs

IF __name__ == "__main__":
    CALL main()
```

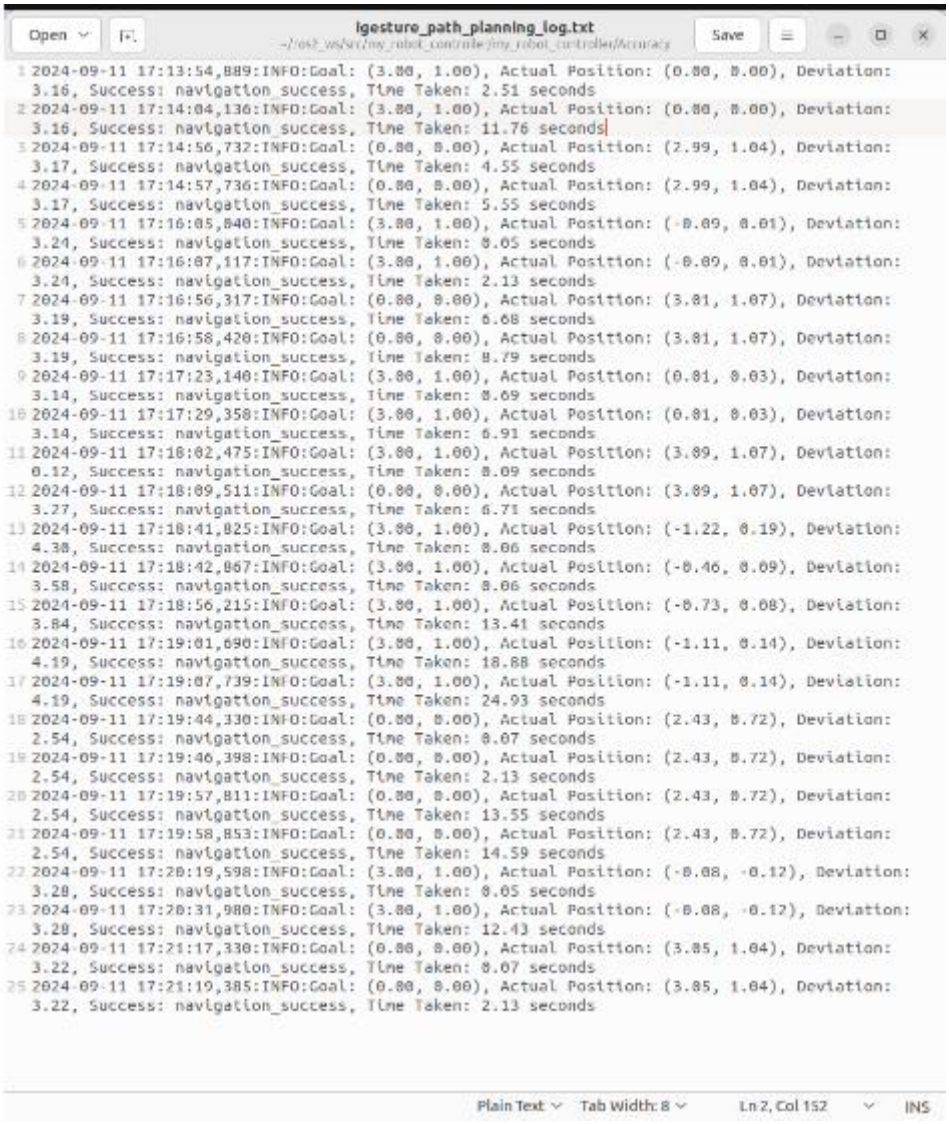


Figure 6-7 Results of the path planning for navigation

$$\text{Average time taken} = \frac{\frac{2.51+11.76+4.55+5.55+0.05+2.13+6.68+8.79+0.69+6.91+0.09+6.71+0.06+0.06}{13.41+18.88+24.93+0.07+2.13+13.55+14.59+0.05+12.43+0.07+2.13}}{25}$$

$$= 6.351 \text{ seconds}$$

$$\text{Average deviation} = \frac{\frac{3.16+3.16+3.17+3.17+3.24+3.24+3.19+3.19+3.14+3.14+0.12+3.27}{+4.38+3.58+3.84+4.19+4.19+2.54+2.54+2.54+2.54+3.28+3.28+3.22+3.22}}{25}$$

$$= 3.141 \text{ centimeters}$$

The robot system using nav2 for navigation can be considered as smart, accurate and responsive as the path planning taking less than 7 seconds of the average time taken and an average deviation of under 4 centimeters. However, its performance may decline in environments with significant noise, such as human activity or obstacles.

6.2 Testing Setup and Result

Gestures detection with their corresponding actions and GUI functionalities

Table 6-1 Hand Gestures with their corresponding actions

Hand(s)	Hand Gesture	Action
1	(0,0,0,0,0): 0	Stop
1	(0,1,0,0,0): 1	Go to (0.0, 0.0) then capture image when arrived
1	(0,1,1,0,0): 2	Go to (3.00, 1.00) then capture image when arrived
1	(0,1,1,1,0): 3	Go to (-1.20, 1.10) then capture image when arrived
1	(0,0,1,1,1): OK	Go to (0.0, 0.0) then capture image when arrived
1	(0,1,1,1,1): 4	Go to (-2.70, -0.40) then capture image when arrived
1	(1,1,1,1,1): 5	Draw circle
1	(1,0,0,0,1): 6	Surveillance mode
1	(1,1,0,0,0): 7	Start recording on the raspi_canvas and go to (-2.70, -0.40) then stop recoding when arrived
1	(1,1,1,0,0): 8	Start recording on the raspi_canvas and go to (0.0, 0.0) then stop recoding when arrived
1	(1,1,1,1,0): 9	Start recording on the raspi_canvas and go to (3.00, 1.00) then stop recoding when arrived
1	(1,0,0,0,0): 10	Start recording on the raspi_canvas and go to (-1.20, 1.10) then stop recoding when arrived
1	(0,0,0,0,1): 11	Capture image
1	(0,0,0,1,1): 12	Record video on the raspi_canvas by drawing one circle then stop recording
1	(1,0,0,1,1): 13	Start recording on the raspi_canvas
1	(1,1,0,1,1): 14	Stop recording
2	(0,0,0,0,0), (0,0,0,0,0)	Stop
2	(1,0,0,0,0), (1,0,0,0,0)	Move forward
2	(0,0,0,0,1), (0,0,0,0,1)	Move backward
2	(0,0,0,0,1), (0,0,0,0,0)	Speed up
2	(0,0,0,0,0), (0,0,0,0,1)	Slow down
2	(0,1,0,0,0), (0,0,0,0,0)	Turn left
2	(0,0,0,0,0), (0,1,0,0,0)	Turn right

All the captured images and recorded videos will be saved in the directory as shown below:

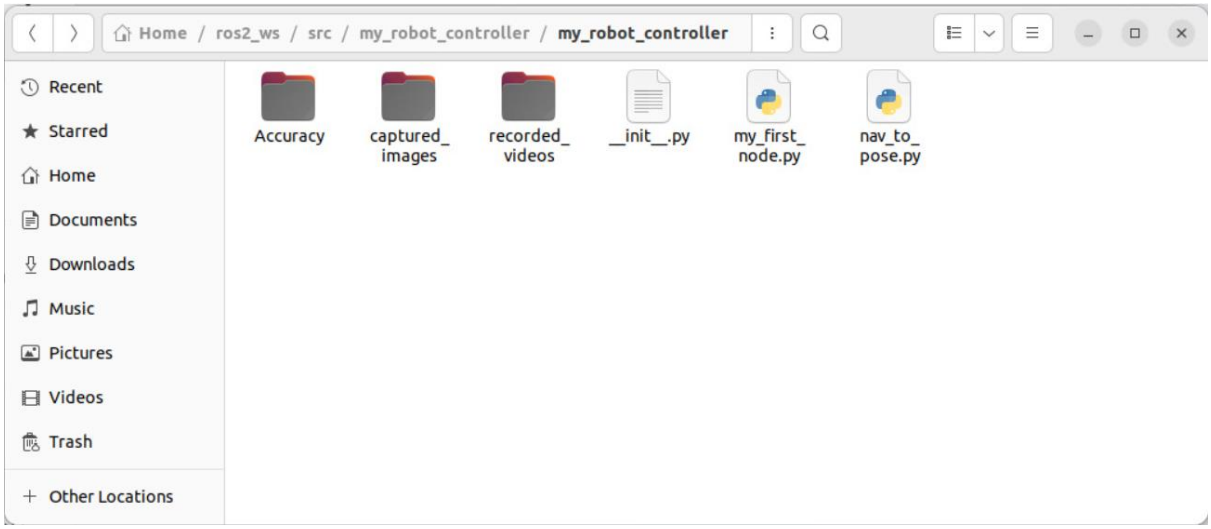


Figure 6-8 Directory of the saving captured images and recorded videos

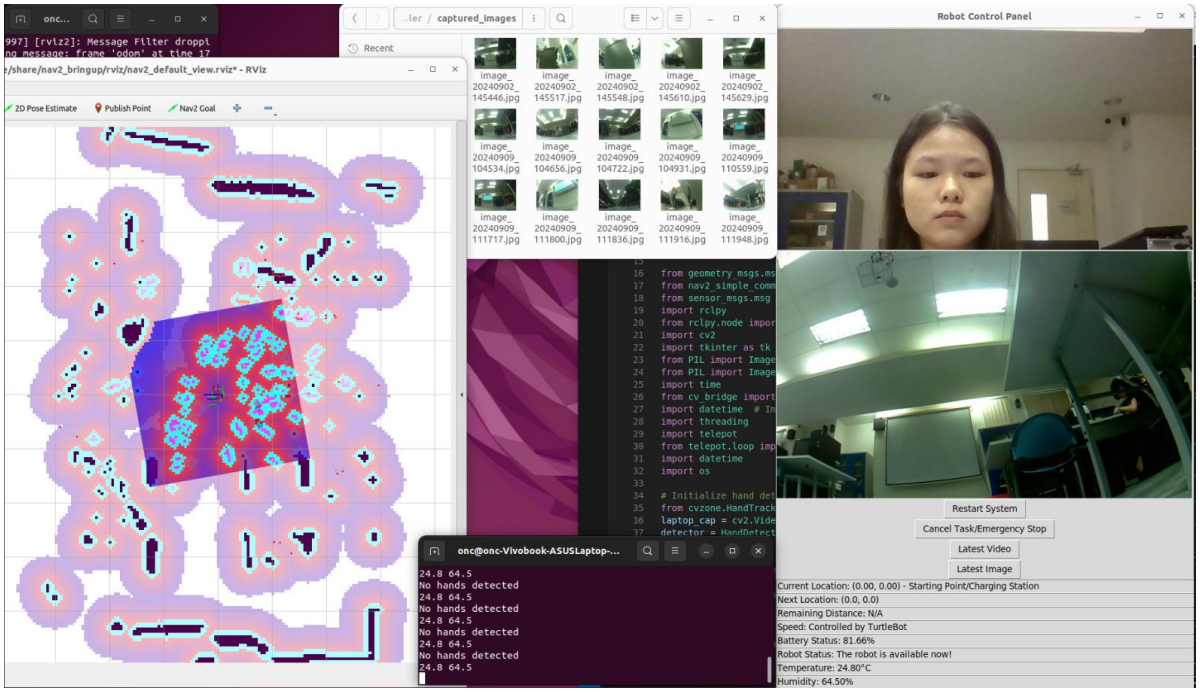


Figure 6-9 Complete work for the robot system with GUI

After running the Python script, a GUI appears with various components, including laptop_canvas, raspi_canvas, and several buttons (Restart, Cancel All Tasks, Display Latest Video, Display Latest Image). It displays information such as the robot's current and next location, remaining distance, speed, battery status, robot status, temperature, and humidity. The starting point is (0.0, 0.0). Most values on the GUI are updated in real-time through ROS 2 topics by publishing or subscribing to the relevant data.

```python
# Initialize ROS 2 node
def init_ros():
    rclpy.init()
    navigator = BasicNavigator()
    node = rclpy.create_node('robot_controller')

    # Subscribe to battery status topic
    node.create_subscription(
        BatteryState,
        '/battery_state',
        battery_callback,
        10
    )

    # Subscribe to temperature status topic
    node.create_subscription(
        Temperature,
        '/temperature',
        temperature_callback,
        10
    )

    # Subscribe to humidity status topic
    node.create_subscription(
        RelativeHumidity,
        '/humidity',
        humidity_callback,
        10
    )

    global velocity_publisher
    velocity_publisher = node.create_publisher(
        Twist,
        '/cmd_vel',
        10
    )

    # Subscribe to Raspberry Pi camera topic
    node.create_subscription(
        Image,
        '/image_raw',
        live_callback,
        10
    )

    # Subscriber for AMCL pose to get the current location
    node.create_subscription(
        PoseWithCovarianceStamped,
        '/amcl_pose',
        pose_callback,
        10
    )
```

Figure 6-10 python code to publish and subscribe to ros2 topics

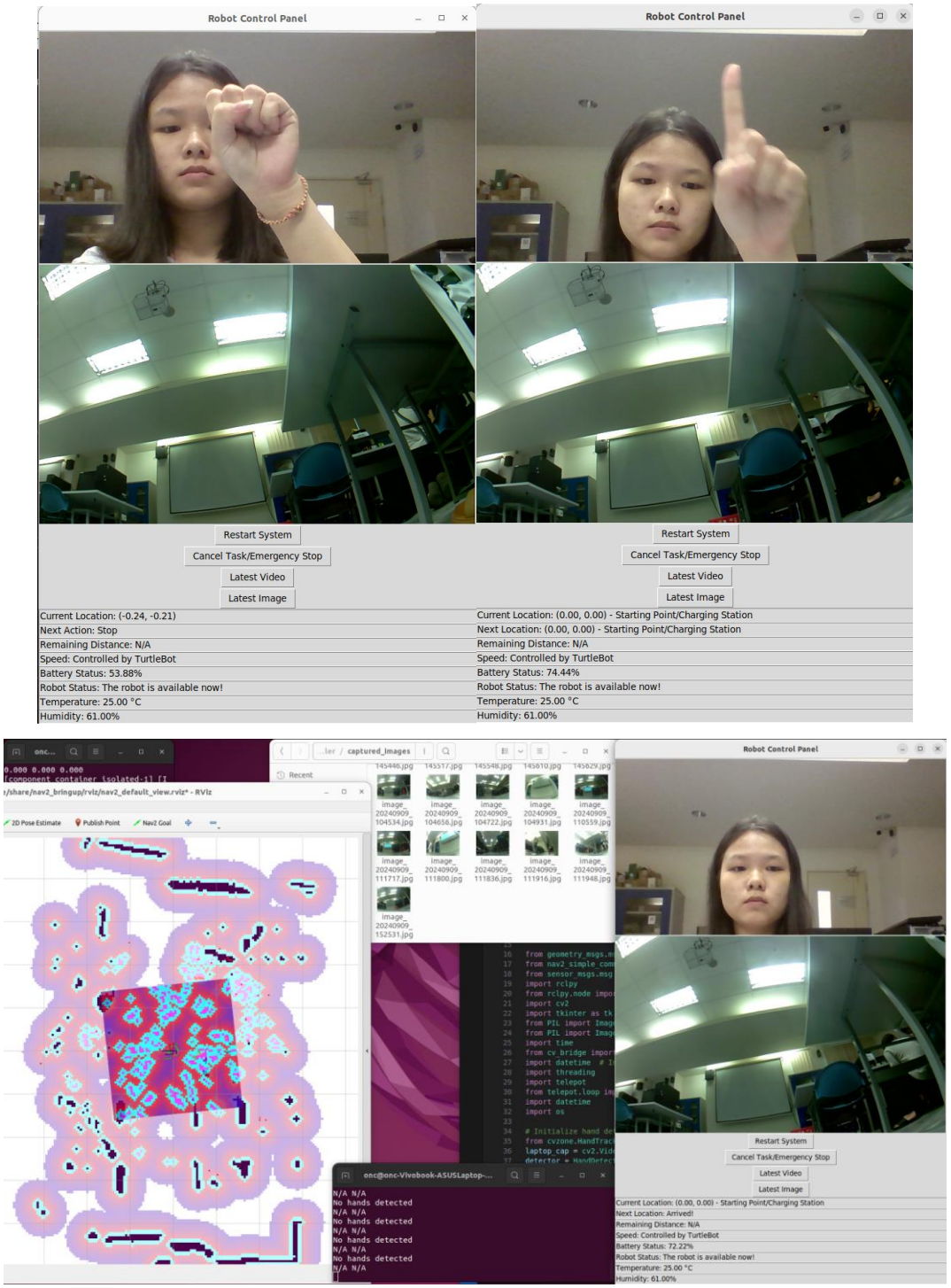


Figure 6-11 Robot system's GUI (1)

When the hand gesture "0" is detected from the laptop canvas, the robot will stop immediately, and the GUI will update the next action to "stop." Upon detecting gesture "1," the robot will navigate to the coordinates (0.0, 0.0), update the next location, status, and remaining distance, then capture an image upon arrival and mark the location as "done." The same process of capturing an image and updating the next location to "done" applies to gestures "2," "3," "4," and "OK."

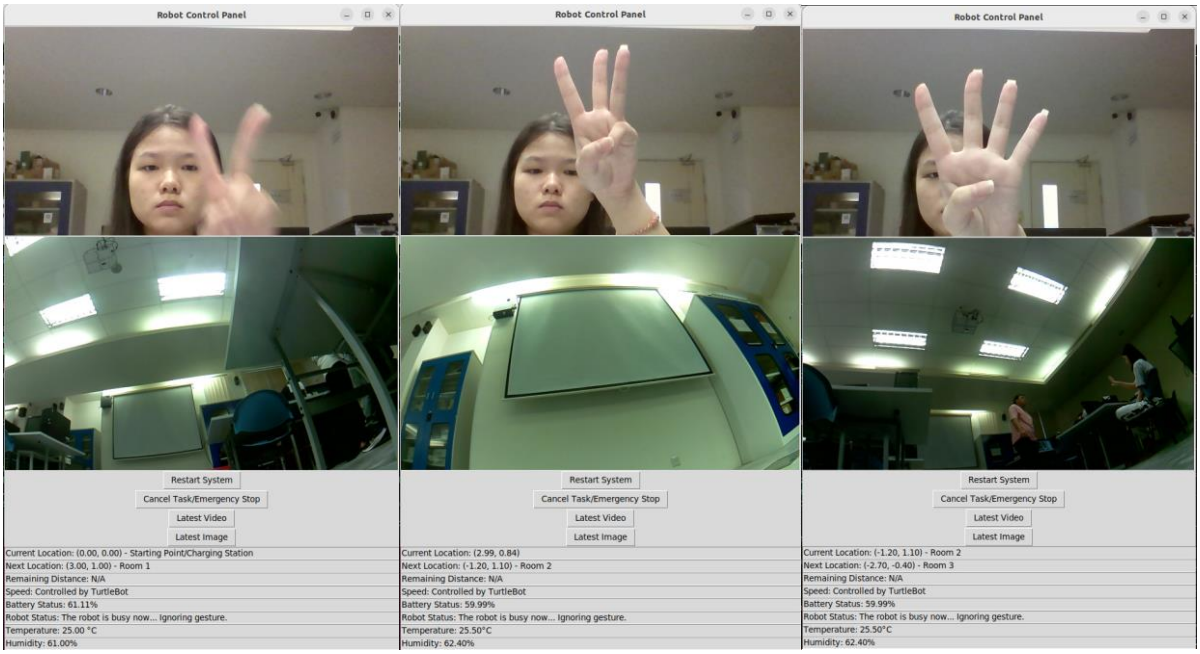


Figure 6-12 Robot system's GUI (2)

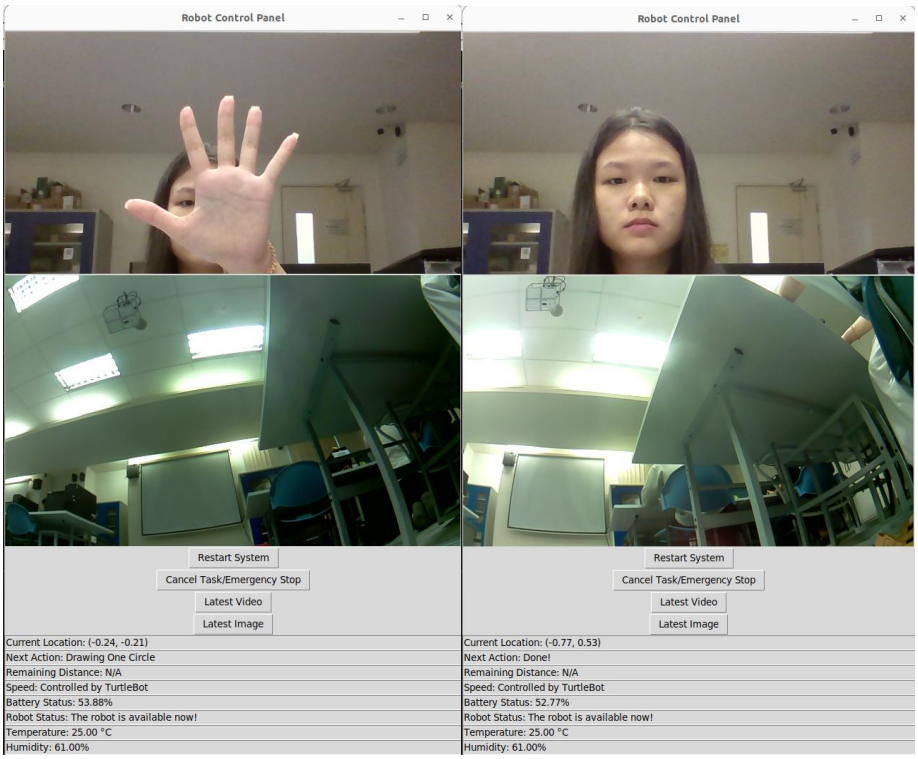


Figure 6-13 Robot system's GUI (3)

When gesture number 5 is detected, the TurtleBot3 will move in a circular path once and then stop. Simultaneously, the GUI will update to reflect this action.

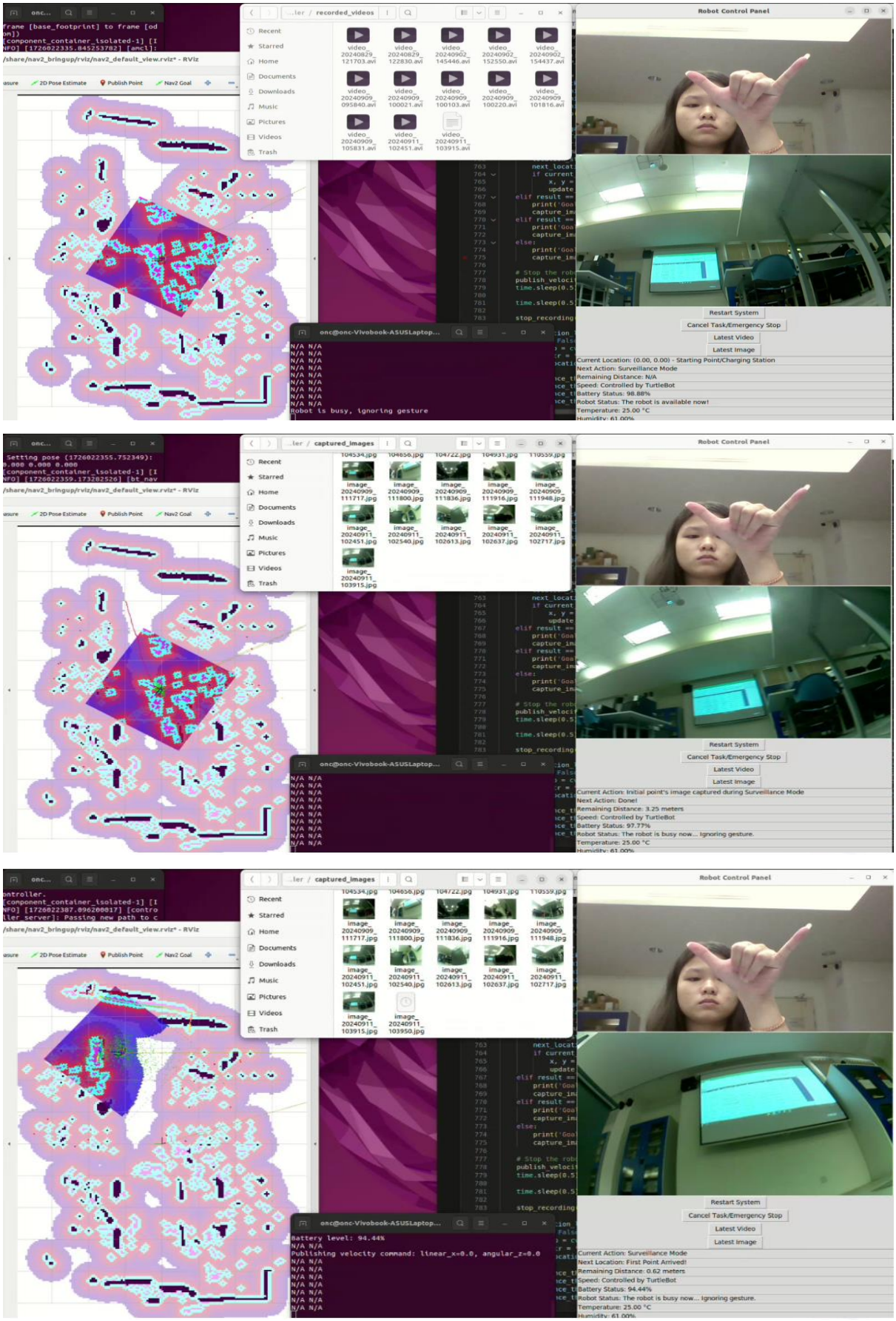


Figure 6-14 Robot system's GUI (4)

Figure 6-15 Robot system's GUI (5)

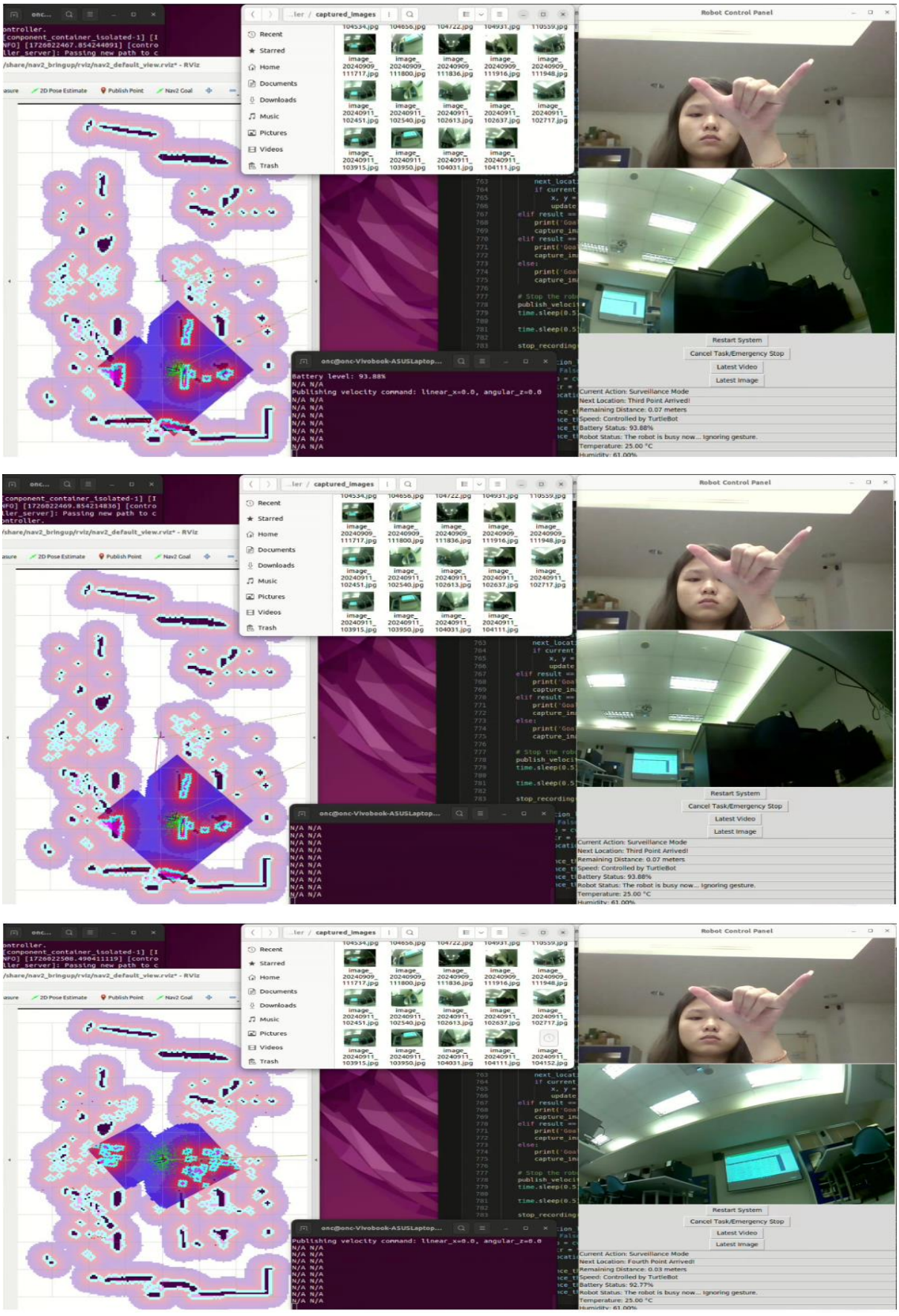


Figure 6-16 Robot system's GUI (6)

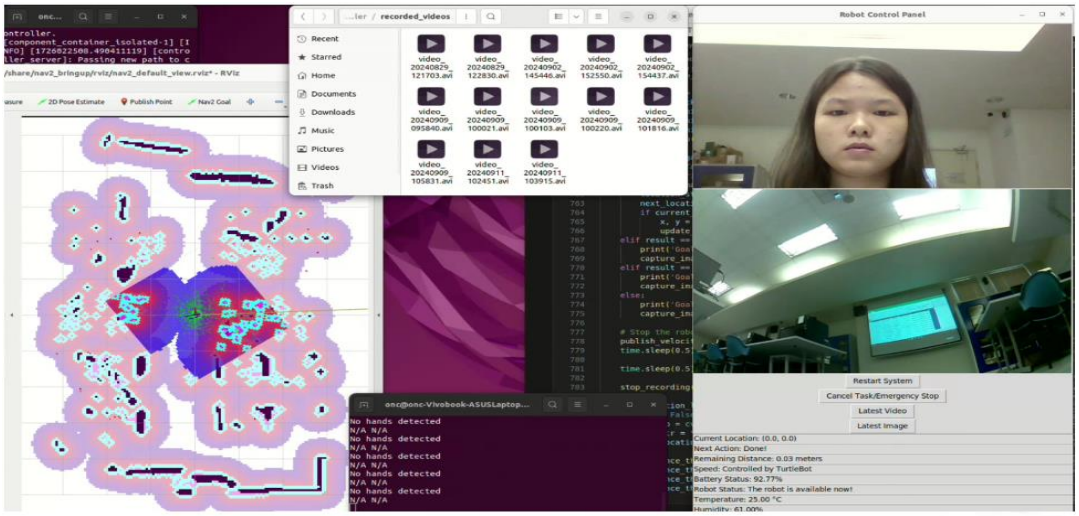


Figure 6-17 Robot system's GUI (7)

When gesture number 6 is detected, the TurtleBot3 activates surveillance mode. It begins recording and captures an initial image, then navigates through a series of waypoints: (0.0, 0.0) to (3.00, 1.00), followed by (-1.20, 1.10), then (-2.70, -0.4), and finally returns to the initial point (0.0, 0.0). An image is captured at each waypoint, and the recording stops upon returning to the initial point. The GUI updates after each action.

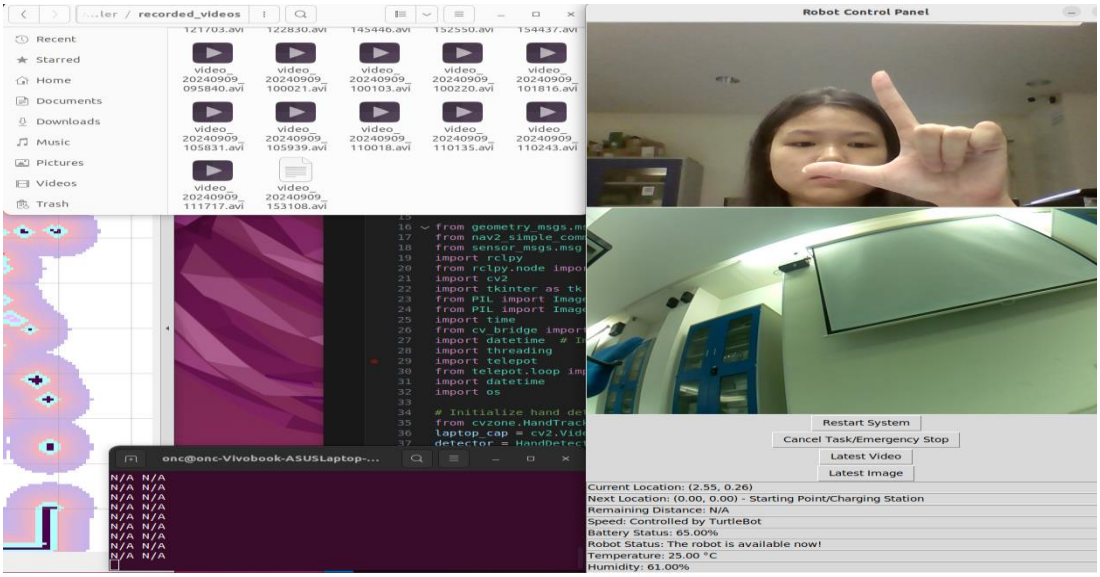


Figure 6-18 Robot system's GUI (8)

When gesture numbers 7, 8, 9, or 10 are detected, the robot will start recording and "7" will move to the location (0.0, 0.0). It will update its next location, status, and remaining distance during the movement. Once it arrives, the robot stops recording and updates the next location status to "done." The recording and status update functions are consistent across all four gestures.

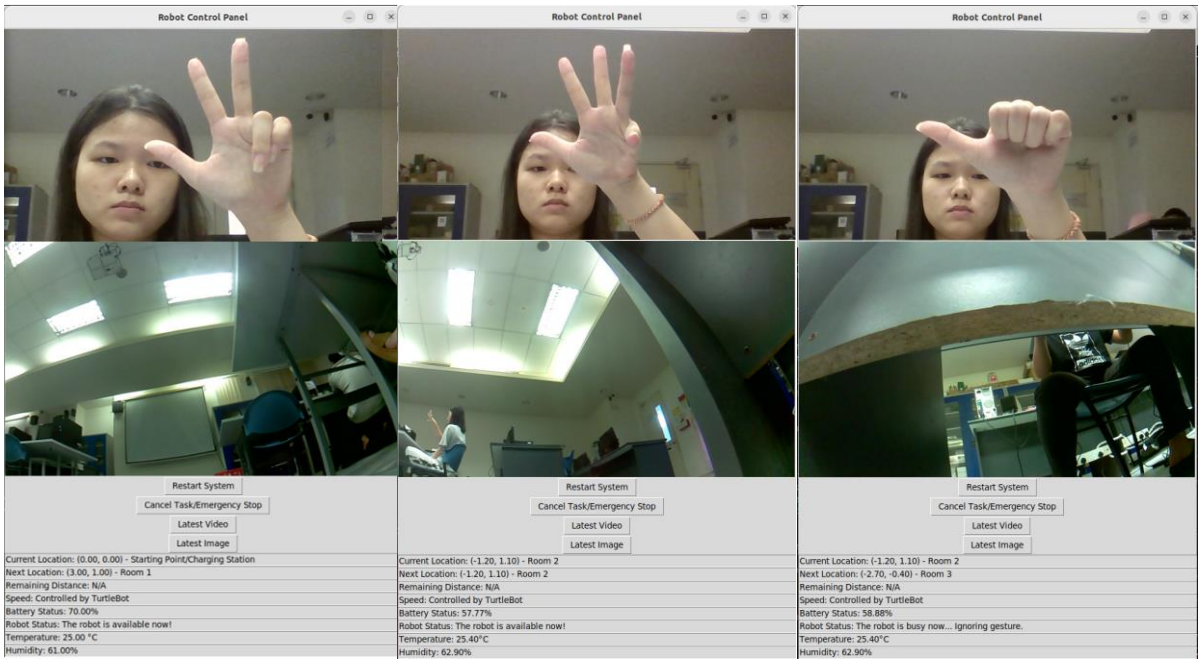


Figure 6-19 Robot system's GUI (9)

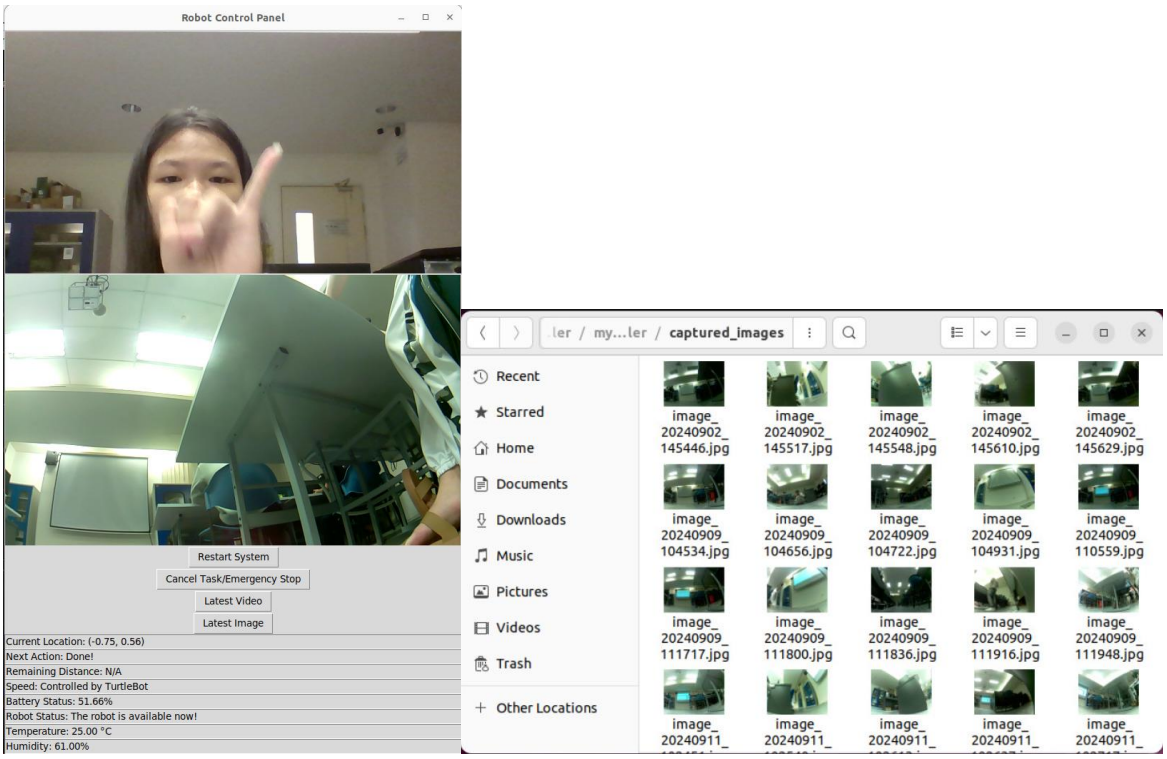


Figure 6-20 Robot system's GUI (10)

When gesture number 11 is detected, an image is immediately captured and saved to a designated directory. The GUI then updates to display the next action as "done" once the image capture is completed.

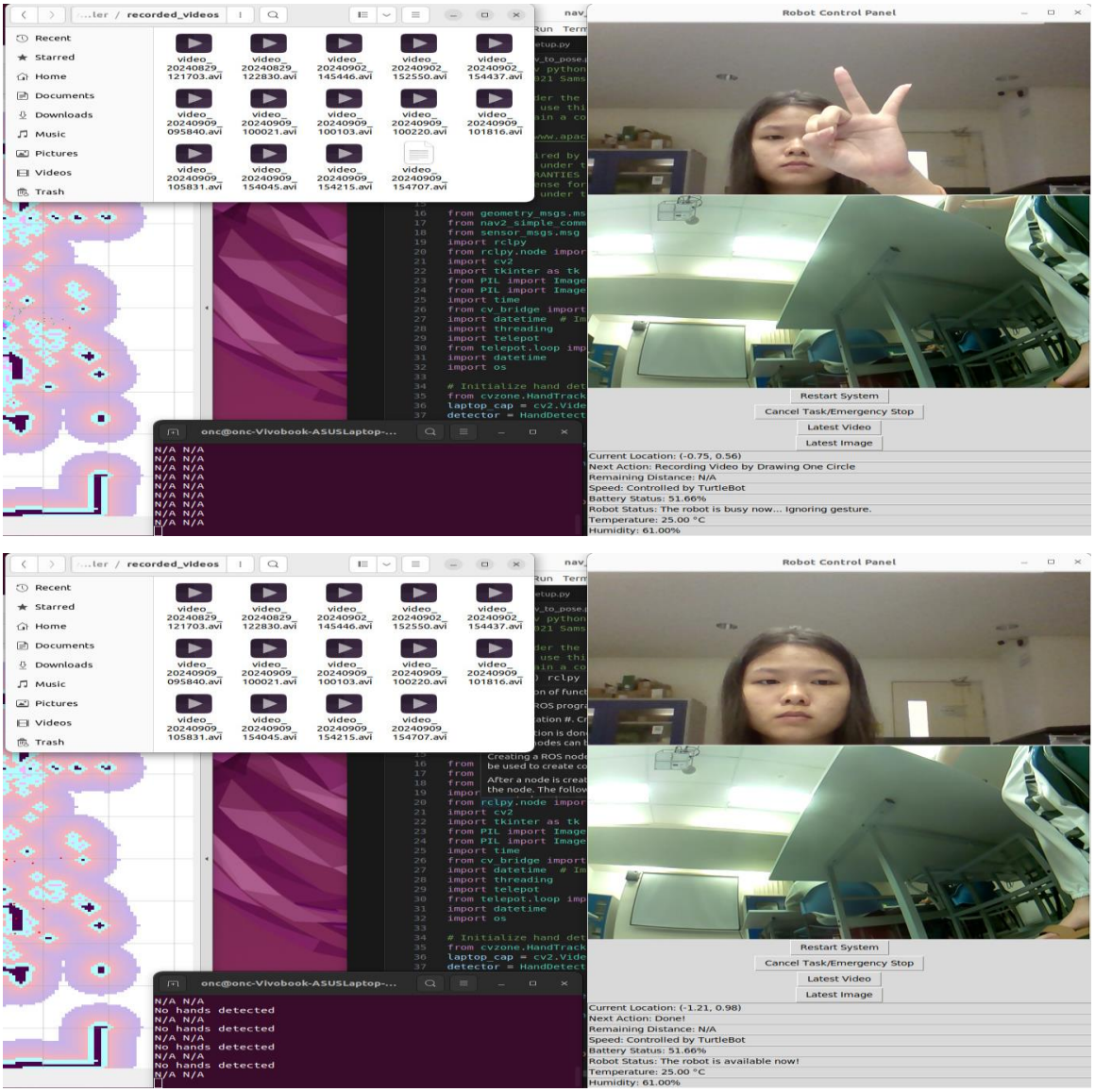


Figure 6-21 Robot system's GUI (11)

When gesture number 12 is detected, the TurtleBot3 will start recording and begin moving in a circular path. Recording will stop once the TurtleBot3 halts. Additionally, the GUI will update to show "Done" on the next action.

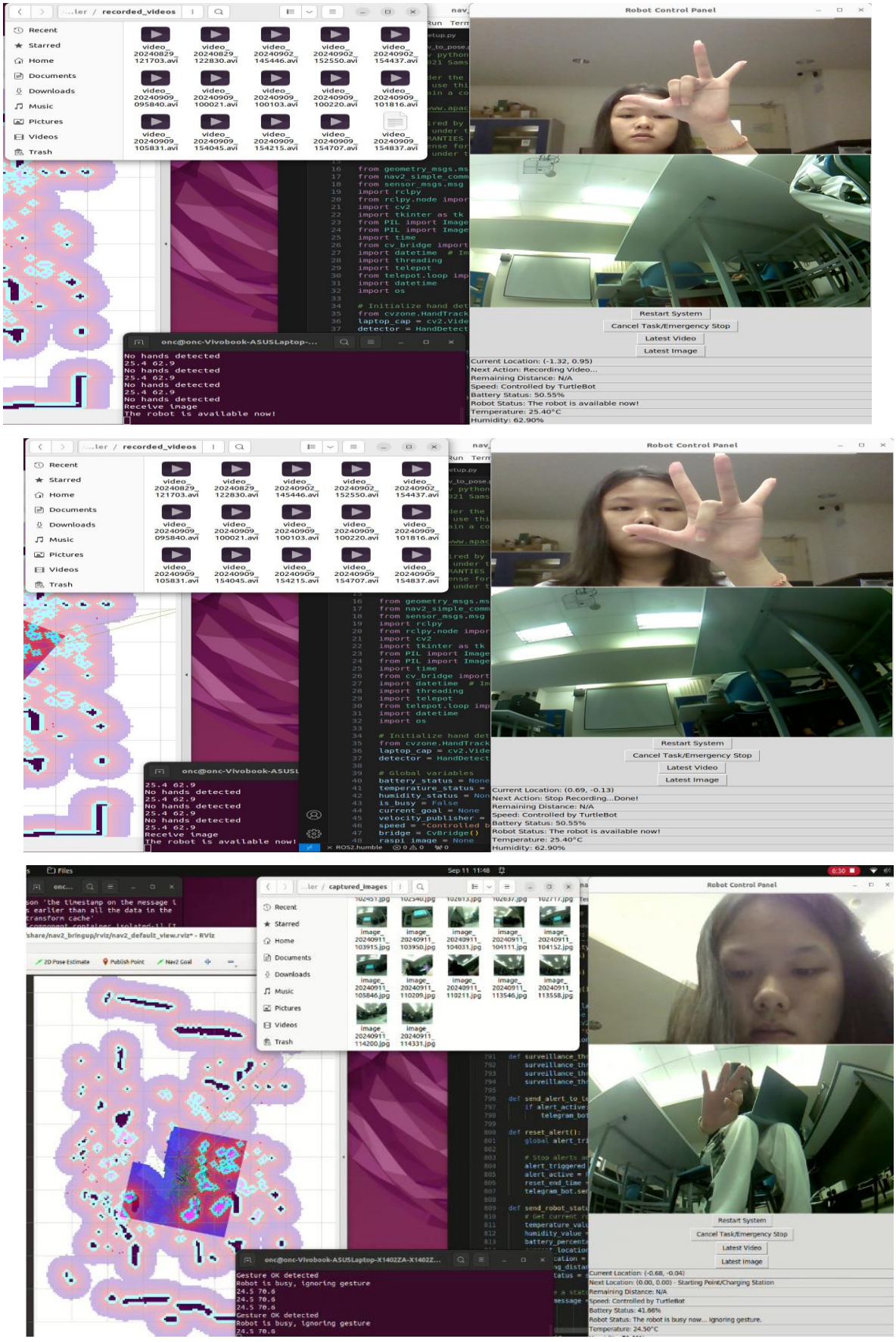


Figure 6-22 Robot system's GUI (12)

In this system, gesture number 13 initiates recording, while gesture number 14 halts it. The "OK" gesture, detected by the Raspi_canvas, commands the TurtleBot3 to return to its starting point (0.0, 0.0). This functionality is beneficial for business applications, such as in restaurants, where a robot can deliver food to tables. Customers can signal the robot with the "OK" gesture to prompt it to return to the counter for further tasks.

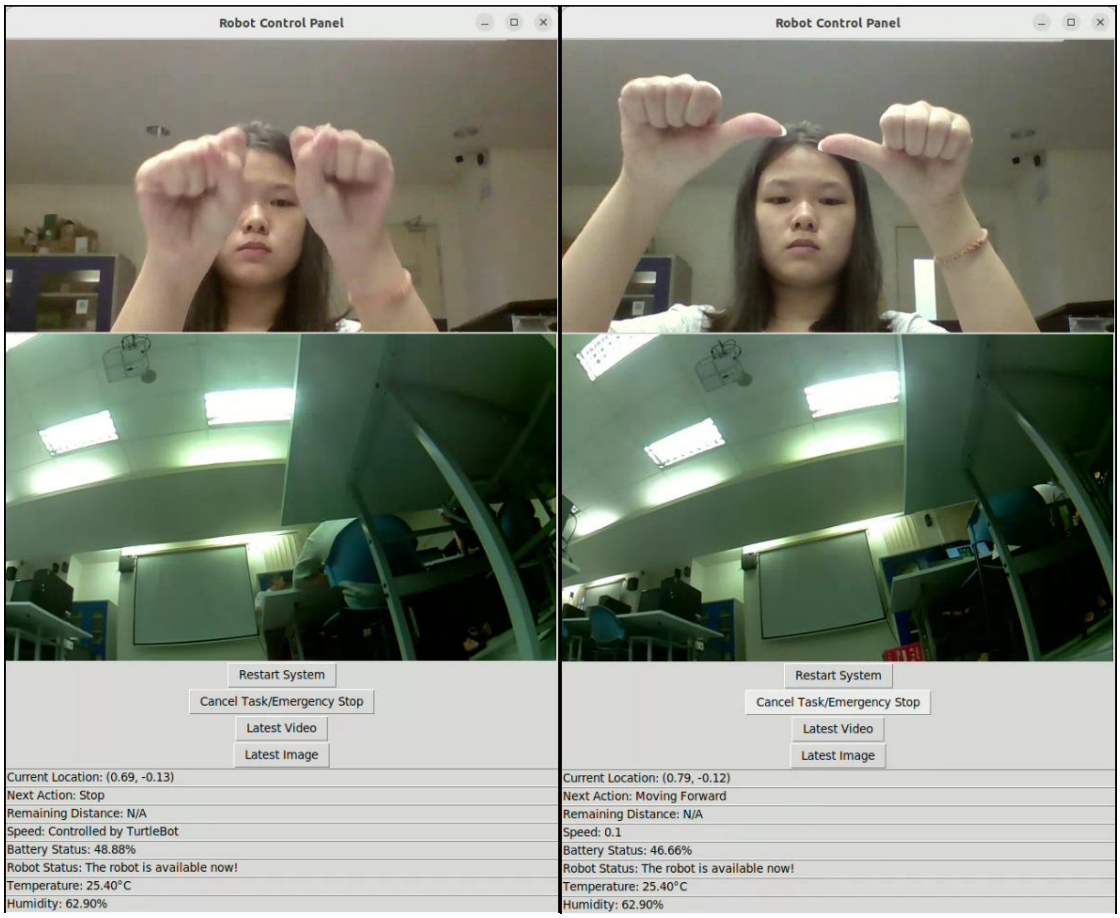


Figure 6-23 Robot system's GUI (13)

When two hands are detected, the system will respond according to Table 16-1 with commands to stop, move forward, move backward, turn left, turn right, speed up, or slow down. Additionally, the speed displayed on the GUI will be updated accordingly.

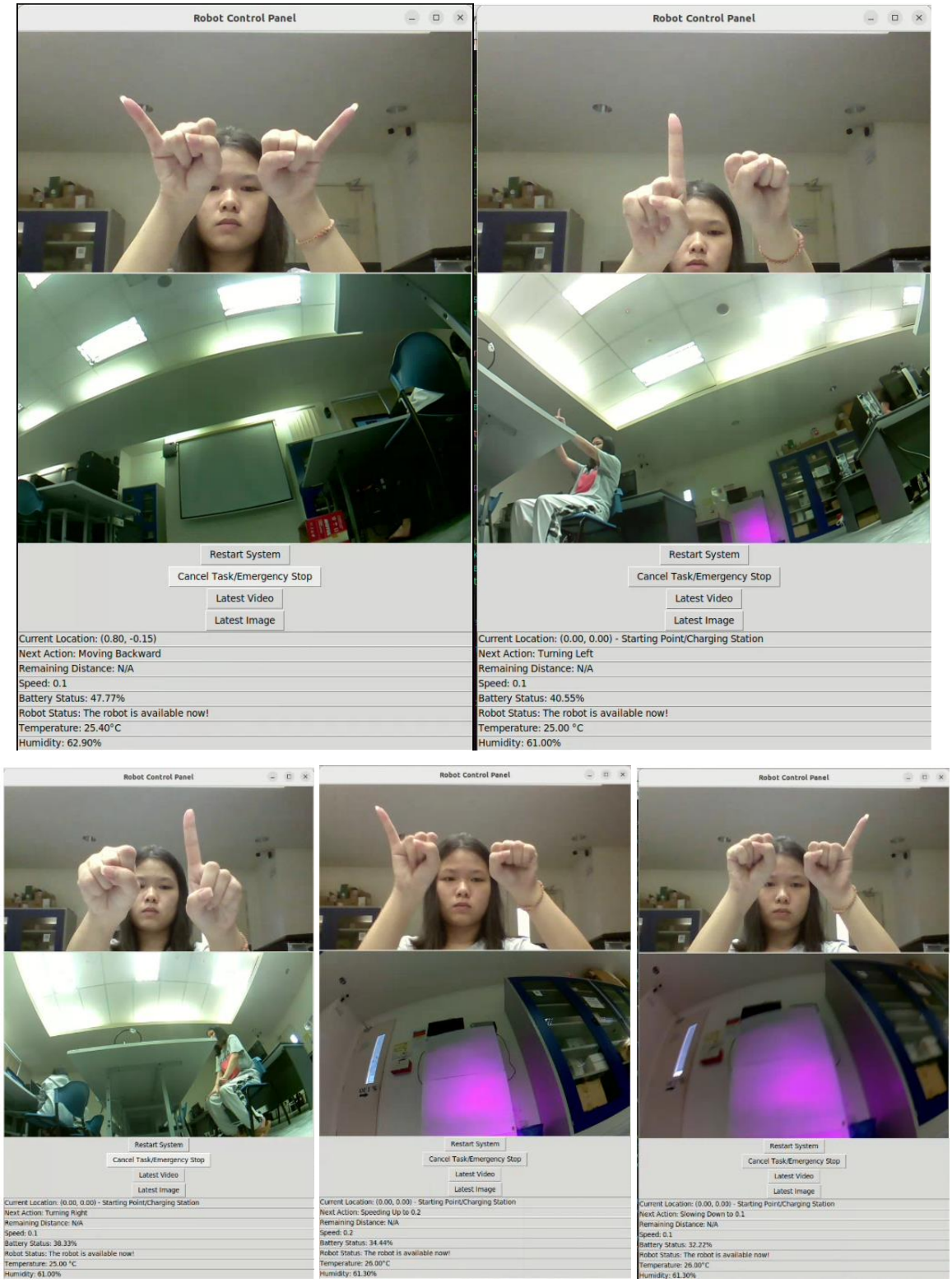


Figure 6-24 Robot system's GUI (14)

Other GUI functionalities

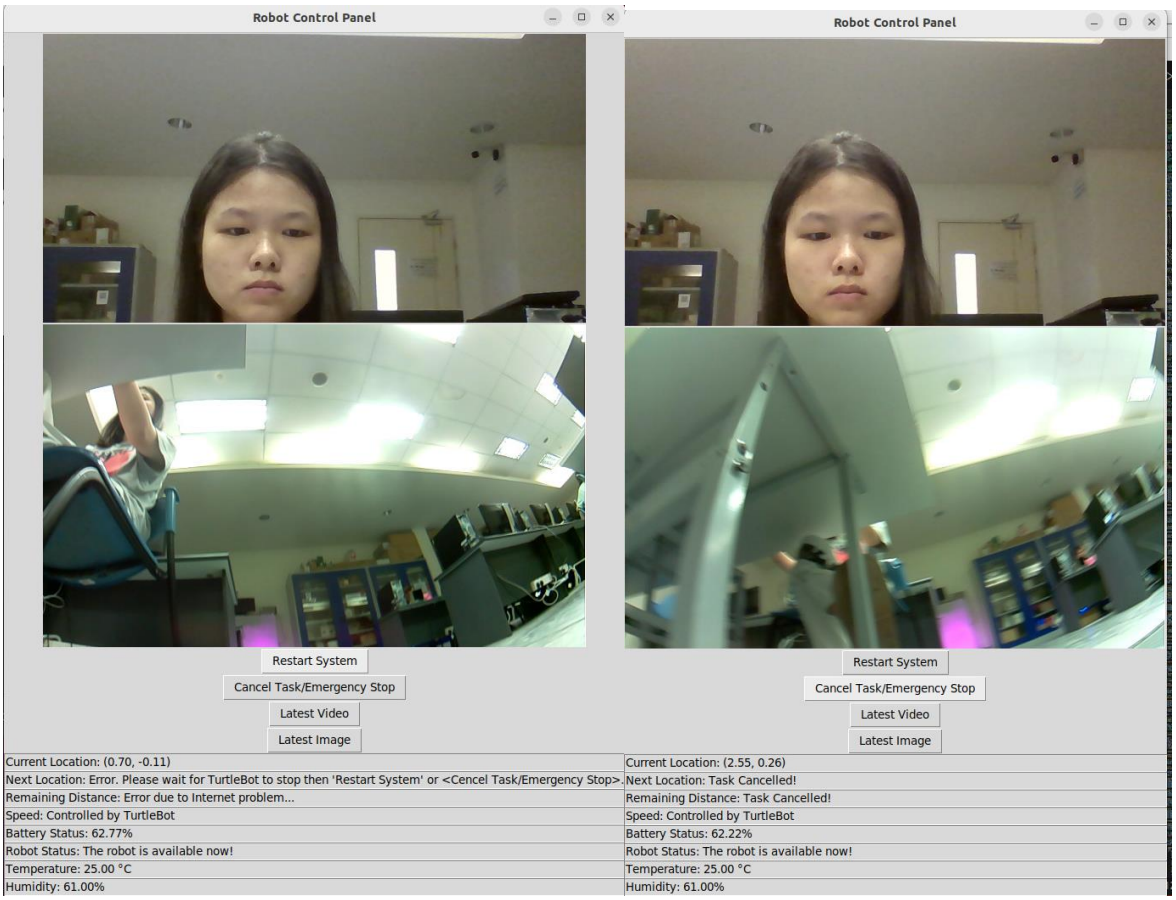


Figure 6-25 Robot system's GUI (15)

When an internet issue prevents accessing the remaining distance during navigation, users can either wait for the TurtleBot3 to reach its goal and then click the "restart system" button to resume normal operations, or they can directly click the "cancel task/emergency stop" button to immediately halt all tasks and stop the TurtleBot3 if an error occurs.

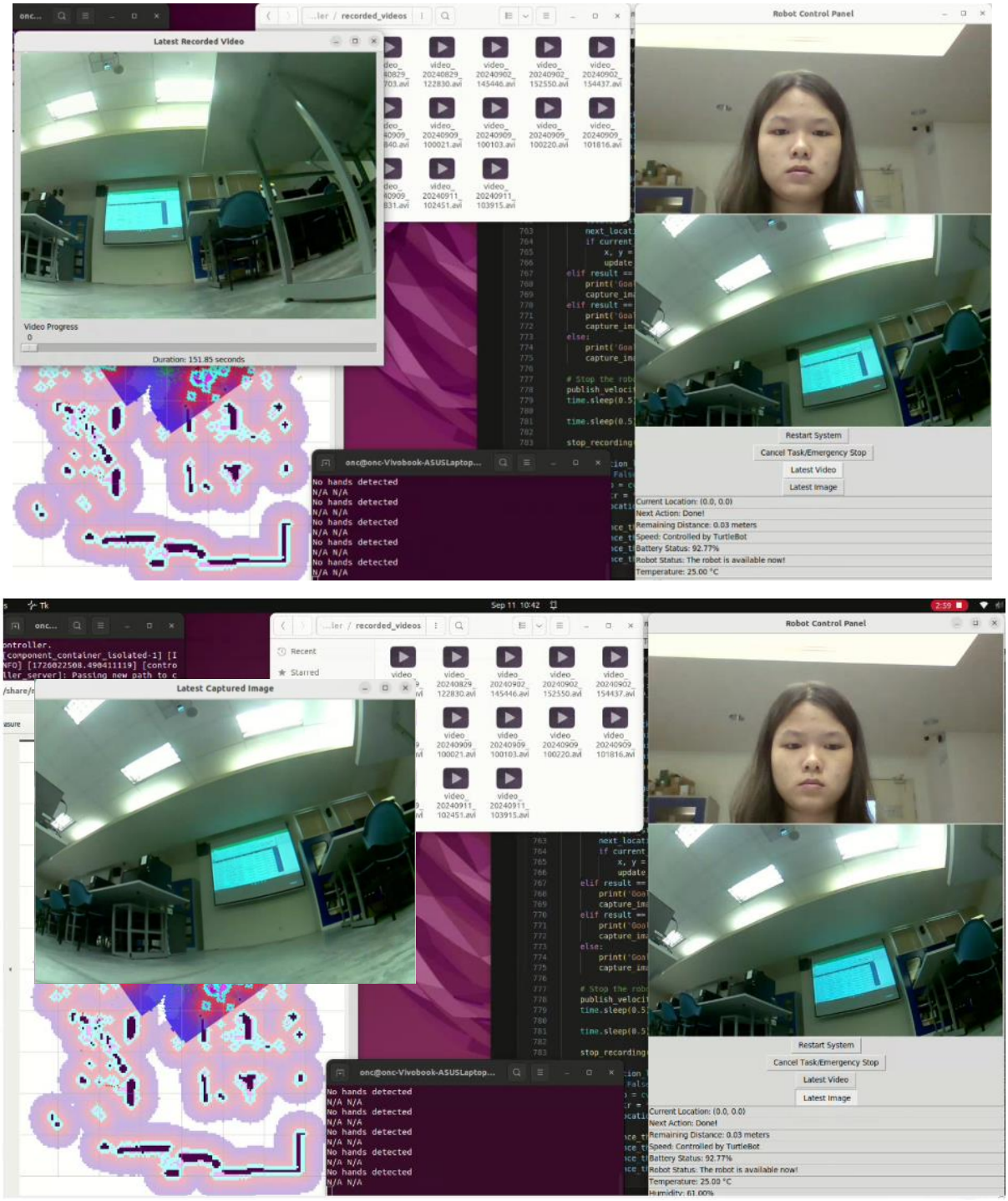


Figure 6-26 Robot system's GUI (16)

The system allows users to view the most recent captured image or recorded video by clicking the "latest image" or "latest video" buttons. If no image or video has been captured or recorded since the system started, a window will display a "no video/image found" message.

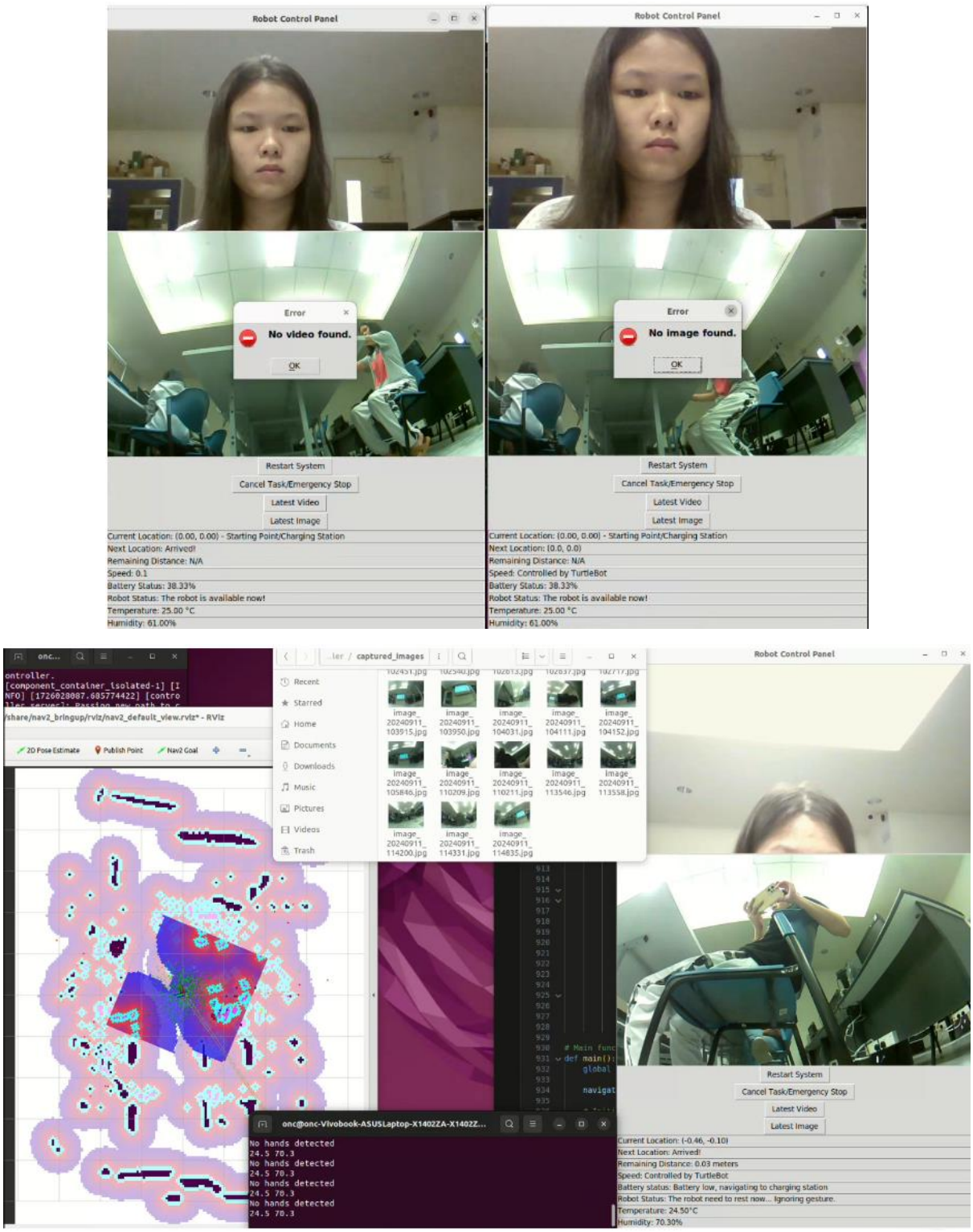


Figure 6-27 Robot system's GUI (17)

When the TurtleBot3's battery is low, it will automatically return to its charging station located at coordinates (0.0, 0.0). Additionally, it will notify users to charge the robot via the GUI.

Telegram functionalities

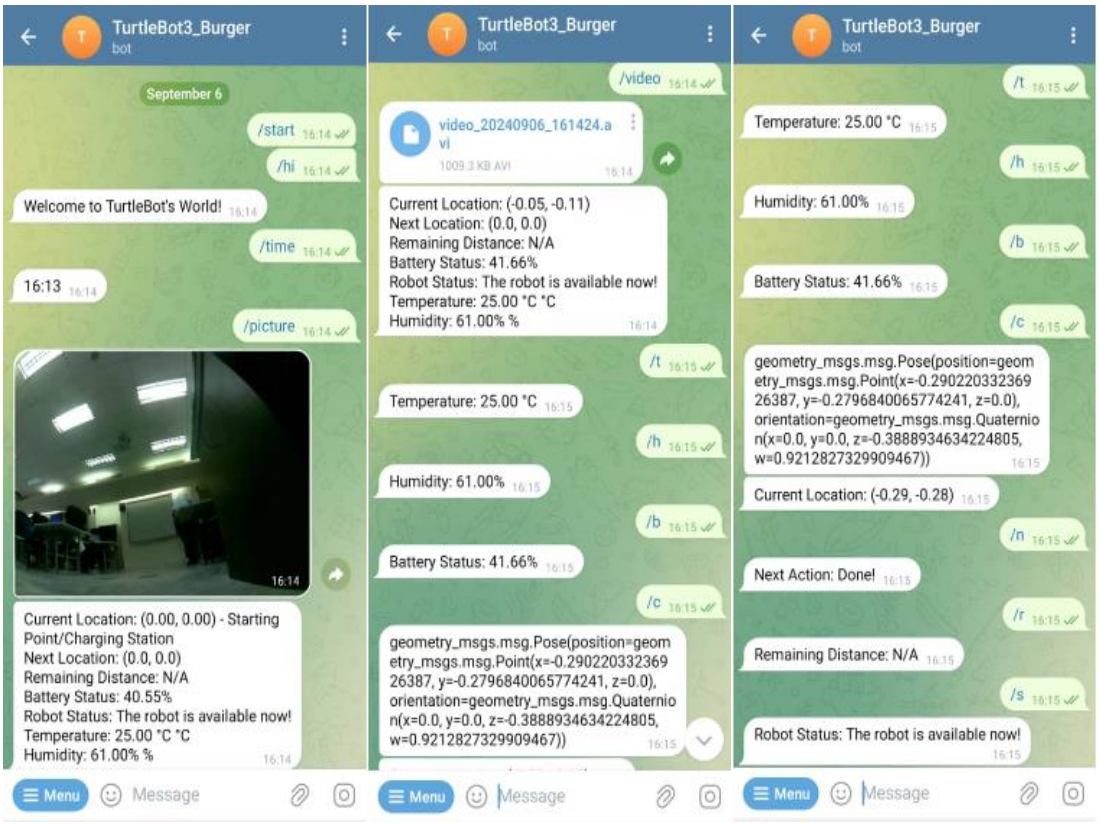


Figure 6-28 Results of Telegram (1)

When users select data command from the Telegram command menu, they receive information from the robot system or can control the TurtleBot3.

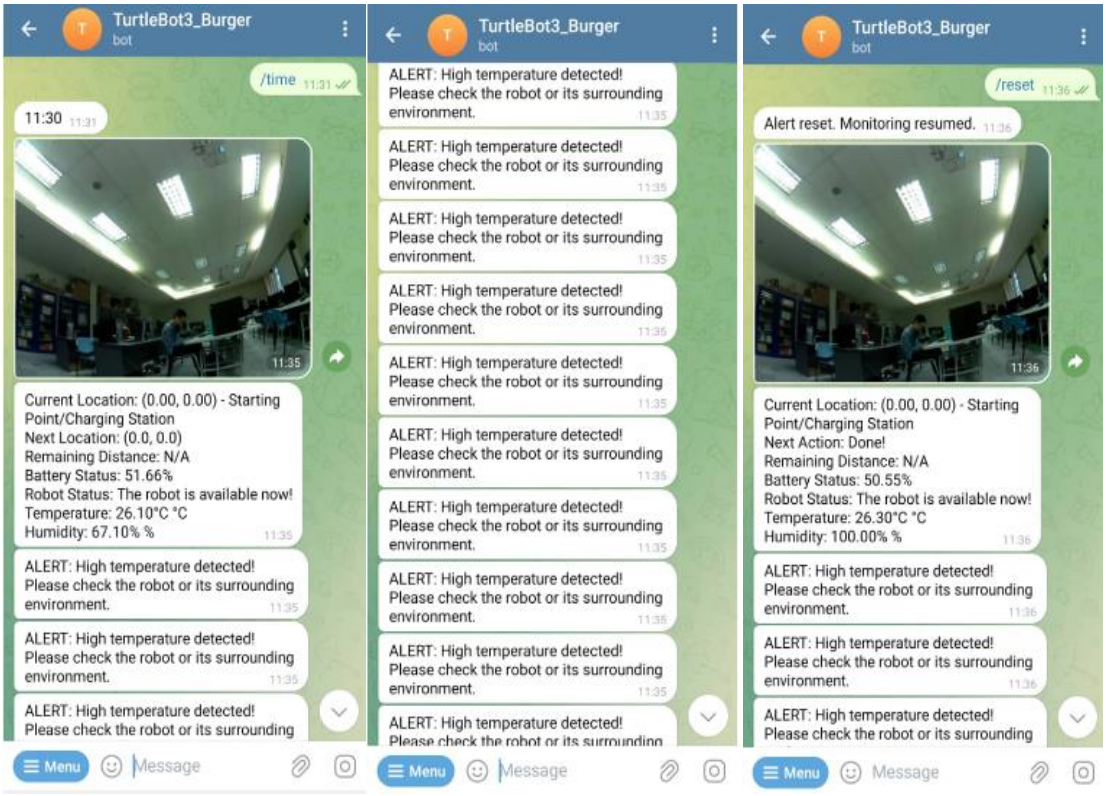


Figure 6-29 Results of Telegram (2)

When the DHT22 detects an overheat environment, the system will notify users or their contacts via a Telegram bot. Initially, an image along with data values will be sent, followed by continuous alert messages until the user issues the /reset command. After resetting, the alerts will pause for 10 seconds and resume the same steps if the overheat condition persists again; otherwise, they will stop sending alert messages.

Figure 6-30 Results of Telegram (3)

The TurtleBot3 will capture images or record videos whenever a hand gesture is detected. These images or videos will be sent to a Telegram bot for further processing or review.

6.3 Project Challenges

The main challenges encountered in this project were related to the integration of multiple technologies and the development of a highly responsive system. Specific challenges include:

1. **Gesture Recognition Accuracy**: It found difficult to achieve consistently high accuracy in gesture detection across a wide range of locations, lighting situations, and human variances. To ensure that the system could recognize gestures consistently in real time, the deep learning model needed to be optimized and fine-tuned for varied inputs.
2. **Real-Time Performance**: Real-time performance was important for effective human-robot interactions. Reducing latency when processing hand gestures proved difficult, especially on resource-constrained platforms such as Raspberry Pi. To minimize delays, the system requires thorough optimization of both hardware and software components.
3. **Hardware and Software Integration**: Combining hardware components like TurtleBot3, Raspberry Pi, DHT22 sensor, and a camera module with software (ROS2 and custom gesture recognition models) created compatibility concerns that had to be addressed through debugging and adjustments.
4. **Environmental Sensitivity**: Environmental factors such as lighting and background noise influenced the system's effectiveness, affecting gesture detection and robot navigation. Robustness under different situations necessitated periodic system adjustment.
5. **Internet Connectivity Issues**: Delays were also caused by inconsistent or slow internet connections, especially when receiving the stream video from raspberry pi camera and the system employed IoT and cloud-based services for remote monitoring and control. This impacted real-time performance and responsiveness, demanding increases in network stability and connection speed to ensure smooth operations.

6.4 Objectives Evaluation

The objectives of the project were successfully evaluated through extensive testing. Below is a summary of the results:

1. **Accuracy of Gesture Recognition**: The gesture recognition system achieved an average accuracy of **70-100%**, depending on the gesture and the surrounding conditions. For simple movements like "Move forward," accuracy was rather high, but it fell for more complex gestures or in darker situations.
2. **Real-Time Performance**: The system's response time from gesture recognition to robot action was calculated to be an average of **0.003162 seconds**, making the interactions practically real-time. This performance level fulfils the criteria for a smooth human-robot interaction.
3. **User Interface and Feedback**: The GUI gave real-time updates on system status, such as temperature, humidity, and robot operations. Users could simply control the robot with hand gestures, and feedback mechanisms such as **GUI and IoT** contributed to a user-friendly experience.
4. **Gesture-to-Action Mapping**: All defined gestures were **successfully** mapped to their corresponding robot movements. The technology accurately converted hand signals into navigation and other commands, allowing for efficient interaction with the robot.

6.5 Concluding Remark

In conclusion, the research met its primary goal of creating a real-time gesture detection system for human-robot interaction. Despite technological hurdles, the system was successfully installed, with acceptable levels of accuracy and responsiveness. The combination of deep learning, IoT, and robotics resulted in an easy framework for commanding the robot via natural gestures. Future enhancements should focus on improving system robustness in a variety of situations and optimizing hardware for broader applications.

Chapter 7

CONCLUSION AND RECOMMENDATION

7.1 Conclusion

In conclusion, by creating a real-time gesture detection system, this project has addressed the urgent demand for improved human-robot interaction. The first issue arose from the shortcomings of conventional human-robot interfaces, which frequently depend on laborious input techniques and obstruct smooth communication between humans and robots. Driven by the opportunity to optimize and enhance this communication, the project aimed to develop a solution utilizing cutting-edge technologies and techniques.

Convolutional neural networks (CNNs) [30], the deep learning models, was the main component of the suggested solution for gesture identification challenges. The system aims to enable natural communication between people and robots by utilizing deep learning to accurately understand and respond to human gestures in real-time. By leveraging cutting-edge technologies such as deep learning, computer vision, and IoT, the system allowed for intuitive, non-verbal communication with a robot, enhancing the user experience.

Many original concepts were developed during the project, mostly in the fields of model optimization and data pre-processing. Methods like model quantization and data augmentation were investigated in order to improve the robustness and effectiveness of the gesture detection system. Furthermore, enhancing the overall user experience and system performance was greatly aided by the integration of feedback systems and user interface design. The successful implementation of gesture-to-action mapping and a user-friendly interface facilitated smooth and efficient interactions.

In summary, by creating a real-time gesture detection system, the project has advanced human-robot interaction significantly. Through tackling the stated issue and utilizing creative solutions, the project has established the groundwork for further investigation and advancement in this fascinating domain. The future of human-robot interaction holds great promise due to the immense possibility for additional invention and improvement of gesture recognition systems as technology progresses. Overall, the project's objectives were met, contributing to advancements in human-robot interaction.

7.2 Recommendation

For future improvements, the following recommendations are made:

1. **Enhancing Accuracy in Diverse Conditions**: The system's accuracy can be increased by increasing the training dataset to cover a wider range of hand gestures and surrounding variables. Implementing adaptive learning models may enable the system to self-improve over time.
2. **Optimizing Hardware for Real-Time Performance**: Using more powerful hardware platforms with more processing capacity, such as GPUs, could reduce latency and speed up gesture detection and robot reaction.
3. **Improving Environmental Robustness**: The system should be strengthened by including sensors that adapt for environmental factors such as lighting and background noise. Noise reduction and multi-modal sensor integration techniques have the potential to boost dependability even more.
4. **Expanding System Applications**: The gesture recognition system can be used to other fields, such as healthcare, education, and entertainment, where intuitive, hands-free interaction with robots is extremely advantageous.
5. **Enhancing Internet Connectivity**: To overcome delays caused by internet connectivity concerns, it is advised to invest in more robust and quicker internet connections, particularly when employing IoT and cloud-based services. Edge computing, which processes data locally, can also help reduce reliance on internet bandwidth while maintaining real-time responsiveness in the system.

These recommendations aim to further enhance the usability, scalability, and performance of the system for real-world applications.

REFERENCES

[1] D. Translations, “Is Video Game Development Different by Country?,” Day Translations Blog, May 17, 2019. https://www.daytranslations.com/blog/gamedevelopment-country/

[2] IBM, “What is Natural Language Processing? | IBM,” www.ibm.com, 2023. https://www.ibm.com/topics/natural-language-processing

[3] “Top 45 Artificial Intelligence (AI) Interview Questions & Answers,” Edureka. https://www.edureka.co/blog/interview-questions/artificialintelligence-interview-questions/

[4] S. Goyal, “The History of C++ (With Timeline Infographic) // Unstop (formerly Dare2Compete),” unstop.com. https://unstop.com/blog/history-ofcpp

[5] “Programming with MATLAB,” www.mathworks.com. https://www.mathworks.com/products/matlab/programming-withmatlab.html#:~:text=MATLAB%20is%20a%20high%2Dlevel

[6] “Why should we use MATLAB (Matrix Laboratory),” Yorku.ca, 2019. https://www.yorku.ca/jdc/Matlab/Lesson1.html

[7] R. Keim, “Understanding Colour Models Used in Digital Image Processing,” ALL ABOUT CIRCUITS, Aug. 17, 2018. https://www.allaboutcircuits.com/technical-articles/understanding-colormodels-used-in-digital-imageprocessing/#:~:text=The%20key%20to%20understanding%20RGB,%2C%20green%2C%20and%20blue%20light.

[8] C. Marcello, A. Eng, and Marangi, “POLITECNICO DI TORINO Development of an automated benchmark for the analysis of Nav2 controllers Candidate Federica SCHENA,” 2024. Accessed: Sep. 09, 2024. [Online]. Available: https://webthesis.biblio.polito.it/30975/1/tesi.pdf

[9] Y. Name, “ROBOTIS e-Manual,” ROBOTIS e-Manual. https://emanual.robotis.com/docs/en/platform/turtlebot3/features/#specificatio ns

[10] opensource.com, “What is a Raspberry Pi?,” Opensource.com, 2012. https://opensource.com/resources/raspberry-pi

[11] “5MP Camera Module Jectse OV5647 5MP (Resolution 2592 x 1944) High Definition Camera Module Board Wide Angle 175° for Raspberry Pi B 3/2 : Amazon.de: Business, Industry & Science,” Amazon.de, 2024. https://www.amazon.de/-/en/Camera-Module-Resolution-Definition-Raspberry/dp/B07SXT1H8K (accessed Sep. 10, 2024).

[12] Available: https://www.sparkfun.com/datasheets/Sensors/Temperature/DHT22.pdf

[13] N. Gita, "What Is Ubuntu? A Quick Beginner's Guide," Hostinger Tutorials, Apr. 01, 2022. https://www.hostinger.com/tutorials/what-is-ubuntu

[14] "Ubuntu – Ubuntu Packages Search," packages.ubuntu.com. https://packages.ubuntu.com/

[15] "ROS 2 Documentation — ROS 2 Documentation: Foxy documentation," docs.ros.org. https://docs.ros.org/en/foxy/index.html

[16] "Client libraries — ROS 2 Documentation: Rolling documentation," docs.ros.org. https://docs.ros.org/en/rolling/Concepts/Basic/About-ClientLibraries.html

[17] E. Kim, J. Shin, Y. Kwon, and B. Park, "EMG-Based Dynamic Hand Gesture Recognition Using Edge AI for Human–Robot Interaction," Electronics, vol. 12, no. 7, p. 1541, Mar. 2023, doi: https://doi.org/10.3390/electronics12071541.

[18] J. E. Solanes, A. Muñoz, L. Gracia, and J. Tornero, "Virtual Reality-Based Interface for Advanced Assisted Mobile Robot Teleoperation," Applied Sciences, vol. 12, no. 12, p. 6071, Jun. 2022, doi: https://doi.org/10.3390/app12126071.

[19] M. Inayat and U. Khan, "Hand Gesture Detection & Recognition System." Available: https://www.diva-portal.org/smash/get/diva2:519237/FULLTEXT01.pdf

[20] J. Siby, "Hand Gesture Recognition," IJITR) INTERNATIONAL JOURNAL OF INNOVATIVE TECHNOLOGY AND RESEARCH, vol. 3, no. 2, pp. 1946–1949, 2015, Available: https://core.ac.uk/download/pdf/228547068.pdf

[21] Robotis e (no date) Manual. Available at: https://emanual.robotis.com/docs/en/parts/controller/opencr10/ (Accessed: 18 April 2024).

[22] GfG (2024) What is opencv library?, GeeksforGeeks. Available at: https://www.geeksforgeeks.org/opencv-overview/ (Accessed: 18 April 2024).

[23] What is visual studio code? (no date) Educative. Available at: https://www.educative.io/answers/what-is-visual-studio-code (Accessed: 18 April 2024).

[24] Simple commander API¶ (no date b) Simple Commander API - Nav2 1.0.0 documentation. Available at: https://navigation.ros.org/commander_api/index.html (Accessed: 18 April 2024).

[25] Robotis e (no date) Manual. Available at: https://emanual.robotis.com/docs/en/platform/turtlebot3/quick-start/ (Accessed: 18 April 2024).

[26] Instructables, “Set Up Telegram Bot on Raspberry Pi,” Instructables, Aug. 19, 2015. https://www.instructables.com/Set-up-Telegram-Bot-on-Raspberry-Pi/ (accessed Sep. 10, 2024).

[27] “- YouTube,” Youtu.be, 2024. https://youtu.be/va7o7wzhEE4?si=MxLgAZ8Bkb6oDIv3 (accessed Sep. 12, 2024).

[28] learning_topic/learning_topic/topic_webcam_sub.py • GuYueHome/ros2_21_tutorials - Gitee.com, “ learning_topic/learning_topic/topic_webcam_sub.py • GuYueHome/ros2_21_tutorials - Gitee.com, ” Gitee, 2022. https://gitee.com/guyuehome/ros2_21_tutorials/blob/master/learning_topic/learning_topic/topic_webcam_sub.py (accessed Sep. 12, 2024).

[29] J. Jordan, “Evaluating a machine learning model.,” Jeremy Jordan, Jul. 21, 2017. https://www.jeremyjordan.me/evaluating-a-machine-learning-model/

[30] L. Craig, “CNN vs. RNN: How are they different?,” SearchEnterpriseAI, Aug. 08, 2023. https://www.techtarget.com/searchenterpriseai/feature/CNN-vs-RNN-How-they-differ-and-where-they-overlap

FINAL YEAR PROJECT WEEKLY REPORT

(Project II)

Trimester, Year: T3, Y3	**Study week no.: 2**
Student Name & ID: ONG NIAM CHI 20ACB05969	
Supervisor: DR TEOH SHEN KHANG	
Project Title: DEVELOPMENT OF A REAL-TIME GESTURE RECOGNITION SYSTEM FOR HUMAN-ROBOT	

1. WORK DONE

[Please write the details of the work done in the last fortnight.]

This week's primary goal was to explore and better understand the technological requirements for constructing a real-time gesture detection system for human-robot interaction. Several internet resources, including tutorials and research articles, were used to gain information on programming approaches and machine learning algorithms suited for gesture recognition. In addition, instructional films from outlets such as YouTube were evaluated to help with TurtleBot3 configuration and knowledge of its sophisticated functionalities.

2. WORK TO BE DONE

Begin writing the basic code for the gesture detection system in Python, and then experiment with several algorithms to see which ones work best for the desired application. Prepare to create a rudimentary GUI to interact with the TurtleBot 3.

3. PROBLEMS ENCOUNTERED

Encountered difficulties in assessing the reliability of the online resources consulted, as there were inconsistencies in the information found. There was also uncertainty regarding the applicability of certain algorithms to the project's specific requirements.

4. SELF EVALUATION OF THE PROGRESS

The research phase was productive, providing a broad understanding of the requirements and challenges associated with gesture recognition systems. However, the need for more reliable and vetted resources became apparent, and further guidance from experienced professionals will be sought in the upcoming weeks to ensure the chosen approach is sound and applicable to the project.

TEOH

Supervisor's signature

Student's signature

FINAL YEAR PROJECT WEEKLY REPORT

(Project I)

Trimester, Year: T3, Y3	**Study week no.: 4**
Student Name & ID: ONG NIAM CHI 20ACB05969	
Supervisor: DR TEOH SHEN KHANG	
Project Title: DEVELOPMENT OF A REAL-TIME GESTURE RECOGNITION SYSTEM FOR HUMAN-ROBOT	

1. WORK DONE

[Please write the details of the work done in the last fortnight.]

This week's focus was on getting started developing the graphical user interface (GUI) for the gesture recognition system. Efforts were focused on mapping hand gestures to navigation commands for the TurtleBot3. In order to efficiently implement the GUI, various libraries and programming frameworks were studied. Initial tests were carried out to determine the system's ability to read gestures and regulate the robot's motions.

2. WORK TO BE DONE

Continue to refine the GUI and improve the accuracy of gesture mapping with navigation commands. Plan to include more control features such as speed modification and robot status monitoring.

3. PROBLEMS ENCOUNTERED

The challenges were choosing the right tools and libraries for GUI development and guaranteeing interoperability with existing gesture detection techniques.

4. SELF EVALUATION OF THE PROGRESS

Good progress was achieved in building the GUI's core features. However, additional work is required to improve gesture recognition accuracy and ensure seamless connection with the TurtleBot3 control system.

TEOH

Supervisor's signature

Student's signature

FINAL YEAR PROJECT WEEKLY REPORT

(Project I)

Trimester, Year: T3, Y3	**Study week no.: 6**
Student Name & ID: ONG NIAM CHI 20ACB05969	
Supervisor: DR TEOH SHEN KHANG	
Project Title: DEVELOPMENT OF A REAL-TIME GESTURE RECOGNITION SYSTEM FOR HUMAN-ROBOT	

1. WORK DONE

[Please write the details of the work done in the last fortnight.]

Gestures have been successfully translated to the TurtleBot3 navigation commands. The system can now manage the robot's movement in all directions (forward, backward, left, and right), as well as its speed. New functions were introduced, such as displaying the robot's battery status, current stance, and speed through ros2 topic list. This represented a big step towards completing the gesture control system's fundamental functionalities.

2. WORK TO BE DONE

Begin integrating new sensors and improving system performance to ensure smoother functioning. Plan to add environmental sensors to the system's capabilities.

3. PROBLEMS ENCOUNTERED

Minor issues with synchronization between gesture commands and robot responses were observed, requiring adjustments in the code.

4. SELF EVALUATION OF THE PROGRESS

Overall, this week was successful as core functionalities were achieved. Future work will focus on optimizing performance and adding more features to enhance system capabilities.

TEOH

Supervisor's signature

Student's signature

FINAL YEAR PROJECT WEEKLY REPORT

(Project I)

Trimester, Year: T3, Y3	**Study week no.: 8**
Student Name & ID: ONG NIAM CHI 20ACB05969	
Supervisor: DR TEOH SHEN KHANG	
Project Title: DEVELOPMENT OF A REAL-TIME GESTURE RECOGNITION SYSTEM FOR HUMAN-ROBOT	

1. WORK DONE

[Please write the details of the work done in the last fortnight.]

This week's setback was caused by a corrupted microSD card, which required a system rebuild. The memory swap is still needed to make the system rebuild smoothly. Despite the delay, the system was successfully recovered, and work began to integrate the Raspberry Pi camera as a topic /image_raw in the ROS2 environment to record visual data for gesture detection and surrounding environment monitoring.

2. WORK TO BE DONE

Continue to refine camera integration and expand image processing capabilities to improve gesture detection accuracy from both laptop and raspberry pi cameras.

3. PROBLEMS ENCOUNTERED

The corruption of the microSD card resulted in downtime and a temporary halt in development. There were also minor issues with configuring the camera topic in ROS2.

4. SELF EVALUATION OF THE PROGRESS

Despite the unexpected delay, the ability to quickly rebuild the system and continue with the camera integration showed resilience. The progress is on track to complete further implementation tasks.

TEOH

Supervisor's signature

Chi

Student's signature

FINAL YEAR PROJECT WEEKLY REPORT

(Project I)

Trimester, Year: T3, Y3	**Study week no.: 10**
Student Name & ID: ONG NIAM CHI 20ACB05969	
Supervisor: DR TEOH SHEN KHANG	
Project Title: DEVELOPMENT OF A REAL-TIME GESTURE RECOGNITION SYSTEM FOR HUMAN-ROBOT	

1. WORK DONE

[Please write the details of the work done in the last fortnight.]

All capabilities, including gesture control, camera integration, and the GUI, were successfully implemented and tested. Threads were introduced to enable the system to work smoothly and without lag. In addition, a DHT22 sensor was added as /temperature and /humidity ros2 topics so that can be subscribed to monitor ambient conditions, increasing the system's capabilities.

2. WORK TO BE DONE

Connect the DHT22 sensor data to a remote monitoring platform like Telegram and make final refinements to the code.

3. PROBLEMS ENCOUNTERED

Some challenges were faced with optimizing the system for multi-threading, but these were resolved by adjusting the thread management strategy.

4. SELF EVALUATION OF THE PROGRESS

The project is progressing well, with all key functionalities now in place. The system is performing reliably, and the addition of multi-threading has improved overall performance.

TEOH

Supervisor's signature

Chi

Student's signature

FINAL YEAR PROJECT WEEKLY REPORT

(Project I)

Trimester, Year: T3, Y3	**Study week no.: 12**
Student Name & ID: ONG NIAM CHI 20ACB05969	
Supervisor: DR TEOH SHEN KHANG	
Project Title: DEVELOPMENT OF A REAL-TIME GESTURE RECOGNITION SYSTEM FOR HUMAN-ROBOT	

1. WORK DONE

[Please write the details of the work done in the last fortnight.]

We successfully linked the DHT22 sensor to Telegram for remote monitoring and finished the last coding adjustments. The system is now fully operational, with all expected functions such as gesture control, camera feed, and environmental monitoring.

2. WORK TO BE DONE

Conduct extensive testing to ensure that all capabilities work as intended under a variety of scenarios. Prepare for the final system evaluation.

3. PROBLEMS ENCOUNTERED

Minor issues arose in integrating the DHT22 sensor data with the Telegram platform, but these were resolved through debugging.

4. SELF EVALUATION OF THE PROGRESS

This week marked the completion of the development phase, with all objectives met. The system is ready for final testing and evaluation.

TEOH

Supervisor's signature

Student's signature

FINAL YEAR PROJECT WEEKLY REPORT

(Project I)

Trimester, Year: T3, Y3	**Study week no.: 13**
Student Name & ID: ONG NIAM CHI 20ACB05969	
Supervisor: DR TEOH SHEN KHANG	
Project Title: DEVELOPMENT OF A REAL-TIME GESTURE RECOGNITION SYSTEM FOR HUMAN-ROBOT	

1. WORK DONE

[Please write the details of the work done in the last fortnight.]

Focused on determining the system's correctness and reliability throughout various scenarios. The performance of gesture detection and robot control was thoroughly tested. The final report was created, describing every aspect of the project, from development to implementation.

2. WORK TO BE DONE

Complete the final report, considering input from testing, and prepare for any final presentations or demonstrations.

3. PROBLEMS ENCOUNTERED

Minor discrepancies were observed during testing, particularly under low-light conditions, which required some last-minute adjustments to the camera settings.

4. SELF EVALUATION OF THE PROGRESS

The project has reached its final stages successfully. The testing results were satisfactory, and the final report is nearing completion. The system is ready for deployment or further enhancements as needed.

TEOH

Supervisor's signature

Student's signature

POSTER

UNIVERSITI TUNKU ABDUL RAHMAN
FACULTY OF INFOTMATION AND COMMUNICATION TECHNOLOGY
Prepared by: Ong Niam Chi. Supervisor: DR TEOH SHEN KHANG

Development of a Real-Time Gesture Recognition System for Human-Robot Interaction

Abstract

This project focuses on developing a real-time gesture recognition system for human-robot interaction (HRI), integrating cutting-edge technologies like deep learning, computer vision, IoT, and robotics. The system enables intuitive communication between humans and robots through natural hand gestures, enhancing interaction in various fields such as home services, security, and entertainment. Despite challenges in ensuring accuracy, real-time performance, and hardware integration, the system demonstrates promising results in improving HRI.

Methodology, Phase 2

- Develop & Implement Gesture Recognition Model
- Integration of Gesture Recognition Model, TurtleBot3, Raspberry Pi, sensors, and a camera module with ROS2
- Design of User Interface to provide feedback such as robot status, sensors data to user
- Testing & Evaluation on gesture recognition accuracy and real-time responsiveness.

Objectives

- Implement & develop an accurate gesture recognition system using deep learning to identify hand gestures.
- Achieve real-time performance for smooth human-robot interaction.
- Create a user-friendly interface with real-time feedback.
- Map recognized gestures to robot actions for effective control and monitoring.

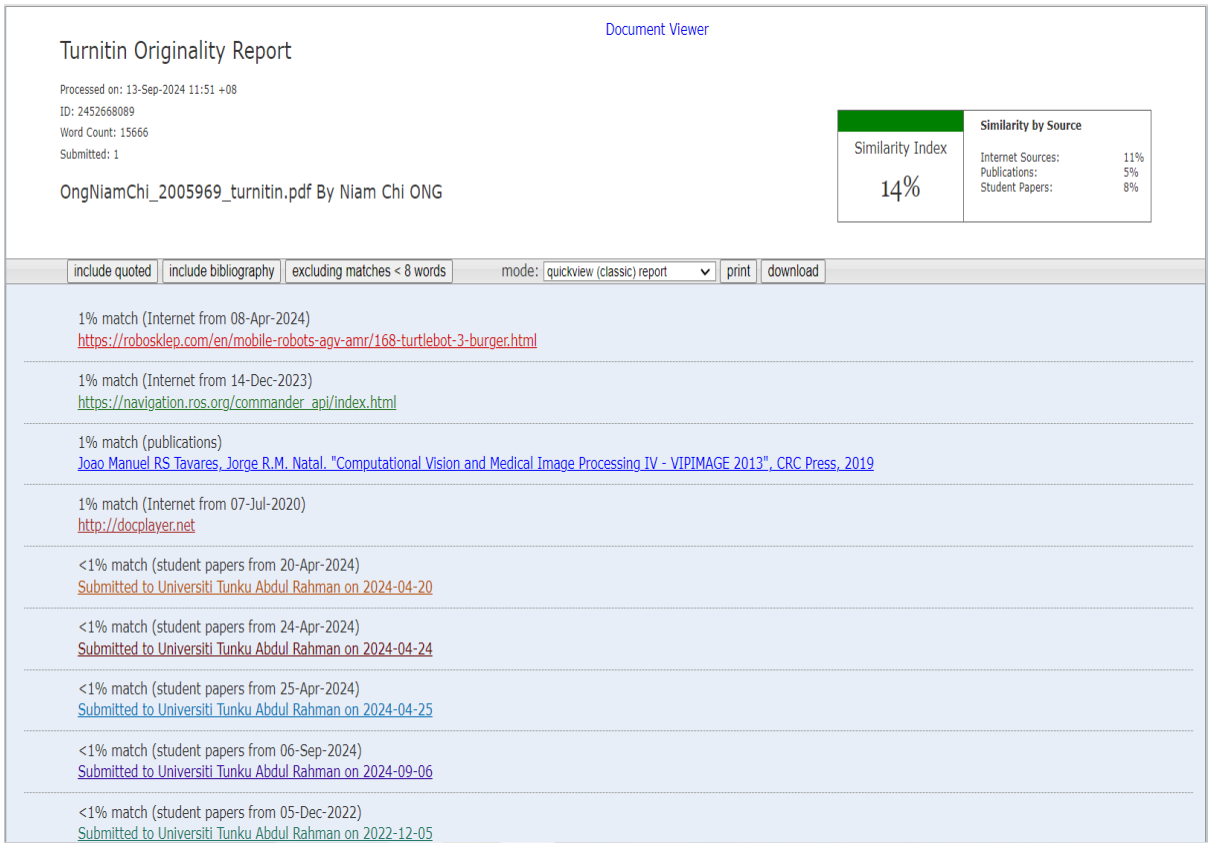
Result

The system achieved an accuracy of 70-100% in recognizing hand gestures depending on the gesture and environment.Real-time performance was achieved with an average response time of 0.003162 seconds, enabling seamless interaction between users and the robot. The user interface successfully provided feedback on robot actions, sensor data, and gestures, ensuring ease of use.

Conclusion

The project successfully demonstrated the feasibility of using gesture recognition for human-robot interaction. Despite challenges related to internet connectivity, environmental conditions, and hardware integration, the system showed promising results in accuracy and responsiveness. Future work should focus on improving robustness, addressing internet-related delays, and expanding the system's applications in fields like healthcare and entertainment.

PLAGIARISM CHECK RESULT



Document Viewer

Turnitin Originality Report

Processed on: 13-Sep-2024 11:51 +08
ID: 2452668089
Word Count: 15666
Submitted: 1

OngNiamChi_2005969_turnitin.pdf By Niam Chi ONG

Similarity Index	Similarity by Source	
14%	Internet Sources:	11%
	Publications:	5%
	Student Papers:	8%

include quoted | include bibliography | excluding matches < 8 words | mode: quickview (classic) report | print | download

1% match (Internet from 08-Apr-2024)
https://robosklep.com/en/mobile-robots-agv-amr/168-turtlebot-3-burger.html

1% match (Internet from 14-Dec-2023)
https://navigation.ros.org/commander_api/index.html

1% match (publications)
Joao Manuel RS Tavares, Jorge R.M. Natal. "Computational Vision and Medical Image Processing IV - VIPIMAGE 2013", CRC Press, 2019

1% match (Internet from 07-Jul-2020)
http://docplayer.net

<1% match (student papers from 20-Apr-2024)
Submitted to Universiti Tunku Abdul Rahman on 2024-04-20

<1% match (student papers from 24-Apr-2024)
Submitted to Universiti Tunku Abdul Rahman on 2024-04-24

<1% match (student papers from 25-Apr-2024)
Submitted to Universiti Tunku Abdul Rahman on 2024-04-25

<1% match (student papers from 06-Sep-2024)
Submitted to Universiti Tunku Abdul Rahman on 2024-09-06

<1% match (student papers from 05-Dec-2022)
Submitted to Universiti Tunku Abdul Rahman on 2022-12-05

Universiti Tunku Abdul Rahman			
Form Title : Supervisor's Comments on Originality Report Generated by Turnitin for Submission of Final Year Project Report (for Undergraduate Programmes)			
Form Number: FM-IAD-005	Rev No.: 0	Effective Date: 01/10/2013	Page No.: 1of 1

FACULTY OF INFORMATION AND COMMUNICATION TECHNOLOGY

Full Name(s) of Candidate(s)	ONG NIAM CHI
ID Number(s)	2005969
Programme / Course	BACHELOR OF INFORMATION TECHNOLOGY (HONOURS) COMPUTER ENGINEERING
Title of Final Year Project	DEVELOPMENT OF A REAL-TIME GESTURE RECOGNITION SYSTEM FOR HUMAN-ROBOT

Similarity	**Supervisor's Comments (Compulsory if parameters of originality exceed the limits approved by UTAR)**
Overall similarity index: 14 **%** **Similarity by source** Internet Sources: 11 % Publications: 5 % Student Papers: 0 %	ok
Number of individual sources listed of more than 3% similarity: 0	ok

Parameters of originality required, and limits approved by UTAR are as Follows:

(i) Overall similarity index is 20% and below, and

(ii) Matching of individual sources listed must be less than 3% each, and

(iii) Matching texts in continuous block must not exceed 8 words

Note: Parameters (i) – (ii) shall exclude quotes, bibliography and text matches which are less than 8 words.

Note: Supervisor/Candidate(s) is/are required to provide softcopy of full set of the originality report to Faculty/Institute

Based on the above results, I hereby declare that I am satisfied with the originality of the Final Year Project Report submitted by my student(s) as named above.

TEOH

Signature of Supervisor

Name: Dr. Teoh Shen Khang

Date: 13 September 2024

Signature of Co-Supervisor

Name: _____

Date: _____

UNIVERSITI TUNKU ABDUL RAHMAN

FACULTY OF INFORMATION & COMMUNICATION TECHNOLOGY
(KAMPAR CAMPUS)

CHECKLIST FOR FYP2 THESIS SUBMISSION

Student Id	20ACB05969
Student Name	ONG NIAM CHI
Supervisor Name	DR TEOH SHEN KHANG

TICK (√)	DOCUMENT ITEMS Your report must include all the items below. Put a tick on the left column after you have checked your report with respect to the corresponding item.
√	Title Page
√	Signed Report Status Declaration Form
√	Signed FYP Thesis Submission Form
√	Signed form of the Declaration of Originality
√	Acknowledgement
√	Abstract
√	Table of Contents
√	List of Figures (if applicable)
√	List of Tables (if applicable)
√	List of Abbreviations (if applicable)
√	Chapters / Content
√	Bibliography (or References)
√	All references in bibliography are cited in the thesis, especially in the chapter of literature review
√	Weekly Log
√	Poster
√	Signed Turnitin Report (Plagiarism Check Result - Form Number: FM-IAD-005)
√	I agree 5 marks will be deducted due to incorrect format, declare wrongly the ticked of these items, and/or any dispute happening for these items in this report.

*Include this form (checklist) in the thesis (Bind together as the last page)

I, the author, have checked and confirmed all the items listed in the table are included in my report.

(Signature of Student)
Date: 13/9/2024