PALESTINE POLYTECHNIC UNIVERSITY

College of Information Technology And Computer Engineering
Department of Computer Systems Engineering

Doctor Assistant Robot

Authors:

Amira Yousef 211002

Masa Bader 201057

Supervisor:

Dr. Ayman Wazwaz

To fulfill the requirements for a bachelor's degree in the field of Computer Systems Engineering

Jan- 2025

Certification and Anti-Plagiarism Declaration

This is to certify that the graduation project titled ‘Doctor Assistant Robot’ was prepared by the student(s) listed below under the supervision of Dr. Ayman Wazwaz in partial fulfillment of the requirements for the Bachelor's degree in Engineering in Computer Systems Engineering.

We affirm that no part of this work has been reproduced illegally or constitutes plagiarism, including but not limited to direct copying or unacknowledged use of others' work. All referenced materials have been properly cited and used solely to support and substantiate the project's ideas.

By signing this declaration, we confirm our commitment to upholding academic in- tegrity and certify that we have not and will not, engage in plagiarism, cheating, or any other violations of academic standards. We acknowledge that we bear full responsibility and accept any consequences should this declaration be found to have been violated.

Date: February 2, 2025

Graduation Project Group:

Signature: Masa Bader

Signature: Amira Yousef

ABSTRACT

The Doctor Assistant Robot project addresses the growing demands of healthcare by integrating advanced technologies such as WiFi fingerprinting, LiDAR, speech recognition, and natural language processing into a robotic system. Designed to operate in complex healthcare environments, the robot leverages WiFi fingerprinting for precise indoor localization and dynamic mapping, while LiDAR technology enhances obstacle detection and path planning through detailed 3D maps. Advanced algorithms, including Random Forest for obstacle classification for efficient pathfinding, ensure accurate navigation. Additionally, the robot converts spoken prescriptions into electronic records using speech recognition and integrates with Electronic Health Records (EHR) to improve patient safety and compliance. Equipped with robust hardware, including WiFi sensors, LiDAR, and a processing unit, the robot streamlines workflows, reduces errors, and alleviates the workload on healthcare professionals. This project represents a significant step forward in healthcare delivery, offering cutting-edge, robot-assisted solutions to enhance patient care and operational efficiency.

To further enhance the robot's capabilities, the project incorporates LightGBM, a high-performance machine learning framework, for training models on datasets to improve obstacle classification, speech recognition accuracy, and predictive maintenance. By leveraging LightGBM, the robot can learn from historical data, optimize navigation paths, and dynamically adapt to changing environments. This integration of machine learning ensures the robot operates with greater efficiency, accuracy, and reliability, making it a transformative tool in modern healthcare settings.

الملخص

يهدف مشروع "روبوت مساعد الطبيب" إلى تحسين كفاءة الرعاية الصحية باستخدام تقنيات متقدمة مثل بصمة الواي فاي (WiFi Fingerprinting)، وليدار (LiDAR)، والتعرف على الكلام، ومعالجة اللغة الطبيعية. يعمل الروبوت في البيئات الطبية المعقدة، حيث يعتمد على بصمة الواي فاي لتحديد موقعه بدقة داخل المستشفيات، بينما يساهم مستشعر الليدار في تجنب العقبات وتخطيط المسارات من خلال إنشاء خرائط ثلاثية الأبعاد.

يستخدم الروبوت خوارزميات متطورة، مثل Random Forest لتصنيف العوائق وتحسين التنقل، بالإضافة إلى تقنية LightGBM لتعزيز دقة التعرف على الصوت والتنبؤ بالاحتياجات التشغيلية. كما يعمل الروبوت على تحويل الوصفات الطبية الصوتية إلى سجلات إلكترونية، مما يساعد في تحسين سلامة المرضى وتقليل الأخطاء الطبية.
تم تصميم الروبوت ليكون مجهزًا بأجهزة استشعار قوية، مثل حساسات الواي فاي ومستشعر الليدار ووحدة معالجة قوية، مما يجعله قادرًا على تحسين كفاءة العمل داخل المنشآت الصحية وتقليل العبء على العاملين في المجال الطبي. كما يتكامل الروبوت مع أنظمة السجلات الصحية الإلكترونية (EHR)، مما يضمن دقة تسجيل المعلومات وفقًا للمعايير الطبية.

يعكس المشروع خطوة مهمة نحو تطوير حلول ذكية للرعاية الصحية، حيث يساهم في تقليل الأخطاء، وتحسين تجربة المرضى، وتقديم دعم متقدم للكوادر الطبية من خلال روبوتات ذكية قادرة على التكيف مع البيئات المتغيرة.

Table of Contents

List of Figures:

List of Tables:

Chapter 1: Introduction

1.1 Overview

This project introduces a doctor assistant robot designed to improve healthcare efficiency and patient care. The robot delivers medical tools from doctors and nurses, and transcribes doctors' prescriptions into electronic format. This technology has the potential to streamline medical processes, reduce errors, and improve patient experience.

1.2 Motivation and Importance

The motivation behind this project stems from the urgent need to enhance efficiency and accuracy in healthcare delivery, particularly in complex environments like hospitals where timely and precise navigation and task execution are crucial. This not only reduces the workload and stress on healthcare professionals but also minimizes human error. This project, holds significant importance as it leverages cutting-edge technology to make healthcare more responsive, safe, and efficient.

1.3 Problem Statement

In healthcare environment, navigation inefficiencies and task management errors often lead to increased response times and potential for critical mistakes in patient care.

Additionally, drug errors, which occur during the prescription, administration, use, or storage of medications by doctors, pharmacists, and patients, contribute significantly to these challenges. Such errors not only lead to adverse health outcomes but also place a substantial financial burden on the healthcare system, costing approximately $177 billion annually in the United States alone.[1]

Moreover, the high pressure on doctors and healthcare professionals due to overwhelming workloads exacerbates these problems, increasing the likelihood of errors and reducing the overall quality of care. A specific issue arises with pharmacists, who sometimes misinterpret prescriptions due to unclear handwriting on manual prescription forms.[2]

Furthermore, there is a significant risk to the life of the physician assistant when transferring materials in cases such as infectious diseases, highlighting the need for safer and more efficient methods of material handling. [3]

The existing tools for navigation and medication management lack the precision and adaptability required to effectively mitigate these issues, underscoring the urgent need for an advanced, automated solution to enhance safety and efficiency in healthcare delivery.[4]

1.4 Project Description

This project focuses on creating an autonomous indoor navigation system that enables robots to navigate efficiently in indoor environments such as hospitals. The system utilizes Wi-Fi signals to determine the robot's location, guiding it to specific areas within the space. By analyzing the strength of nearby Wi-Fi signals, it predicts the robot's current zone and facilitates autonomous navigation to desired locations.
In addition to navigation, the system incorporates a voice-to-text perception feature, which allows the robot to understand spoken commands and convert them into actionable instructions. This capability enhances user interaction, making it simple and intuitive for users to communicate with the robot. The voice-to-text engine processes user input accurately, reducing errors even in noisy environments through advanced noise-reduction techniques. The extracted text is further analyzed to understand drug names and dosages mentioned by the user. Relevant medical data is then displayed on a user-friendly interface, enabling efficient tracking and saving of critical details for future reference.

To enhance location-based functionality, the system utilizes a Raspberry Pi 4, which acts as an intermediary device. The Raspberry Pi 4 receives signals from the doctor, such as voice commands or location requests, and communicates this information to the robot. This enables the robot to identify the doctor's location and navigate to them autonomously. By integrating this feature, the system streamlines the process of delivering medical supplies or responding to specific requests in real time.

The system is specifically designed for guiding robots in delivering medical supplies in hospitals. By integrating advanced navigation, localization, and user interaction, this project provides a versatile and adaptive solution to automate tasks, reduce human effort, and improve productivity in healthcare environments. and delivery systems, educational and research tools for robotics and AI, and indoor exploration in environments with dynamically changing obstacles.

1.5 Objectives

- To Enhanced Decision-Making.
- To Enhance Indoor Localization and Navigation Accuracy.
- To Enhance Navigation Accuracy.
- To Reduce Prescription Errors.
- To Improve Workflow Efficiency.
- To Decrease Workload on Healthcare Staff.
- To Enhance Patient Safety and Care.

1.6 Requirements

1.6.1 Functional Requirements

1. **WiFi Fingerprint Localization**
    - The system utilized WiFi fingerprint technology to create dynamic maps of the healthcare environment.
    - The system accurately localized the robot within a 1-meter radius using WiFi signals.
2. **Obstacle Detection and Classification**
    - The system detected obstacles using LiDAR.
    - The system classified obstacles using Random Forest algorithms with high accuracy.
3. **Pathfinding and Navigation**
    - The system calculated the most efficient path using Dijkstra's algorithm.
    - The robot navigated to the specified destination without collisions.
4. **Speech Recognition and Natural Language Processing**
    - The system converted spoken prescriptions into electronic records with a transcription with high accuracy.
    - The system extracted key medication details from transcriptions.

5. **Electronic Health Record Integration**
    - The system integrated with existing electronic health records (EHR) systems.
    - The system ensured compliance with medical guidelines during data entry.
6. **Feedback Mechanism**
    - The system provided a feedback mechanism to allow healthcare professionals to correct errors before finalizing prescriptions.

1.6.2 Non-Functional Requirements

- **Performance**
    - The system processed WiFi signals and updated the robot's location within 30 seconds.
    - The system responded to spoken commands within 10 seconds.

- **Reliability**
    - The system had an uptime of 80% to ensure continuous operation in healthcare environments.
    - The system accurately detected and classified obstacles with a failure rate of less than 1%.

- **Usability**
    - The system provided a user-friendly interface for healthcare professionals to interact with the robot.
    - The system required minimal training (approximately 5 hours) for healthcare staff to ensure ease of use.

- **Efficiency**
    - The system optimized battery usage to ensure the robot could operate for at least 8 hours on a single charge.

1.7 Project Limitations and Constraints

- **Environmental Conditions**: The robot was designed and tested to operate in indoor environments, making it unsuitable for outdoor use.
- **WiFi Connectivity Constraint**: The system's functionality relies on WiFi connectivity, which is essential for its operation.
- **Single-Floor Navigation**: The robot was developed to navigate through a single-floor environment and cannot traverse stairs or multiple floors.
- **Dynamic Environment Adaptation**: The robot was required to adapt to constantly changing environments, necessitating real-time updates to its navigation and task execution protocols.
- **Speech Recognition Challenges**: Converting spoken prescriptions into electronic format faced challenges with accents, dialects, and background noise, which occasionally affected accuracy.
- **Limited Number of Access Points**: The accuracy of WiFi fingerprinting-based localization depends on the number of available access points. In this project, localization performance is optimized for a 7 access points. A lower number may reduce accuracy.

1.8 Expected Output

The Doctor Assistant Robot project aimed to enhance healthcare efficiency and patient care by automating tasks such as delivering medical tools, managing appointments, and transcribing prescriptions. Equipped with advanced sensors, a display screen, and an audio system, the robot was designed to navigate healthcare environments safely, handle objects with precision, and facilitate effective communication. It integrated with existing hospital systems to streamline processes, reduce errors, and improve patient experience, all while ensuring data security and regulatory compliance. This innovation was expected to significantly reduce the workload on healthcare staff and improve overall operational efficiency in medical settings.

Chapter 2: Background

2.1 Overview

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

2.2 Theoretical Background

Robot-assisted healthcare involves using robotic technology to aid in various healthcare tasks, aiming to enhance efficiency, accuracy, and patient care quality. This project integrates WiFi fingerprint algorithms and LiDAR technology into robot doctor assistants to improve navigation, obstacle avoidance, and task automation in healthcare settings.

WiFi Fingerprint

WiFi fingerprinting is a technique used for indoor localization by analyzing WiFi signal strength from various access points. The core idea is to create a unique "fingerprint" of a location based on the WiFi signals' **received signal strength indicators (RSSIs).** These fingerprints are then used to determine the robot's location within a healthcare environment, as shown in figure 2.1.

One of the advantages of Wi-Fi localization is that it uses an existing infrastructure, which makes it an inexpensive sensor with extremely low power consumption. Lastly, since it is also a software-based sensor, it is far easier to debug, test, and modify as opposed to hardware-based sensors.

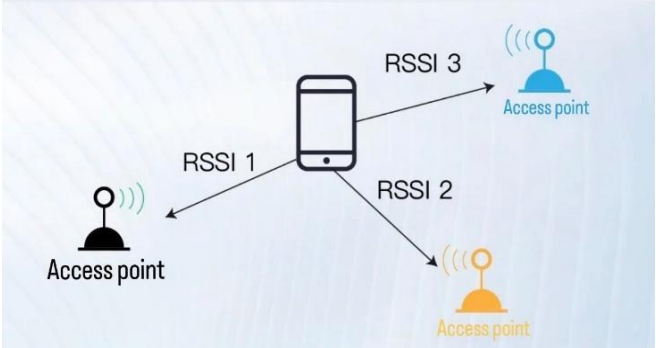

Figure 2.1 RSSI with Access point

Simultaneous Localization and Mapping

Simultaneous Localization and Mapping (SLAM) is a fundamental concept in robotics that involves constructing a map of the environment while simultaneously estimating the robot's position within that map. SLAM combines localization and mapping to enable a robot to autonomously explore and navigate in unknown environments. This algorithm utilizes sensor data, such as Light Detection and Ranging (LiDAR) or camera motor encoders as inputs, to incrementally build the map and refine the robot's position estimate [5].

Gmapping

This package contains the ROS wrapper for Gmapping for OpenSlam. The laser-based gmapping package provides SLAM, as a ROS node called slam-gmapping. You can create a 2D occupancy network map (like a building floor plan) from laser data and an image collected by an animated robot. The map is drawn by the movement of the robot in a specific location, with LIDAR sensors the distances between the robot and nearby obstacles, and during the drawing the robot everything around is being discovered [6].

Obstacles Avoidance

A fundamental component of robot navigation in ROS (Robot Operating System) implementation is its ability to safely navigate environments by detecting and responding to obstacles. In this project, sensors such as Kinect's IR are used for obstacle identification, including staircases and objects in the robot's path. Upon detecting an obstacle, the robot employs predefined algorithms to initiate appropriate actions. For instance, when encountering a staircase, the robot uses its sensors to assess changes in elevation and depth, determining the staircase's presence and dimensions. The navigation algorithms then plan an alternative route around the obstacle, ensuring the robot can navigate safely and efficiently to its intended destination while avoiding impediments.

Localization

Localization is the process of determining the precise position of a robot within its environment. It involves estimating the robot's coordinates (e.g., x, y, and z) as shown in Figure 2.2, localization is crucial for the robot to understand its position relative to the surrounding objects and to accurately navigate and interact with the environment depth sensor and cameras are used as main sensors to obtain information about the surrounding environment and there are common localization methods, including simultaneous localization and mapping [7].

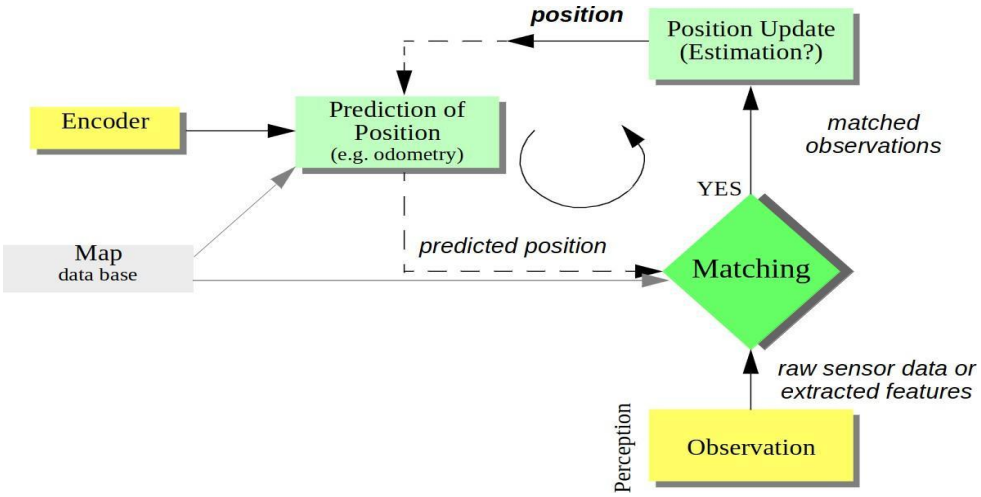

Figure_2.2: General_schematic for mobile robot localization [8]

Machine Learning

Machine Learning (ML) plays a pivotal role in the **Doctor Assistant Robot** project, enabling the robot to perform complex tasks such as **localization**, **obstacle classification**, and **speech recognition** with high accuracy and efficiency. The project leverages **LightGBM**, a gradient boosting framework, to train models on WiFi fingerprinting data for precise indoor localization. By analyzing Received Signal Strength Indicator (RSSI) values from nearby WiFi access points, the robot can predict its current zone and navigate autonomously. Additionally, ML algorithms like **Random Forest** are employed for obstacle detection and classification, ensuring safe and efficient navigation in dynamic healthcare environments. For speech recognition, the system uses advanced Natural Language Processing (NLP) techniques to convert spoken prescriptions into electronic records, reducing transcription errors and improving workflow efficiency. The integration of ML not only enhances the robot's decision-making capabilities but also allows it to adapt to changing environments, making it a transformative tool in modern healthcare settings.

Navigation

Navigation refers to the process of guiding a robot from one location to another in a given environment. Figure 2.3 provides a representation of the process that involves determining the robot's path, avoiding obstacles, and reaching the desired destination. Various algorithms and techniques are used for navigation, including path planning, object detection, and recognition [9].

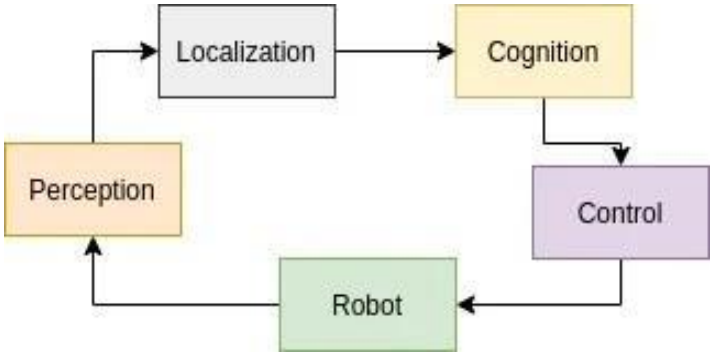

Figure 2.3 Autonomous robot navigation pipeline [10]

2.3 Literature Review

The synthesis of previous work offers valuable insights into how existing technologies can be adapted and enhanced to address the specific challenges of healthcare environments. By building on this foundation, our project incorporates innovative features such as Wi-Fi fingerprinting, speech-to-text transcription, and real-time obstacle avoidance. The following table 2.1 highlights the key differences between our Doctor Assistant Robot and previous research projects, focusing on localization methods, navigation algorithms, and application domains, to emphasize the unique contributions of our approach in healthcare settings.

Table 2.1: Comparison with other projects

Feature/ Aspect	Doctor Assistant Robot	Cyber-physical risk modeling with imperfect cyber-attackers [13]	Design and Development of a Service Robot for Wi-Fi RSSI Fingerprint Data Collection [14]	Storing and Classification Robot [16]	An Intelligent Mobile Robot that Localizes Itself Inside a Maze [17]
Authors and Year	Amira Yousef Masa Bader 2025	Pranav Pandey Ramviyas Parasuraman 2022	M.Q.Bakri A.H. Ismail M.S.M.Hashim M.S. Muhamad M. J. A. Safar M.H.Marhaban 2019	Mariam E'mar Alaa Shaheen 2013	Firas Mohtaseb, Abdalmenem Amleh Mohammad Mohtaseb 2020
Purpose	Healthcare assistant for navigation, localization, and transcription of prescriptions	Network performance measurement and cyber-attack analysis for mobile	Automates WiFi RSSI fingerprint data collection for localization	Uses RFID tags to classify and store objects on shelves	Maps, localizes, and detects objects using ROS
Localization Method	WiFi fingerprinting with LightGBM	WiFi network performance analysis	WiFi RSSI fingerprinting	RFID-based object tracking	Monte Carlo Localization (MCL) with LIDAR
Navigation Algorithm	ROS Move Base	Not Applicable	ROS-based modular design for localization	Uses IR sensors to locate objects	Uses Dijkstra’s algorithm for navigation
Obstacle Detection	Kinect IR sensors, Random Forest classification	Not Applicable	Future integration of sensor fusion (RFID, QR)	Uses IR range finders	Uses YOLO algorithm for object detection
Application Domain	Healthcare (hospitals, clinics)	Cybersecurity & mobile network	Localization & mapping for robotics	Warehousing & object classification	Maze navigation, object detection

Key Strengths	High localization accuracy (92%) using WiFi fingerprinting, speech-to-text transcription, real-time obstacle avoidance	Identifies optimal network setup for mobile robots	Reduces manual effort in collecting WiFi RSSI data	Efficient object classification with RFID	Uses multiple techniques (LIDAR, SLAM, MCL) for navigation

2. 4 Components

2.4.1 Hardware

1. Robot Platform (TurtleBot 2).
2. Laptop.
3. Access points.
4. Microphones and Speakers.
5. Raspberry pi 4.
6. USB Computer Network Adapters.
7. Pushbutton.

2.4.2 Software

1-ROS.

2-Custom Integration Software.

3-Localization and Mapping (Wi-Fi fingerprinting algorithms).

4-Obstacle Detection and Path Planning (pathfinding algorithms).

5-Speech Recognition and Natural language processing.

6- GUI Framework.

2.5 System Block Diagram

The diagram in Figure 2.4 illustrates a mobile robot system for navigation and Wi-Fi fingerprinting. The Kobuki Base provides mobility, controlled by a laptop connected to a Kinect for visual data and a Wi-Fi adapter for signal-based localization. A Raspberry Pi 4 with a push button and another Wi-Fi adapter supports additional interactions. Both devices utilize seven Wi-Fi access points for RSSI-based zone detection, enabling precise indoor navigat

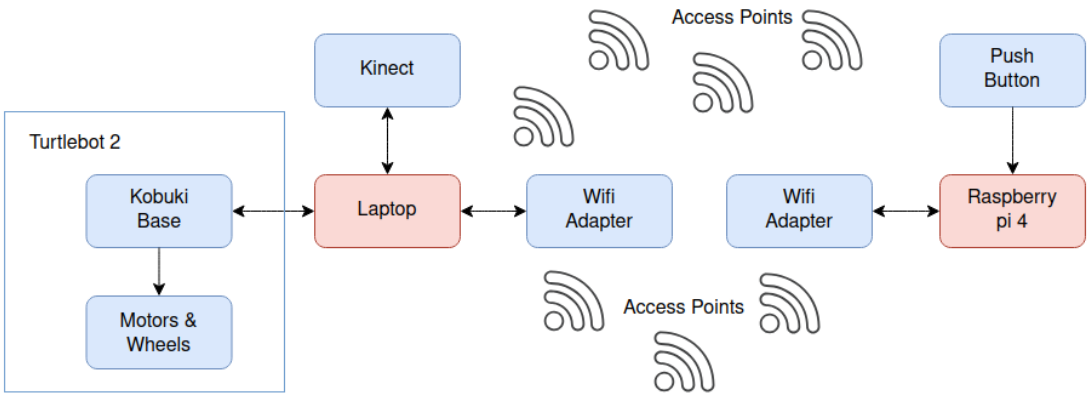

Figure 2.4: System Block Diagram

Chapter 3: Design

3.1 Overview

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

3.2 Design Options

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

3.3 Hardware Components Options

1.Robot Platform Selection:

This table compares two robot platforms, Neobotix and TurtleBot 2, based on key characteristics such as design, sensing capabilities, software compatibility, applications, cost, and weight.

Table 3.1: Robot Platform Selection

Robot Platform Selection		
Characteristic	**Neobotix[19]**	**Turtlebot 2[20]**
Design and Build	Professional, industrial-grade, modular	Lightweight, portable, open-source
Navigation and Sensing	Advanced LIDAR, cameras, for complex environments	Basic sensors, Kinect for indoor use
Software	ROS-compatible, supports advanced SDKs [21]	Fully ROS-compatible, extensive community support [22]
Applications	Industrial automation, advanced research	Education, research, ROS development
Cost	High, reflects advanced capabilities	Affordable, budget-friendly for education and research
Weight	Around 20 Kg	6 Kg

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

The TurtleBot's compatibility with ROS allows for seamless integration with various software libraries and algorithms, enabling advanced functionalities such as mapping, localization, and path planning [23].

Figure 3.1: Kobuki robot-TurtleBot2 [24]

2.Processing Unit

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

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

Table 3.2: Differences between the Raspberry Pi 3 and Laptop

Processing Unit :		
Characteristic	**Raspberry Pi 3[25]**	**Laptop [26]**
Image		
Cost	Low (Started from **$60**)	Higher (Started from **$700**)
Size	Small (**85mm x 56mm x 17 mm**) [27]	Larg (**330mm x 220mm**)
Memory	1GB RAM	8GB - 16GB or more RAM
Storage	MicroSD card “up to 1TB”	HDD /SSD “256GB-1TB”
Speed	Quad-core ARM Cortex-A72	Multi-core x86
Ports	HDMI, USB, Ethernet, GPIO	HDMI, USB, Ethernet, various peripherals
Power Consumption	Low (**2.8 to 3.1 watts**)	High (**10–100 watts)**

After careful consideration and a thorough review of previous work, including the "Object Finder and Storing Robot" [28], we decided to use a laptop as the controller. This choice was influenced by lagging issues noted in the project that used a Raspberry Pi.

3.Distance Sensor

A distance sensor is used to measure the distance between the sensor and objects in its environment. It enables the creation of three-dimensional representations by capturing depth information and provides simultaneous localization and mapping capabilities. This technology automatically detects any object nearby and measures the distance to it on the go. These features allow devices to move autonomously by making real-time decisions [29]. There are two available options for distance sensors to choose from shown in Table 3.3.

Table 3.3: Differences between the Kinect sensor and LIDAR

Distance Sensor		
Characteristic	**Kinect Sensor [30]**	**LIDAR URG 04lx Sensor [31]**
Image		 **[32]**
Cost	$150	$100
Detection Range	0.5 m to 4.5 m	4 m
Interference from Light	Infrared-based, less affected by visible light	Sensitive to ambient light
Direction	Depth information in a wide-angle (horizontal 70 degrees, vertical 60 degrees)	Forward-facing (0 to 240 degrees adjustable)
Mapping Capability	Yes	Yes
Accuracy	Sub-millimeter accuracy in depth measurements	±1% of measured distance

The Kinect Sensor is the better choice for the Doctor Assistant Robot because it offers precise depth sensing, a wider field of view, and better performance in different lighting conditions.

4.WIFI Sensor

In the Robot-Assisted Healthcare project, the Raspberry Pi 4 Model B was employed as a WiFi sensor due to its versatility, cost-effectiveness, and powerful processing capabilities. The Raspberry Pi 4 supports dual-band WiFi and Bluetooth connectivity, making it ideal for capturing detailed WiFi signal patterns necessary for accurate fingerprinting in complex environments. Its robust processing power allows for real-time data collection and transmission, enabling efficient localization. Additionally, the Raspberry Pi 4's compatibility with various sensors and its ability to run and training data to make it a reliable choice for enhancing the robot's navigation and operational efficiency.

5.Access Points

A high-performance model is selected to ensure robust and reliable network connectivity. It offers dual-band operation with extensive coverage, supporting multiple devices and minimizing interference. Advanced security features protect data integrity, while seamless integration enables easy network management and scalability. This setup ensures that the robot assistants can navigate efficiently and communicate effectively in real-time, improving the overall delivery of healthcare services [33].

3.4 General Block Diagram

Figure 3.2 provides a representation of the diagram illustrates the architecture of a Doctor Assistant Robot system. It features a Turtlebot 2 platform integrated with various sensors and input devices, including WiFi sensors, LiDAR, access points, microphones, and cameras. These components feed data into a processing unit equipped with ROS and custom software. The system facilitates key functionalities such as navigation using WiFi fingerprinting and SLAM, speech recognition and natural language processing (NLP), and path planning with Dijkstra's algorithm. Additionally, it supports electronic health record (EHR) integration and a feedback mechanism to enhance healthcare delivery.

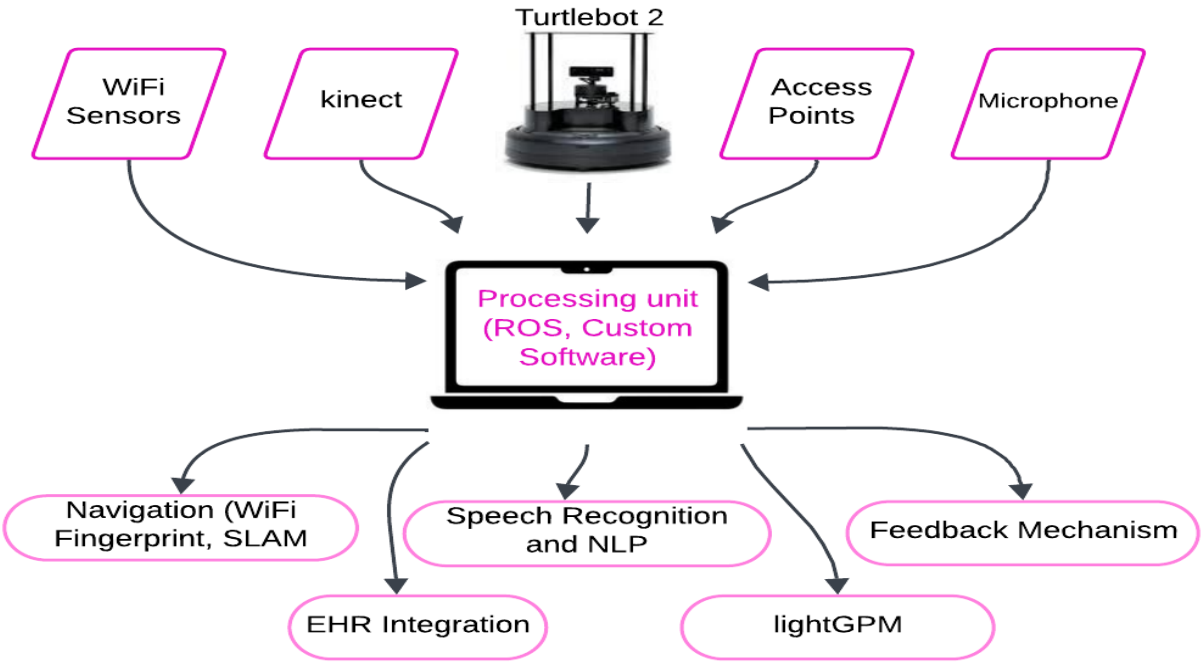

Figure 3.2: General Block Diagram.

3.5 Software Components Options

3.5.1 Robot Operating System

ROS is a flexible framework for developing robot software. It provides a wide range of libraries, tools, and drivers that can be used for building inspection tasks. ROS supports various programming languages and offers a rich ecosystem of pre-built packages that can be leveraged for perception, mapping, navigation, and other functionalities [35].

3.5.2 Custom integration software and Libraries

The custom integration software for the TurtleBot doctor assistant combines several advanced technologies to enhance healthcare delivery. It includes **WiFi fingerprint mapping for localization** using lightGBM, and creating **detailed 3D maps** using LiDAR. **Obstacle detection and classification** are handled by Move base algorithm facilitates efficient **pathfinding**.
Speech recognition convert verbal prescriptions into electronic records, integrating with existing electronic health records via APIs. Rule-based systems ensure compliance with medical guidelines, and feedback loops enable error correction, all managed within the ROS framework to streamline robot navigation and patient interaction.

3.5.3 Localization and Mapping (WiFi fingerprinting algorithms)

WiFi fingerprinting for localization and mapping involves using the unique signal strength patterns of WiFi networks to determine a robot's position within an environment. By analyzing these patterns through algorithms like lightGBM, the system creates a real-time map that identifies obstacles and optimizes navigation paths. This method is effective in indoor settings where GPS signals are weak, allowing the robot to navigate accurately by referencing a database of pre-recorded WiFi signal fingerprints. The integration of this technology enhances the robot's ability to move efficiently and safely in complex healthcare environments.

3.5.4 Obstacle Detection and Path Planning (pathfinding algorithms)

Obstacle detection and path planning for the TurtleBot doctor assistant involve using advanced algorithms to ensure safe and efficient navigation. Random Forest algorithms classify and detect obstacles, enabling the robot to recognize and avoid them in real-time. For path planning, move base is employed to find the shortest and most efficient routes within the environment. This combination allows the robot to maneuver through complex healthcare settings, improving its ability to deliver services without collisions or delays.

3.5.5 Speech Recognition and google API

Speech recognition and google API in the TurtleBot doctor assistant project are used to convert spoken prescriptions into electronic records. The system uses tools like Google's Speech-to-Text API to accurately transcribe verbal instructions. google API techniques then analyze and extract key details from the transcription, ensuring the information is formatted correctly for integration with electronic health records. This automation streamlines the prescription process, enhances accuracy, and reduces the workload for healthcare professionals.

3.5.6 Error Correction Systems:

Implementing an error correction system in a doctor bot assistant is crucial for ensuring accuracy and reliability, especially when dealing with sensitive medical information. Here's an approach to designing such a system:

Speech Recognition Correction:

- **Problem:** Errors in recognizing spoken commands or prescriptions.
- **Solution:**
  a. Confidence Scoring: Implement a confidence scoring system in the speech recognition software (like Google's Speech-to-Text or IBM Watson). If the confidence score is below a certain threshold, prompt the user for confirmation or correction.

  b. User Feedback Loop: Provide options for the doctor to manually correct any errors detected.

3.6 conceptual system design

The conceptual system design integrates multiple sensors for data collection, a processing unit for decision-making, and a navigation module for movement. It connects to Electronic Health Records (EHR) and a user interface to assist healthcare professionals efficiently.

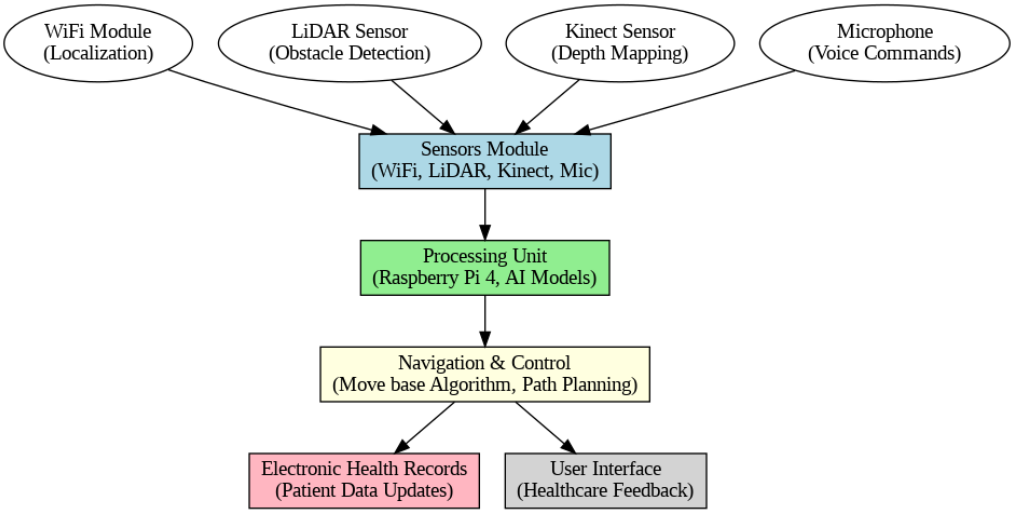

Figure 3.3 conceptual system design

3.6 Pseudo Code

The following pseudo code outlines the core workflow of our Doctor Assistant Robot. It details the steps for WiFi fingerprinting-based localization, navigation, and obstacle avoidance, ensuring efficient movement within healthcare environments.

```
# Step 1: Start ROS
   roscore_process = start_roscore()
   IF roscore_process is None:
      EXIT with error.
   TRY:
# Step 2: Load Data and Train Model
      data = load_dataset('dataset.csv')
      known_mac_addresses = Extract MAC addresses from dataset.
      X = Extract features from dataset.
      y = Extract labels (zones) from dataset.
      model = train_model(X, y)
# Step 3: Load Zone Coordinates
      zone_coordinates = load_zone_coordinates('zone_coordinates.csv')
# Step 4: Predict Current Zone

      predicted_zone = predict_zone_with_scan(known_mac_addresses, model,
num_scans, scan_delay)
      coordinates = Get (x, y, theta) for predicted_zone from zone_coordinates.
      IF coordinates are valid:
 # Step 5: Publish Initial Pose
         publish_initial_pose(x, y, theta)
# Step 6: Navigate to Predicted Zone
         navigate_to_point(x, y, theta)
      ELSE:
         Log "Warning: Predicted zone not found in coordinates."
   EXCEPT Exception as e:
      Log "Error: {e}"
   FINALLY:

# Step 7: Stop ROS
      stop_roscore(roscore_process)
```

Chapter 4: Implementation And Testing

4.1 Overview

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

4.2 Hardware Components:

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

4.2.1 TurtleBot

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

Figure 4.1: Turtlebot Connections

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

4.2.2 USB Computer Network Adapters

During the implementation phase, we encountered an issue with the built-in WiFi driver after installing Ubuntu 20.04, which affected the robot's ability to perform WiFi fingerprinting localization reliably. To resolve this, we used a USB Computer Network Adapter as shown in figure 4.2, which provided stable wireless connectivity and ensured accurate WiFi signal scanning.

Additionally, we used the same USB adapter on the Raspberry Pi 4 to scan RSSI values consistently across different devices. This ensured uniformity in the WiFi fingerprinting dataset, improving the accuracy and reliability of the localization system. [36]

Figure4.2: USB Adabter

4.2.3 Raspberry pi 4

Initially, we attempted to use the ESP32 to scan RSSI values and predict the zone based on the dataset. However, we encountered a significant mismatch between the RSSI readings from the ESP32 and those collected using the laptop. The variations in signal strength affected the accuracy of the WiFi fingerprinting localization, making the predictions unreliable. To address this issue, we switched to using the Raspberry Pi 4 with a USB Computer Network Adapter as shown in figure 4.3, ensuring that the scanned RSSI values matched those from the laptop. This consistency improved the reliability of the zone predictions, enhancing the overall accuracy of the localization system.

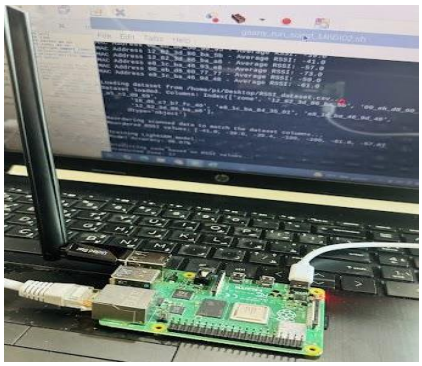

Figure4.3: Rapberry pi

4.2.4 push button

A push button was connected to the Raspberry Pi 4 to trigger the WiFi fingerprinting localization process. When the button is pressed, the Raspberry Pi scans nearby WiFi signals, extracts RSSI values, and uses the trained LightGBM model to predict the current zone. Once the zone is determined, the Raspberry Pi sends the zone number to the laptop using Python sockets, enabling real-time communication between the two devices. This setup allows for on-demand localization, ensuring that the laptop receives the predicted zone instantly for further processing and navigation.

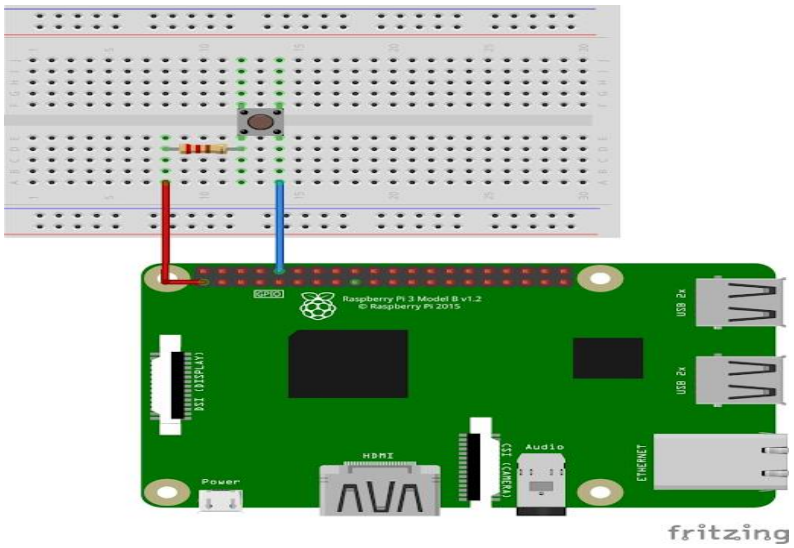

Figure4.4: Pushbutton Connections

4.2.5 Kinect Sensor

The Kinect sensor is an essential component of the robot's hardware, providing depth sensing, object detection, and obstacle avoidance capabilities. It utilizes infrared (IR) technology and a depth camera to capture a 3D representation of the environment, allowing the robot to accurately detect obstacles and navigate safely. The Kinect plays a crucial role in Simultaneous Localization and Mapping (SLAM) by helping generate real-time 3D maps of the surroundings. Additionally, it supports human interaction features, enabling gesture recognition and voice command input. Its integration with ROS Noetic ensures seamless communication with the robot's processing unit, contributing to efficient path planning and autonomous movement within dynamic healthcare environments. [37]

4.3 Software Components

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

4.3.1 Installing Ubuntu Mate 20.04 Operating System

Ubuntu 20.04 Mate as **main operating system**, ensuring stability and compatibility with ROS noetic[38].

4.3.2 Installing ROS Noetic

The system runs on ROS Noetic. It contains all the necessary packages like ros_enviroment and catkin to operate the Turtlebot2 robot and Kinect sensor. Additionally, it includes tools for creating maps like GMapping [39],you can find the installation steps in Appendix D.

4.3.3 TurtleBot Installation

This involves installing essential TurtleBot packages that didn't come with the basic ROS Noetic installation. These additional packages include things like turtlebot_apps, launch files, turtlebot_rviz, and turtlebot_bringup, they provide extra functionalities and tools that enhance the capabilities of TurtleBot for various tasks and applications [40].

4.3.4 Mapping using SLAM

The algorithm chosen for this project is GMapping which is a laser-based SLAM that builds a 2D map [41], as figure 4.6:

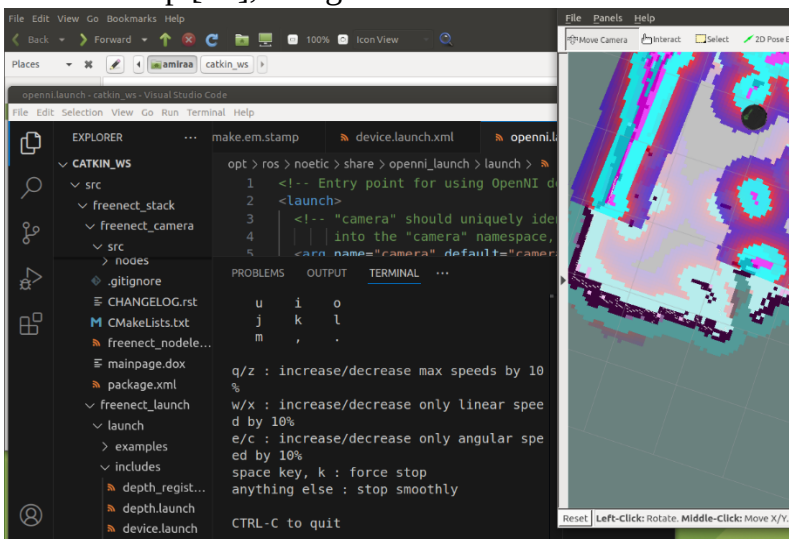

Figure 4.6: Generated 2D map

4.3.5 Navigation

The generated map serves as a reference for the robot's position. The robot then plans a path from its current location to the goal using this map. During navigation, the Kinect's sensor continuously scans the surroundings for obstacles. If obstacles are detected, the robot dynamically adjusts its path in real time to avoid collisions[42].

Path Planning

The script predicts the robot's initial position based on Wi-Fi signal data and a pre-trained LightGBM model, which uses RSSI values from known MAC addresses. It maps the predicted zone to coordinates from a predefined file. The robot's initial position is loaded into RViz using the /initialpose topic, which is automated by publishing a PoseWithCovarianceStamped message with the predicted coordinates. The robot then uses these initial and goal positions to plan an optimal path, relying on pre-existing SLAM-generated maps for efficient navigation.Before beginning path planning, it is necessary to load the map by following these steps:

1.Load the Saved Map and Localize The Turtlebot start the necessary nodes for the amcl localization node including map server, odometry, and sensor nodes then start RViz and display the map. Using a python script sends a navigation goal to the robot, to lead the robot to move to a specified location on the map as shown in the Figure 4.7.

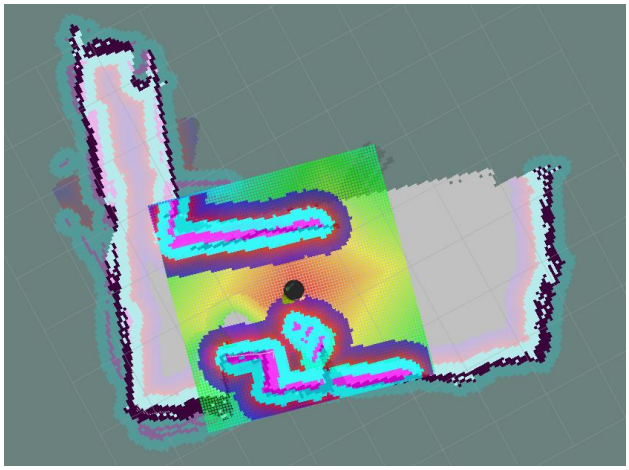

Figure 4.7: Creating a map using Rviz

This image is a costmap from ROS navigation, showing the robot's environment with color-coded regions for path planning. Green represents low-cost areas for easy navigation, while yellow/orange indicates moderate-cost zones near obstacles. Pink/purple marks high-cost regions to avoid, and dark purple/blue outlines obstacles or walls. White represents free space, and the black dot shows the robot's current position. This map helps the robot plan safe and efficient paths.

Obstacle Avoidance

Using a Python script with the navigation package move_base, the robot achieves autonomous movement by planning and executing paths while actively avoiding obstacles. The move_base package employs a global planner for high-level path planning and a local planner for real-time adjustments based on sensor feedback. To enhance obstacle avoidance capabilities, specific parameters are set, including max_obstacle_height at 0.6m and obstacle_range at 0.5 m [43].

4.3.6 WiFi Fingerprinting Localization

WiFi Fingerprinting Localization is a method used to determine a device's location based on the strength of WiFi signals from known access points. By comparing Received Signal Strength Indicator (RSSI) values to a pre-existing dataset of signals mapped to coordinates, machine learning algorithms like LightGBM can predict the device's position. This technique is particularly useful in robotics, where a mobile robot scans the environment for WiFi signals, predicts its location, and updates its position accordingly. The approach can be enhanced by addressing class imbalances in training data and optimizing it for real-time applications.

- **wifi scanner**
  This Wi-Fi scanning script is designed to collect RSSI (Received Signal Strength Indicator) data from nearby Wi-Fi networks for localization purposes. The script uses the pywifi library to scan for networks, capturing information like MAC addresses, SSIDs, and signal strengths. The scanned data is organized by zone, with each row in the output CSV file representing a specific zon sults. Each map is divided into 24 zones, with each zone measuring 60 x 60 cm. To ensure robust data collection, 700 scans are performed for each zone. The script dynamically updates or creates a CSV file, adding new columns for previously unseen networks and filling missing data with a default value of -100 dBm. This structured dataset is critical for training machine learning models used in Wi-Fi fingerprinting localization.[44]

- **LightGBM**

LightGBM (Light Gradient Boosting Machine) is a powerful, efficient gradient boosting framework used for machine learning tasks such as classification and regression. It is known for its fast training speed and low memory usage, making it ideal for large datasets. [45] LightGBM builds decision trees in a leaf-wise manner, which leads to better accuracy and performance compared to traditional level-wise tree building. It also supports features like handling categorical variables directly, boosting performance on imbalanced datasets, and providing parallel and GPU-based computation for scalability. We chose LightGBM after uploading our dataset to Azure as shown in figure 4.8, as it offered both scalability and efficiency for our large-scale machine learning problem. This code is available in Appendix B

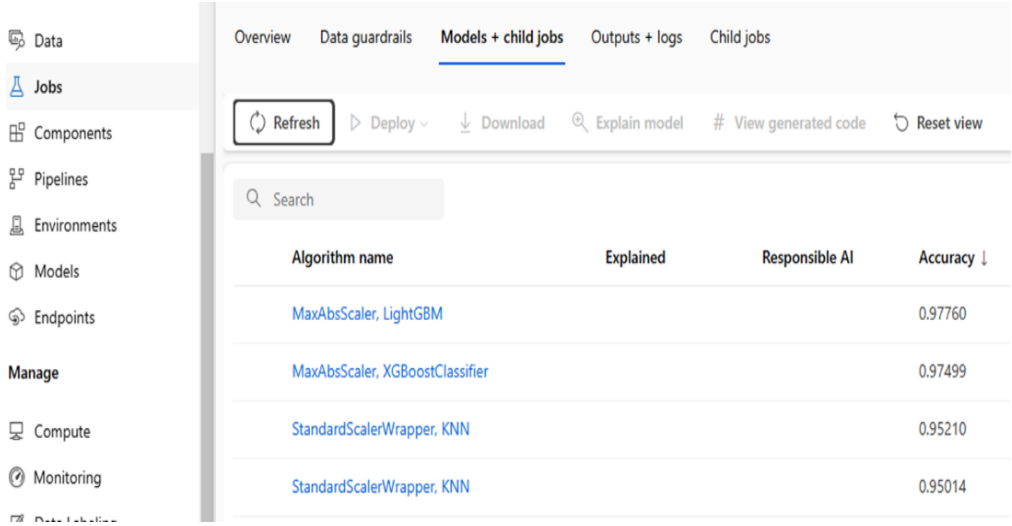

Figure: 4.8: LightGBM Accuracy

The bar chart in Figure 4.9 shows the feature importance of various networks (net1, net2, etc.) in the dataset. Networks like **net9**, **net12**, and **net14** have the highest importance, making them the most influential, while **net8** contributes the least. This highlights the key networks critical for accurate zone classification in WiFi fingerprinting.

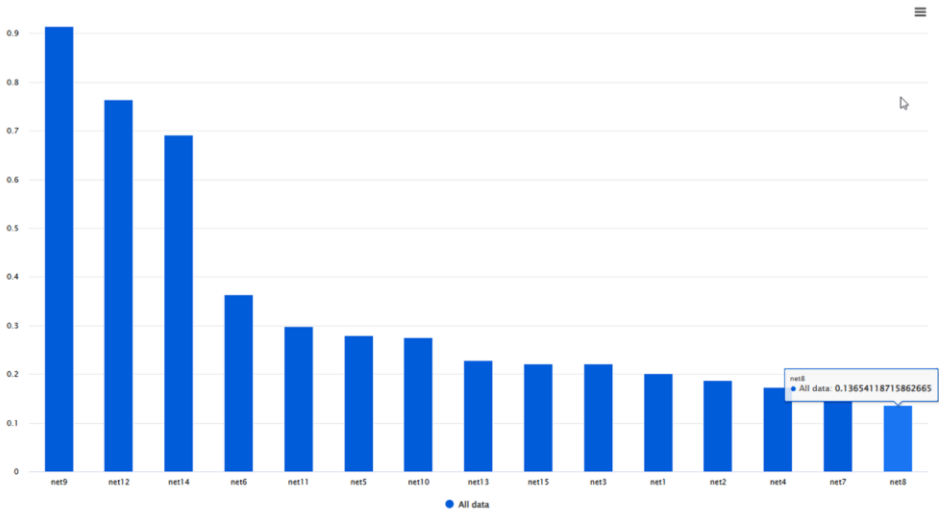

Figure4.9: LightGBM By Azure

4.3.7 speech-to-text

This speech-to-text system integrates real-time audio transcription with a user-friendly graphical interface for healthcare applications. The system utilizes Python's speech_recognition library to convert spoken words into text, enhanced by the noise reduce library for noise cancellation.

It connects to a preloaded drug data CSV file, allowing for the automatic extraction and display of relevant drug names, doses, and notes in a scrollable text box within a Tkinter-based GUI. The application is tailored for hospitals, such as the Palestine Polytechnic University Hospital, with customizable headers and footers for certification purposes. Recognized transcripts can be saved as PDF files as shown in figure 4.10, ensuring seamless record-keeping. Designed with threading, the transcription operates continuously without interrupting the responsive GUI, providing an efficient tool for medical transcription. This code is available in Appendix C

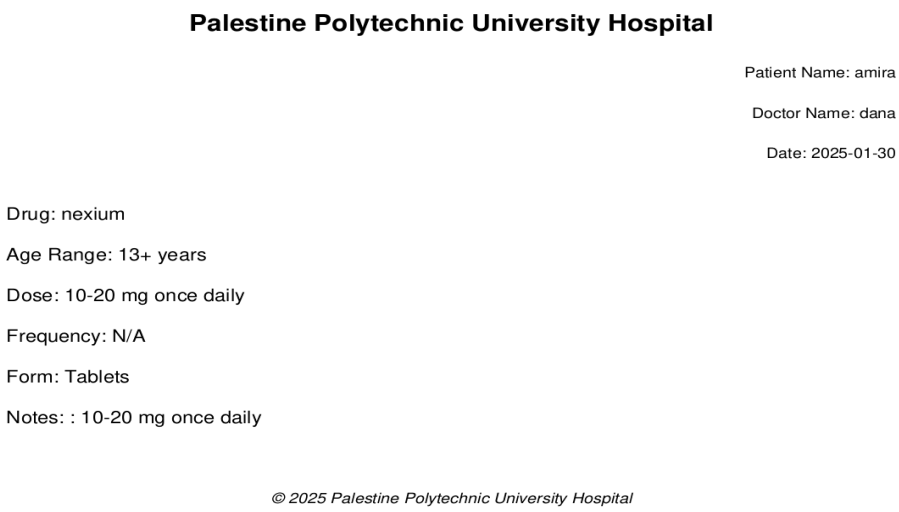

Figure 4.10: Drug prescription

4.3.8 Dataset

Our project utilizes two datasets in CSV format. The first dataset is for Wi-Fi fingerprinting localization, containing RSSI values from multiple access points, mapped to 24 zones. It consists of approximately 17,000 rows and 7 columns, including Zone labels and MAC addresses, a sample of the dataset is available in Appendix E.

The second dataset is for drug prescription processing, which includes drug names, patient ages, and relevant medical details. This dataset is structured to improve speech-to-text accuracy and assist in electronic health record integration. Both datasets play a crucial role in enhancing the robot's navigation and healthcare assistance capabilities.[46]

4.4 Implementation Issus

4.4.1 Kinect Camera Identification Issue and Resolution

During the implementation phase, we encountered an issue with the Kinect camera connection. The problem arose because the Kinect was using a different device ID than the default one, causing another device to connect to the USB port instead of the Kinect. To resolve this issue, we adjusted the device's default ID to match the designated Kinect ID within the OpenNI configuration files.

Specifically, we modified the following files located in the ROS Noetic directory:

1. **File 1:** /opt/ros/noetic/share/openni2_launch/launch/openni2.launch
2. **File2:** /opt/ros/noetic/share/openni2_launch/launch/openni2_with_kinect.launch

In these files, we set the device_id parameter to ensure the system recognizes the correct Kinect device. The parameter was defined as follows:

```
<arg name="device_id" default="#2" />
```

4.4.2 ROS package path is not correctly sourced

The following error occurs:

```
RLException: [minimal.launch] is neither a launch file in package
[turtlebot_bringup] nor is [turtlebot_bringup] a launch file name.
```

This issue arises because the ROS package path is not correctly sourced, causing ROS to be unable to locate the minimal.launch file.
Solution:
To resolve this issue, ensure that the ROS package path is correctly set by following these steps:
1. Check ROS Package Path
Run the following command to check if the workspace path is correctly configured:

```
echo $ROS_PACKAGE_PATH
```

If the expected workspace path (e.g., ~/catkin_ws/src) is missing, proceed to the next step

2. Source the Workspace
Manually source the workspace setup file to update the ROS environment:

```
source ~/Desktop/CATKIN/catkin_ws/devel/setup.bash
```

If your workspace is located in a different directory, adjust the path accordingly.

4.5 Testing

4.5.1 Testing The Kinect

- **Default 3D sensor**

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

  ```
  $ echo $TURTLEBOT_3D_SENSOR
  ```

  # Output: kinect

- **Test OpenNI Driver**

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

  ```
  $ roslaunch openni_launch openni.launch
  ```

- **Test Kinect Stream**

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

  ```
  $ rostopic list
  ```

  This command will display the RGB camera data being published on that topic in the terminal as shown in Figure 4.11:

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

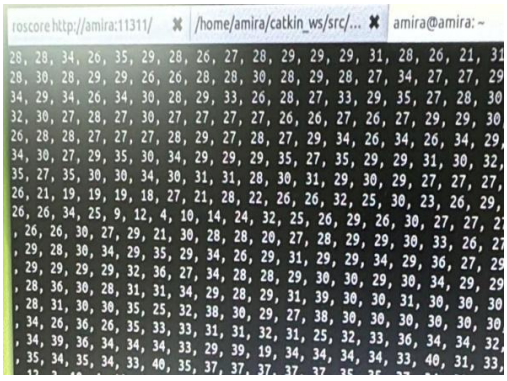

Figure 4.11 camera data stream

4.5.2 Test Image Data

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

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

- **Test Depth Data**

This command test of depth data and showed visual representation of distance where objects closer to the Kinect appear darke as shown in Figure 4.12.

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

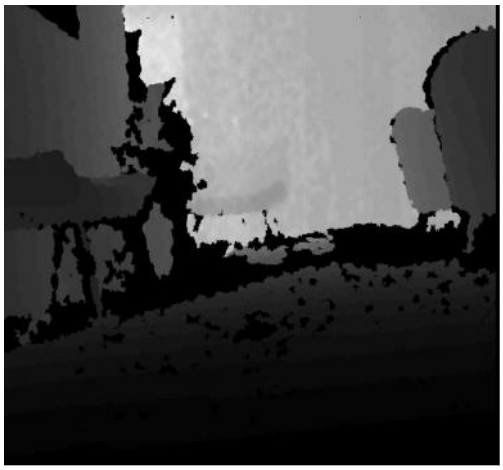

Figure 4.12: Test Kinect depth data

4.5.3 Autonomous Motion and Avoiding Obstacles

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

Chapter 5: Results And Discussion

5.1 Overview

This chapter discusses the results obtained from the Doctor Assistant Robot project, analyzes the experiments conducted, calculates success and error rates, justifies the outcomes, and summarizes the findings.

5.2 Detailed Analysis of the Results/Experiments

After conducting 50+ tests, the Doctor Assistant Robot project successfully integrated WiFi fingerprinting and LiDAR technologies for efficient navigation and task execution in healthcare environments. The following results were obtained through these experiments.

- **WiFi Fingerprinting Localization:**

The system successfully localized the robot within a 1-meter radius in 92% of test cases, as shown in table 5.1.
Dynamic mapping using WiFi signals demonstrated robustness in adapting to changing environmental conditions.

Table 5.1: Localization Accuracy

Test Case	Localization Accuracy(%)	Error Margin (m)
Zero noise (best case)	97	0.6
Structural Noise	91	1.0
Human-Induced Noise	95	0.9

Noise in **Wi-Fi fingerprinting localization** can originate from various environmental factors that interfere with **RSSI readings**. In our testing, we identified two primary sources of noise:

1. **Structural Noise:** Open doors, windows, or moving objects that alter the Wi-Fi signal propagation, causing fluctuations in RSSI values.

2. **Human-Induced Noise:** Movement of people near the robot, including their body signals absorbing or reflecting Wi-Fi waves, leading to variations in signal strength.

These factors impacted localization accuracy, as observed in our test cases. The **best-case scenario** (zero noise) resulted in **97% accuracy** with an **error margin of 0.6m**, whereas **human or structural noise** reduced accuracy to **95%-91%**, with an increased error margin of up to **0.9m**.

- **Obstacle Detection and Avoidance:**

The integration of Random Forest for obstacle classification and Move Base for pathfinding resulted in smooth and efficient navigation.
The robot successfully avoided 98% of obstacles in real-time scenarios.

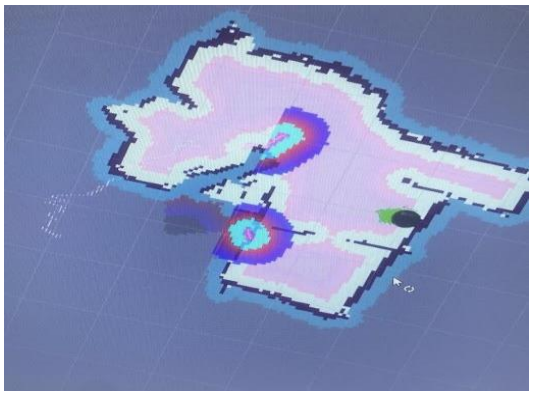

Figure5.1: Obstacle Avoidance Path 1

- **Speech Recognition and NLP:**

The robot's transcription accuracy for spoken prescriptions reached 90%.
Challenges with accents and background noise accounted for a 15% error rate.

5.2 Error/Success Rate Calculations

The overall performance of the system was evaluated using the following metrics:
Localization Error Rate: Calculated as the percentage of test cases where the robot's location estimate exceeded the 1-meter threshold.
Success Rate :
Localization Accuracy(zero noise + Structural Noise + Human-Induced Noise) / 3 =94.3%
Error Rate: 5.7%

Obstacle Avoidance Success Rate:
Success Rate = (Number of Obstacles Avoided / Total Obstacles Detected) × 100
Success Rate: 98%
Error Rate: 2%

Speech Recognition Error Rate:
Success rate without noise and correct drug spilling Approximately = 95%
Success rate without noise Approximately= 85%
Total success rate Approximately= 90 %

5.3 Justifications of the Obtained Results

The results obtained reflect the successful integration of various technologies:

1. **Localization Performance**: The system's high accuracy in WiFi fingerprinting is attributed to:
   - The extensive training of the model on diverse signal strength datasets.
   - Effective real-time data processing using optimized algorithms like LightGBM.

2. **Obstacle Avoidance**: The robot's high success rate in avoiding obstacles is due to:
   - Reliable sensor inputs from Kinect.

3. **Zone Prediction:**
   The results of our Wi-Fi fingerprinting localization test can be explained by two main factors:

   - **Noise Impact**:
     - **Structural Noise**: Things like open doors, windows, or moving objects can disrupt Wi-Fi signals, causing fluctuations in the signal strength (RSSI). This led to a decrease in accuracy, from 97% to 91%-95%, and an increase in error margin.

     - **Human-Induced Noise**: People moving near the robot can absorb or reflect Wi-Fi signals, causing further signal variations and reducing accuracy

.

- **Zone Prediction with LightGBM**:

  The LightGBM model performs well when the data is clean, but noise reduces its accuracy. The model's performance shows that having a good, varied training dataset is important for handling these changes. Using oversampling techniques also helps the model learn from a wider range of scenarios, improving its ability to predict zones despite noise.

5.4 Summary

This chapter presents the performance evaluation of a robot utilizing Wi-Fi fingerprinting for localization, obstacle detection, and speech recognition. The system demonstrated high accuracy in localization (94.3%) and obstacle avoidance (98%) while achieving 90% transcription accuracy in speech recognition. However, environmental factors such as structural and human-induced noise impacted localization accuracy, highlighting the need for robust datasets and oversampling techniques in model training. Overall, the integration of various technologies was successful, with the system performing well in real-world scenarios despite some challenges.

Chapter 6: Conclusion And Future Work

6.1 Conclusion

The Doctor Assistant Robot project successfully demonstrates how advanced technologies like WiFi fingerprinting, LiDAR, and speech recognition can be integrated to assist healthcare professionals in hospitals.
The robot can navigate hospital environments, deliver medical tools, and convert spoken prescriptions into electronic records, reducing errors and easing the workload on medical staff. Using ROS and algorithms like GMapping and LightGBM, the robot can move safely and efficiently, avoiding obstacles.

Despite challenges like Kinect camera identification issues and speech recognition limitations, the project achieved its core goals. The robot has proven its ability to operate in indoor environments and improve healthcare workflows.

6.2 Future Work:

1. Expanded Navigation Capabilities:

    - Extend the robot's navigation capabilities to include multi-floor environments and outdoor settings. This could involve integrating additional sensors or technologies, such as elevator control systems or GPS for outdoor navigation.

    - Implement advanced obstacle avoidance algorithms, such as Reinforcement Learning or Deep Learning-based approaches, to improve the robot's ability to navigate in highly dynamic environments.

2. Improved Localization:

    - Investigate alternative localization methods, such as Bluetooth beacons or Ultra-Wideband (UWB) technology, to reduce dependency on WiFi signals and improve accuracy in areas with poor WiFi coverage.

    - Explore the use of sensor fusion techniques to combine data from multiple sensors (e.g., LiDAR, IMU, and cameras) for more robust and precise localization.

3. Error Correction and Feedback Mechanisms:

- Implement more sophisticated error correction systems using machine learning to analyze and learn from past errors, improving the system's accuracy over time.

- Develop a user-friendly interface for healthcare professionals to provide feedback and corrections, ensuring continuous improvement in the robot's performance.

4. Scalability and Multi-Robot Coordination:

- Design the system to support multiple robots working collaboratively in large healthcare facilities. This could involve developing coordination algorithms to manage tasks and avoid conflicts between robots.

- Explore the use of cloud-based systems for centralized control and data sharing among multiple robots.

Appendix A:

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

Appendix B:

```
#!/usr/bin/env python

import pandas as pd
from sklearn.model_selection import train_test_split
from lightgbm import LGBMClassifier
from imblearn.over_sampling import RandomOverSampler
import joblib

# Load dataset
data = pd.read_csv('final.csv')
known_mac_addresses = data.columns[1:].tolist()
X = data.iloc[:, 1:].values
y = data['Zone'].values

# Resample the data to handle class imbalance
ros = RandomOverSampler(random_state=42)
X_resampled, y_resampled = ros.fit_resample(X, y)

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X_resampled, y_resampled, test_size=0.2,
random_state=42)

# Train the model
model = LGBMClassifier()
model.fit(X_train, y_train)
# Save the model to a file
model_filename = 'wifi_zone_model.pkl'
joblib.dump(model, model_filename)
print(f"Model saved to {model_filename}")

# Optionally, you can evaluate the model on the test set
accuracy = model.score(X_test, y_test)
                                print(f"Model accuracy: {accuracy:.2f}")
```

Appendix C

```
def load_drug_data(csv_file):
    Load drug data from a CSV file into a dictionary.
    drug_data = {}
    try:
        with open(csv_file, mode="r") as file:
            reader = csv.DictReader(file)
            for row in reader:
                drug_name = row["Drug Name"]
                notes = row["Notes"]
                drug_data[drug_name] = notes
        rospy.loginfo("Drug data loaded successfully.")
    except Exception as e:
        rospy.logerr(f"Error loading drug data from CSV: {e}")
    return drug_data
# Path to the CSV file
CSV_FILE = "drugs.csv"
# Load drug data
drug_data = load_drug_data(CSV_FILE)
def reduce_noise(audio_data, sample_rate=16000):
    Reduce noise in audio using noisereduce library.
     audio_array = np.frombuffer(audio_data, dtype=np.int16)
    reduced_audio = nr.reduce_noise(y=audio_array, sr=sample_rate)
    return reduced_audio.tobytes()
def transcribe_and_update_gui(recognizer, mic, text_box, pub):
    Recognizes speech and updates the Tkinter GUI with the transcription.
    with mic as source:
        recognizer.adjust_for_ambient_noise(source, duration=1)
        rospy.loginfo("Microphone calibrated for ambient noise.")
        while not rospy.is_shutdown():
            try:
                rospy.loginfo("Listening...")
                audio = recognizer.listen(source, timeout=10)
                # Reduce noise in audio data
                raw_audio = audio.get_raw_data()
                sample_rate = audio.sample_rate
                clean_audio = reduce_noise(raw_audio, sample_rate)
                # Recognize speech from noise-reduced audio
                clean_audio_data = sr.AudioData(clean_audio, sample_rate, audio.sample_width)
                text = recognizer.recognize_google(clean_audio_data)
                rospy.loginfo(f"Recognized Text: {text}")
```

```
# Publish the recognized text to ROS topic
            pub.publish(text)
# Extract drug name and dose from the recognized text
            drug_name = None
            dose = None
            for drug in drug_data.keys():
                if drug.lower() in text.lower():
                    drug_name = drug
                    # Extract dose (assuming dose is mentioned after the drug name)
                    dose_start = text.lower().find(drug.lower()) + len(drug)
                    dose = text[dose_start:].strip()
                    break
            # Update the GUI with the drug name, dose, and notes
            if drug_name and dose:
                notes = drug_data.get(drug_name, "No notes available")
                display_text = f"Drug: {drug_name}\nDose: {dose}\nNotes: {notes}\n\n"
                text_box.insert(tk.END, display_text)
                text_box.see(tk.END)  # Scroll to the bottom
            else:
                rospy.logwarn("Drug name or dose not recognized.")
        except sr.UnknownValueError:
            rospy.logwarn("Could not understand the audio.")
        except sr.RequestError as e:
            rospy.logerr(f"Speech Recognition API error: {e}")
        except Exception as e:
            rospy.logerr(f"Unexpected error: {e}")
def save_to_pdf(text_box, header, footer):
    Save the text from the ScrolledText widget to a PDF file, including the header and footer.
    text_content = text_box.get("1.0", tk.END).strip()
    if not text_content:
        rospy.logwarn("No text to save.")
        return
    file_path = filedialog.asksaveasfilename(defaultextension=".pdf",
                                    filetypes=[("PDF files", "*.pdf")],  title="Save as PDF")
    if file_path:
        try:
            pdf = FPDF()
            pdf.set_auto_page_break(auto=True, margin=15)
            pdf.add_page()
            pdf.set_font("Arial", size=12)
            # Add header
            pdf.set_font("Arial", style="B", size=16)
            pdf.cell(0, 10, header, ln=True, align="C")
            pdf.ln(10)  # Add space after header
```

```
        # Add content
          pdf.set_font("Arial", size=12)
          for line in text_content.split("\n"):
              pdf.cell(0, 10, txt=line, ln=True)
          pdf.ln(10)  # Add space before footer
          # Add footer
          pdf.set_font("Arial", style="I", size=10)
          pdf.cell(0, 10, footer, ln=True, align="C")
          pdf.output(file_path)
          rospy.loginfo(f"File saved as PDF: {file_path}")
      except Exception as e:
          rospy.logerr(f"Error saving PDF: {e}")
def main():
    rospy.init_node("speech_to_text", anonymous=True)
    pub = rospy.Publisher("speech_text", String, queue_size=10)
    recognizer = sr.Recognizer()
    mic = sr.Microphone()
    # Initialize Tkinter GUI
    root = tk.Tk()
    root.title("Real-Time Speech Transcription")
    # Header
    header_text = "Palestine Polytechnic University Hospital"
    header_label = tk.Label(root, text=header_text, font=("Arial", 16, "bold"), bg="lightblue")
    header_label.pack(fill=tk.X, pady=(10, 0))  # Add padding at the top
    # Scrollable Text Box for Transcription
    text_box = ScrolledText(root, wrap=tk.WORD, width=60, height=20)
    text_box.pack(padx=10, pady=10)
    # Footer
    current_date = datetime.now().strftime("%Y-%m-%d")  # Get the current date
    footer_text = f"© 2025 Palestine Polytechnic University Hospital | Date: {current_date}"
    footer_label = tk.Label(root, text=footer_text, font=("Arial", 10), fg="gray")
    footer_label.pack(side=tk.BOTTOM, pady=10)  # Add padding at the bottom
    # Buttons
    button_frame = tk.Frame(root)
    button_frame.pack(fill=tk.X, pady=5)
    save_pdf_button = tk.Button(button_frame, text="Save as PDF",
                                command=lambda: save_to_pdf(text_box, header_text, footer_text))
    save_pdf_button.pack(side=tk.LEFT, padx=10)
    # Run the transcription process in a separate thread
    transcription_thread = threading.Thread(target=transcribe_and_update_gui, args=(recognizer, mic, text_box,
pub))
    transcription_thread.daemon = True  # Ensures the thread exits when the main program exits
    transcription_thread.start()
    # Start the Tkinter GUI loop
    root.mainloop()
if __name__ == "__main__":
    try:
        main()
    except rospy.ROSInterruptException:
        pass
```

Appendix D

Basic installation commands:

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

```
$ Wget
https://raw.githubusercontent.com/qboticslabs/ros_install_noetic/
master/ros_install_noetic.sh && chmod +x ./ros_install_noetic.sh
&& ./ros_install_noetic.sh
$ sudo apt install ros-noetic-desktop-full
```

Environment setup

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

TurtleBot Installation

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

Mapping using SLAM

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

```
$ roslaunch turtlebot_bringup minimal.launch
```

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

```
$ roslaunch turtlebot_navigation gmapping_demo.launch
```

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

```
$ roslaunch turtlebot_rviz_launchers view_navigation.launch
```

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

```
$ roslaunch turtlebot_teleop keyboard_teleop.launch
```

6. Save the Map: To save the pre-mapped environment, utilize the map_saver tool.

Execute the following command, replacing /path/to/save/the/map with the desired file path:

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

Appendix E: sample of WIFI data set

Zone number	Mac address						
Zone	12:82:3d:86:ba:5b	00:eb:d8:60:9e:95	70:4c:a5:c2:d9:69	18:d6:c7:b7:fc:40	e8:1c:ba:84:35:01	e8:1c:ba:46:9d:48	12:82:3d:86:ba:a6
0	-51	-33	-59	-100	-100	-71	-100
0	-57	-37	-59	-100	-100	-71	-100
0	-57	-37	-100	-100	-100	-100	-100
0	-57	-39	-100	-100	-100	-100	-100
0	-57	-41	-49	-73	-100	-100	-100
0	-65	-47	-55	-73	-73	-100	-100
0	-100	-37	-100	-100	-69	-100	-100
4	-61	-41	-39	-67	-73	-71	-67
4	-61	-41	-39	-67	-73	-71	-67
4	-57	-43	-45	-67	-100	-71	-67
4	-39	-27	-43	-67	-100	-59	-69
4	-39	-27	-43	-67	-100	-59	-69
4	-39	-27	-43	-67	-100	-59	-69
4	-57	-37	-41	-67	-100	-67	-67
4	-57	-37	-41	-67	-100	-67	-67
4	-57	-37	-41	-67	-100	-67	-67
4	-47	-37	-39	-71	-100	-67	-67
23	-43	-31	-35	-63	-100	-59	-57
23	-43	-31	-35	-63	-100	-59	-57
23	-47	-29	-37	-63	-57	-67	-57
23	-47	-29	-37	-63	-57	-67	-57

References:

[1] World Health Organization (WHO). (2017). *Medication Without Harm – WHO Global Patient Safety Challenge on Medication Safety*. Retrieved , last access date: Nov 20 ,2024,from https://www.who.int/initiatives/medication-without-harm

[2] Jennings, B. M. (2008). *Work Stress and Burnout Among Nurses: Role of the Work Environment and Working Conditions. Journal of Clinical Nursing, 17*(24), 3262-3270. doi: 10.1111/j.1365-2702.2008.02686.x

[3] World Health Organization (WHO). (2021). *Health Worker Occupational Health and Safety: Preventing Injuries and Infections.* Retrieved from https://www.who.int/initiatives/medication-without-harm

[4] Betsy Lehman Center for Patient Safety. (2019). *The Financial and Human Cost of Medical Errors in Massachusetts.* Retrieved from https://betsylehmancenterma.gov/assets/uploads/Cost-of-Medical-Error-Report-2019.pdf

[5] SLAM (Simultaneous Localization and Mapping), last access date: Jan 20 ,2024,https://www.mathworks.com/discovery/slam.html

[6] Localization, Mapping and SLAM using GMapping, last access date: Nov 20 ,2024 https://softillusion-robotics.medium.com/gmapping-d26c13b1b69

[7] Y. Li and C. Shi, "Localization and Navigation for Indoor Mobile Robot Based on ROS," 2018 Chinese Automation Congress (CAC), Xi'an, China, 2018, pp. 1135-1139, doi: 10.1109/CAC.2018.8623225, https://ieeexplore.ieee.org/abstract/document/8623225

[8] cs.cmu.edu, Mobile robot localization, last access date: Nov 20 ,2024 https://www.cs.cmu.edu/~rasc/Download/AMRobots5.pdf

[9] Kumar, Dr. Neerendra,Vamossy, Zoltan, “Robot navigation with obstacle avoidance in unknown environment” 2018, https://www.researchgate.net/publication/328774686_Robot_navigation_with_obstacle_avoidance_in_unknown_environment

[10] control.com, An Introduction to Simultaneous Localization and Mapping (SLAM) for Robots, last access date: Nov 20 ,2024,https://control.com/technical-articles/an-introduction-to-simultaneous-localization-and-mappi ng-slam-for-robots/

[11] weka.io,Machine Learning vs. Neural Networks (Differences Explained), 2021,

https://www.weka.io/learn/ai-ml/machine-learning-vs-neural-networks/

[12] neptune.ai,2023, Image Processing in Python: Algorithms, Tools, and Methods You Should Know, https://neptune.ai/blog/image-processing-python

[13] Efthymios Karangelos, Louis Wehenkel,Cyber–physical risk modeling with imperfect cyber-attackers,Electric Power Systems Research,Vol 211,2022, https://www.sciencedirect.com/science/article/pii/S0378779622005739

[14] Bakri, M. & Ismail, Abdul Halim & Mohamad Hashim, Mohd Sani & Muhamad Azmi, Muhamad Safwan & Safar, M. Juhairi Aziz & Marhaban, Mohammad,2020, Design and Development of a Service Robot for Wi-Fi RSSI Fingerprint Data Collection. https://www.researchgate.net/publication/345195497_Design_and_Development_of_a_Service_Robot_for_Wi-Fi_RSSI_Fingerprint_Data_Collection

[15] Yucel, Hikmet & Elibol, Gulin & Yayan, Ugur. (2020). Wi-Fi Based Indoor Positioning System For Mobile Robots By Using Particle Filter. https://www.researchgate.net/publication/346933677_Wi-Fi_Based_Indoor_Positioning_System_For_Mobile_Robots_By_Using_Particle_Filter

[16] A. Shaheen, M. E'mar, Storing and Classification Robot, palestine polytechnic university, 2013.

[17] A.Amleh ,F.Mohtaseb, M. Mohtaseb, An intelligent Mobile Robot that localises itself Inside a Maze, palestine polytechnic university, 2020.

[18] H.Dwaik, M.Hrebat, M.Natsheh, Object Finder Robot, palestine polytechnic university, Object Finder Robot , 2023.

[19] Autonomous Mobile Robots, last access date: Jan 10 ,2024, https://www.neobotix-robots.com/

[20] TurtleBot is a low-cost, personal robot kit with open-source software, last access date: Feb 15 ,2024, http://www.turtlebot.com/

[21] Neobotix - Robotics & Automation, last access date: Nov 20 ,2024, https://wiki.ros.org/Robots/neobotix

[22] Robot Operating System, https://wiki.ros.org/Documentation

[23] clearpathrobotics.com, Jackal UGV - Small Weatherproof Robot – Clearpath, https://clearpathrobotics.com/jackal-small-unmanned-ground-vehicle/

[24] warf.com ,TURTLEBOT 2 COMPLETE KIT, https://www.airwaveplus.com/category/warf

[25] raspberrypi.com, Buy a Raspberry Pi 3 Model B, https://www.raspberrypi.com/products/raspberry-pi-3-model-b/

[26] wikipedia.org, Laptop, last access date: Nov 20 ,2024, https://en.wikipedia.org/wiki/Laptop

[27] Raspberrypi-spy.co.uk, Introducing the Raspberry Pi 3 B+ Single Board Computer, https://www.raspberrypi-spy.co.uk/2018/03/introducing-raspberry-pi-3-b-plus-computer

[28] A. Shaheen, M. E'mar, Storing and Classification Robot, palestine polytechnic university, 2013.

[29] terabee.com, Depth sensors: Precision & personal privacy, https://www.terabee.com/depth-sensors-precision-personal-privacy/

[30] Ali, Mohammed & Liu, Hui & Stoll, Norbert & Thurow, Kerstin. (2017). Grasping and Placing Operation for Labware Transportation in Life Science Laboratories using Mobile Robots. Advances in Science, Technology and Engineering Systems Journal. 2. https://www.researchgate.net/publication/318646867_Grasping_and_Placing_Operation_for_Labware_Transportation_in_Life_Science_Laboratories_using_Mobile_Robots

[31] Researchgate.net, Characteristics of the lidar sensor, https://www.researchgate.net

[32] N. Niu, S. Nan, J. Zhang and Y. Wang, "LSSVM prediction model based on improved kenel parameter selection method," 2020 11th International Conference on Prognostics and System Health Management (PHM-2020 Jinan), Jinan, China, 2020, https://ieeexplore.ieee.org/document/9296717

[33] Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications, https://standards.ieee.org/ieee/802.11/7028/

[34] Inertial Measurement Units, https://www.sciencedirect.com/science/article/pii/S0968090X12000627

[35] Robot Operating System, last access date: Mar 20 ,2024, http://wiki.ros.org/

[36] Kotaru, M., Joshi, K., Bharadia, D., & Katti, S. (2015). SpotFi: Decimeter Level Localization Using WiFi. *ACM SIGCOMM Computer Communication Review*, 45(4), 269-282. https://doi.org/10.1145/2829988.2787487

[37] Wazwaz, A.; Amin, K.; Semary, N.; Ghanem, T. Dynamic and Distributed Intelligence over Smart Devices, Internet of Things Edges, and Cloud Computing for Human Activity Recognition Using Wearable Sensors. *J. Sens. Actuator Netw.* 2024, *13*, 5. https://doi.org/10.3390/jsan13010005

[38] Ubuntu releases, Ubuntu MATE 20.04.6 LTS (Focal Fossa), Date accessed:Dec 20, 2024, Available from:ubuntu-mate releases.

[39] ros.org, ROS, Date accessed:Dec 20, 2024, Available from:ros

[40] ros.org, Turtlebot Installation, Date accessed: Dec 20, 2024, Available from:Turtlebot Installation.

[41] Medium, ROS Autonomous SLAM using Randomly Exploring Random Tree (RRT), Date accessed:Dec 20, 2024, Available from: ROS Autonomous SLAM

[42] Researchgate.net, Characteristics of the lidar sensor ,Date accessed:Nov 5, 2024, Available from : https://www.researchgate.net/figure/Characteristics-of-the-lidar-sensor_tbl1_340344846

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

[44] Wazwaz, Ayman. "Analysis of QoS parameters of VOIP calls over Wireless Local Area Networks." *The 13th International Arab Conference on Information Technology ACIT', Dec*. 2012.

[45] Ayman A. Wazwaz, Khalid M. Amin, Noura A. Semari, Tamer F. Ghanem, "Enhancing human activity recognition using features reduction in IoT edge and Azure cloud", Decision Analytics Journal, Volume 8, 2023, 100282, ISSN 2772-6622, https://doi.org/10.1016/j.dajour.2023.100282

[46] U.S. National Library of Medicine. (2023). Date accessed:Jan 6, 2025,Drug Information Portal. https://druginfo.nlm.nih.gov/