

Article

Web-Based Real-Time Alarm and Teleoperation System for Autonomous Navigation Failures Using ROS 1 and ROS 2

Nabih Pico ^{1,2,*} , Giovanni Mite ^{3,†}, Daniel Morán ^{3,†}, Manuel S. Alvarez-Alvarado ² , Eugene Auh ¹  and Hyungpil Moon ^{1,*} 

¹ Department of Mechanical Engineering, Sungkyunkwan University, Suwon 16419, Republic of Korea; egauh@skku.edu

² Facultad de Ingeniería en Electricidad y Computación, Escuela Superior Politécnica del Litoral (ESPOL), Campus Gustavo Galindo, Guayaquil 09-01-5863, Ecuador; mansalva@espol.edu.ec

³ Facultad de Ingeniería Mecánica y Ciencias de la Producción, Escuela Superior Politécnica del Litoral (ESPOL), Campus Gustavo Galindo, Guayaquil 09-01-5863, Ecuador; gwmite@espol.edu.ec (G.M.); damogarc@espol.edu.ec (D.M.)

* Correspondence: npico@skku.edu (N.P.); hyungpil@skku.edu (H.M.)

† These authors contributed equally to this work.

Abstract: This paper presents an alarm system and teleoperation control framework, comparing ROS 1 and ROS 2 within a local network to mitigate the risk of robots failing to reach their goals during autonomous navigation. Such failures can occur when the robot moves through irregular terrain, becomes stuck on small steps, or approaches walls and obstacles without maintaining a safe distance. These issues may arise due to a combination of technical, environmental, and operational factors, including inaccurate sensor data, sensor blind spots, localization errors, infeasible path planning, and an inability to adapt to unexpected obstacles. The system integrates a web-based graphical interface developed using frontend frameworks and a joystick for real-time monitoring and control of the robot's localization, velocity, and proximity to obstacles. The robot is equipped with RGB-D and tracking cameras, a 2D LiDAR, and odometry sensors, providing detailed environmental data. The alarm system provides sensory feedback through visual alerts on the web interface and vibration alerts on the joystick when the robot approaches walls, faces potential collisions with objects, or loses stability. The system is evaluated in both simulation (Gazebo) and real-world experiments, where latency is measured and sensor performance is assessed for both ROS 1 and ROS 2. The results demonstrate that both systems can operate effectively in real time, ensuring the robot's safety and enabling timely operator intervention. ROS 2 offers lower latency for LiDAR and joystick inputs, making it advantageous over ROS 1. However, camera latency is higher, suggesting the need for potential optimizations in image data processing. Additionally, the platform supports the integration of additional sensors or applications based on user requirements.

Keywords: teleoperation system; web framework; real-time alarm; joystick design



Academic Editor: Zhuming Bi

Received: 30 January 2025

Revised: 8 March 2025

Accepted: 24 March 2025

Published: 26 March 2025

Citation: Pico, N.; Mite, G.; Morán, D.; Alvarez-Alvarado, M.S.; Auh, E.; Moon, H. Web-Based Real-Time Alarm and Teleoperation System for Autonomous Navigation Failures Using ROS 1 and ROS 2. *Actuators* **2025**, *14*, 164. <https://doi.org/10.3390/act14040164>

Copyright: © 2025 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (<https://creativecommons.org/licenses/by/4.0/>).

1. Introduction

The advancements in mobile robot operational techniques have been aimed at enhancing autonomy, allowing robots to perform tasks independently with minimal human intervention. Modern autonomous robots leverage sophisticated decision-making strategies powered by artificial intelligence, enabling them to plan, execute, and adjust their actions based on dynamic environmental factors. These robots navigate complex environments, whether in warehouses, industries, shopping centers, or universities, handling tasks such

as load transportation, patrolling, and customer assistance. The ability to autonomously follow paths and avoid obstacles is crucial for robots to operate effectively in both controlled and dynamic environments. However, ensuring safe and efficient navigation requires the integration of trajectory planning [1,2], where robots calculate the optimal paths based on data from various sensors, including LiDAR and depth cameras. These sensors provide valuable real-time data that allow robots to adjust their movements as they encounter obstacles, ensuring they can navigate efficiently even in unfamiliar or changing environments. In tasks like food or room service deliveries, where time and accuracy are critical, trajectory planning plays a vital role in determining the fastest and safest paths to achieve the goals [3]. Robots rely heavily on perception systems to understand their surroundings and gather the data necessary for decision-making. Using sensors, robots continuously collect real-time information about their environment, enabling them to map spaces [4,5], detect obstacles, and identify hazards such as steps or stairs on their paths [6]. Perception also involves continuously monitoring variables such as speed, position, and object mapping, which are critical for real-time decision-making and trajectory adjustments. Examples include researchers using RADAR for human tracking [7] or predicting human behaviors [8]. Control mechanisms are also at the heart of mobile robot operations, ensuring that robots function optimally in both autonomous and teleoperated modes. In autonomous mode, decision-making strategies integrate real-time sensor feedback to control robot movements and adjust routes as needed. Algorithms such as MPC, RPP, DWA, and RL are common approaches for autonomous navigation [7,9]. On the other hand, operators continuously monitor critical robot variables, such as speed, position, and environmental mapping, to ensure that the robot performs as expected. If problems arise, such as delayed feedback or sensor mismanagement, the operator can intervene and make the necessary adjustments [10–13]. Effective control requires the integration of embedded systems, sensors, and actuators for navigation algorithms. Delays or mismanagement of sensor data can increase the risk of trajectory failures, crashes, or falls, making reliable control systems vital for robot safety and efficiency.

Today, artificial intelligence and traditional navigation algorithms enable autonomous decision-making without human intervention. However, it is recommended to implement a system that can send alerts and report events that endanger the stability of the robot and cannot be resolved by current autonomous algorithms. Such events include scenarios where the robot moves over irregular terrain, becomes stuck on small steps in its path, or collides with walls due to localization failures, leading to a loss of the correct path during navigation. Therefore, teleoperation systems provide a crucial backup mechanism when autonomous decision-making is insufficient or not possible due to limited real-time data or unexpected obstacles. In these cases, human operators step in to guide the robot manually, ensuring its continued operation and safety. Even though teleoperation enhances robot reliability, it poses challenges such as latency and the demanding task of monitoring multiple variables simultaneously.

Based on previous approaches, this work aims to integrate information from onboard sensors, including cameras, LiDAR, and IMU, into a teleoperation architecture featuring a web-based real-time alarm system and a physical device (joystick) that communicates with ROS 1 or ROS 2, as shown in Figure 1. The goal is to compare both robot operating systems and provide effective assistance while facilitating the monitoring of mobile robots within autonomous navigation plans.

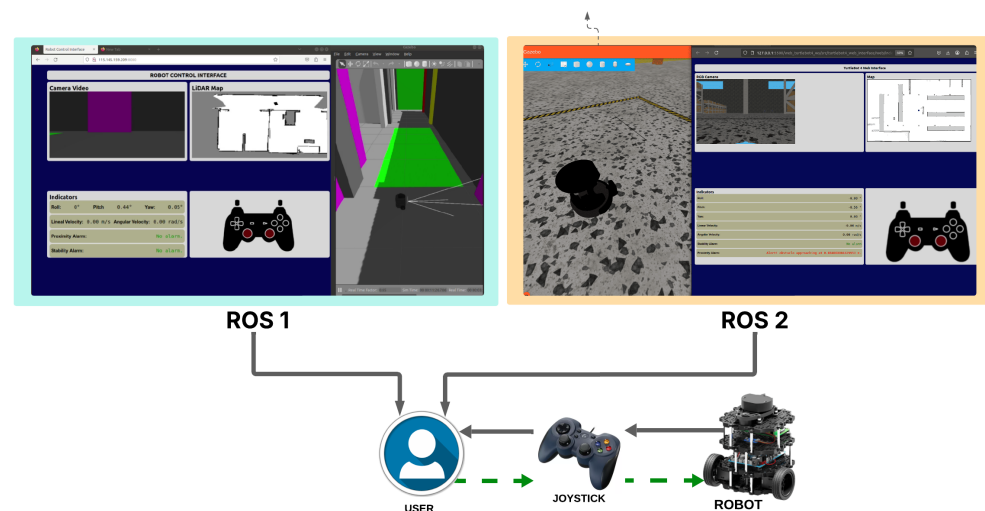


Figure 1. Web interfaces for a teleoperation system: the left side using ROS 1 and the right side using ROS 2.

The key contributions of this work include the following:

- A frontend framework for ROS 1 and ROS 2 that enables users to access real-time sensor and environmental data, including the robot's trajectory, area mapping, camera inputs, position, and both linear and rotational speed.
- An alarm system that monitors: (a) the vibration and inclination thresholds of the robot, caused by irregular terrain or climbing attempts that may compromise its stability, and (b) the distance thresholds between the robot and nearby objects, indicating potential collisions. Alerts are displayed on the web interface and joystick, with increasing vibrations signaling anomalies, ensuring the operator receives timely notifications about critical events.
- Simulations and experiments involving the web interface and joystick design, which enable increased vibration feedback based on alarms triggered during navigation tasks.

This paper is organized as follows: Section 2 reviews previous research on teleoperation and HID systems in mobile robots. Section 3 outlines the problem formulation and system architecture, explaining the robot's sensor integration and joystick design. Section 4 discusses the ROS 1 and ROS 2 architecture for the system, the alarm, and the communication between the joystick, robot, and graphical interface. Section 5 explain the haptic joystick design and ROS vinculation. Section 6 presents the simulations and experiments, and Section 7 details the conclusion of this work.

2. Related Work

Robotics has undergone significant evolution, with companies increasingly investing in robotic solutions to improve efficiency and functionality. In mobile robot systems, which are multiple-input-multiple-output (MIMO) systems, uncertainty can significantly affect performance. Addressing it is crucial for ensuring robust and reliable performance in dynamic environments, where challenges may arise from sensor inaccuracies, dynamic obstacles, and the complexity of the environment itself. Several approaches have been proposed to handle uncertainty, including probabilistic models, uncertainty propagation techniques, and robust planning methods.

One common approach is the use of probabilistic robotics, which incorporates uncertainty into state estimation and decision-making. Methods such as Kalman filters and Particle filters are widely used to estimate the robot's position and correct for errors due

to noisy sensor measurements [14]. These techniques provide a framework for filtering out uncertainty, allowing robots to make informed decisions about their movement and environment.

Simultaneous Localization and Mapping (SLAM) is another technique that addresses uncertainty by combining sensor data to create maps while simultaneously estimating the robot's position within the environment [15,16]. This is handled through probabilistic data association and optimization techniques, which ensure that the robot can operate effectively even when some parts of the map are less certain due to noisy sensor readings.

Another approach is robust path planning, where planners take into account uncertainties in the environment, such as dynamic obstacles, and optimize the robot's trajectory to minimize risks. Robust methods often rely on optimization frameworks and safety margins to ensure that the robot can adapt to unexpected changes in the environment [17,18]. In the case of sliding in autonomous navigation, this is often addressed by selecting a constant switching gain through trial and error. However, if uncertainty increases suddenly without an automatic mechanism to adjust the switching gains, the robot may deviate from its intended trajectory. To address these issues, a robust adaptive control law has been proposed that allows the robot to track the desired trajectory even in the presence of uncertainties and external forces [19,20]. This approach improves the robot's stability and ability to adapt to changing environmental conditions.

Significant advances have been made in optimizing autonomous navigation algorithms, such as reinforcement learning (RL) [21], to plan actions under constraints like the field of view (FOV) of sensors [22] and navigation in limited spaces, where traditional approaches face challenges in navigating and reaching the goal [23]. However, despite improvements in autonomous navigation, perception limitations often require human intervention to ensure task completion. This necessity has driven the continued development of robotic teleoperation systems.

Robotic teleoperation has played a crucial role in facilitating robot navigation and operator control. The integration of SLAM technology has improved robot localization within its working environment, ensuring greater safety and precision. During the COVID-19 pandemic, SLAM using RGB and depth images was employed in hospital robots to reduce cross-infection and enhance operational efficiency [4]. In construction, an autonomous 3D spatial data collection method using cameras and lasers was presented, tested on a ground robot for structural mapping [5]. Additionally, RGB-D cameras have been employed to improve robotic performance in irregular terrains by detecting and modeling steps, slopes, and stairs for climbing and descending tasks [6].

Virtual reality (VR) has also been explored for improving teleoperation. A two-armed teleoperated robot controlled through novel VR techniques was tested across different age groups to assess VR's impact on teleoperation performance [12]. Similarly, a teleoperation system using HTC Vive Pro 2 VR goggles was developed to control a KUKA youBot robot. This system included a Unity-based virtual environment for user training, integrated with ROS for seamless operation [24].

Other innovative approaches to teleoperation include multimodal control and gesture-based interaction. A system was developed that allows operators to control robots using hand movements, integrating IMU-based arm tracking and computer vision for hand position recognition [25]. Stability in a bilateral teleoperation system designed to operate mobile robots across different countries via the internet has also been analyzed [26].

Wheel slippage in mobile robots during teleoperation can cause instability in movement. To address this, the traditional dynamic model has been extended to account for slippage and enhance performance [11]. Additionally, an Android application using the User Datagram Protocol (UDP) for two-way communication with ROS has been devel-

oped, enabling the teleoperation of a Turtlebot3 robot in both simulated and real environments [27]. A taxonomy of design guidelines for robotic teleoperation interfaces has also been provided, offering a general framework based on an extensive literature review [28].

Modern teleoperation systems aim to assist operators in making optimal control decisions. An assisting system for a teleoperated robotic gripper that suggests optimal grasps based on quality measures has been developed [13]. Furthermore, a Genetic Adaptive User Interface (GAUI) that employs an unsupervised learning algorithm to optimize teleoperation performance in robots with many degrees of freedom has been introduced [29]. Based on these previous approaches, the next section presents the problem formulation and the proposed teleoperation system for robotic applications using ROS 1 and ROS 2.

3. Problem Formulation and Teleoperation System

Significant advancements in autonomous robot navigation have been made, enabling robots to navigate through diverse environments without human intervention. However, despite these advancements, gaps remain in their ability to respond to unexpected events in real-world scenarios. These limitations in autonomy can hinder a robot's operation, particularly in complex or dynamic settings. Teleoperation is crucial in such situations, as it allows human operators to intervene and help the robot overcome obstacles, ensuring safe navigation. Our goal is to create an architecture that provides assistance during robotic tasks, which can be replicated and scaled for various robotic applications using ROS 1 or ROS 2. This work focuses on scenarios where robots operating in indoor environments encounter failures in navigation algorithms, leading to collisions with walls or objects, instability on irregular terrain, or situations where the robot becomes stuck. In such cases, the teleoperation system issues alerts upon detecting potentially dangerous situations, such as imminent collision events or excessive robot vibrations.

3.1. Risky Cases

In collision prevention approaches, as illustrated in Figure 2a, it has been proposed that robots should possess the capability to emit proximity alarms, focusing on their distance to both static and dynamic objects. Static objects refer to immovable items in the environment, such as walls, furniture, or other obstacles that could block the robot's path or cause a collision. Dynamic agents include people or other mobile objects that may pose a threat to the robot's trajectory. The system continuously processes LiDAR data to evaluate the distance to these agents and generates alerts when the robot is at risk of colliding with any of them, especially when within a threshold distance of less than 25 cm from any obstacle.

In vibration-handling approaches, as shown in Figure 2b, the robot's orientation is monitored using data from the IMU, focusing on situations that could lead to falls or instabilities within its working environment. To assess the robot's orientation, Euler angles are tracked in real time, with particular emphasis on the pitch and roll angles, as these provide critical information about the robot's potential risk of tipping over. Such risks may arise while navigating uneven terrain, climbing slopes, or encountering obstacles that could destabilize it. When the pitch or roll angle exceeds a predefined threshold value, the robot emits an alarm through the "stability alarm" indicator to alert operators to potential instability.

By continuously tracking these parameters, the system provides valuable real-time data to help prevent accidents. The integrated feedback alert system notifies the operator of potential hazards through visual and/or vibration signals, prompting immediate action. This feedback system is crucial for reducing response times when the robot becomes stuck or encounters issues, ensuring that the operator can quickly and effectively intervene to

maintain the safety and reliability of the robot's operation during failures in autonomous navigation algorithms.

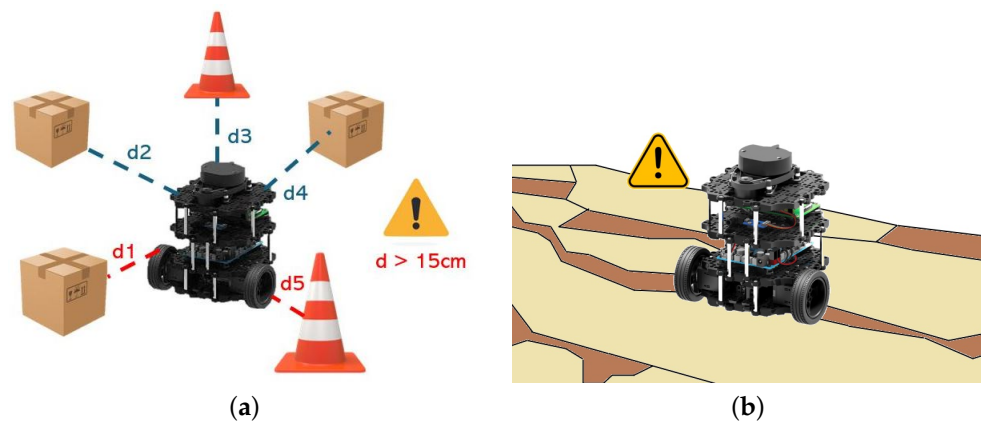


Figure 2. Scenarios for alarm activation. (a) Collision prevention case; (b) High vibration case.

3.2. Teleoperation System Architecture

The teleoperation system architecture, depicted in Figure 3, integrates the robot's computer and sensors with a web-based graphical interface and a joystick device, enabling users to monitor critical robot data in real time. The robot and sensors transmit data to the main computer via an SSH port, which is connected to the local network through a Wi-Fi connection. The main computer processes the data for the web interface and receives joystick input.

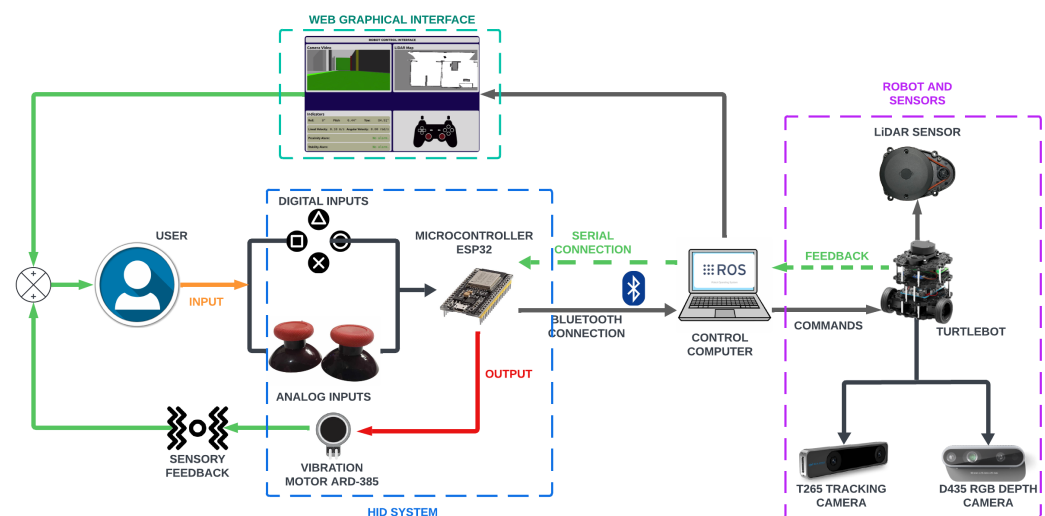


Figure 3. Overview of the system architecture for monitoring and teleoperation control.

The robot's LiDAR sensors, IMU, and depth camera continuously capture data about its surroundings. The sensors are compatible with ROS, including the Real-Time Tracking Camera Intel RealSense T265, which measures the robot's orientation, and the D435 RGB-D camera, which provides real-time images of the environment. Both cameras operate at 30 Hz with a resolution of 640×480 pixels. The LDS01 2D LiDAR sensor operates at 5 Hz with a 360-degree field of view, measuring the distance between the robot and surrounding objects, generating a map of the environment, and assisting in localization in conjunction with odometry, which operates at 25 Hz for precise motion tracking.

These data are visualized on the graphical interface, allowing users to monitor the robot's position, proximity to obstacles, and other essential environmental factors. Addi-

tionally, the system incorporates two types of alarms: proximity alarm and robot stability monitoring. Both alarms alert the user through the web interface and vibrations on the joystick, which are triggered by a vibration motor (ARD-385) included in its design.

The collision alarm activates when the robot approaches an obstacle (less than 25 cm away). This alarm notifies the user through visual cues on the interface and triggers joystick vibrations, with intensity inversely proportional to the robot's distance from the obstacle. The stability alarm is activated when the robot navigates over irregular terrain, potentially compromising its stability. This alarm provides both visual and vibration feedback through the joystick, allowing the teleoperator to take timely action to maintain safety and functionality. Therefore, the interface ensures that relevant information is easily accessible, supporting effective teleoperation control in emergency situations.

4. Software Design

This section outlines each stage involved in developing the proposed solution, specifically the programming of the teleoperation system. It covers ROS communication with the graphical user interface. Python 3.8 for ROS 1 and Python 3.10 for ROS 2 are used as the programming languages for node development.

4.1. Definition of the ROS Environment

The teleoperation system utilizes ROS nodes to enable seamless communication between sensors, computers, and users. In this work, both ROS 1 and ROS 2 were used to compare their structure, design, and performance. Ubuntu 20.04 LTS with ROS 1 Noetic was used along with the SDK for the TurtleBot3 robot within the Gazebo Classic simulation environment. Meanwhile, Ubuntu 22.04.5 LTS with ROS 2 Humble was utilized for TurtleBot 4, using the Gazebo Ignition Fortress simulator during the simulation. Figure 4 presents the ROS architecture diagrams, highlighting the nodes used in the robot's teleoperation system. These diagrams illustrate the interconnections between the nodes (green), their interactions with peripheral sensors and the joystick (pink), the messages transmitted via topics (yellow), and their integration with the web interface (orange). The rosbriidge server node enables real-time communication, ensuring that the entire system operates through the ROS backend, where all processing is performed natively on the control computer. The web interface displays real-time alerts and system status updates, while the joystick provides vibration feedback, both assisting the operator in making informed decisions regarding the robot's control.

The primary difference between ROS 1 and ROS 2 lies in their communication frameworks. ROS 1 relies on the ROS Master for centralized communication, using TCP as the default transport protocol, with the option to use UDP for certain applications. In contrast, ROS 2 employs DDS (Data Distribution Service), which uses UDP by default through the RTPS (Real-Time Publish-Subscribe) protocol, enabling decentralized peer-to-peer communication.

Both ROS 1 and ROS 2 support external WebSocket communication through the rosbriidge suite, enabling external clients to interact with nodes. In ROS 1, the ROS Master serves as a centralized directory that manages nodes and topics. However, this introduces a critical limitation: if the ROS Master fails, the entire system becomes nonfunctional. On the other hand, DDS eliminates the need for a central directory. It enables real-time, decentralized communication where nodes discover and interact with each other directly, making ROS 2 more robust and better suited for real-time robotic applications. The decentralized nature of DDS allows for low-latency communication and supports configurable Quality of Service (QoS) policies for greater control over reliability and performance.

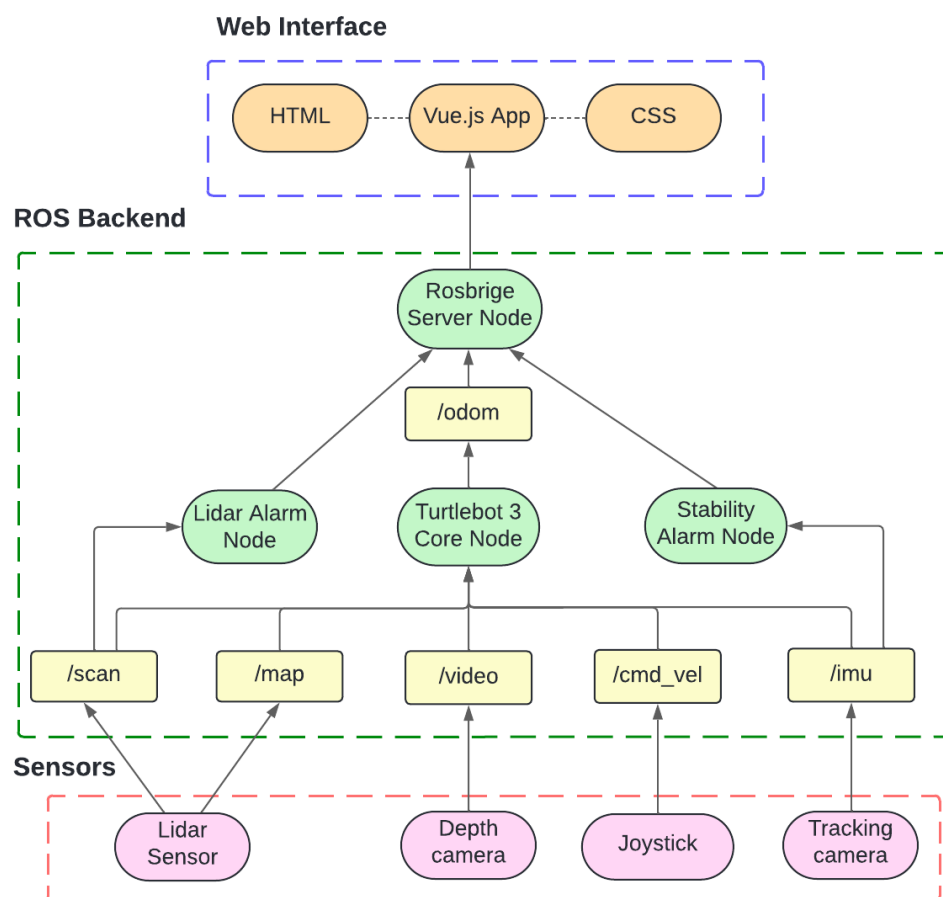


Figure 4. ROS nodes architecture.

4.2. Proximity Alarm Function

We implement the proximity alarm function as shown in Algorithm 1. In this function, the LDS01 LiDAR provides distance measurements within a 360-degree field of view in 2D, which are published through the `/scan` topic. In each iteration, we calculate the minimum distance $d_{\min}$, representing the closest detected obstacle (steps 1–11). From steps (1–3), the procedure begins by initializing $d_{\min}$ and setting the threshold to 50 cm. This threshold value represents the initial distance at which the proximity alarm is triggered, indicating when the robot is approaching an obstacle. The `ExtractDistancesBelowLimit` function filters out distances that exceed the threshold (step 4). Steps 5 to 9 describe the iteration process through all distances, where $d_{\min}$ holds the smallest value, which corresponds to the closest detected obstacle. In step 10, we return $d_{\min}$, which now represents the minimum distance between the robot and any obstacle within its field of view.

In the `ActivateAlarm` procedure (steps 12–25), we first retrieve the minimum distance $d_{\min}$ by calling the `MinDistanceObs` procedure. We then compare $d_{\min}$ with the predefined proximity threshold of 50 cm. If $d_{\min}$ is smaller than 50 cm, we trigger the proximity alarm and set the `ActiveAlarm` variable to true step 17. This alerts the teleoperator that the robot is approaching an obstacle. In steps 19 to 21, if $d_{\min}$ is smaller than 25 cm, it indicates that the robot is very close to the obstacle. In this case, the intensity of the vibration feedback increases proportionally to alert the operator about the critical proximity. If no alarm is triggered, the `ActiveAlarm` variable is set to false in steps 22–23.

These threshold values were determined based on the physical properties of the robot. The maximum velocity of the robot is 0.22 m/s, and a potential collision risk arises when the robot moves from 50 cm to 25 cm or below. This process ensures that the robot provides

real-time feedback to the teleoperator about its proximity to obstacles, allowing the operator to intervene if necessary.

Algorithm 1 Proximity Alarm Function

```

1: MINDISTANCEOBS gets the minimum distance to an obstacle
2: procedure MINDISTANCEOBS(ScanMsg)
3:    $d_{\min}, d_{\text{threshold}} \leftarrow \infty, 25 \text{ cm}$ 
4:   distances  $\leftarrow$  EXTRACTDISTANCESBELOWLIMIT(ScanMsg)
5:   for d in distances do
6:     if  $d < d_{\min}$  then
7:        $d_{\min} \leftarrow d$ 
8:     end if
9:   end for
10:  return  $d_{\min}$ 
11: end procedure
12: ACTIVATEALARM triggers the proximity alarm if the condition is met
13: procedure ACTIVATEALARM(ScanMsg)
14:    $d_{\text{threshold}} \leftarrow 50 \text{ cm}$  ▷ Initial threshold for proximity alarm
15:    $d_{\min} \leftarrow$  MINDISTANCEOBS(ScanMsg)
16:   if  $d_{\min} < d_{\text{threshold}}$  then
17:     TRIGGERFIRSTALERT ▷ First alert triggered at 50 cm distance
18:   end if
19:   if  $d_{\min} < 25 \text{ cm}$  then
20:     INCREASEVIBRATIONINTENSITY ▷ Increase vibration intensity if within 25 cm
21:     ActiveAlarm  $\leftarrow$  True
22:   else
23:     ActiveAlarm  $\leftarrow$  False
24:   end if
25: end procedure

```

4.3. Stability Alarm Implementation

The T265 tracking camera provides the robot orientation in quaternion form. These quaternions are converted into Euler angles, which are then published to the web interface. Terrain-induced vibrations cause fluctuations in the robot's roll and pitch angles, potentially affecting its stability and navigation. To alert the user of these instabilities and enable timely corrective actions, a threshold value, β , is set to 2 degrees. This threshold is based on the detected angle when the robot encounters cables with a diameter of 0.5 cm, which affects its climbing ability. If the roll or pitch angle exceeds this threshold, an alarm is triggered. The alarm remains constant depending on the magnitude and persistence of the oscillations.

The stability alarm function, as shown in Algorithm 2, is described as follows: In steps 1–6, the `GetEulerAngle` procedure extracts the quaternion data from the odometry in simulation or the IMU in the experiments and converts them into Euler angles (roll, pitch, and yaw). The alarm is activated in steps 7–8, followed by setting the (β) threshold value for the roll and pitch angles, and the current roll, pitch, and yaw angles are retrieved using the `GetEulerAngle` procedure in steps 9–11. Steps 12–15 set flags for oscillation detection (both initialized to false). In steps 16–25, the algorithm checks for oscillations by comparing the current roll and pitch values to their thresholds and previous values. If an oscillation is detected in either the roll or pitch, the respective flag is set to true. In steps 26–30, if an oscillation is detected, the alarm is triggered by setting `ActiveAlarm` to true; otherwise, it remains false. Finally, in steps 31–33, the previous roll and pitch values are updated to the current ones, and the process repeats. This ensures that the robot's stability is constantly monitored, with alarms triggered when necessary.

Algorithm 2 Stability Alarm Function with Oscillation Check

```

1: GETEULERANGLE Quaternions are transformed into Euler angles.
2: procedure GETEULERANGLE(OdomMsg)
3:   Quaternion  $\leftarrow$  EXTRACTQUATERNION(OdomMsg)
4:   Roll, Pitch, Yaw  $\leftarrow$  QUATERNIONTOEULER(Quaternion)
5:   return Roll, Pitch, Yaw
6: end procedure
7: ACTIVATEALARM triggers the vibration alarm if the value for the roll or pitch angle
  exceeds the threshold  $\beta$  and detects oscillations.
8: procedure ACTIVATEALARM(OdomMsg)
9:   Pitchthreshold  $\leftarrow$   $\beta$  degrees
10:  Rollthreshold  $\leftarrow$   $\beta$  degrees
11:  Roll, Pitch, Yaw  $\leftarrow$  GETEULERANGLE(OdomMsg)
12:  PreviousRoll  $\leftarrow$  Previous Roll value
13:  PreviousPitch  $\leftarrow$  Previous Pitch value
14:  OscillatingRoll  $\leftarrow$  False
15:  OscillatingPitch  $\leftarrow$  False
16:  if Roll > Rollthreshold and PreviousRoll < Rollthreshold then
17:    OscillatingRoll  $\leftarrow$  True
18:  else if Roll < Rollthreshold and PreviousRoll > Rollthreshold then
19:    OscillatingRoll  $\leftarrow$  True
20:  end if
21:  if Pitch > Pitchthreshold and PreviousPitch < Pitchthreshold then
22:    OscillatingPitch  $\leftarrow$  True
23:  else if Pitch < Pitchthreshold and PreviousPitch > Pitchthreshold then
24:    OscillatingPitch  $\leftarrow$  True
25:  end if
26:  if OscillatingRoll or OscillatingPitch then
27:    ActiveAlarm  $\leftarrow$  True
28:  else
29:    ActiveAlarm  $\leftarrow$  False
30:  end if
31:  PreviousRoll  $\leftarrow$  Roll
32:  PreviousPitch  $\leftarrow$  Pitch
33: end procedure

```

4.4. Design and Connection Between ROS and the Web Interface

In this phase, the web interface's aesthetic design and functional structure were developed and implemented. The layout and styling were created using standard web development tools, including HTML and CSS, to ensure a clean and user-friendly appearance. The Vue.js JavaScript framework was utilized to dynamically manage the interface elements and enable interaction between ROS and the user. Establishing robust communication between the native ROS engine running on the control computer and the web interface was a critical aspect of this phase. To achieve this, the *roslaunch* server was employed, serving as a middleware that facilitates communication between ROS-based systems and non-ROS environments, such as web browsers or mobile applications, via WebSocket or HTTP protocols. The *roslaunch* server simplifies integration by eliminating the need for a full ROS environment on the client side. For the connection, the default port 9090 of the *roslaunch* server was utilized, ensuring stable and efficient data exchange. After successfully establishing the communication bridge, the connection was configured by defining ROS subscribers and publishers to interact with the necessary topics. Each topic was mapped to corresponding elements within the web interface to display relevant data or receive user commands. This mapping ensures real-time synchronization between the robot and the user interface. To facilitate the development of the web-based application, the Robot Web Tools suite was integrated into the project. Robot Web Tools is an open

source collection of JavaScript libraries and resources specifically designed to simplify the development of web applications for robots. By leveraging this library in the JavaScript source code of the interface, connections to various ROS topics were efficiently established, enabling seamless interaction between the web interface and the ROS backend.

For example, a connection to the `/cmd_vel` topic, which publishes messages of type `geometry_msgs/Twist`, was implemented. This allowed real-time display of the robot's velocity components (x , y , and z) through intuitive indicators in the interface. Similarly, the `/map` topic, which publishes messages of type `/nav_msgs/OccupancyGrid`, was processed to create a visual reconstruction of the robot's environment. Using Robot Web Tools' `OccupancyGridClient` feature, the interface rendered the map data dynamically, providing a detailed and interactive visualization of the robot's surroundings. This integrated design ensures that the web interface not only functions as a visual tool but also enables real-time bidirectional communication, allowing operators to monitor and control the robot effectively. The approach balances functionality, efficiency, and user experience, making it a vital component of the teleoperation system. The graphical interface was designed with four sections, as shown in Figure 5, which include the video feed from the robot's camera; the map built from LiDAR sensor data (including real-time localization within the map); indicators of the robot's general status, such as position, linear and angular velocity, and alarms; and finally, a section featuring a digital joystick that displays the commands provided by the user in real time.

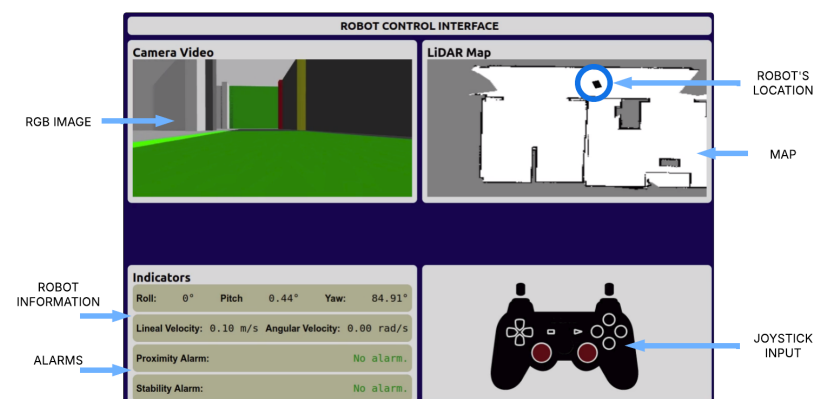


Figure 5. Web interface for teleoperators.

5. Haptic Joystick Design

5.1. Joystick Electronic and Control System Design

In Algorithm 3, the joystick's digital and analog inputs are handled as follows: the digital inputs are read using the `ReadPins` function, where the states of the buttons are captured. Specifically, the digital buttons are represented by $B_1, B_2, \dots, B_8$, and their states (pressed or released) are stored in the `Buttons` variable. These button states are used to trigger certain actions or transitions in the robot's behavior, such as changing modes or activating specific functions. For the analog inputs, the code reads the positions of two joysticks using the `AnalogRead` function. The analog x and y axes of the first joystick are captured as `JoystickX1` and `JoystickY1`, while the x and y axes of the second joystick are represented by `JoystickX2` and `JoystickY2`. These analog values correspond to continuous changes in the joystick's position, allowing smooth and variable control of the robot's movement. `JoystickX1` and `JoystickY1` represent the first joystick's horizontal and vertical movements, respectively, while `JoystickX2` and `JoystickY2` serve the same purpose for the second joystick. These digital and analog inputs are then formatted together in the `FormatData` procedure, which combines `Buttons`, `JoystickX1`, `JoystickY1`,

“JoystickX2”, and “JoystickY2” into a single data packet. This data packet is subsequently sent over Bluetooth through the “SendBluetooth” procedure, enabling the receiving system to interpret the robot’s controls and adjust its actions in real time. The system seamlessly integrates the binary button presses with continuous joystick movements, providing both discrete and smooth control over the robot’s functions.

Algorithm 3 Joystick operation

```

1: procedure READINPUTS
2:    $Buttons \leftarrow \text{ReadPins}([B_1, B_2, \dots, B_8])$  ▷ Read digital button states
3:    $JoystickX1, JoystickY1 \leftarrow \text{AnalogRead}(J1_x, J1_y)$ 
4:    $JoystickX2, JoystickY2 \leftarrow \text{AnalogRead}(J2_x, J2_y)$ 
5:   return  $Buttons, JoystickX1, JoystickY1,$ 
6:      $JoystickX2, JoystickY2$ 
7: end procedure
8: procedure FORMATDATA( $Buttons, JX1, JY1, JX2, JY2$ )
9:    $Data \leftarrow \text{Concatenate}(Buttons, JX1, JY1, JX2, JY2)$ 
10:  return  $Data$  ▷ Format data as a single packet
11: end procedure
12: procedure SENDBLUETOOTH( $Data$ )
13:    $\text{BluetoothWrite}(Data)$  ▷ Send data via Bluetooth
14: end procedure
15: procedure MAINLOOP
16:    $\text{InitBluetooth}("ESP32\_controller")$ 
17:   while True do
18:      $Buttons, JX1, JY1, JX2, JY2 \leftarrow \text{READINPUTS}$ 
19:      $Data \leftarrow \text{FORMATDATA}(Buttons, JX1, JY1, JX2, JY2)$ 
20:      $\text{SENDBLUETOOTH}(Data)$ 
21:     PROXIMITY ALARM FUNCTION ▷ Call Algorithm 1 here
22:     STABILITY ALARM FUNCTION ▷ Call Algorithm 2 here
23:      $\text{Delay}(10)$  ▷ Sampling rate adjustment
24:   end while
25: end procedure

```

Figure 6 illustrates the schematic operation and communication types for data acquisition and transmission. It is crucial to consider the specific parameters and features required for the controller. Sensor data are received over serial communication using an ESP32 microcontroller, while a Bluetooth connection is employed to send commands to the robot. These data are filtered and validated to activate the sensory feedback system. The primary objective of the joystick is to enable a human operator to teleoperate the robot. Additionally, the joystick provides sensory feedback, with alarms that increase proportionally based on events such as potential loss of stability or the need for collision avoidance. The ARD385 vibration motor is used to deliver this feedback, offering tactile alerts to the operator and enhancing the overall usability of the system. The robot will monitor vibration oscillations in roll (ϕ) and pitch (θ) angles. The ARD385 will be activated at low intensity and will progressively increase as the tilt angle grows. If the variations in ϕ (roll) or θ (pitch) exceed a threshold within a short period of time (Δt), the robot will be considered to be oscillating. In our case, Δt is set to 0.25 s, and a variation in either ϕ (roll) or θ (pitch) greater than 2 degrees is used as the threshold. The vibration intensity will increase proportionally to the magnitude of the oscillation. Sensor alarms will be processed to capture continuous changes in the robot’s behavior and relay this information to the joystick device to alert the teleoperator. This will allow the operator to smooth robot control via analog inputs and detect binary actions through digital inputs, which can trigger specific commands or states, such as movement or stopping actions. This combined approach ensures that the

robot responds dynamically to both its proximity to obstacles and its orientation, providing real-time feedback to the user and maintaining stability during operation.

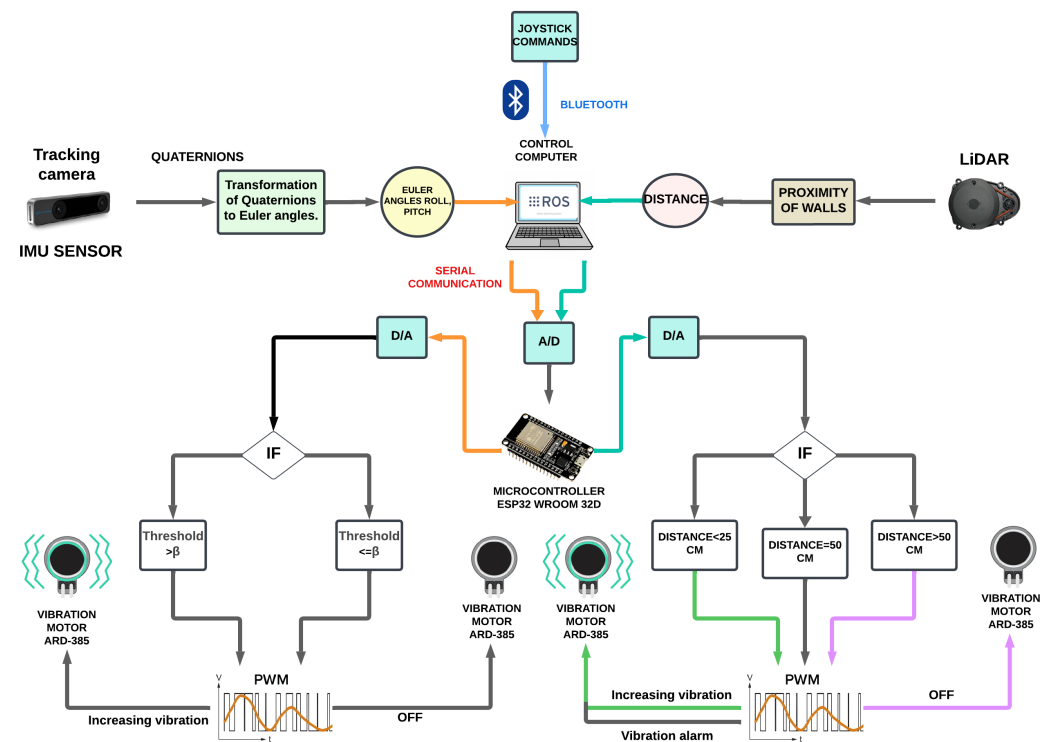


Figure 6. Vibration motor logic controller.

5.2. Joystick–Robot Vinculation

To connect the joystick to ROS, the system first ensured that Linux recognized the joystick. Then, two ROS nodes were launched: the Joy Node, which captures input from the joystick's buttons and axes and publishes it to the `/joy` topic, and the TeleopTwistJoy Node, which converts the joystick input into Twist velocity commands and publishes them to the `/cmd_vel` topic. The TeleopTwistJoy Node can be configured to map specific joystick axes to linear and angular velocity and adjust joystick sensitivity. To verify the mapping, the `/jstest` command was used to check the joystick's axes and button inputs. After confirming proper mapping, ROS Noetic or ROS Humble was used to link the joystick input to the robot's velocity control, with a TurtleBot3 or TurtleBot4 model simulated in Gazebo. The vertical axis of the left joystick controlled linear velocity, and the horizontal axis of the right joystick controlled angular velocity. Testing confirmed that joystick inputs were correctly published to `/joy` and converted into Twist messages on the `/cmd_vel` topic for robot control. Algorithm 4 details the operation of the joystick and graphical interface with ROS. During this stage, the user transmits data from the joystick to set the robot's velocity. Additionally, the system calls the web video server to stream real-time camera footage and uses rosbridge to facilitate communication between nodes and update data on the web server.

Algorithm 4 Joystick and Ghrapical Interface ROS Vinculation

```

1: procedure PROCESSJOYSTICKINPUT(JoystickMsg)
2:   Linear  $\leftarrow$  MapAxis(JoystickMsg.axes[1])
3:   Angular  $\leftarrow$  MapAxis(JoystickMsg.axes[0])
4:   return Linear, Angular
5: end procedure
6: procedure PUBLISHVELOCITY(Linear, Angular)
7:   TwistMsg.linear.x  $\leftarrow$  Linear
8:   TwistMsg.angular.z  $\leftarrow$  Angular
9:   Publish(TwistMsg)
10: end procedure
11: procedure JOYSTICKTELEOPNODE
12:   InitNode("teleopjoystick")
13:   Subscribe("/joy", Joy, Callback)
14:   procedure CALLBACK(JoystickMsg)
15:     Linear, Angular  $\leftarrow$  PROCESSJOYSTICKINPUT(JoystickMsg)
16:     PUBLISHVELOCITY(Linear, Angular)
17:   end procedure
18:   Spin()
19: end procedure
20: procedure INTERFACE VINCULATION
21:   InitNode("WebVideoServer")
22:   InitNode("RosbridgeServer")
23:   Launch("mainHMTLinter face")
24:   SubscribeTopics("/joy", "/map", "/cmd_vel", "/odom")
25: end procedure

```

6. Simulation and Experiments

In the following section, we present a comprehensive analysis of the experiments and the results obtained in both simulated and real environments within the context of our work. The behavior and performance of the system's graphical interface, the haptic feedback joystick, and the proposed alarms of our teleoperation system were analyzed. Figure 7 summarizes the implementation of the teleoperation system, including its design and testing process. For physical tests, a real TurtleBot3 was used in both proximity alarm and stability handling approaches. Our work conducted simulations in both ROS 1 (Noetic) and ROS 2 (Humble). The construction of the joystick was tested, as it is the primary element through which the operator interacts with the robot. It was verified that the joystick was recognized by the Linux operating system and that the buttons were properly calibrated, ensuring that pressing them triggered the expected actions as defined by the teleoperation system parameters.

Both ROS versions were used to map the joystick topic to the topic controlling the robot's speed. The web development interface was designed based on Section 4. A simulation was performed using a TurtleBot3 model in Gazebo Classic and a TurtleBot4 model in Gazebo Ignition. In this setup, the vertical movement axis of the left joystick controlled the robot's linear velocity, while the horizontal movement axis of the right joystick controlled the angular velocity. The proximity and destabilization alerts mentioned earlier were tested using both the web page and the joystick alarms to enable the teleoperator to respond effectively in both simulated and physical scenarios.

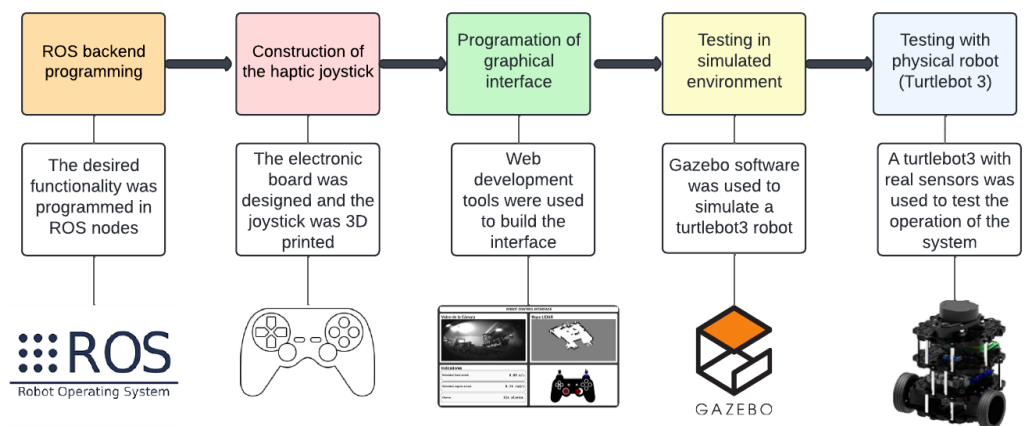


Figure 7. Implementation of the teleoperation system.

6.1. System Adaptation to Sudden Environmental Changes and Failures

The proposed system is designed to detect and respond to sudden environmental changes and system failures in real time, particularly those caused by terrain variations affecting stability and potential collision risks. Figure 8 illustrates the overall system during autonomous navigation. The robot navigates autonomously using its sensors, relying on mapping, localization, trajectory planning, and control algorithms to determine its speed. Simultaneously, the system processes data from onboard sensors, including RGB-D and tracking cameras, 2D LiDAR, and odometry sensors. It detects anomalies by analyzing variations in sensor readings, such as sudden changes in the LiDAR distance that indicate new obstacles or abnormal IMU data that suggest instability. When such conditions arise, the system triggers immediate visual alerts on the web interface. Additionally, a vibration-based warning system enhances operator awareness by increasing vibration intensity as the robot approaches an obstacle or loses stability.

The teleoperation system serves as a monitoring and backup system, assisting with navigation tasks when the primary autonomous navigation system fails or requires intervention to overcome unexpected obstacles. During teleoperation, the robot follows speed commands from the user instead of the autonomous navigation system. This ensures that the robot can still reach its goal with user assistance. The system supports external inputs and considers other events, making the architecture adaptable to future risks or unforeseen situations.

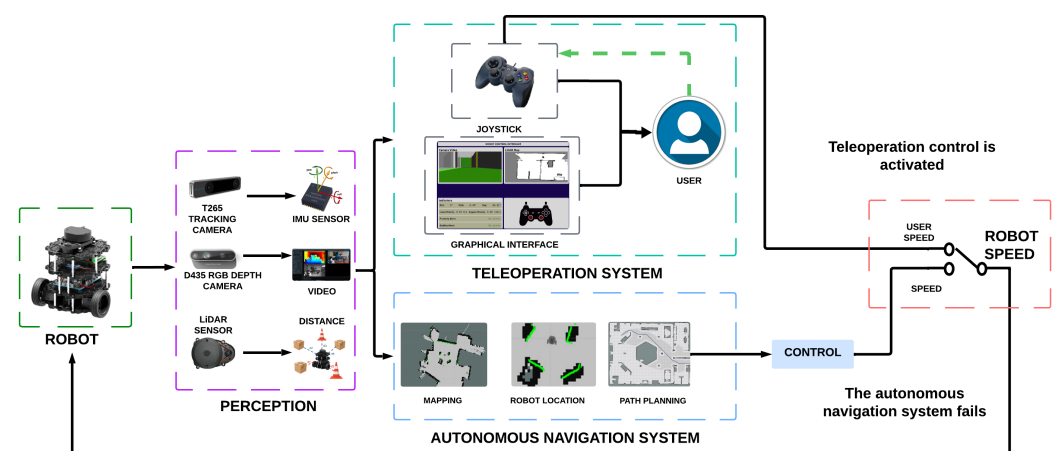


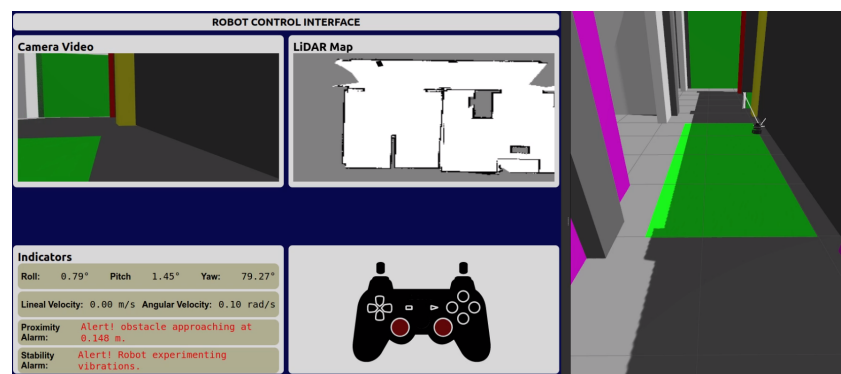
Figure 8. Emergency teleoperation system during autonomous navigation.

6.2. Simulated Environment

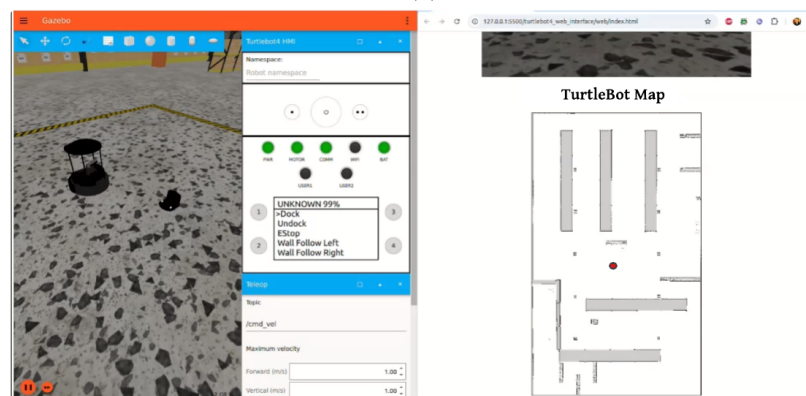
Figure 9a,b illustrate the simulation environments in ROS 1 and ROS 2 used for testing, focusing on two key scenarios:

(a) The robot navigates autonomously and receives an alert when approaching a wall. A simulated LiDAR sensor was utilized to measure distances to obstacles in the environment and validate the functionality of the proximity alarm. As described in previous sections, this alarm publishes alerts to the Lidar_Alarm topic when an obstacle is detected within the predefined threshold.

(b) The robot traverses over irregular terrain. During this task, the joystick provides haptic feedback through vibrations, and the web interface displays alerts to notify the user of stability variations.



(a)



(b)



(c)

Figure 9. Cont.

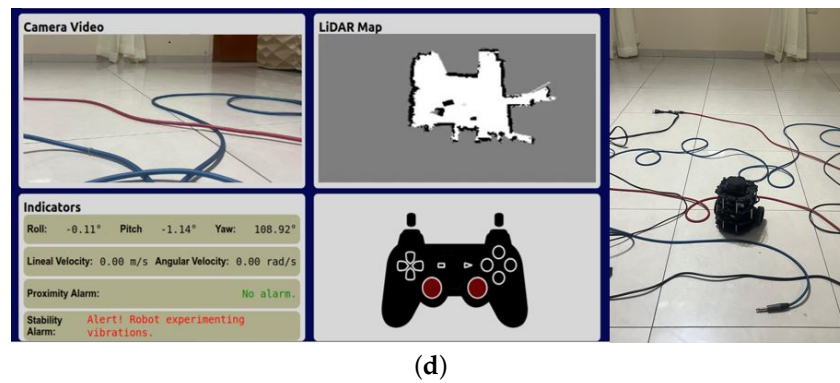


Figure 9. Detailed view of the object detection process using a radar sensor. Each step highlights a specific aspect of the detection and tracking pipeline. (a) Simulation test in ROS 1; (b) Simulation test in ROS 2; (c) Experimental test of collision detection in ROS 1; (d); Experimental stability test in ROS 1.

In both scenarios, the teleoperator can intervene to guide the robot along a safer path. The robot's movements were successfully controlled using the joystick by leveraging tools from the joy package. As shown in Figure 9a, the robot is approaching a wall while also descending a small irregularity in the floor, represented by the green surface. Figure 9b presents the TurtleBot 4 operating within a ROS 2 communication framework, displaying its position on a map to help the user identify its location within the environment. This simulation setup serves as a baseline for testing robotic applications and web-based communication, enabling real-time remote control and user interaction.

6.3. Experimental Results

The real-world environment test was conducted using a TurtleBot3 Burger running ROS 1, with a router that covered approximately 25 m. The robot was connected to a computer where the ROS environment was initialized, and the graphical interface was displayed. Figure 9c,d illustrates the results during the navigation task. In Figure 9c, the robot was placed in a room with a flat surface but several scattered obstacles. This scenario was used to test the robot's control and the functionality of the proximity alarm when approaching obstacles within a defined range. For this experiment, we placed four boxes very close to the robot to observe its behavior. As expected, the robot displayed on the interface the shortest distance to an obstacle. Since all four boxes were relatively close and within the alert range, the robot detected multiple points at distances of less than 15 cm in different directions. However, it was able to correctly measure and display the shortest distance on the screen, confirming that the proximity alarm was successfully developed.

In the second scenario Figure 9d, the robot was placed in a room with several thick cables distributed across the floor to create an uneven terrain, allowing us to test the functionality of the robot's stability alarm. As the robot moved over the cables, we were able to verify the correct operation of this feature. One particular aspect of this scenario was that, during the experiment, although the robot's camera could clearly see the cables placed on the floor, they did not appear on the map generated by the robot's LiDAR sensor. This was because the sensor was positioned on the upper part of the robot, while the cables were on the ground. Since this is a 2D LiDAR, it only measures distances to obstacles at the same height as the sensor. Therefore, implementing this alarm can be very useful for the robot operator, as it serves as a complement to the LiDAR, alerting the operator to obstacles that the sensor cannot detect. The teleoperation system and all its components are shown working together, including the joystick for controlling the robot, the graphical interface for monitoring and interaction, and the proximity and destabilization alarms functioning

in real time. This integration demonstrates the system's capability to operate effectively in real-world environments, adapting to various scenarios and challenges.

6.4. Latency Analysis in Simulation and Real-World Experiments

The latency of the teleoperation system in both simulation and real-world experiments using ROS 1 and ROS 2 was measured, focusing on LiDAR, camera, and joystick communication with the web interface over a local network. Table 1 summarizes the latency results, allowing for a comparison between simulation and real-world scenarios. In simulation, LiDAR latency in ROS 2 (0.0105 s) is lower than in ROS 1 (0.0123 s), indicating a slight improvement in sensor data transmission. However, camera latency is higher in ROS 2 (0.1581 s) compared to ROS 1 (0.0958 s), suggesting additional delays in processing camera data under simulation conditions. Joystick latency remains low in both cases, with ROS 1 at 0.0009 s and ROS 2 at 0.0080 s. In real-world experiments, LiDAR latency is again lower in ROS 2 (0.04243 s) compared to ROS 1 (0.04992 s), confirming improved efficiency in sensor data transmission. Joystick latency is also reduced in ROS 2 (0.0012 s) versus ROS 1 (0.0036 s), enhancing real-time teleoperation responsiveness. However, camera latency in ROS 2 (0.15199 s) is higher than in ROS 1 (0.0971 s), consistent with the simulation findings. These results demonstrate that ROS 2 offers lower latency for LiDAR and joystick inputs, making it advantageous for real-time teleoperation. However, the slightly higher camera latency in ROS 2 suggests potential optimizations for image data processing. Overall, the proposed system achieves real-time performance in both simulation and real-world experiments, ensuring efficient teleoperation and alarm response mechanisms for safer autonomous navigation.

Table 1. Latency of the teleoperation system in simulation and real experiments using ROS 1 and ROS 2.

	Version	LiDAR		Camera		Joystick	
		Time (s)	Latency	Time (s)	Latency	Time (s)	Latency
SIM	ROS 1 - Web	60.0324	0.0123	60.1821	0.0958	60	0.0009
	ROS 2 - Web	60.0238	0.0105	60.4245	0.1581	60	0.0080
REAL	ROS 1 - Web	60.3155	0.04992	60.1964	0.0971	60	0.0036
	ROS 2 - Web	60.1648	0.04243	60.2254	0.15199	60	0.0012

7. Conclusions

This project successfully demonstrated the teleoperation of a robot using a ROS-based software architecture integrated with a WebSocket communication protocol. This approach ensured real-time data transmission, enabling seamless execution of various system functions. The teleoperation interface was designed to efficiently incorporate sensor feedback from LiDAR, IMU, and depth cameras, providing operators with structured, real-time information. The integration of multiple sensors proved essential for enhancing situational awareness, enabling operators to make informed decisions based on comprehensive environmental data. Additionally, the implementation of a haptic feedback system using joystick vibrations further optimized operator response times. By delivering immediate and intuitive alerts to critical events, this system reduced reliance on visual or auditory cues, significantly improving operational safety.

Overall, this work demonstrates the potential for innovation in teleoperation, contributing to improved performance, safety, and adaptability in increasingly complex operational settings while providing a foundation for future applications. Such systems are

essential in environments where autonomous navigation algorithms may fail due to limited environmental perception or require human intervention during task execution.

Author Contributions: Formal analysis, N.P. and H.M.; Investigation, N.P., G.M. and D.M.; Methodology, N.P. and H.M.; Software, N.P., G.M. and D.M.; Supervision, H.M.; Validation, M.S.A.-A., E.A. and H.M.; Visualization, M.S.A.-A., E.A. and H.M.; Writing—original draft, N.P., G.M. and D.M.; Writing—review and editing, E.A. and H.M. All authors have read and agreed to the published version of the manuscript.

Funding: This work was supported by the Technology Innovation Program (2410002650, Development of a mobile manipulator robot system based on multi-collaboration for manipulating and assembling 500 kg large and heavy parts) funded By the Ministry of Trade Industry & Energy (MOTIE, Korea).

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: The original contributions presented in this study are included in the article. Further inquiries can be directed to the corresponding author.

Conflicts of Interest: The authors declare no conflicts of interest.

References

1. Gul, F.; Mir, I.; Mir, S. Efficient environment exploration for multi agents: A novel framework. In Proceedings of the AIAA SCITECH 2023 Forum, Online, 23–27 January 2023; p. 1088.
2. Zhao, T.; Li, P.; Yuan, Y.; Zhang, L.; Zhao, Y. Trajectory Re-Planning and Tracking Control for a Tractor–Trailer Mobile Robot Subject to Multiple Constraints. *Actuators* **2024**, *13*, 109. [\[CrossRef\]](#)
3. Jeon, J.; Jung, H.r.; Luong, T.; Moon, H. Task-Motion Planning System for Socially Viable Service Robots Based on Object Manipulation. *Biomimetics* **2024**, *9*, 436. [\[CrossRef\]](#) [\[PubMed\]](#)
4. Fang, B.; Mei, G.; Yuan, X.; Wang, L.; Wang, Z.; Wang, J. Visual SLAM for robot navigation in healthcare facility. *Pattern Recognit.* **2021**, *113*, 107822. [\[PubMed\]](#)
5. Kim, P.; Chen, J.; Kim, J.; Cho, Y.K. SLAM-driven intelligent autonomous mobile robot navigation for construction applications. In Proceedings of the Advanced Computing Strategies for Engineering: 25th EG-ICE International Workshop 2018, Lausanne, Switzerland, 10–13 June 2018; Proceedings, Part I 25; Springer: Cham, Switzerland, 2018; pp. 254–269.
6. Pico, N.; Soriano, D.; Auh, E.; Velasquez, W.; Shin, J.; Moon, H. Accurate Stair Measurement Method for Autonomous Robot Navigation using RGB-D Camera. In Proceedings of the 2024 24th International Conference on Control, Automation and Systems (ICCAS), Jeju, Republic of Korea, 29 October–1 November 2024; IEEE: Piscataway, NJ, USA, 2024; pp. 1567–1572.
7. Pico, N.; Montero, E.; Vanegas, M.; Erazo Ayon, J.M.; Auh, E.; Shin, J.; Doh, M.; Park, S.H.; Moon, H. Integrating Radar-Based Obstacle Detection with Deep Reinforcement Learning for Robust Autonomous Navigation. *Appl. Sci.* **2024**, *15*, 295. [\[CrossRef\]](#)
8. Rudenko, A.; Palmieri, L.; Herman, M.; Kitani, K.M.; Gavrila, D.M.; Arras, K.O. Human motion trajectory prediction: A survey. *Int. J. Robot. Res.* **2020**, *39*, 895–935. [\[CrossRef\]](#)
9. Auh, E.; Jung, H.; Pico, N.; Choi, H.; Koo, J.; Moon, H. Model Predictive Contouring Control for Four-Wheel Independent Steering and Driving Mobile Robots. In Proceedings of the 2024 IEEE International Conference on Real-time Computing and Robotics (RCAR), Alesund, Norway, 24–28 June 2024; pp. 259–264. [\[CrossRef\]](#)
10. Canaza Ccari, L.F.; Adrian Ali, R.; Valdeiglesias Flores, E.; Medina Chilo, N.O.; Sulla Espinoza, E.; Silva Vidal, Y.; Pari, L. JVC-02 Teleoperated Robot: Design, Implementation, and Validation for Assistance in Real Explosive Ordnance Disposal Missions. *Actuators* **2024**, *13*, 254. [\[CrossRef\]](#)
11. Li, W.; Guo, J.; Ding, L.; Wang, J.; Gao, H.; Deng, Z. Teleoperation of wheeled mobile robot with dynamic longitudinal slippage. *IEEE Trans. Control Syst. Technol.* **2022**, *31*, 99–113. [\[CrossRef\]](#)
12. Grabowski, A.; Jankowski, J.; Wodzyński, M. Teleoperated mobile robot with two arms: The influence of a human-machine interface, VR training and operator age. *Int. J. Hum.-Comput. Stud.* **2021**, *156*, 102707.
13. Fine, T.; Zaidner, G.; Shapiro, A. Grasping assisting algorithm in tele-operated robotic gripper. *Appl. Sci.* **2021**, *11*, 2640. [\[CrossRef\]](#)
14. Thrun, S. Probabilistic algorithms in robotics. *Ai Mag.* **2000**, *21*, 93.
15. Cadena, C.; Carlone, L.; Carrillo, H.; Latif, Y.; Scaramuzza, D.; Neira, J.; Reid, I.; Leonard, J.J. Past, present, and future of simultaneous localization and mapping: Toward the robust-perception age. *IEEE Trans. Robot.* **2016**, *32*, 1309–1332.

16. Ebadi, K.; Bernreiter, L.; Biggie, H.; Catt, G.; Chang, Y.; Chatterjee, A.; Denniston, C.E.; Deschênes, S.P.; Harlow, K.; Khattak, S.; et al. Present and future of slam in extreme environments: The darpa sub challenge. *IEEE Trans. Robot.* **2023**, *40*, 936–959.
17. Wong, C.; Yang, E.; Yan, X.T.; Gu, D. An overview of robotics and autonomous systems for harsh environments. In Proceedings of the 2017 23rd International Conference on Automation and Computing (ICAC), Huddersfield, UK, 7–8 September 2017; IEEE: Piscataway, NJ, USA, 2017; pp. 1–6.
18. Sun, Y.; Ren, D.; Lian, S.; Fu, S.; Teng, X.; Fan, M. Robust path planner for autonomous vehicles on roads with large curvature. *IEEE Robot. Autom. Lett.* **2022**, *7*, 2503–2510.
19. Alakshendra, V.; Chiddarwar, S.S. Adaptive robust control of Mecanum-wheeled mobile robot with uncertainties. *Nonlinear Dyn.* **2017**, *87*, 2147–2169. [[CrossRef](#)]
20. Cui, M.; Liu, W.; Liu, H.; Jiang, H.; Wang, Z. Extended state observer-based adaptive sliding mode control of differential-driving mobile robot with uncertainties. *Nonlinear Dyn.* **2016**, *83*, 667–683.
21. Zhu, K.; Zhang, T. Deep reinforcement learning based mobile robot navigation: A review. *Tsinghua Sci. Technol.* **2021**, *26*, 674–691.
22. Montero, E.; Ghergherehchi, M.; Song, H.S. Memory-driven deep-reinforcement learning for autonomous robot navigation in partially observable environments. *Eng. Sci. Technol. Int. J.* **2025**, *62*, 101942.
23. Pico, N.; Montero, E.; Amirbek, A.; Auh, E.; Jeon, J.; Alvarez-Alvarado, M.S.; Jamil, B.; Algabri, R.; Moon, H. Human and environmental feature-driven neural network for path-constrained robot navigation using deep reinforcement learning. *Eng. Sci. Technol. Int. J.* **2025**, *64*, 101993.
24. Galarza, B.R.; Ayala, P.; Manzano, S.; Garcia, M.V. Virtual reality teleoperation system for mobile robot manipulation. *Robotics* **2023**, *12*, 163. [[CrossRef](#)]
25. Li, S.; Jiang, J.; Ruppel, P.; Liang, H.; Ma, X.; Hendrich, N.; Sun, F.; Zhang, J. A mobile robot hand-arm teleoperation system by vision and imu. In Proceedings of the 2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), Las Vegas, NV, USA, 25–29 October 2020; IEEE: Piscataway, NJ, USA, 2020; pp. 10900–10906.
26. Penizzotto, F.; Slawinski, E.; Mut, V. Analysis and experimentation of a mobile robot teleoperation system over internet. *IEEE Lat. Am. Trans.* **2014**, *12*, 1191–1198.
27. Szymańska, E.; Petrović, L.; Marković, I.; Petrović, I. Mobile robot teleoperation via Android mobile device with UDP communication. In Proceedings of the 2021 44th International Convention on Information, Communication and Electronic Technology (MIPRO), Opatija, Croatia, 27 September–1 October 2021; IEEE: Piscataway, NJ, USA, 2021; pp. 1143–1148.
28. Adamides, G.; Christou, G.; Katsanos, C.; Xenos, M.; Hadzilacos, T. Usability guidelines for the design of robot teleoperation: A taxonomy. *IEEE Trans. Hum.-Mach. Syst.* **2014**, *45*, 256–262. [[CrossRef](#)]
29. Wijayasinghe, I.B.; Saadatzi, M.N.; Peetha, S.; Popa, D.O.; Cremer, S. Adaptive Interface for Robot Teleoperation using a Genetic Algorithm. In Proceedings of the 2018 IEEE 14th International Conference on Automation Science and Engineering (CASE), Munich, Germany, 20–24 August 2018; pp. 50–56. [[CrossRef](#)]

Disclaimer/Publisher’s Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.