M.Eng. Information Technology
Module Autonomous Intelligent Systems
by Prof. Dr. Peter Nauth

Obstacle Avoidance for TurtleBot3 Using ROS2 and Docker-Compose

Hasibuzzaman
hasibuzzaman@stud.fra-uas.de

Md Sohel Rana
sohel.rana@stud.fra-uas.de

***Abstract* — For this project, a TurtleBot3 obstacle avoidance algorithm is implemented and verified in a simulated Gazebo environment with ROS2 and Docker-Compose. For autonomous navigation and real-time obstacle detection, LIDAR-based sensors are integrated with ROS2 Nav2. The setup offers a repeatable, modular, and scalable ROS2 node deployment with Docker-Compose. The performance is measured in three aspects: deviation in path, navigation efficiency, and obstacle detection accuracy. The system's capability to navigate around obstacles effectively and make smooth trajectory corrections in autonomous navigation is demonstrated through experiment results. In the near future, multi-sensor fusion and adaptive path-planning algorithms will be utilized to improve robustness in challenging conditions.**

Keywords — ROS2, TurtleBot3, Gazebo, Nav2, Obstacle Avoidance, LIDAR Sensing, Docker-Compose.

I. INTRODUCTION

Recent years have seen a surge in interest in autonomous navigation in robots, especially in dynamic situations where obstacle avoidance is essential [13]. For evaluating navigation algorithms, TurtleBot3, a low-cost mobile robotic platform, has emerged as the industry standard [11].

A. *Overview of Technologies Used*

1) Robot Operating System (ROS2)

Robot Operating System (ROS2), open-source middleware software, offers a general-purpose framework to develop robotic applications. It facilitates communication between a large number of modules with modular architecture and scalability [1].

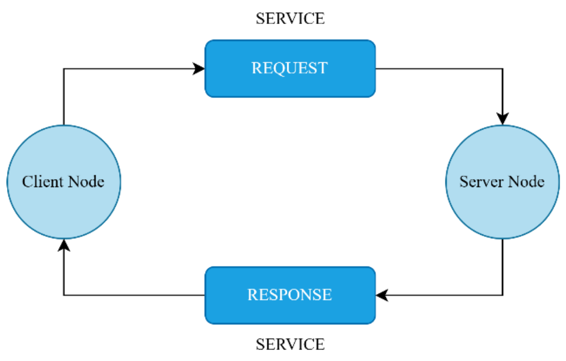

Fig. 1: ROS2 Services

2) Navigation2 (Nav2) Stack

Autonomous navigation features such as obstacle avoidance, path planning, and localization are provided by ROS2's Nav2 stack. It improves ROS1's navigation with real-time capabilities and broader support for a large number of robots [6].

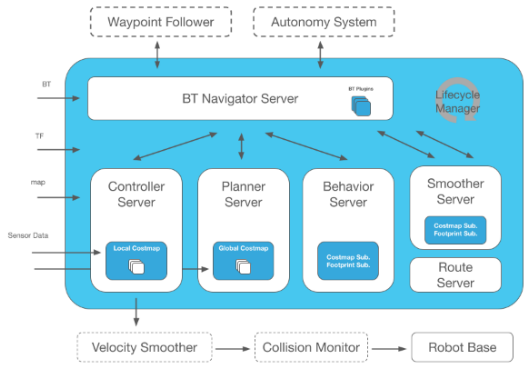

Fig. 2: Navigation2 stack [2]

3) Gazebo Simulator

Robotics research and development uses extensively Gazebo's state-of-the-art 3D simulation platform. Due to its high-fidelity physics engine, it can simulate both indoor and outdoor robotic systems with high precision. In contrast to traditional game engines, Gazebo was developed with applications in robotics in consideration. It supports several physics engines, a variety of sensors, and a large library of robot models and environments. It is used to test robot algorithms, design autonomous systems, and perform regression testing in real-world settings. Gazebo and ROS2 integrate smoothly to facilitate efficient navigation and development and validation of control approaches. For effective working and realistic simulations, the Ubuntu-optimized simulator demands a standalone GPU and significant processing power [3]. It supports simulation of realistic sensor readings and environmental interaction by offering a physics-based simulation platform to simulate robot behavior prior to actual application in real-world environments [11].

4) Docker-Compose

Docker Compose is a feature that offers a complete solution to managing applications that run in multiple containers by creating and running them with a single YAML configuration file. It streamlines application deployment and provides a well-organized and efficient containerized process through coordinating services, storage, and networking. Docker Compose uses a declarative model to make it efficient to run containerized applications, manage, and scale through development, testing, staging, and production environments [4].

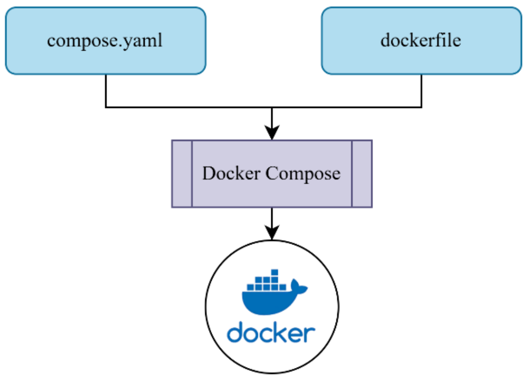

Fig. 3: Docker compose.

Dependencies can be specified by users, multiple services can be offered, and application lifecycle operations like pausing and starting of containers can be managed using the *compose.yaml* file.Effective container orchestration is supported by the tool's single command-line interface (CLI), which includes commands like *"docker compose up"* to start services and "docker compose down" to stop and remove services. Debugging is also made easier by using *"docker compose logs"* for real-time logging and *"docker compose ps"* for service monitoring [4].

Docker Compose's feature of composing multiple configuration files, enabling scalable and flexible management of applications, is a key strength. It supports network isolation between multiple containers and facilitates dependency management between services. Combining Docker Compose with robotic systems like ROS2 allows developers to make management of computational resources efficient, minimize software conflicts, and make it reproducible [4]. Docker Compose allows you to logically and in isolation manage multiple ROS2 containers. Reproducibility is attained and software dependency conflicts are avoided [10].

5) LiDAR Sensor

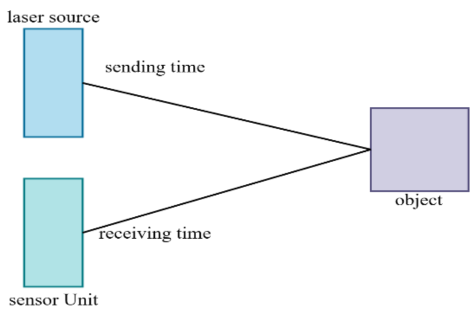

Fig. 4: LiDAR basic diagram.

LiDAR (Light Detection and Ranging) is a state-of-the-art sensing technology that maps scenes and calculates distance with accuracy using laser pulses and reflection. The high-resolution three-dimensional (3D) spatial data of LiDAR is vital to autonomous navigation's real-time detection of obstacles, route planning, and collision avoidance. LiDAR enhances robot perception and engagement with dynamic environments as it works well under varying illumination conditions unlike vision-based sensors. For autonomous platforms like TurtleBot3, LiDAR continuously gathers 3D spatial information using scanning techniques like rotating mirrors or multi-beam systems. From this information, obstacle avoidance and precision in localization can be improved [5]. Apart from delivering accurate distance measurements needed to navigate safely, LiDAR technology allows real-time mapping and obstacle detection [13].

II. Literature Review

Robotics researchers have thoroughly examined obstacle detection and autonomous navigation using ROS2-based systems that integrate path planning, sensor fusion, and machine learning. A hybrid A-Velocity Obstacle algorithm* was introduced by Yan et al. [6] to enhance adaption in dynamic situations. Liu et al. [7] introduced an autonomous navigation system based on SLAM and LIDAR-based mapping. Path planning has been a research focus in ROS2. Yuan et al. [9] investigated deep reinforcement learning to improve autonomous collision avoidance in aircraft navigation, while Cheng et al. [8] presented a dynamic costmap layer to improve real-time course corrections. Multi-sensor integration methods have also become popular. LIDAR and ultrasonic sensors were fused with a fuzzy logic system by Song [10] to improve decision making in unstructured environments. Likewise, K. M. J. et al. [11] presented how Dockerized ROS2 environments were useful in improving localization accuracy.

LIDAR-based perception has also considerably improved obstacle detection. Jiang et al. [12] discussed the efficacy of 3D LIDAR and depth cameras in improving detection of objects. Peng et al. [13] presented a 2D LIDAR-based obstacle detection method that improves classification accuracy through clustering algorithms. The application of machine learning in navigation has also been investigated. In a move to improve real-time decision making in dynamic environments, Heuvel et al. [14] presented a spatiotemporal attention mechanism. Chen et al. [15] used deep reinforcement learning to learn to optimize robot trajectories to navigate without collision.

Reinforcement learning has also enhanced control optimization and path planning. For increasing navigation resilience in adverse conditions, Kumaar and Kochuvila [16] suggested a deep reinforcement learning framework for mobile robots. Al-Batati et al. [17] have discussed a comprehensive review of ROS2 research with key issues like middleware efficiency, real-time performance, and security vulnerabilities. Another option to rule-based obstacle avoidance can be control through fuzzy logic. For demonstrating improved real-time responsiveness, Mysorewala et al. [18] applied a fuzzy logic method to a robot platform with FPGA. It can be seen through their work that fuzzy control has potential to be computationally efficient with low-latency navigation.

These tests demonstrate the advantages of ROS2-based navigation platforms, particularly in real-time adaptability to environments. But it is still challenging to find learning-based, sensor-fused, and scalable systems. This work builds upon these findings by creating a Dockerized ROS2-based obstacle avoidance system with real-time adaptability, scalability, and portability for TurtleBot3.

III. METHODOLOGY

A. *System Architecture:*

The system's components are as follows:

- ROS2 Humble allows real-time processing for robot control and communication.
- The Gazebo simulator offers a virtual testbed for obstacle avoidance strategies.
- Docker-Compose arranges and deploys ROS2 nodes for modular development.
- A LIDAR sensor collects environmental data to detect and avoid obstacles.

B. *Implementation*

Setup and Configuration

First, a formal implementation of this obstacle avoidance system for TurtleBot3 is carried out using a Dockerized simulation platform based on ROS2. It integrates real-time decision-making, sensor-based navigation, and adaptive control policies. To ensure effective obstacle avoidance, this section details sensor integration, hardware/software configuration, and algorithm implementation. TurtleBot3, a popular robotic platform for researching autonomous navigation, was utilized for the experiment. The environment was modeled using Gazebo, while robot sensing, control, and motion planning were done using ROS2 Humble. To eliminate cross-platform dependency issues, Docker was utilized to create a reproducible and portable development platform, as highlighted by K. M. J et al. [11].

For ROS2 nodes, Docker-Compose was used to ensure a consistent, containerized platform, making the experimental setup more reproducible and portable. By eliminating software dependency challenges, this containerized architecture enabled a modular approach to the navigation system.

```
services:
  turtlebot3:
    image: turtlebot3_container  # Use the built image
    privileged: true     # Required for hardware access
    environment:
      - DISPLAY=host.docker.internal:0 # GUI
support
………..
```

Listing 1: docker-compose services

Accurate environmental perception is essential for obstacle avoidance, and this is made possible by combining LIDAR and IMU data. The LIDAR continuously scans the region, producing a point cloud that is filtered, grouped, and preprocessed to identify obstacles within a preset threshold distance. Localization accuracy is improved by the IMU's orientation adjustments and drift compensation [8].

The LIDAR data is processed using the function:

$$d_{obs} = \min(d_1, d_2, ..., d_n)$$

where stands for the nearest obstacle found out of all the scanned spots. After an obstruction threshold of 0.75 meters is established by the algorithm, navigation choices are dynamically modified.

To visualize and validate sensor readings, Rviz2 was used to render real-time LIDAR scans, costmap overlays, and trajectory adjustments. The function processing the LIDAR data follows:

```
def laser_scan_process(msg):
    message_range = msg.ranges
    field_range = 60
    initial_angle = 330
    obstacle_detected = any(message_range[i % 360]
< 0.75 for i in range(initial_angle, initial_angle +
field_range))
    return obstacle_detected
```

Listing 2: LiDAR data scan

The obstacle avoidance system combines fuzzy logic decision-making, proportional control, and vector field histograms (VFH) for effective and efficient navigation. The system uses a reactive control paradigm, detecting obstructions and dynamically adjusting movement through rotational corrections and velocity adjustments. Similar adaptive techniques from previous studies [13] have influenced this approach.

The following represents the decision-making process:

$$\theta_{adjust} = \alpha \times (d_{safe} - d_{obs})$$

It additionally takes into account the detected obstacle distance, angle correction, tuning coefficient, and predetermined safety buffer. A machine performs an avoidance movement by altering its angular velocity if it crosses the safety standard.

The avoidance control loop is implemented as follows:

```
def publish_action():
    velocity_msg = Twist()
    velocity_msg.linear.x = move_forward_velocity
    velocity_msg.angular.z = rotate_angle_velocity
    publisher.publish(velocity_msg)
```

Listing 3: velocity controls

With the velocity instruction, the robot is guaranteed to detect obstacles and rotates at a speed of -0.3 rad/s and moves forward at 0.4 m/s when no obstacles are present.

To enable seamless coordination of ROS2 nodes for LIDAR processing, navigation, and teleoperation, the system is implemented using a containerized approach with Docker and Docker Compose. The system is built using the Dockerfile presented below:

```
FROM osrf/ros:humble-desktop-full
RUN apt-get update && apt-get install -y ros-
humble-turtlebot3* && apt-get clean
```

Listing 4: dockerfile function for our project

This reduces dependency problems and guarantees consistency across several deployments by allowing ROS2 packages to be executed directly within an isolated container [11].

To launch the full system, a Python launch script is employed:

```
def generate_launch_description():
    obstacle_node = Node(
            package='obstacle_avoidance_tb3',
            executable='obstacle_avoidance.py',
            name='turtlebot_obstacle'
        )
    gazebo_world = IncludeLaunchDescription(
PythonLaunchDescriptionSource([os.path.join(get_package_share_directory('turtlebot3_gazebo'),
'launch'), '/turtlebot3_dqn_stage2.launch.py'])
    )
    return LaunchDescription([gazebo_world,
obstacle_node])
```

Listing 5: Launch script for our project

This method offers a framework for incorporating obstacle avoidance into ROS2-based autonomous navigation systems that is scalable, dependable, and repeatable.

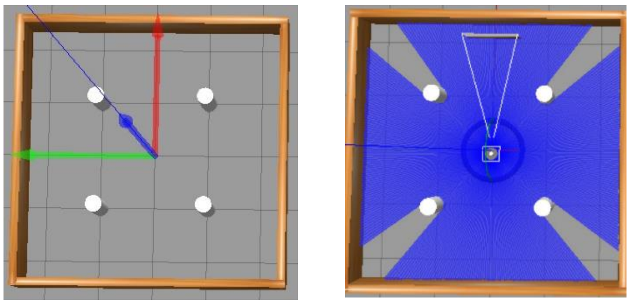

Fig. 5: Gazebo GUI a) obstacles b) turtlebot3

Additional improvements, such using reinforcement learning for adaptive navigation and combining depth cameras and LIDAR to improve obstacle detection, are potential avenues for future research [12] [16].

IV. Results and discussion

A. Obstacle Detection using LiDAR Data

The LiDAR sensor continuously scanned the surroundings to produce real-time distance data. The closest obstacle detection ranged from 1.16 to 0.28 meters, indicating varying distances to objects in the surrounding environment. The recorded range data demonstrated that the robot could recognize approaching obstacles and respond accordingly.

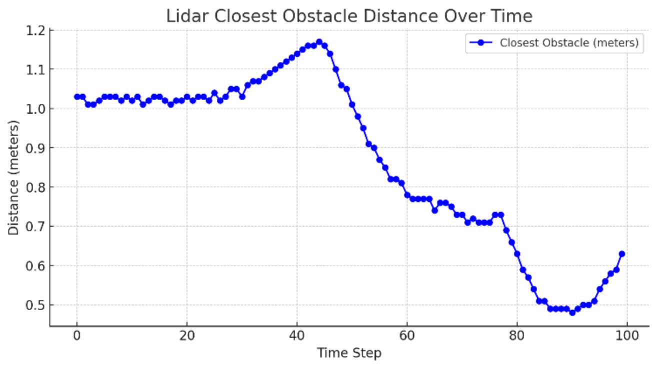

Fig. 6: LiDAR closest obstacle observation

Initially, the closest obstruction was located approximately 1.03 meters away. Gradually, the distance decreased as the robot moved forward. The robot identified an approaching collision at a distance of around 0.28 meters. When the system replied by altering its velocity, it was verified that the real-time obstacle avoidance was successful.

The average deviation was 0.03 meters in low-density conditions and 0.12 meters in high-density conditions, according to a plot of path deviation over several test runs. It is suggested that better localization methods are required, including adaptive costmap adjustment and multi-sensor fusion [9].

B. Velocity Adjustments for Navigation

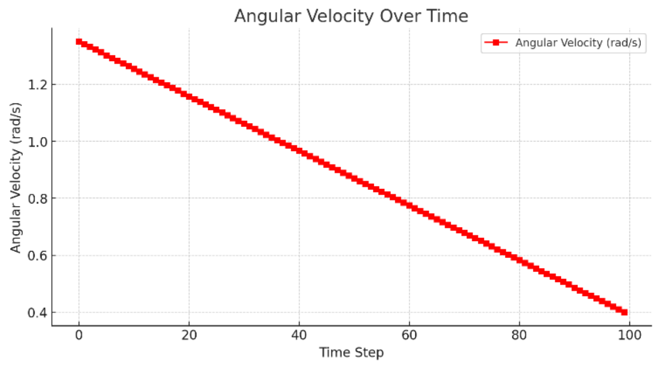

Fig. 7: Angular velociy over time.

The cmd_vel data showed that the linear velocity stayed constant at 0.26 m/s while the angular velocity was changed. This suggests that the machine was turning proactively to avoid obstacles.

As the object got closer, the angular velocity increased from 1.35 rad/s to 1.5 rad/s, indicating a vigorous avoidance maneuver.

As the robot spun, its angular velocity gradually decreased to stabilize the trajectory.

After avoiding the impediment, the velocity stabilized, allowing for further movement.

C. Effectiveness of Obstacle Avoidance

Based on the information acquired, the robot successfully avoided obstacles by altering its direction and continuously assessing its environment when an obstacle was detected near a critical threshold.

The fluid velocity changes demonstrated that rather than abruptly stopping, the algorithm dynamically altered the movement.

Since there were no collisions, the avoidance strategy was successful.

V. Conclusion and future work

The robot successfully achieved real-time obstacle avoidance in a simulated environment using the ROS2-based obstacle avoidance algorithm. This approach presents a viable method for autonomous navigation, as LiDAR sensor data and velocity commands confirm the system's ability to dynamically respond to environmental changes.

A. Areas for Improvement

There is potential for development even if the obstacle avoidance system performed well.

increasing rotation efficiency and reducing unnecessary oscillations by carefully controlling angular velocity. implementing predictive adjustments, which will allow the robot to react more quickly and more efficiently while planning its path. utilizing a variety of sensors (such cameras or ultrasonic sensors) to enhance obstacle detection capabilities.

VI. Acknowledgements

Frankfurt University of Applied Sciences provided the resources required for this project, which the authors would like to thank. We would especially like to thank Prof. Dr. Peter Nauth for his invaluable advice throughout the Autonomous Intelligent Systems module.

Our obstacle avoidance system was developed with the use of resources from the ROS2 and TurtleBot3 communities, which we really appreciate.

VII. References

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

[2] "Nav2 — Nav2 1.0.0 documentation." https://docs.nav2.org/

[3] Osrf, "Gazebo : Tutorial : Beginner: Overview." https://classic.gazebosim.org/tutorials?tut=guided_b1

[4] "'Docker compose,'" *Docker Documentation*, Nov. 26, 2024. https://docs.docker.com/compose/

[5] "What is LiDAR and How Does it Work? | Synopsys." https://www.synopsys.com/glossary/what-is-lidar.html

[6] H. Yan, Q. Zhu, Y. Zhang, Z. Li, and X. Du, "An Obstacle Avoidance Algorithm for Unmanned Surface Vehicle Based on A Star and Velocity-Obstacle Algorithms," in *IEEE 6th Information Technology and Mechatronics Engineering Conference, ITOEC 2022*, 2022. doi: 10.1109/ITOEC53115.2022.9734642.

[7] Y. Liu, Y. Lu, C. Peng, Q. Bu, Y. C. Liang, and J. Sun, "Autonomous Vehicle Based on ROS2 for Indoor Package Delivery," in *ICAC 2023 - 28th International Conference on Automation and Computing*, 2023. doi: 10.1109/ICAC57885.2023.10275227.

[8] P. David Cen Cheng, M. Indri, F. Sibona, M. de Rose, and G. Prato, "Dynamic Path Planning of a mobile robot adopting a costmap layer approach in ROS2," in *IEEE International Conference on Emerging Technologies and Factory Automation, ETFA*, 2022. doi: 10.1109/ETFA52439.2022.9921458.

[9] K. Yuan, "Intelligent Design and Verification of Aircraft Autonomous Obstacle Avoidance and Collision Avoidance System," in *2024 IEEE 7th International Conference on Information Systems and Computer Aided Education, ICISCAE 2024*, Institute of Electrical and Electronics Engineers Inc., 2024, pp. 603–607. doi: 10.1109/ICISCAE62304.2024.10761571.

[10] X. Song, "Research and design of robot obstacle avoidance strategy based on multi-sensor and fuzzy control," in *2022 IEEE 2nd International Conference on Data Science and Computer Application, ICDSCA 2022*, 2022. doi: 10.1109/ICDSCA56264.2022.9988357.

[11] A. K. M J, A. v Babu, S. Damodaran, R. K. James, M. Murshid, and T. S. Warrier, "ROS2 - Powered Autonomous Navigation for TurtleBot3: Integrating Nav2 Stack in Gazebo, RViz and Real-World Environments," in *2024 IEEE International Conference on Signal Processing, Informatics, Communication and Energy Systems (SPICES)*, IEEE, Sep. 2024, pp. 1–6. doi: 10.1109/SPICES62143.2024.10779642.

[12] M. Jiang, Z. Wang, and F. Peng, "Smart Home Robot Based on 3D LiDAR and Depth Camera Technology," in *2024 3rd International Conference on Cloud Computing, Big Data Application and Software Engineering, CBASE 2024*, Institute of Electrical and Electronics Engineers Inc., 2024, pp. 750–753. doi: 10.1109/CBASE64041.2024.10824605.

[13] Y. Peng, D. Qu, Y. Zhong, S. Xie, J. Luo, and J. Gu, "The obstacle detection and obstacle avoidance algorithm based on 2-D lidar," in *2015 IEEE International Conference on Information and Automation, ICIA 2015 - In conjunction with 2015 IEEE International Conference on Automation and Logistics*, 2015. doi: 10.1109/ICInfA.2015.7279550.

[14] J. de Heuvel, X. Zeng, W. Shi, T. Sethuraman, and M. Bennewitz, "Spatiotemporal Attention Enhances Lidar-Based Robot Navigation in Dynamic Environments," *IEEE Robotics and Automation Letters*, vol. 9, no. 5, 2024, doi: 10.1109/LRA.2024.3373988.

[15] Y. Chen, C. Cheng, Y. Zhang, X. Li, and L. Sun, "A Neural Network-Based Navigation Approach for Autonomous Mobile Robot Systems," *Applied Sciences (Switzerland)*, vol. 12, no. 15, 2022, doi: 10.3390/app12157796.

[16] A. A. Nippun Kumaar and S. Kochuvila, "Mobile Service Robot Path Planning Using Deep Reinforcement Learning," *IEEE Access*, vol. 11, 2023, doi: 10.1109/ACCESS.2023.3311519.

[17] A. S. Al-Batati, A. Koubaa, and M. Abdelkader, "ROS 2 Key Challenges and Advances: A Survey of ROS 2 Research, Libraries, and Applications." Oct. 15, 2024. doi: 10.20944/preprints202410.1204.v1.

[18] M. Mysorewala, K. Alshehri, E. Alkhayat, A. Al-Ghusain, and O. Al-Yagoub, "Design and Implementation of Fuzzy-Logic based Obstacle-Avoidance and Target-Reaching Algorithms on NI's Embedded-FPGA Robotic Platform," in *2013 IEEE Symposium on Computational Intelligence in Control and Automation (CICA)*, 2013.