An E2ED-based Approach to Custom Robot Navigation and Localization

Andrés Moreno, Daniel Páez, Fredy Martínez
Universidad Distrital
Francisco José de Caldas
Bogotá D.C., Colombia

***Abstract*—Simultaneous mapping and localization or SLAM is a basic strategy used with robots and autonomous vehicles to identify unknown environments. It is of great attention in robotics due to its importance in the development of motion planning schemes in unknown and dynamic environments, which are close to the real cases of application of a robot. This is why, in parallel with research, they are also important in specialized training processes in robotics. However, access to robotic platforms and laboratories is often complex and costly, with high demands on time and resources, particularly for small research centers. A more efficient and affordable approach to working with autonomous algorithms and motion planning schemes is often the use of the ROS-Gazebo simulator, which allows high integration with customized non-commercial robots, and the possibility of an end-to-end design (E2ED) solution. This research addresses this approach as a training and research strategy with our ARMOS TurtleBot robotic platform, creating an environment for working with navigation algorithms, in localization, mapping, and path planning tasks. This paper shows the integration of ROS into the ARMOS TurtleBot project, and the design of several subsystems based on ROS to improve the interaction in the development of service robot tasks. The project's source code is available to the research community.**

***Keywords*—End-to-End design; localization; navigation; path planning; robotics; SLAM**

I. Introduction

Robotics is a field of research in constant growth, the need for support, cost reduction, safety, and reliability drives the development and study of problems related to its application in real tasks. There are major unsolved problems that hinder its wide use in applications such as service robotics (the area closest to our research), in general, related to interaction, autonomy, safety, and reliability [1]. Many research centers around the world are actively working in these niches, but in many cases, access to robotic platforms and specialized laboratories is restricted for reasons of cost, availability, and resources related to this activity. A working scheme of good acceptance in the scientific community is the use of high-performance simulators, which can replicate very faithfully the behavior of a robot, one of the most popular platforms today in this regard is ROS OS (Robotic Operating System).

The operating system has more than 10 years of growth and interaction in robotic programming, and its particular details such as interoperability and open source generation provide services comparable to those of an operating system, such as hardware abstraction, low-level device control, inter-process message passing and packet management [2]. Though is not an operating system, it provides the same services to its users as an operating system does.

In today's social development, the implementation of mobile robots for problem-solving in industrial and non-industrial environments is of great relevance [3]. Applications in rural or agricultural areas, as well as those involving direct interaction with humans [4], [5], are of great current interest. In this respect, mobile robotic platforms have been proposed that can be used in a greenhouse to transport vegetables [6], [7], in some cases with safe interaction systems with humans. Another study proposes the control and tracking of trajectories based on simulations and experiments in real-time on the ROS platform, where the objective is to validate the effectiveness of the proposed control algorithm and compare it with a modified hybrid PID dynamic controller with feedback [8]. In these cases, ROS was fundamental in the design and performance evaluation stages. As for the concrete problem of mapping and navigation, whether in the land, water, or air vehicles, ROS allows SLAM simulation in the working environment, thus reconstructing the map and dynamically planning a trajectory to the target destination [9], [10].

The use of this type of software tool becomes essential when control schemes integrate a large variety of sensors [11]. Not only does it facilitate the preliminary performance evaluation of the scheme (reducing costs and implementation time), but it also speeds up the overall implementation time of the schemes. In assistive robotics, this is key, as new interaction schemes and control algorithms are continually being developed and need to be evaluated quickly [12]. These applications involve both the interaction problem and the navigation problem, which in many cases involve complex control algorithms, whose performance and coordination must be evaluated before implementation on real platforms [13], [14]. The emulation under ideal conditions, therefore, allows having an additional design tool that guarantees time and design reduction throughout the whole process [15].

Our research project is part of a macro project aimed at the development of robotic solutions for assistance tasks, particularly in applications related to the care of people (elderly and children). In previous years, we have developed a modular mobile platform with load capacity for integration with a manipulator robot (industrial application) and/or an anthropomorphic robot for services [16], [17]. Previous prototypes of this robot had been integrated with ROS to provide ease of handling and simulation of behaviors [18], [19]. At the time, the use of the platform with ROS in structured motion planning applications was justified due to the simplification

in the integration of navigation algorithms. This project seeks the same benefit in a more complex robot, which additionally requires greater availability of access for members of the research group. However, the application of ROS in more complex robot also opened the door to new problems [20].

Similar projects to the one proposed in this research have been previously developed on other platforms [21]. ROS is commonly used to facilitate the design, analysis, and tuning of control algorithms by taking advantage of its modularity and master-slave scheme of work. Our work differs from some of them, such as [22], [23], in the sense that it prefers coding in an open environment with Python support, unlike MatLab, which conforms to a closed platform with expensive licensing. Still, as in its previous uses, it allows the easy use of simulators such as Gazebo. We did not solve all the issues mentioned above, but we did accomplish our goal: We programmed robots using the ROS framework.

II. Problem Statement

The ARMOS TurtleBot robot is a robotic platform designed and implemented by the ARMOS research group to develop assistance tasks and human support (service robot) (Fig. 1). Although it was conceived as a mobile platform for other systems, it has load restrictions (maximum value defined as 10 kg), and movement restrictions derived from its structure and motors. It has four 9 V DC motors with geared motors, driving a caterpillar on a differential structure. Each of the motors has a starting torque of 9.5 kg, no-load speed of 150 RPM (revolutions per minute), starting current of 4.5 A, and nominal working current of 1.2 A (200 mA no-load). These motors have a Hall sensor for shaft position estimation.

Fig. 1. ARMOS TurtleBot Robot.

The working model of the motors is in pairs, i.e., the same control signal activates simultaneously the two motors on the right side, and another signal activates simultaneously the two motors on the left side. This scheme improves the robot's total torque while also simplifying its model and control scheme without sacrificing displacement capacity.

The structure is composed of an aluminum base, a material chosen for its lightweight, which reduces the total weight of the platform. Object detection is done through sensors around the robot. The robot has nine SHARP GP2Y0A21YK IR infrared sensors, which are distributed throughout the robot structure. Other peripherals included in the robot are:

- Four contact buttons, that function as impact detectors.
- A TCS3200 color sensor located at the bottom of the robot.
- An inductive sensor LJ12A3-4-Z/BY located in the lower part of the robot to detect metallic elements.

For the control system, two STM32L432KCU6 microcontrollers are used to process sensor and actuator signals, and a Raspberry Pi 4 card is used as the control unit, which will also have the ROS OS operating system.

The ARMOS TurtleBot robot was designed with diversity and a wide variety of peripherals capable of interacting and communicating with each other through the master-slave level configuration. In addition, it allows the simple integration of new peripherals with a LiDAR sensor. The initial tasks of the robot contemplate navigation problems in dynamic and unknown environments, the reason why in this project we intend to implement ROS in the platform and evaluate its performance in basic navigation tasks that contemplate the use of these peripherals in a real environment.

III. Methods

The ARMOS TurtleBot vehicle platform can be analyzed by separating it into two sections. The first one is composed of position sensors. These are in charge of identifying bodies or objects close to the trajectory performed by the robot. Additionally, the drives of these devices prevent damage to the robot's electronic circuits (Fig. 2).

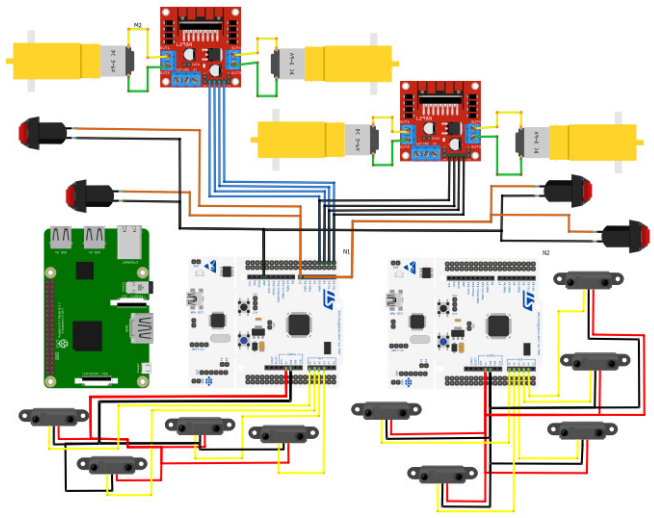

Fig. 2. Proximity Sensor Connection.

The second section is composed of proximity sensors. The main purpose of these devices is to specify the characteristics of the materials that are in the path (floor) and that could

modify their behavior according to the programmed task. There is the detection of metallic materials, and the characterization of materials according to their color (Fig. 3).

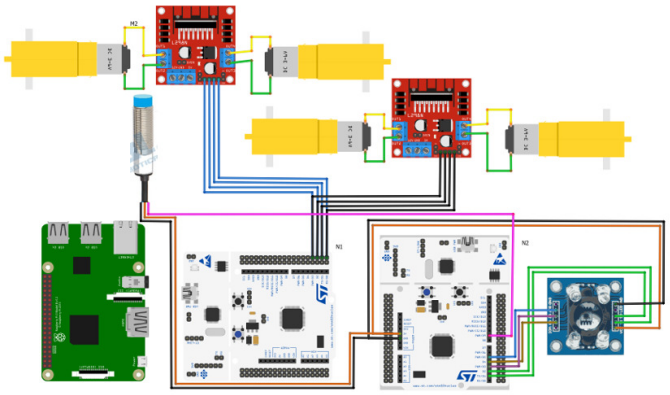

Fig. 3. Connection of Color and Inductive Sensors.

To integrate ROS into the robotic platform we have sought to replicate its structure within the software. All the physical components, considering their specific capabilities, as well as the communication, signal collection, and handling capabilities, have been programmatically replicated on ROS. The correct description of the robot requires a correct description of each peripheral, which was linked to specific laboratory tests. In this way the robot architecture was developed, starting from its control unit, the Raspberry Pi 4 board, which is responsible for managing all the information. Fig. 4 shows the complete architecture of the main systems and subsystems involved in the robotic platform.

The first step is to establish serial communication with the devices or peripherals of the ARMOS TurtleBot platform. To start the interaction with the ROS OS framework, the following command is entered in a Raspberry Pi 4 terminal (on the robot):

```
$ roscore
```

In this way, the operating system starts the communication. The initial part corresponds to the user generated to work on this terminal (pi@raspberrypi:). Thus, if the connection is successful, the system responds as shown in Fig. 5.

This corresponds to the workspace and communication verification process of the master node in ROS OS. At this moment is when the communication or start of the ROS OS system is successfully obtained. The next step is to continue with the opening of the workspace defined for the nodes and commands that are entered, this can be done through the command line as follows.

```
$ cd ~/catking_ws/ &&source devel\setup.bash
$ cd rosrun rosserial_server serial_node _port:=dev/ttyACM0
```

The command line corresponding to rosserial is the one that indicates the communication through language or programs generated through Arduino IDE 18.2, which is the version implemented for the project.

The next step consists of the communication of the peripherals or sensors, both position, and proximity. For this mobile application, the programming is done under the C++ language, where it should be noted the confirmation of one of the main objectives of the ROS OS system, the recycling of code and interoperability between the various programming options available at the moment.

For the inductive sensor LJ12A3-4-Z/BY whose function, as detailed above, is to enable the robot to determine the existence (detection) of metallic material on the surface on which the ARMOS TurtleBot robot is moving (floor), the corresponding command line is as follows:

```
$ rostopic echo inductive_sensor
```

The metal detection and the message received by ROS OS are shown in Fig. 6.

The contact sensor or pushbutton has the objective of detecting contact with an object, which should produce a change in the speed and direction of the robot according to the navigation strategy. The command line to control this sensor is as follows:

```
$ rostopic echo switch_limit
```

The response or communication by ROS OS in case of contact detection is shown in Fig. 7.

The proximity sensor used is the Sharp GP2Y0A21YK IR. There are a total of nine of these sensors, which are in charge of establishing the distances to obstacles or objects around the robot. Their signal is visualized through a graph generated by the operating system, the tool is called rqt_plot. The following are the lines of code and their corresponding response for three sensors (Fig. 8 and 9).

```
$ rqt_plot / sensor3/range: /sensor4/range: /sensor5/range
```

The last sensor coupled to the system is the TC3200 color sensor. The color identification method consists of 64 photodiodes equally distributed in four colors (red, green, blue, and white). Its operating principle is based on the conversion of signals (electronic circuit) from current to frequency, where a specific frequency is assigned to the detected color, thus with the constant determination of three frequencies (red, green, and blue) and the determination of the frequency range during the three readings, the conditions to be programmed in the control unit are obtained. The frequency ranges are specified in Table I.

TABLE I. FREQUENCY RANGE FOR COLOR DETERMINATION BY THE SENSOR

Colors according to frequency range			
Red	R<50	V>90	A>90
Blue	R>85	V>80	A<50
Green	R>90	V<120	A>58

In the robot, the color sensor is managed by a STM32L432KCU6 microcontroller. This microcontroller establishes the communication with ROS OS. To query the

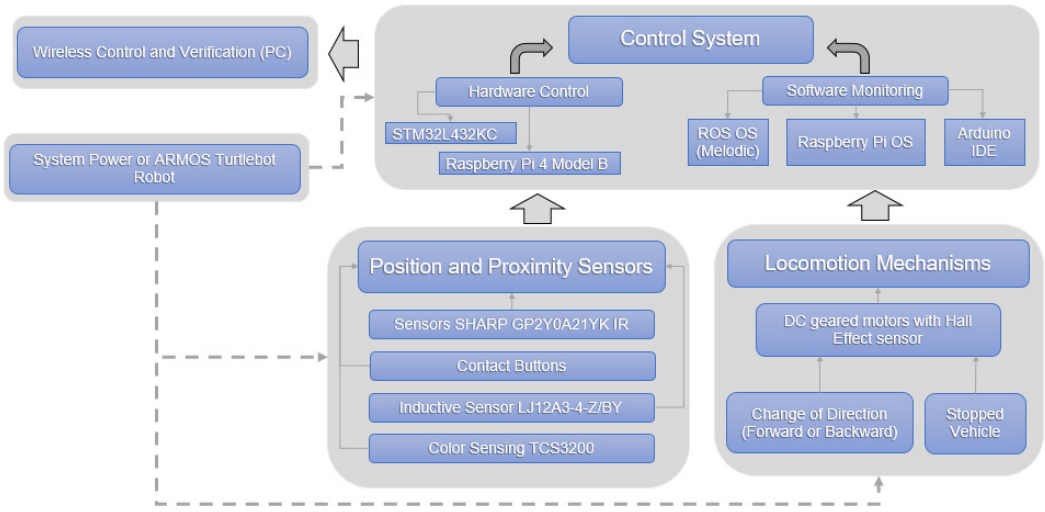

Fig. 4. Control Scheme: Hardware and Software.

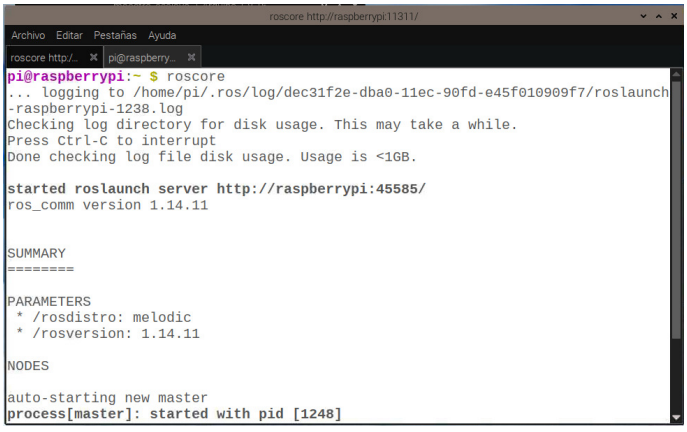

Fig. 5. Roscore Start.

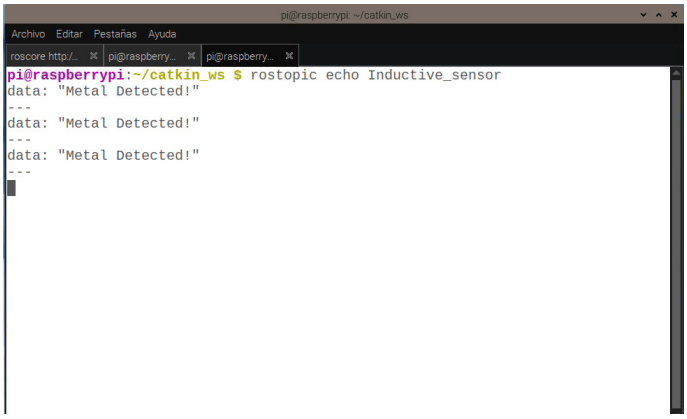

Fig. 6. Metal Detection Report from the Inductive Sensor.

reading, a message is sent from ROS similar to the one shown below.

```
$ rostopic echo color_sensor
```

Actuator management is also configured from ROS. The monitoring and instructions of the DC motors, as well as the Hall effect sensors coupled to each of them, can also be easily managed from this environment. These sensors prove to be an essential tool that allows the feedback of the movement of each motor, thus reducing errors in autonomous navigation tasks in real environments. Three basic commands were designed for its operation: go forward, stop and reverse. These commands are translated into specific movements performed by each of the motors, and controller through Hall effect sensors. The instruction for their execution is as follows:

```
$ rostopic pub forward std_msgs/empty
```

IV. Results and Discussion

To evaluate the performance of the system, a test environment was built to assess the behavior of each of the robot's sensors, as well as its actuators and communication characteristics. The following is a brief description of this test environment, as well as the expected interaction with the ARMOS Turtlebot robot, its displacement parameters, and the defined trajectory path.

The scenario where the peripheral integration tests are performed on the ARMOS Turtlebot robot has a flat surface (Fig. 10). In the center of the area, there is a box composed of different materials with characteristics recognizable by the devices embedded in the robot. The navigation of the robot was programmed with restrictions on its displacement, to evaluate the interaction of sensors and actuators in the ROS OS operating system. The navigation conditions for the execution of the experiment are detailed below.

The displacement of the robot starts with the control signal generated from the control unit installed on the robot

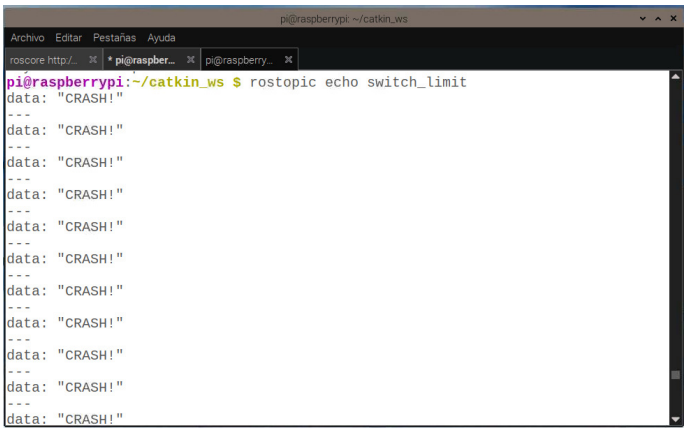

Fig. 7. Impact Report from the Contact Sensor.

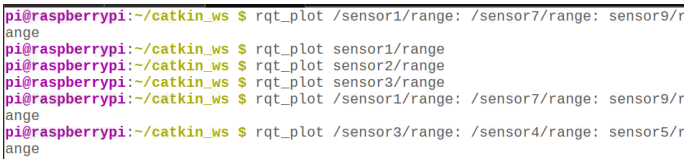

Fig. 8. IR Sensors Report.

itself. From there, the ROS connection is established with the peripherals as described above. As soon as the communication is established and the work folder is opened, the trajectory is started with the instruction from the corresponding command line. At this point, the ROS connection has been established between the control unit and other peripherals.

The specific mechanism to intervene during the whole test is the caterpillar wheels equipped with geared motors and coupled to Hall effect sensors. These devices are used to move the robot to carry out the trajectory. The path is defined on a surface adapted with characteristic colored materials, recognizable by the TC3200 color sensor. The first zone corresponds to the blue material, the sensor acquires this information and transmits it directly to ROS OS, which is in charge of processing the data and sending a color detection message. In this case, the text specifying the color of the detected object is displayed on the terminal.

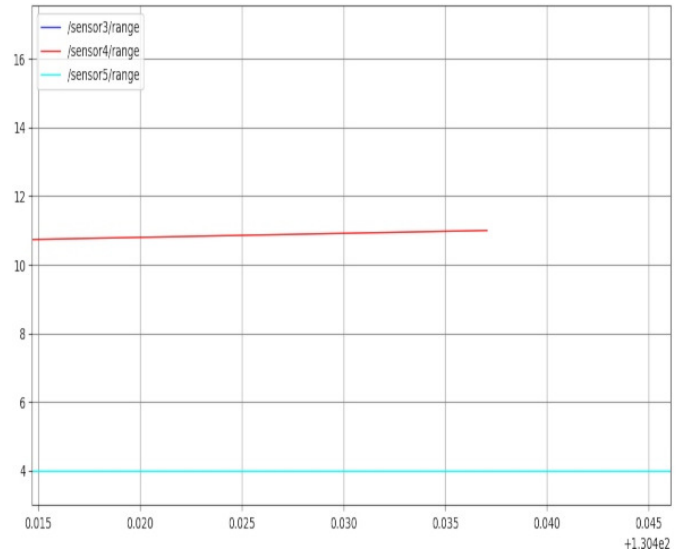

Fig. 9. Distance Curves Reported by the IR Sensor

Fig. 10. Test Environment Setup.

Sharp GP2Y0A21YK IR infrared proximity sensors have a maximum detection range of 80 cm. For the proposed test application, this detection is conditioned to a maximum of 20 to 25 cm, and the control is done graphically in real-time through the rqt_plot tool of ROS OS, making the recognition of object or obstacle and its distance. Once this condition is met, the programmed instruction is the change of direction, and to perform this action by the robot, a rotation is performed on its axis (right) which occurs whenever the motors on each side rotate at the same speed and in opposite directions (one side of the robot moves forward and the other backward).

Starting up in the new direction, the surface on which it is running is colored green, which is detected by the TC3200 color sensor. Thereupon the system and its programming will issue a message on the terminal indicating that the second green area has been detected. While the vehicle is running, a green-colored area is detected by the TC3200 color sensor.

When a metallic material is detected, it is time for the intervention of the inductive sensor LJ12A3-4-Z/BY, which causes two different interactions in the programming and control of the robot. The first consists of the ROS instruction that indicates the detection of metal, performing a change of direction or rotation on the axis. Additionally, the data of the metal material is sent so that a message is transmitted to the terminal indicating the detection of this element.

It changes direction in a 90-degree turn with a new rotation on its axis in relation to its previous position. The robot is moving over a third displacement zone, this time in red. As before, this information is reflected through ROS OS using a message indicating the detection of a colored area.

The test or experiment ends with the intervention of the contact sensors that aim to detect the impact of an obstacle on the robot. These are activated when changing state (bi-valued sensor), which complements the reading of the distance sensor formed by the nine infrared sensors IR Sharp GP2Y0A21YK, or even substitute it when the obstacle is below the line of sensing of the infrared. The action defined at this point is that the robot stops immediately.

This navigation circuit was developed by the robot multiple times to verify the consistency of operation and results. One of these tests can be seen in the following link (Fig. 11).

https://youtu.be/uh5wzhIwi6g

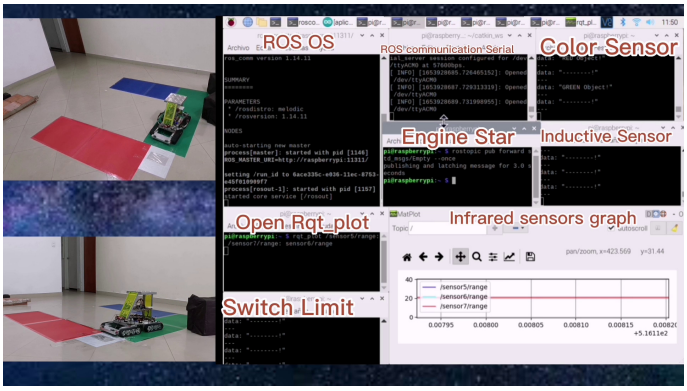

Fig. 11. ROS-Controlled Robot Performance Test.

Communication between ROS OS and the sensors allows it to know where they are and allows for these devices to communicate between them automatically, as a message that is generated by one device will be transmitted automatically to all other devices. When running all the peripherals and devices associated with the robot, it is evident that the communication between ROS OS and these works correctly because there are no crashes or failures when all these devices are communicating at the same time in the system. This feature guarantees the robustness of the system and allows for validating it for evaluation tests of more complex navigation algorithms.

The inductive sensor detects the metal correctly, and the message display transmitted by ROS OS through the terminal shows the warning message approximately one second after the sensor detected the metal surface. In the case of the contact sensor, it was triggered immediately when the robot had a collision with the obstacle that was not detected by the IR ranger sensors located at the front of the robot. The warning message displayed by the ROS system was almost instantaneous when the collision occurred (about a 120 ms delay).

As for the motors, when the command was executed in ROS OS on the terminal to move the wheels of the robot and produce forward motion, it was evident in all tests that there was a delay in the response of the motors of approximately three seconds after the command was executed. The same was true for moving the wheels backward or for stopping the movement of the motors completely. After almost three seconds of waiting to start or stop the movement, the motors responded appropriately.

The most important sensor in the reactive control tasks is the ranger IR sensor. Each sensor in the system was individually tested for interaction with ROS OS. The object detection tests were verified in the rqt_plot tool that is integrated in ROS OS, which allows to view two-dimensional plots. In this tool, a distance vs. time graph is displayed, where a continuous line appears referring to the distance in cm, in which the ranger sensor is detecting an obstacle. The individual tests showed that the sensors detected the objects acceptably with an error of approximately one to two centimeters. Back in the test application, where the sensors were acting in conjunction with the other devices on the robot, it was found that the reading accuracy error increased to almost three centimeters for some sensors. When trying to visualize the nine signals detected by each of the sensors in the rqt_plot tool, the operating system of the raspberry pi generated delay in the response and sometimes even total crash of the system. So to avoid this problem, it was decided to visualize in the rqt_plot tool only some of the IR, but to use all of them for mapping tasks. In the tests, the response of the three ranger sensors located at the front of the robot is visualized. The ranger sensors were programmed to detect objects within a range of 0.2 to 0.25 m, and then instantly generate action on the robot's motors. Furthermore, when they detected an object in this distance range, the motors reacted within two to three seconds after the sensor had detected the object.

The color sensor was evaluated individually with ROS OS before being incorporated with the other devices and sensors on the robot. The ROS readings were accurate over the three colors that the sensor measures. When the sensor was incorporated into the test task, the color readings began to differ significantly from the previous ones, resulting in erroneous readings in the application. The possible interference of communication delays on the performance of this sensor is evaluated in this case.

In this study, the results of implementing ROS as a handling strategy for the ARMOS TurtleBot robot were achieved and shown. The strategies used can be generalized and further investigated on this platform as well as on other custom robotic platforms not initially designed with ROS in mind. This work can be further enhanced with the use of navigation algorithms and optimization techniques that reflect the comparative performance of each case. This study shows that ROS can be used as a suitable programming environment for a robot and should be the one platform from which further work is don

V. Conclusion

The constant innovation in design, programming, and control in mobile robotics leads to the generation of a wide variety of algorithms, compromising the understanding and suitability of robotic applications as a whole. A design methodology already proven and widely studied is the versatility offered by the ROS OS operating system in the different stages of the development of control schemes for robots. This tool combines the ease of code recycling and the interaction and communication of different types of programming languages. This paper presents and explains the integration of the ROS OS (Melodic) operating system on the ARMOS TurtleBot mobile robot and its peripherals. The platform is designed to perform navigation tasks or activities, thus obtaining synchronization, data transfer, and information management in real-time on surfaces with suitable characteristics to be detected by the different devices that make up the robot. The proposed purpose corresponds to the integration of the devices that make possible the displacement and recognition of the robot in a given environment. The systems involved according to the architecture are Raspberry Pi OS, ROS OS (Melodic), and some microcontrollers as embedded systems that directly manipulate sensors and motors, with this architecture the compatibility and possibility of integration of peripherals are confirmed. The simultaneous interaction of the proximity peripherals and position sensors through the ROS OS operating system is visualized through windows or terminals utilizing messages or interactively in 2D graphics, confirming the interoperability and real-time control of the environment surrounding the robot when it performs navigation tasks. The presence of windows and graphic interfaces for its control demonstrates the possibility of remote real-time observation and monitoring, providing immediate feedback about the robot's environment. No communication problems are detected that could prevent its use as a general strategy in the development of navigation algorithms. On the contrary, the possibility of integrating additional peripherals such as a LiDAR sensor or a digital camera onboard is observed.

Acknowledgment

This work was supported by the Universidad Distrital Francisco José de Caldas, specifically by the Technological Faculty. The views expressed in this paper are not necessarily endorsed by Universidad Distrital. The authors thank all the students and researchers of the research group ARMOS for their support in the development of this work.

References

[1] B. Liu and C. Liu, "Path planning of mobile robots based on improved RRT algorithm," *Journal of Physics: Conference Series*, vol. 2216, no. 1, p. 012020, mar 2022.

[2] R. Alonso, A. Bonini, D. R. Recupero, and L. D. Spano, "Exploiting virtual reality and the robot operating system to remote-control a humanoid robot," *Multimedia Tools and Applications*, vol. 81, no. 11, pp. 15 565–15 592, feb 2022.

[3] A. Shahoud, D. Shashev, and S. Shidlovskiy, "Visual navigation and path tracking using street geometry information for image alignment and servoing," *Drones*, vol. 6, no. 5, p. 107, apr 2022.

[4] M. Quiroz, R. Patiño, J. Diaz-Amado, and Y. Cardinale, "Group emotion detection based on social robot perception," *Sensors*, vol. 22, no. 10, p. 3749, may 2022.

[5] P. Wang, R. Ye, J. Zhang, and T. Wang, "An eco-driving controller based on intelligent connected vehicles for sustainable transportation," *Applied Sciences*, vol. 12, no. 9, p. 4533, apr 2022.

[6] E.-T. Baek and D.-Y. Im, "ROS-based unmanned mobile robot platform for agriculture," *Applied Sciences*, vol. 12, no. 9, p. 4335, apr 2022.

[7] R. Xu and C. Li, "A modular agricultural robotic system (MARS) for precision farming: Concept and implementation," *Journal of Field Robotics*, vol. 39, no. 4, pp. 387–409, jan 2022.

[8] A. D. Sabiha, M. A. Kamel, E. Said, and W. M. Hussein, "ROS-based trajectory tracking control for autonomous tracked vehicle using optimized backstepping and sliding mode control," *Robotics and Autonomous Systems*, vol. 152, p. 104058, jun 2022.

[9] M. Facerias, V. Puig, and E. Alcala, "Zonotopic linear parameter varying SLAM applied to autonomous vehicles," *Sensors*, vol. 22, no. 10, p. 3672, may 2022.

[10] H. M. P. C. Jayaweera and S. Hanoun, "Path planning of unmanned aerial vehicles (UAVs) in windy environments," *Drones*, vol. 6, no. 5, p. 101, apr 2022.

[11] Y. Guo, X. Fang, Z. Dong, and H. Mi, "Research on multi-sensor information fusion and intelligent optimization algorithm and related topics of mobile robots," *EURASIP Journal on Advances in Signal Processing*, vol. 2021, no. 1, nov 2021.

[12] J. A. C. Panceri, É. Freitas, J. C. de Souza, S. da Luz Schreider, E. Caldeira, and T. F. Bastos, "A new socially assistive robot with integrated serious games for therapies with children with autism spectrum disorder and down syndrome: A pilot study," *Sensors*, vol. 21, no. 24, p. 8414, dec 2021.

[13] A. A. Umar and J.-S. Kim, "Nonlinear model predictive path-following for mecanum-wheeled omnidirectional mobile robot," *The transactions of The Korean Institute of Electrical Engineers*, vol. 70, no. 12, pp. 1946–1952, dec 2021.

[14] B. Boroujerdian, R. Ghosal, J. Cruz, B. Plancher, and V. J. Reddi, "RoboRun: A robot runtime to exploit spatial heterogeneity," in *2021 58th ACM/IEEE Design Automation Conference (DAC)*. IEEE, dec 2021.

[15] S. Chen, W. Zhou, A.-S. Yang, H. Chen, B. Li, and C.-Y. Wen, "An end-to-end UAV simulation platform for visual SLAM and navigation," *Aerospace*, vol. 9, no. 2, p. 48, jan 2022.

[16] C. Penagos, L. Pacheco, and F. Martínez, "Armos turtlebot 1 robotic platform: Description, kinematics and odometric navigation," *International Journal of Engineering and Technology*, vol. 10, no. 5, pp. 1402–1409, 2018.

[17] C. Hernández, D. Giral, and F. Martínez, "Kinematic and dynamic analysis of a differential robotic platform with caterpillar tracks," *(JATIT) Journal of Theoretical and Applied Information Technology*, vol. 99, no. 24, pp. 5993–6003, 2021.

[18] A. Moreno and D. Páez, "Performance evaluation of ros on the raspberry pi platform as os for small robots," *Tekhnê*, vol. 14, no. 1, pp. 61–72, 2017.

[19] F. Martínez, "Turtlebot3 robot operation for navigation applications using ros," *Tekhnê*, vol. 18, no. 2, pp. 19–24, 2021.

[20] A. Yilmaz, E. Sumer, and H. Temeltas, "A precise scan matching based localization method for an autonomously guided vehicle in smart factories," *Robotics and Computer-Integrated Manufacturing*, vol. 75, p. 102302, jun 2022.

[21] S. Abdul-Rahman, M. S. A. Razak, A. H. B. M. Mushin, R. Hamzah, N. A. Bakar, and Z. A. Aziz, "Simulation of simultaneous localization and mapping using 3d point cloud data," *Indonesian Journal of Electrical Engineering and Computer Science*, vol. 16, no. 2, p. 941, 2019.

[22] Z. Chen, S. Yan, M. Yuan, B. Yao, and J. Hu, "Modular development of master-slave asymmetric teleoperation systems with a novel workspace mapping algorithm," *IEEE Access*, vol. 6, no. 1, pp. 15 356–15 364, 2018.

[23] N. Sadeghzadeh-Nokhodberiz, A. Can, R. Stolkin, and A. Montazeri, "Dynamics-based modified fast simultaneous localization and mapping for unmanned aerial vehicles with joint inertial sensor bias and drift estimation," *IEEE Access*, vol. 9, no. 1, pp. 120 247–120 260, 2021.