XLIII CILAMCE
21–25, November, 2022
FOZ DO IGUAÇU – PR – BRAZIL

ROS 2 in the development of an autonomous navigation application for a 4WD mobile robot with GPS, odometry and inertial systems

Pablo F. Salarolli[1], Leonardo G. Batista[1], Gustavo M. de Almeida[1], Rafael P. D. Vivacqua[1], Daniel F. T. Gamarra[2], Marco Antonio de S. L. Cuadros[1]

[1]*Master's program in control and automation engineering, Federal Institute of Espírito Santo*
Avenida dos Sabiás nº 330 - Morada de Laranjeiras, CEP: 29166 - 630 - Serra/ES, Brazil
pablo_salarolli@hotmail.com, leonardo-baptista@live.com, gmaia@ifes.edu.br, rafsat@ifes.edu.br, marcoantonio@ifes.edu.br
[2]*Control and Automation Engineering Course, Federal University of Santa Maria*
Av. Roraima nº 1000 - Camobi, CEP 97105-900, Santa Maria/RS, Brazil
fernandotg99@gmail.com

Abstract. Developing software for robots is a complex task as it involves many fields of knowledge, from sensor drivers for data acquisition, data processing, control loops, artificial intelligence, task management, etc. In addition, the hardware is often built distributed across different devices with different computing capabilities. Based on this, the Robot Operating System (ROS) was developed, which is a framework for developing distributed robot applications. ROS provides a communication layer that abstracts the way communication is performed and allows transparent and distributed access to data. In addition, ROS provides tools for viewing information, compiling, managing packages, etc. A new version called ROS 2 has been launched, and it includes industry-standard features, best practices in software development to better support commercial and research robotic applications. To demonstrate the use of ROS in the development of robotic applications, this paper presents an outdoor autonomous navigation application of a 4WD robot. This application explored the use of microcontrollers to control wheel speed, embedded minicomputer running algorithms for sensor fusion of odometry, inertial sensors and GNSS for localization and navigation algorithms. Navigation tests were conducted in a parking lot.

Keywords: mobile robots, ROS 2, autonomous navigation, robot software.

1 Introduction

The scope and scale of robotics are continually growing, and the wide variation in the type of hardware used by each robot makes this a complex task. These are tasks from the hardware interface level to perception algorithms, abstract reasoning, etc., which make code reuse difficult. There is also the fact that the breadth of knowledge required to implement all these tasks is beyond the capabilities of any researcher. ROS (Robot Operational System) emerged to make easier to develop software for robots, as said by Quigley et al. [1].

Despite the name, ROS is not an operating system. In fact, it is an open-source framework that provides a structured communication layer over operating systems on a heterogeneous network of computers. It also provides libraries and tools to facilitate the creation of robotic applications, message transfer between processes, viewers, device drivers, hardware abstraction, package management, etc., as described by Quigley et al. [1] and ROS [2].

The present work aims to demonstrate the development of an outdoor autonomous navigation robotic application, which includes the use of several technologies. In this application, various resources were explored such as the use of microcontrollers to control wheel speed, an embedded minicomputer running algorithms for sensor fusion of odometry, inertial sensors and GNSS for localization, navigation algorithms such as path planner, trajectory control, local and global costmaps and obstacle avoidance. The work describes which ROS tools and packages were used, demonstrating how the task of designing a complex application like this is simplified using the ROS ecosystem.

2 Architecture

2.1 About the robot used

The robot used is a differential mobile robot with four-wheel drive, with each pair of wheels on each side being driven by a DC motor. The robot is also equipped with two incremental wheel encoders, IMU Bosch BNO055, 2D LiDAR Sick TiM551, RTK GNSS receiver ublox C94-M8P, ESP32 microcontroller, Raspberry PI 4B mini-computer and Wi-Fi access point, as can be seen in Figure 1 as well as the hardware architecture.

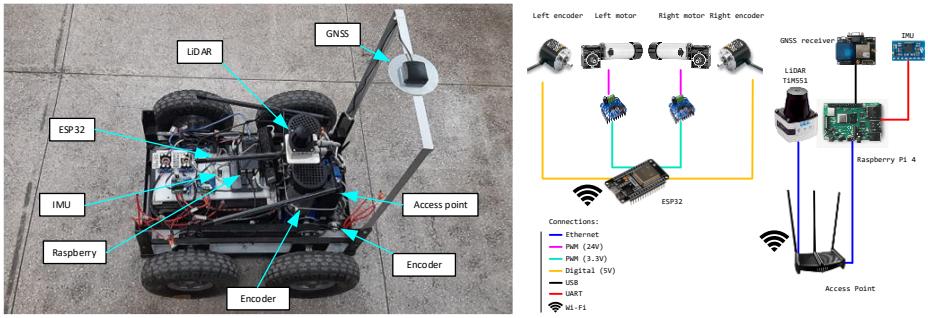

Figure 1. 4WD robot used (left) and hardware architecture (right).

2.2 Sensor measurements and low-level control

Use of miroROS for compute wheel odometry and PID control loops. Reading the encoders, calculating the odometry and executing the wheel speed control loops was performed using microROS, a version of ROS 2 that can be embedded into microcontrollers. The firmware was developed taking advantage of the ESP32's multicore processing capability, using multiple threads for each task. The firmware architecture can be seen in Figure 2. In summary, ESP32 receives wheel speed setpoints and executes the control loops for each wheel and applies the PWM signals to the motors' power drivers. These setpoints are computed based on the `cmd_vel` topic, which consists of linear and angular velocity set points calculated by the autonomous navigation stack or by the joystick. Furthermore, it performs reading of both right and left wheel encoders (two channels each) through interrupts. With pulse counting, the odometry is computed and published via a topic to be accessed in any ROS node.

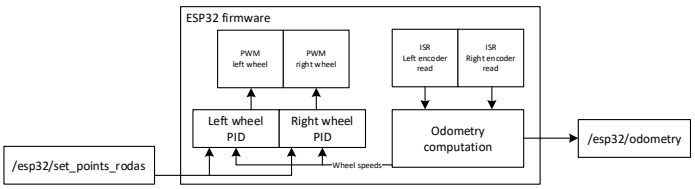

Figure 2. Firmware architecture.

IMU, GNSS and LiDAR. The reading of the other sensors was performed using packages available for ROS 2. The IMU data was read by the package `bno055`, developed by the GitHub user Flynn [3]. The reading of the GNSS receiver was performed using the `ublox` packages, developed by Kumar Robotics [4]. LiDAR reading was performed using the `sick_scan2` package, developed by the sensor manufacturer Sick AG [5].

2.3 Localization and transform setup

Alatise and Hancke [6] claim that the localization system is one of the most important modules to implement an autonomous navigation system. By dealing with data of different types and sampling rates, this is usually a complex task. ROS provides two fundamental packages to assist in the assembly of a robust localization system: `robot_localization` and `tf2`. The `robot_localization` package authors, Moore and Stouch [7] claims

that the package contains state estimation nodes, each of which is an implementation of a non-linear state estimator for robots moving in 3D space, using either the extended Kalman filter (EKF) or the unscented Kalman filter (UKF), in addition to a node that assists in the integration of GNSS data. The `tf2` package is the ROS transform library, which allows the user to track multiple coordinate frames over time, maintaining the relationship between frames in a tree structure buffered in time, and allows the user to transform points, vectors, etc. between any two coordinate frames at any desired point in time, as said the `tf2` author Foote [8].

The localization system was configured in accordance with REP-105, proposed by Meeussen [9], which is a requirement of the *Nav2*, navigation stack used that will be detailed in section 2.4. Basically, it is necessary for the transform tree to have two reference systems, a local one to control and avoid obstacles and a global one to maintain the correct pose estimation on the map, in addition to the static transforms for the sensors. The configured transform tree can be seen in Figure 3 (left).

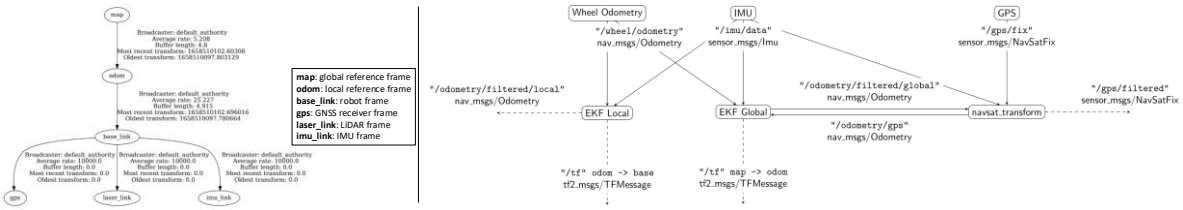

Figure 3. tf2 transform tree (left); Localization system architecture [10] (right).

To reach this transform tree configuration, it is necessary to correctly configure the localization system. The architecture used for the localization system was the one proposed by the `robot_localization` package documentation, as can be seen in Figure 3 (right). In this way, the localization system generates the transform tree expected by the *Nav2* navigation stack, in addition to providing data through topics.

2.4 Navigation stack

The navigation stack used in this work is Nav2, which is the stack presented in "The Marathon 2" paper of Macenski et al. [11], allowing two robots to navigate over 37 miles through high-trafficked areas. The Nav2 framework seeks to find a safe way to have a mobile robot move from point A to point B. It can also be applied in applications like following dynamic points. Nav2 will complete dynamic path planning, compute velocities for motors, avoid obstacles, and structure recovery behaviors. It uses behavior trees (BT) to call modular servers to complete an action, that can be to compute a path, control effort, recovery, or any other navigation related action. The diagram of Figure 4 gives an overview at the structure of Nav2. It has tools to: load, serve, and store maps; localize the robot on the map; plan a path from A to B around obstacles; control the robot as it follows the path; smooth path plans to be more continuous and feasible; convert sensor data into a costmap representation of the world; build complicated robot behaviors using behavior trees; compute recovery behaviors in case of failure; follow sequential waypoints; manage the lifecycle and watchdog for the servers; and plugins to enable your own custom algorithms and behaviors, as described in the Nav2 documentation written by Macenski [12].

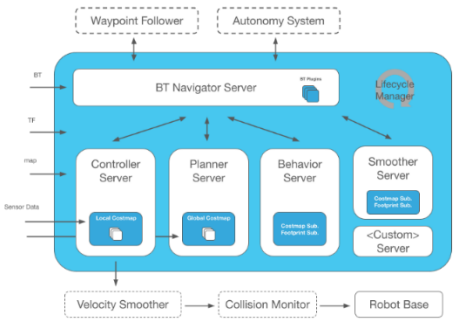

Figure 4. Nav2 architecture. [12]

The expected inputs to Nav2 are TF transformations conforming to REP-105, a map source, a BT XML file, and any relevant sensor data sources. It will then provide valid velocity commands for the motors of a holonomic or non-holonomic robot to follow.

2.5 Other used packages

Other packages used were: `joy` and `teleop_twist_joy` to control the robot through a joystick; `rviz2`, which contains RViz, ROS supervision and visualization software; `rqt_graph` for visualization of the communication graph between nodes through topics.

3 Tests and results

The algorithms mentioned in section 2 were configured and executed in the robot. The fully configured system has approximately 39 nodes communicating through 64 topics.

To evaluate the assembled structure, initially the robot was positioned in a parking lot. Then it was requested the robot to go to a certain point. After it reached the first goal, it was requested to the robot to go to another point, however, in the middle of the way an obstacle appears to test the ability to avoid obstacles dynamically. Figure 5 shows the supervision and testing environment. Figure 6 shows the moment the robot detects and avoids the obstacle. The complete result can be watched in a YouTube video of GAIn [13] (https://youtu.be/AUlz58yum58).

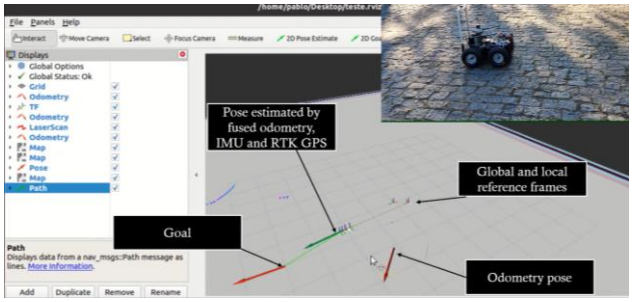

Figure 5. Navigation test: RViz and robot image synchronized.

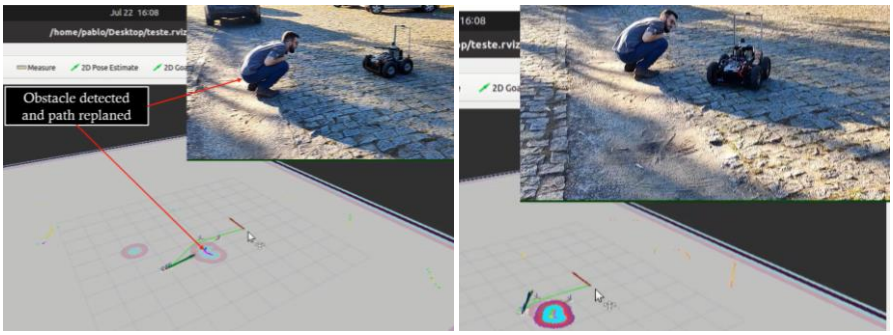

Figure 6. Navigation test: RViz and robot image synchronized at the moment of avoiding an obstacle.

4 Conclusions

ROS 2 was used to develop an outdoor autonomous navigation robotic application. As can be seen in the video of the test performed, the implementation was successful. The strategy of developing applications in a distributed and isolated way contributes to the applications to be decoupled and organized. The use of ROS also contributes to the fact that a lot of the code does not need to be developed again, but rather reused through packages made available by the ROS community.

As future works, GNSS fusion will be better managed, according to covariance of the signal, allowing the

localization system to be free of RTK corrections failures, making navigation even more reliable and smooth. It is also possible to improve the current application to become a ready for use application, i.e., setup the software to run immediately the robot is turned on, allowing the user to use it with the joystick or in an autonomous application. It can also be developed some specific applications for agriculture, surveillance, industrial inspection, etc.

Acknowledgements. The development of this work was supported by the Postgraduate Program in Control and Automation Engineering (ProPECAut) of the Federal Institute of Espírito Santo (Ifes - Campus Serra).

Authorship statement. The authors hereby confirm that they are the sole liable persons responsible for the authorship of this work, and that all material that has been herein included as part of the present paper is either the property (and authorship) of the authors, or has the permission of the owners to be included here.

References

[1] M. Quigley *et al.*, "ROS: an open-source Robot Operating System," 2009.

[2] ROS, "ROS/Introduction - ROS Wiki," *ROS Wiki*, 2018. http://wiki.ros.org/ROS/Introduction (accessed Oct. 24, 2019).

[3] Evan Flynn, "flynneva/bno055: ROS2 driver for Bosch BNO055 using UART or I2C," 2019. https://github.com/flynneva/bno055 (accessed Jul. 24, 2022).

[4] Kumar Robotics, "KumarRobotics/ublox: A driver for ublox gps," 2014. https://github.com/KumarRobotics/ublox (accessed Jul. 24, 2022).

[5] Sick AG, "SICKAG/sick_scan2: sick_scan2 is an open-source project to support the laser scanner of the company SICK using the ROS2 framework," 2018. https://github.com/SICKAG/sick_scan2 (accessed Jul. 24, 2022).

[6] M. B. Alatise and G. P. Hancke, "A Review on Challenges of Autonomous Mobile Robot and Sensor Fusion Methods," *IEEE Access*, vol. 8, pp. 39830–39846, 2020, doi: 10.1109/ACCESS.2020.2975643.

[7] Tom Moore and Daniel Stouch, "robot_localization wiki — robot_localization 2.7.3 documentation," 2014. http://docs.ros.org/en/noetic/api/robot_localization/html/index.html (accessed Jul. 24, 2022).

[8] Tully Foote, "tf2 - ROS Wiki," 2013. http://wiki.ros.org/tf2 (accessed Jul. 24, 2022).

[9] Wim Meeussen, "REP 105 -- Coordinate Frames for Mobile Platforms (ROS.org)," *ROS.org*, 2010. https://www.ros.org/reps/rep-0105.html (accessed Sep. 26, 2019).

[10] Tom Moore and Daniel Stouch, "Integrating GPS Data — robot_localization 2.7.3 documentation," 2014. http://docs.ros.org/en/noetic/api/robot_localization/html/integrating_gps.html (accessed Jul. 24, 2022).

[11] S. Macenski, F. Martin, R. White, and J. G. Clavero, "The marathon 2: A navigation system," *IEEE International Conference on Intelligent Robots and Systems*, pp. 2718–2725, Oct. 2020, doi: 10.1109/IROS45743.2020.9341207.

[12] S. Macenski, "Nav2 — Navigation 2 1.0.0 documentation," 2021. https://navigation.ros.org/ (accessed May 06, 2022).

[13] GAIn, "ROS 2 in the development of an autonomous robot application - YouTube," 2022. https://www.youtube.com/watch?v=AUlz58yum58 (accessed Jul. 24, 2022).