DESIGNING AN ALGORITHM AND DEVELOPING AN INTEGRATED NAVIGATION SYSTEM IN ROS2 AND NAV2 FOR INTRALOGISTICS ACTIVITIES WITH AN AUTONOMOUS MOBILE ROBOT (AMR)

DABIJA Mihai-Daniel, Conf. Dr. Eng. ABAZA Bogdan Felician
College: FIIR, Specialty: IAII, Year of studies: IV, e-mail: dabija.mihai97@gmail.com

ABSTRACT: The purpose of this paper is to investigate the implementation of a navigation system based on the Nav2 module, using the ROS2 framework. This research is realized on an AMR type robot with a predefined hardware architecture.

Keywords: ROS2, Nav2, AMR

1. Introduction

The paper focuses on the design of an algorithm and the development of a navigation system integrated into ROS2 and Nav2 for intralogistics activities with an Autonomous Mobile Robot (AMR). The case study considers the transfer of boxes with semi-finished products and parts between workstations (machine tools) in a specific location within the FIIR Faculty. The work will be conducted by implementing the ROS2 Humble framework (Robot Operating System). This framework comprises a set of software libraries and tools to build robotic applications, ranging from drivers and state-of-the-art algorithms to powerful development tools, providing the necessary open-source instruments for specific developments [1]. The development of the topic involves the following stages:

- installation of the ROS2 Humble framework on a virtual machine required for simulation;
- installation of the compatible navigation module – Nav2;
- programming and configuration of the onboard computer installed on the AMR robot within the ROS2 environment;
- development of the navigation algorithm in the simulation environment on the virtual machine;
- development of the navigation algorithm configured for the existing hardware subsystems of AMR;
- testing the navigation algorithm in both the simulation environment and on the AMR vehicle;

To solve the task, knowledge in the following areas is required:

- robotics, for analyzing the AMR, enabling its transformation into a set of data that the framework can understand;
- programming, for installing the framework, the module, configuring the robot in ROS2, data transfer, and processing;
- logistics, for understanding the logistics processes involving the robot and implementing them in software to make them more efficient.

Using the ROS2 framework, it was consider to utilize too, some specialized software packages for simulation, such as Webots and Gazebo, with the aim of testing the implementation of the navigation algorithm with the specialized Nav2 module first in a simulation environment and then implementing and testing it on the physical AMR vehicle with a known hardware configuration.

2. Definition of the simulation model for intralogistics activities

The simulation model is designed to ensure the navigation algorithm of an AMR for performing the following intralogistics activities [5]:

- putaway: transferring products to the storage area;
- replenishment: replenishing source cells for picking/kitting;
- line feeding: delivering goods and materials to production lines;
- end of line: retrieving finished products from production lines and delivering them to the warehouse.

In this case, it is assumed that for each of the above activities, in the case of a data process, there are predefined routes on which an AMR vehicle can move. At the end of a route, there are passive stations for loading or unloading payloads (box with semi-finished products or parts). Thus, when the AMR vehicle receives a task for performing an activity, it should be able to autonomously move from the battery charging station where it is located, towards Station A, pick up the load box, and move towards Station B at the end of the designated route for the given activity, where it will leave the box. Then, the vehicle can perform another activity or move to the battery charging station. The movement will be done autonomously based on a navigation algorithm (Nav2) that should provide perception, planning, control, localization, and complete environment modeling from the data acquired from sensors. It should also ensure dynamic route planning, motor speed calculation, obstacle avoidance, representation of regions and objects in the route, and structuring of the robot's behavior at a higher level [3].

3. Designing ROS2 simulation environment for navigation

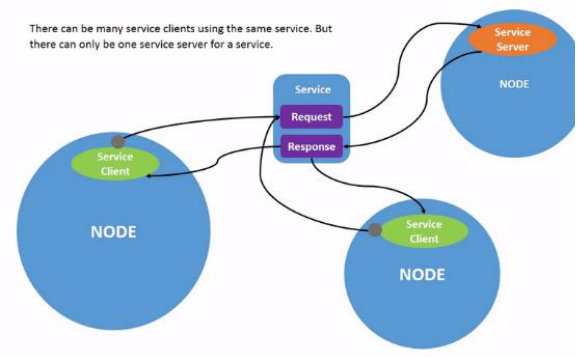

Fig. 1. **Example model of communication between nodes [1]**

```python
class ObstacleAvoider(Node):
    def __init__(self):
        super().__init__('obstacle_avoider')

        self.__publisher = self.create_publisher(Twist, 'cmd_vel', 1)

        self.create_subscription(Range, 'left_sensor', self.__left_sensor_callback, 1)
        self.create_subscription(Range, 'right_sensor', self.__right_sensor_callback, 1)

    def __left_sensor_callback(self, message):
        self.__left_sensor_value = message.range

    def __right_sensor_callback(self, message):
        self.__right_sensor_value = message.range

        command_message = Twist()

        command_message.linear.x = 0.1

        if self.__left_sensor_value < 0.9 * MAX_RANGE or self.__right_sensor_value < 0.9 * MAX_RANGE:
            command_message.angular.z = -2.0

        self.__publisher.publish(command_message)
```

Fig. 2. **Node model**

In ROS2, the program is structured into nodes. Each node should be responsible for a single purpose, modular, for example, controlling wheel motors or publishing sensor data from a laser rangefinder. Each node can send and receive data from other nodes through topics, services, actions, or parameters. A complete robotic system consists of many nodes working together. In ROS2, a single executable [6] (C++ program, Python program, etc.) can contain one or more nodes. Nodes communicate with each other through topics [7]. Topics represent communication channels where sensors, nodes, servers, and other modules post data, commands, information, etc. The figure below presents an example node for obstacle avoidance.

Defining a navigation system in the Nav2 module requires following a well-defined architecture.

Fig. 3 represents the architecture of the navigation system, consisting of the following input parameters:

- BT (Behavior-Tree), which represents a task switching and decision-making tool;
- map (predefined and constructed map that the system constantly updates and uses for path creation);

- TF (Transformation Library), which is a library that allows tracking multiple coordinate frames over time. It maintains relationships between coordinate frames in a time-stamped tree structure and enables the transformation of points, vectors, etc., and data from sensors, modules, and other installed equipment;
-

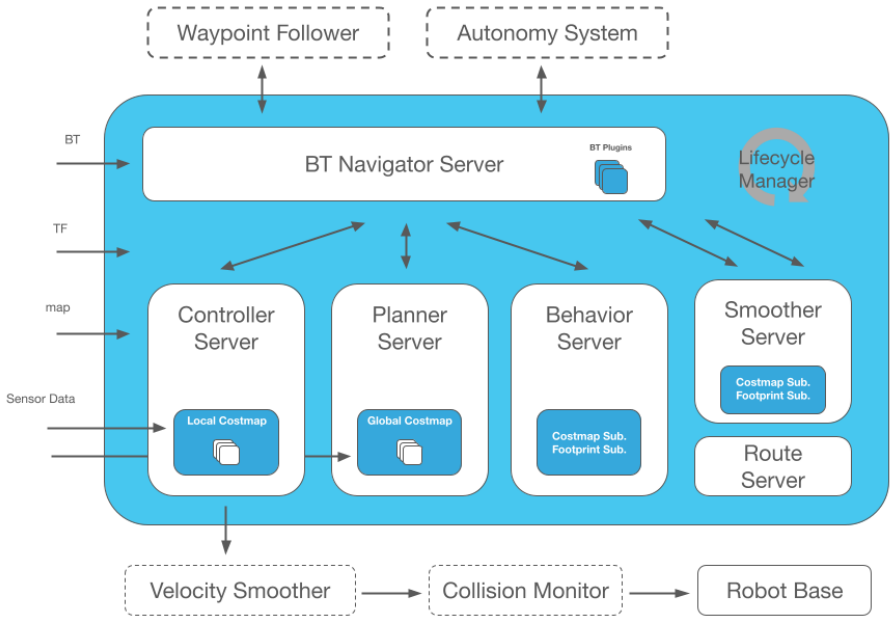

Fig. 3. Navigation system architecture [3]

According to this architecture, these input data feed into several specialized servers:

- BT Navigation Server (this server represents the system responsible for decision-making related to BT);
- Controller Server (this server, based on the data provided by sensors and information received from the BT Navigation Server, generates the output data that will be transmitted to the next module for robot control);
- Planner Server (implements the server to handle planner requests for stack and hosts a map of plugin implementations. An objective and a planner plugin name will be required to use and call the corresponding plugin to compute a path to the objective);
- Behavior Server (implements the server to handle behavior requests for recovery and hosts a vector of plugins that implement various C++ behaviors. It is also possible to implement independent behavior servers for each custom behavior, but this server allows multiple behaviors to share resources such as cost maps and TF buffers to reduce incremental costs for new behaviors);
- Smoother Server (the Smoother server implements the server to handle smooth path requests and hosts a vector of plugins that implement various C++ smoothing systems. The server exposes an action interface for smoothing with multiple smoothing devices that share resources such as cost maps and TF buffers).

These servers send commands or information to specific nodes or exchange commands and information with nodes to perform actions. These nodes are:

- Waypoint Follower (The Waypoint Follower module implements a way to track waypoints. It takes a set of ordered reference points to follow and then attempts to navigate to them in order. It also hosts a reference point task execution plugin that can be used to perform custom behavior at a reference point, such as waiting for user instructions, or picking up a

box. If a reference point is not reached, the stop_on_failure parameter will determine whether to continue to the next point or stop);

- Autonomy System (represents the CPU responsible for running the program);
- Velocity Smoother (is a package that contains a component node of the lifecycle for smoothing speeds sent from Nav2 to robot controllers. The purpose of this package is to implement velocity, acceleration, and deadband smoothing from Nav2 to reduce wear on robot motors and hardware controllers by smoothing accelerations/motions that might be present with control efforts from local trajectory planners);
- Collision Monitor (The Collision Monitor is a node that provides an additional level of robot safety. It performs several tasks related to collision avoidance using data received from sensors, bypassing cost maps and trajectory planners to monitor and prevent potential collisions at the emergency stop level);
- Robot Base (represents the physical structure of the robot, along with the processor, motor drivers, and other components).

4. Implementation and Testing of the Algorithm

The implementation and testing of the algorithm were carried out based on a customized hardware and software architecture consisting of a computer with Ubuntu 22.04.2 LTS operating system. To implement and test the algorithm, a specific working environment called Workspace was defined, which contains a folder with a predefined structure compatible with ROS2. Within this working environment, specific nodes were installed, documented, analyzed, compiled, and progressively tested, including motor control nodes for the AMR, encoder data reading nodes, IMU9250 inertial sensor data reading nodes, and encoder nodes integrated in Andy Mark 2964 NeveRest 40 motors.

Fig. 4. First simulation in Webots

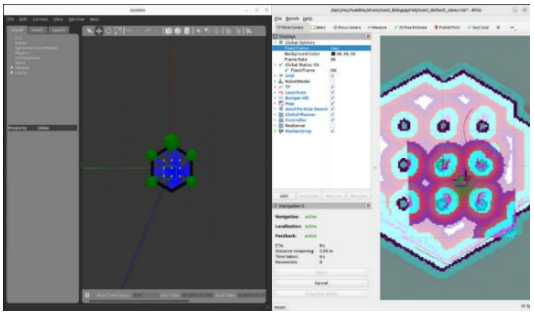

Fig. 5. a

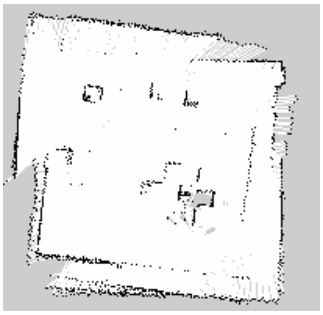

Fig. 5. b

Figure 4 shows the first simulation in Webots after compiling the ROS2 framework and the Webots module. The simulation represents a virtual space in Webots, which is rectangular in shape and enclosed by exterior walls. Within space, four cuboidal obstacles are defined, and the simulation with a predefined robot model, e-puck, needs to avoid them during its movement. After the first simulation, the next step was to define the individual robot model.

These activities are performed and tested beforehand in Webots and Gazebo simulations to address any potential issues, verify the application's compliance with the intended results, and ensure the compatibility of the robot model defined in the URDF file with the physical robot.

The development of the algorithm involves the following stages:

- Defining the robot's map: Using mapping sensors such as LiDAR, a map of the environment in which the robot is intended to navigate can be obtained. This map will be used by the Nav2 module to plan routes and avoid obstacles.
- Defining the robot's initial position: The robot needs to know its initial position to navigate successfully. This can be achieved using a localization system, specifically

SLAM (Simultaneous Localization and Mapping). This module represents an automatic system for localization and map creation, using landmarks and provided data.

- Configuring navigation parameters: These parameters can be set and edited through a YAML configuration file. The YAML file represents a code that can be written using the Python language.
- Programming the robot's behavior: Using Nav2, the robot's behavior can be programmed to navigate autonomously in the environment. This can include route planning, obstacle avoidance, and parking maneuvers. The programming is done in Python language.
- Testing and debugging: Testing involves navigating the robot in the defined environment in Webots and Gazebo simulations, simulating actions and commands that may occur, to verify its compliance and make necessary modifications.

In the ROS2 design of the navigation simulation environment, the following objectives were considered:

- Defining a ROS2 Workspace compatible with Webots specific to the AMR in question.
- Defining TF2 (tf2 is the transformation library, which allows the user to keep track of multiple coordinate frames over time. tf2 maintains the relationship between coordinate frames in a time-stamped tree structure and enables the user to transform points, vectors, etc., between any two coordinate frames at any desired point in time);
- Defining URDF (Unified Robot Description Format), which represents the file that defines the geometries and physical organization of the robot, written in the XML language;
- Defining navigation sensors for robot odometry, specifically the 9-axis IMU sensor MPU9250 and the encoders integrated in the existing Andy Mark 2964 NeveRest 40 motors;
- Configuring Robot's Footprint in ROS2, so that the navigation module can create routes and avoid existing obstacles, aiming to prevent possible collisions of the robot with obstacles;
- Configuring and simulating navigation, aiming for its subsequent implementation on the physical robot without direct initial testing on it.

Figure 5a shows the first simulation after compiling Gazebo and Nav2. The simulation presents two views. The first one is a 3D view that includes the movement area, obstacles, the test robot, and a visualization of the distance sensors. The second view represents the map known and updated by the robot, with more intense coloring in the updated area. Simultaneously, commands for robot movement can be given in the second part.

Figure 5b represents a map generated by Nav2 in another simulation. In that simulation, a different movement space similar to the one in Figure 4 was used. In that simulation, the robot had no prior information about the map, and it was defined from scratch.

After confirming its proper functioning based on tutorials, tests, and simulations, the development of the specific model for the AMR robot progressed further. It started with the development of the URDF file. As a result, a robot model with two wheels driven by motors at the back and a free wheel at the front, with the initial IMU sensor integrated into it, was created.

Figure 6 shows the first simulation of a customized robot model in Gazebo. This robot has two wheels driven by motors at the back, and in the front, there is a single spherical wheel for movement. The initial sensor is integrated into the robot's body. This model is intended to be modified and improved by:

- Replacing the spherical wheel with two sets of free wheels similar to the ones at the back;
- Adding distance sensors in the front of the robot for obstacle avoidance;

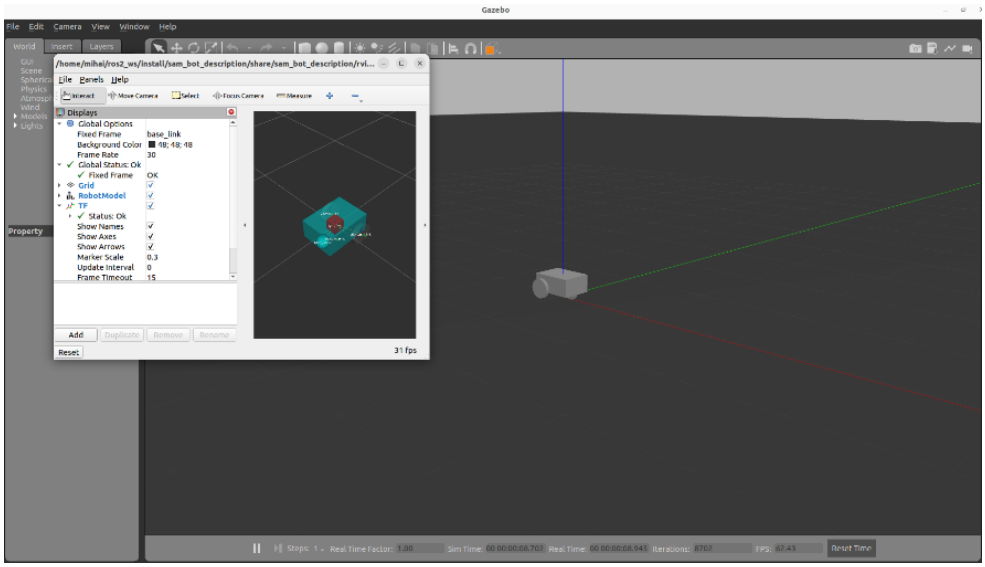

Fig. 6. The first simulation of a customized robot model in Gazebo

5. Conclusions

The following stages have been completed in this study: installation of the ROS2 framework and the Nav2 navigation module on a virtual machine required for simulation, verification, compilation, and testing of the codes, creation of the URDF file for the customized AMR model, simulation of the customized AMR model.

The main achievements are as follows: development of a functional robot model in the ROS2, framework, accurate simulation of the model in Webots and Gazebo, interconnection of specific nodes.

The main challenges encountered were, installation of the framework and modules, writing the URDF file to be correctly understood by ROS2 and other modules, proper testing of nodes and constructive codes, interconnecting the elements.

The following directions for future development are: implementation of the Nav2 module, virtual environment testing of the system with Nav2, implementation and testing of the code on the AMR predefined hardware architecture, testing the proper functioning of the physical model.

6. Bibliography

[1]. https://docs.ros.org/en/humble/
[2]. https://www.cyberbotics.com/#cyberbotics
[3]. Getting Started — Navigation 2 1.0.0 documentation (ros.org)
[4]. https://gazebosim.org/home
[5]. Course: Logistics Network Management, Instructor Conf.dr.ing. Bogdan Felician ABAZA
[6]. S. Macenski, A. Soragna, M. Carroll, Z. Ge, “Impact of ROS 2 Node Composition in Robotic Systems”, IEEE Robotics and Autonomous Letters (RA-L), 2023.
[7]. S. Macenski, T. Foote, B. Gerkey, C. Lalancette, W. Woodall, “Robot Operating System 2: Design, architecture, and uses in the wild,” Science Robotics vol. 7, May 2022.