DOI: https://doi.org/10.25140/2411-5363-2025-3(41)-101-108
UDC 621.8

***Marek Malik*[1]*, Rudolf Jánoš*[2]**

[1]PhD student of the Department of Production Systems and Robotics
Technical University of Košice (Košice, Slovakia)
E-mail: marek.malik@tuke.sk

[2]Associate Professor, Associate Professor of the Department of Production Systems and Robotics
Technical University of Košice, (Košice, Slovakia)
E-mail: rudolf.janos@tuke.sk. **ORCID:** https://orcid.org/0000-0002-5754-9278
ResearcherID: AAH-5449-2019. **Scopus Author ID:** 55016528600

ROBOT OPERATING SYSTEM, MIGRATION, GAZEBO, ROS1, ROS2

*This article describes the migration of software packages from the Robot Operating System 1 (ROS 1) environment to the new ROS2 architecture, using the JetAuto mobile platform as a case study. The work focuses on the practical migration procedure, identifies problems and their solutions. The **j**etauto_interfaces and jetauto_driver packages were sequentially migrated and subsequently tested in the Gazebo simulator with ROS2 Humble. Several issues emerged during implementation, such as incorrect model loading in Gazebo, non-functional URIs in XACRO files, and improperly oriented camera images and point clouds. These errors were resolved by a combination of software tools (ros2 topic, image_rotate, tf2_tools) and modifications to URDF files. The result is a fully functional simulation of the JetAuto robot in ROS2, publishing correct sensor data and enabling basic control. In a broader context, the article reflects global trends in ROS2 adoption and highlights the importance of migration for both academic and industrial environments.*

Keywords: *Robot Operating System; migration; Gazebo; ROS1; ROS2.*
Fig.: 6. References: 10.

Relevance of the research. The Robot Operating System (ROS) has become the “de facto” standard for academic research and development in robotics. Introduced in 2010, ROS1 has found widespread adoption in academia and startups, providing a unified framework for communication between software components, sensor data processing, and actuator control. With the growing complexity of robotic applications, however, the limits of this architecture became apparent, particularly in the areas of deterministic communication, security, and scalability for large-scale multi-robot systems. These shortcomings led to the development of the second-generation ROS2, which introduced fundamental architectural changes. It uses the Data Distribution Service (DDS) as its communication core, supports Quality of Service (QoS) profiles, provides mechanisms for secure communication, and enables deployment in real-time environments [1]. These features make ROS 2 a suitable platform for both research and industrial applications. The importance of the transition to ROS 2 is further emphasized by the fact that official ROS 1 support ended in 2025. Companies such as OTTO Motors and LG Electronics are deploying ROS2 in autonomous mobile robots [2; 3], while academic institutions are integrating it into research on autonomous vehicles and service robots. In this context, migrating existing projects from ROS 1 to ROS 2 becomes a key challenge. It not only enables the preservation of older solutions but also allows leveraging the modern features of ROS 2. This issue is the subject of this work, which documents the migration of the JetAuto mobile platform from ROS 1 to ROS 2 and analyzes the identified problems and chosen solutions.

Problem statement. Migrating software packages from ROS1 to ROS2 is not a direct conversion. Although both follow the same principles of modularity and publish-subscribe communication, technical differences cause many complications. One of the most important differences lies in the build systems: packages developed in Catkin for ROS1 must be adapted to Colcon for ROS2 [4]. This requires rewriting CMakeLists.txt and package.xml, adjusting dependencies, versions, and libraries.

Another challenge is that many ROS1 packages have no official ROS2 ports. For JetAuto, both jetauto_interfaces and jetauto_driver existed only in ROS 1. Functions such as ros, Publisher and ros, Subscriber were replaced by rclcpp, Publisher and rclcpp, Subscription, and initialization methods had to be modified [5].

Simulation-level problems also arose. Initial attempts to spawn the robot in Gazebo loaded an incorrect model variant due to conditional inclusions in the jetauto.xacro file. Furthermore, Gazebo failed to load mesh files referenced via package://URIs, requiring manual path adjustments [4].

Analysis of recent research and publications. Scientific papers show that ROS2 offers clear advantages for distributed systems, but latency and performance depend on the DDS (Data Distribution Service) middleware used [6]. Case studies (e.g., Autoware) demonstrate that migrating complex systems requires a modular approach and the use of the ros1_bridge [7]. Practical experience confirms that the most common problems include errors in TF graphs and incorrect QoS (Quality of Service) profile settings, which lead to unreliable message delivery [8]. These experiences align with the problems we encountered in our work – incorrect transformations between camera frames and the point cloud [9].

Sub-objectives of the Research. The primary goal of this work was to design and validate a methodology for migrating ROS1 packages to ROS2 using JetAuto as a test platform. The objectives were not only technical but also methodological, emphasizing that migration requires systematic debugging and validation.

1. **Migration of communication interfaces.** The first objective was to migrate the **jetauto_interfaces** package, which defines messages and services. This step was essential for building the communication layer upon which all other components depend.
2. **Driver migration.** The second objective was the migration of the **jetauto_driver** package, responsible for hardware communication. Porting it to ROS2 required extensive API modifications and represented the most complex part of the process.
3. **Simulation in Gazebo.** The third objective was to launch the robot in the Gazebo simulator as a test environment independent of real hardware. This allowed the identification of errors in model loading, transformations, and sensor data publishing.
4. **Documentation and recommendations.** The final objective was to document the entire migration process, analyze errors, and present solutions applicable to other developers and researchers. Thus, the work provides not only technical results but also methodological contributions in the form of best practices.

Research objective. The primary goal of this work was to migrate selected packages of the JetAuto mobile platform from the ROS1 environment to ROS2 and verify their functionality in simulation. Particular attention was paid to maintaining compatibility with existing robot functions, testing the correctness of sensor data publishing, and identifying the problems that the migration entailed.

The statement of basic materials. For the software platform, ROS2 Humble Hawksbill was chosen. This distribution was selected due to its long-term support (LTS), which is guaranteed until 2027. In addition to stability, Humble also offers broad support for the Gazebo simulator, SLAM Toolbox, and the Nav2 navigation stack, which are key components for the development of autonomous mobile robots [10]. Humble is also the officially recommended distribution for systems with Ubuntu 22.04, which was the test platform for our experiment.

Testing was conducted on an NVIDIA Jetson Orin 4 GB computer unit, chosen for its optimization for robotic applications and hardware acceleration support [11]. This unit was used to compile and run ROS2 packages, providing a realistic environment for practical deployment.

The JetAuto mobile platform is equipped with several sensors that were directly included in the experiments:

- Slamtec A1 LIDAR, providing 2D laser scans for localization and mapping.
- Orbbec FHD-1080P Astra Pro Plus camera, allowing for image capture and point cloud generation for 3D perception of the environment.
- Four-wheel differential drive with motor units, allowing the testing of motion commands in both simulation and a real environment [12].

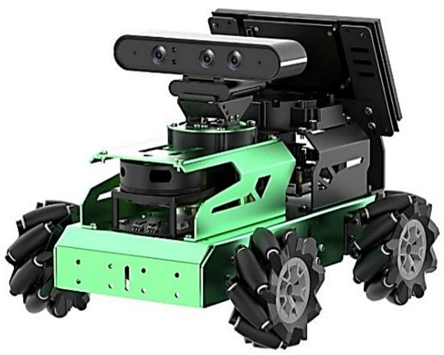
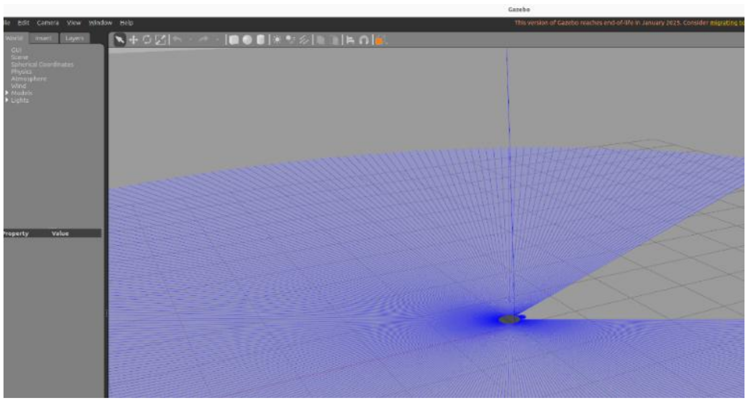


Fig. 1. Jetauto [12]

Work procedure. The migration of the JetAuto mobile platform from ROS1 to ROS2 was carried out step by step. Each stage brought its own set of challenges, which were systematically identified, analyzed, and resolved. This section provides a comprehensive overview of the procedure, serving both as documentation of our work and as a methodological guide for future migrations.

Package migration. The main package to be migrated was identified as a first step. We started with the original JetAuto Simulation project for ROS1. The basic files from this package were gradually taken over, with modifications to CMakeLists.txt and package.xml to make them compatible with Colcon. During the modifications, we also encountered dependencies in the drivers. Some drivers had to be modified or replaced with alternative implementations for ROS 2. It was clear from this stage that migration would not be a mere formal conversion but a complex process requiring interventions in the source code. First, the jetauto_interfaces package, which contains message and service definitions, was migrated. Changes were necessary in CMakeLists.txt and package.xml. Next, the jetauto_driver package was ported. The biggest change was replacing ROS1 API with the ROS 2 API. Errors during the build process had to be resolved by adding dependencies.

First launch of the simulation in Gazebo. During the first attempts to spawn the robot, the wrong model variant was loaded. The problem lay in jetauto.xacro. The solution was to explicitly set the machine, lidar, and depth_camera parameters. Another problem was that Gazebo could not load mesh files via package://. The solution was to set use absolute paths. After successful package compilation, we made our first attempt at a Gazebo simulation. At first glance, the environment appeared empty, but upon closer analysis, we identified that at least some sensors were visualized – especially the LIDAR shown on Fig. 2. RViz2 was also launched, where responses from the basic topics could be displayed.



This version of Gazebo reaches end-of-life in January 2025. Consider migrating to

Fig. 2. Robot spawned with missing network files

Source: developed by the author.

URDF and mesh file modifications. The next step was to adapt the robot's definition in the URDF/XACRO files. It was found that several of the robot's properties, such as dimensions and sensors, needed to be redefined to match reality and be correctly interpreted in ROS2. In detecting the problem, we found that the robot did not have a path to the STL files that defined its 3D model. The paths were originally specified in the package://format, which was not recognized by Gazebo in the ROS 2 environment. The problem was solved by modifying the paths and setting the GAZEBO_MODEL_PATH environment variable. Another problem was that after the second launch, the robot spawned in the simulation with an offset of approximately 5 cm above the surface shown on Fig. 3. This error was related to the incorrect setting of the base link in the URDF and was corrected by modifying the offset in the base frame definition.

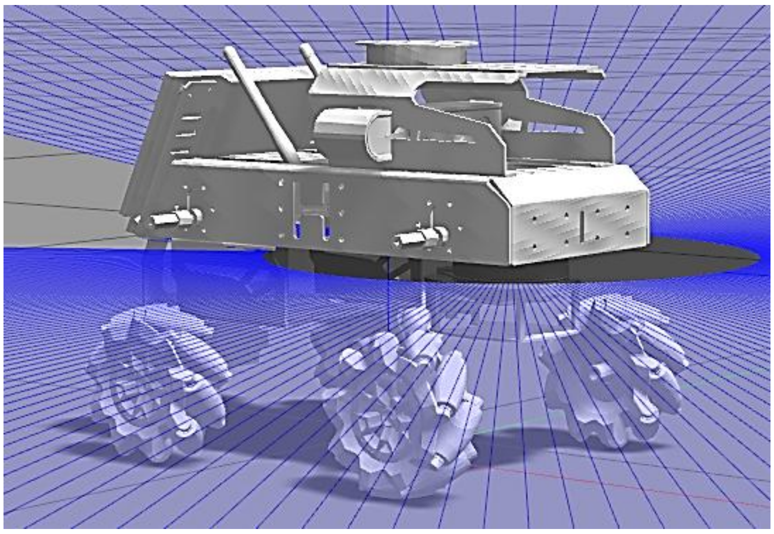


Fig. 3. Robot spawned with missing network files

Source: developed by the author.

Differences between JetAuto versions. Another challenge came from differences between the JetAuto Pro simulation package and the simpler JetAuto hardware configuration shown on Fig. 4. JetAuto Pro includes an arm and additional peripherals, while JetAuto does not. This mismatch caused inconsistencies in the URDF definitions. By exporting the settings and editing the URDF manually, the model was aligned with the actual JetAuto hardware.

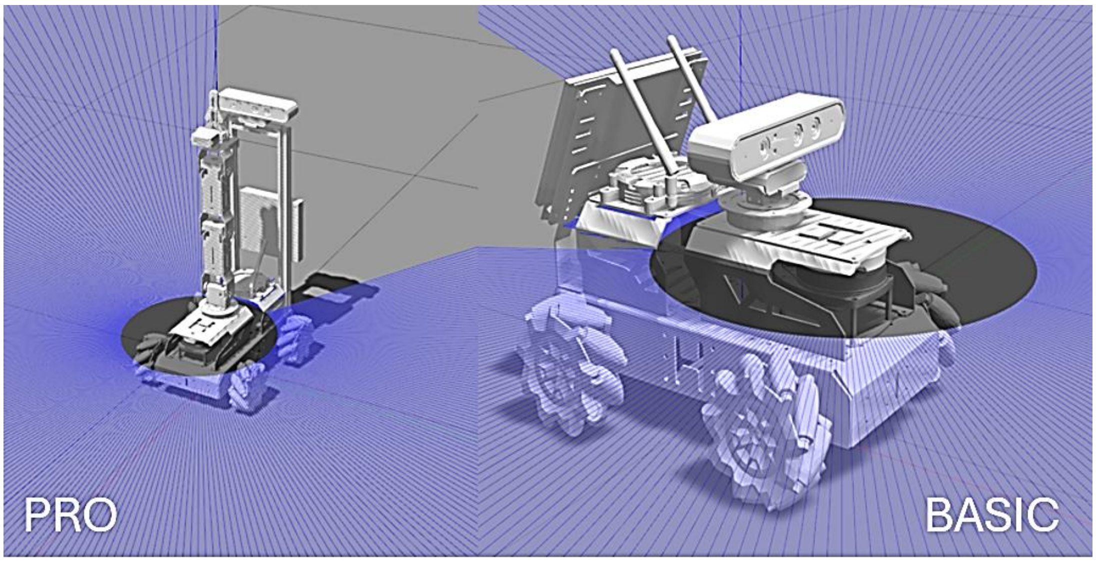


Fig. 4. Comparison of JetAuto Pro and JetAuto in simulation

Source: developed by the author.

Camera and point cloud orientation. One of the most complex issues encountered during the migration was the integration of the depth camera. After replacing the unsupported ROS1 plugin with its ROS2 equivalent, the camera image appeared in RViz2 but was consistently rotated by 180° shown on Figure 5. At first glance, this seemed to be a simple orientation issue that could be solved directly in the URDF description.

Several approaches were attempted:

URDF modifications.

The orientation of the camera link was adjusted multiple times by modifying roll, pitch, and yaw parameters in the camera_link to camera_frame joint. Each adjustment, however, only shifted the problem without solving it: the point cloud remained misaligned relative to the robot's base frame.

Joint type modification.

A second attempt involved changing the definition of joint 2, originally set as a revolute joint, to a fixed joint. This prevented the camera from being virtually "rotated" in simulation. Despite this change, the problem persisted because the underlying TF chain still produced a double rotation between camera_link and camera_frame.

Data validation.

During debugging, the published TF frames were analyzed (ros2 run tf2_tools view_frames). It became evident that the camera frame transformations were inconsistent: even when the URDF was corrected, the resulting visualization showed an image rotated by 180° and a point cloud flipped relative to the environment.

Since structural changes to the URDF did not produce the expected outcome, a software-level solution was chosen. The image_rotate node was introduced into the processing pipeline. This node subscribes to the original camera topic and republishes the image data with corrected orientation.

To ensure consistency between 2D images and 3D point clouds, a new topic called depth_points_optic was created. This topic programmatically rotated the point cloud data derived from the camera, aligning it with the optical frame expected by RViz2 shown on Fig. 6.

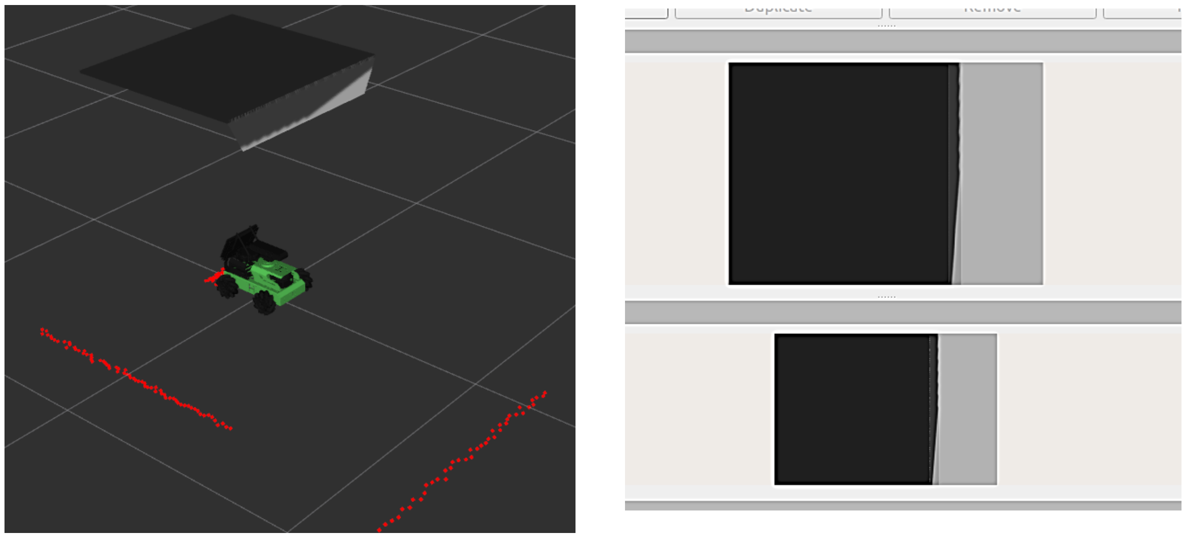


Fig. 5. RViz2 screenshot showing the incorrectly rotated camera image

Source: developed by the author.

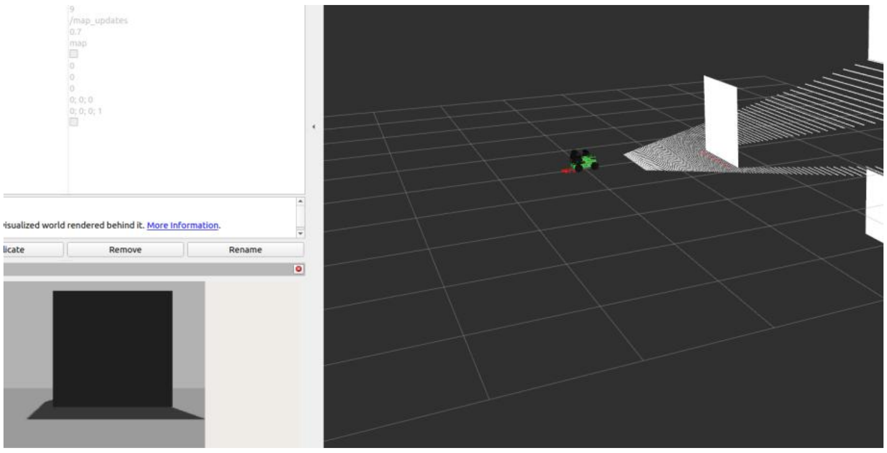


Fig. 6. Correctly oriented image after applying image_rotate and republishing through the new optic topic

Source: developed by the author.

The result was a correctly oriented camera view and point cloud, enabling proper visualization and further use in mapping and navigation.

Final validation. As a final validation step within the procedure, the ROS 2 TF tree was generated and analyzed. The TF tree confirmed that all expected frames–including the base frame, odometry, wheel joints, LIDAR, IMU, and camera frames–were present and connected in a coherent hierarchical structure shown on Fig. 7. This step was crucial to ensure that all transformations were consistent before performing mapping experiments.

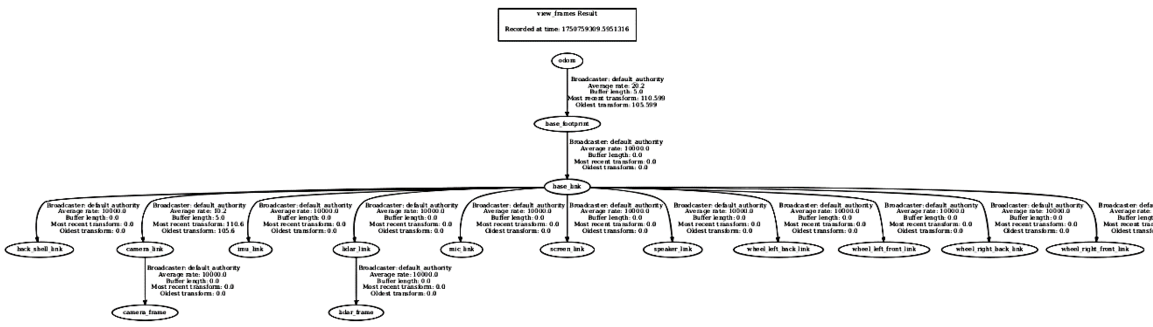

view_frames Result
Recorded at time: 1750759309.5951316
odom
Broadcaster: default_authority Average rate: 20.2 Buffer length: 5.0 Most recent transform: 110.599 Oldest transform: 105.599
base_footprint
Broadcaster: default_authority Average rate: 10000.0 Buffer length: 0.0 Most recent transform: 0.0 Oldest transform: 0.0
base_link
Broadcaster: default_authority Average rate: 10.2 Buffer length: 5.0 Most recent transform: 110.6 Oldest transform: 105.6
back_shell_link
camera_link
imu_link
lidar_link
mic_link
screen_link
speaker_link
wheel_left_back_link
wheel_left_front_link
wheel_right_back_link
wheel_right_front_link
camera_frame
lidar_frame

Fig. 7. TF tree of the JetAuto platform in ROS 2 after migration

Source: developed by the author.

Result. The migration process resulted in a fully functional simulation of the JetAuto mobile platform in ROS2. The **jetauto_interfaces** and **jetauto_driver** packages were successfully ported and compiled without errors. The simulation in Gazebo confirmed that the robot spawned correctly after URDF and mesh path modifications. The LIDAR provided accurate scan data, visualized in RViz2.

The differences between JetAuto Pro and JetAuto were resolved, leading to a model that correctly represented the real hardware. The most critical issue, the camera orientation, was successfully addressed using the **image_rotate** package and the introduction of a new topic for the corrected point cloud. This provided a reliable image feed and accurate 3D perception of the environment.

Finally, the TF tree confirmed that all robot components were properly linked, and transformations were correct. The successful SLAM mapping experiment further validated that the migrated packages, sensors, and transformations worked together seamlessly. The generated occupancy grid maps showed clear boundaries and obstacles, confirming the correctness of LIDAR scans and overall system integration.

Conclusions. This work demonstrated the successful migration of the JetAuto mobile platform from the ROS1 environment to ROS2. The migration covered two essential software packages – jetauto_interfaces and jetauto_driver – which were ported, restructured, and validated in simulation. The main technical challenges included adapting the build system from Catkin to Colcon, resolving API differences between ROS1 and ROS2, correcting URDF/XACRO inconsistencies, and ensuring proper integration of sensors in Gazebo.

The most critical issue, the incorrect orientation of the depth camera and its point cloud was solved through a combined hardware–software approach, including URDF modifications and the use of the *image_rotate* node. Similarly, problems with mesh paths and model variants were eliminated by explicit parameterization and environment variable adjustments. Final validation using TF tree analysis and SLAM mapping confirmed that all robot components were correctly integrated and operational in ROS2.

Beyond the technical results, this study highlights that migration from ROS1 to ROS2 is not a direct conversion but a systematic process that requires iterative debugging and validation. The presented methodology and solutions provide practical guidance for researchers and developers facing similar tasks. In a broader context, the work confirms the necessity of migration to ROS2, which is becoming the global standard in both academic and industrial robotics. Future research will extend this work by integrating higher-level navigation (Nav2), SLAM optimization, and real-world validation on the physical JetAuto platform.

Acknowledgements

This research was supported by project VEGA: 1/0215/23 Research and development of robotic workplaces equipped with industrial and collaborative robots.

References

1. Maruyama, Y., Kato, S., & Azumi, T. (2016). Exploring the performance of ROS2. In *Proceedings of the 13th International Conference on Embedded Software (EMSOFT)*. ACM. https://doi.org/10.1145/2968478.2968502.

2. eProsima. (2021, September 15). *eProsima and Clearpath (OTTO Motors) deploying ROS 2 at scale in production systems*. https://www.eprosima.com/news/eprosima-and-clearpath-deploying-ros-2-at-scale-in-production-systems.

3. Lee, Y., Kim, H., Park, J., & Choi, S. (2021). Development of ROS2-on-Yocto-based thin client robot and edge server interworking system. *Journal of Korea Robotics Society*. https://www.koreascience.or.kr/article/JAKO202109065716450.page.

4. Open Robotics. (2022). *Migrating from ROS 1 to ROS 2 – How-to guide*. In *ROS 2 Humble documentation*. https://docs.ros.org/en/humble/How-To-Guides/Migrating-from-ROS1.html.

5. Black Coffee Robotics. (2022). *ROS 1 to ROS 2 Migration: A Comprehensive Guide*. https://www.blackcoffeerobotics.com/blog/ros-1-to-ros-2-migration.

6. White, R., et al. (2020). *Evaluation of the ROS2 middleware for real-time robotic applications*. arXiv preprint arXiv:2005.02541. https://arxiv.org/abs/2005.02541.

7. Autoware Foundation. (2022). *Autoware ROS 2 migration notes*. https://autowarefoundation.gitlab.io/autoware.auto/AutowareAuto/architecture/migration/.

8. Ekumen Labs. (2022). *Migrating from ROS 1 to ROS 2: what you need to know*. https://ekumenlabs.com/blog/posts/migrating-from-ros1-to-ros2/.

9. ROS 2 image_pipeline package. (2022). *image_rotate Node Documentation*. https://github.com/ros-perception/image_pipeline/tree/ros2/image_rotate.

10. Open Robotics. (2022). *ROS 2 Humble Hawksbill documentation*. https://docs.ros.org/en/humble/.

11. NVIDIA. (2023). *Jetson Orin Nano developer kit* [Product page]. https://developer.nvidia.com/embedded/jetson-orin.

12. *Hiwonder JetAuto AI Robot Kit – NVIDIA Jetson-Powered ROS1/ROS2 Educat*. (n.d.). Hiwonder. https://www.hiwonder.com/products/jetauto?variant=41201592434775.

Отримано 19.09.2025

УДК 621.8

Марек Малік[1], Рудольф Янош[2]

[1]аспірант кафедри виробничих систем та робототехніки
E-mail: marek.malik@tuke.sk
[2]доцент, доцент кафедри виробничих систем та робототехніки
Кошицький технічний університет (Кошице, Словаччина)
E-mail: rudolf.janos@tuke.sk. **ORCID:** https://orcid.org/0000-0002-5754-9278
ResearcherID: AAH-5449-2019. **Scopus Author ID:** 55016528600

РОБОТОТЕХНІЧНА ОПЕРАЦІЙНА СИСТЕМА, МІГРАЦІЯ, GAZEBO, ROS1, ROS2

У цій статті описано міграцію програмних пакетів із середовища Robot Operating System 1 (ROS 1) до нової архітектури ROS2 на прикладі мобільної платформи JetAuto. Робота зосереджена на практичній процедурі міграції, визначає проблеми та їх вирішення. Пакети jetauto_interfaces і jetauto_driver були послідовно перенесені і згодом протестовані в симуляторі Gazebo з ROS2 Humble. Під час реалізації виникло кілька проблем, таких як неправильне завантаження моделі в Gazebo, нефункціональні URI в файлах XACRO, а також неправильно орієнтовані зображення камери та хмари точок. Ці помилки були вирішені за допомогою комбінації програмних інструментів (ros2 topic, image_rotate, tf2_tools) та модифікацій файлів URDF. Результатом є повністю функціональна симуляція робота JetAuto в ROS2, яка публікує правильні дані датчиків і забезпечує базове управління. У ширшому контексті стаття відображає глобальні тенденції у впровадженні ROS2 і підкреслює важливість міграції як для академічного, так і для промислового середовища.

Ключові слова: *робототехнічна операційна система; міграція; Gazebo; ROS1; ROS2.*

Рис.: 6. Бібл.: 10.