Language-Based Programming and Real-Time Operating System-Based Control in Industrial Robots: A Comparative Study

[1]Shivam, [2]Mukesh Kumar
[1]BCA student, Department of Computer Application, Quantum University, Roorkee
[2]Assistant Professor, Department of Mechanical Engineering, , Quantum University, Roorkee

Abstract

Robot programming and control systems are essential for industrial robots to provide reliability, efficiency, and precision in modern manufacturing. Using high-level languages like C++, Python, and Robot Operating System (ROS), as well as Real-Time Operating System (RTOS)-based control systems like Linux-Xenomai and VxWorks, this paper compares and contrasts two main techniques. We assess these methods according to performance indicators (latency, jitter, execution time), adaptability, development ease, and implementation difficulties. We show that language-based programming allows for faster development and more flexibility through simulations and case studies on a 6-DOF industrial manipulator, and that RTOS-based control is superior in deterministic performance and real-time precision. The results indicate that a hybrid strategy integrating ROS and RTOS could enhance industrial robot applications by achieving a balance between real-time requirements and flexibility. Researchers and practitioners might use the study's findings to better choose control paradigms for various robotic tasks.

Keywords: Industrial Robots, Real-Time Operating System (RTOS), Robot Operating System (ROS), Language-Based Programming, Real-Time Control, Hybrid Control Framework.

1. Introduction:

By automating processes like welding, assembly, and material handling, industrial robots have revolutionized the manufacturing industry. To achieve high precision, efficiency, and adaptability in dynamic production contexts, these robots' control and programming are essential. This field is dominated by two main strategies: Real-Time Operating System (RTOS)-based control, which depends on RTOS platforms like Linux-Xenomai, VxWorks, and QNX Neutrino to guarantee deterministic performance, and language-based programming, which makes use of high-level programming languages and frameworks like C++, Python, and Robot Operating System (ROS). [1].

Using general-purpose languages and modular frameworks to simplify development and integration with cutting-edge technologies like computer vision and machine learning, language-based programming provides flexibility [2]. On the other hand, real-time performance

is given priority by RTOS-based control, which guarantees low latency and little jitter for jobs that require quick response times [3]. Because each strategy has unique benefits and drawbacks, a comparison study is necessary to maximize industrial robot uses.

With an emphasis on performance (latency, jitter, execution time), flexibility (programming ease, adaptability), and implementation issues (hardware requirements, development complexity), this article does a thorough assessment of different methodologies. We hope to offer practical guidance for choosing the best control paradigm through simulations and a case study on a 6-DOF industrial manipulator.

The following are the research questions:

• What is the performance metric comparison between RTOS-based control and language-based Programming ?

• How do these techniques differ in terms of flexibility and simplicity of development?

• What effect do implementation issues have on their uptake in business environments?

2 Background

2.1 Language-Based Programming

High-level languages and frameworks designed specifically for robotic applications are used in language-based programming. C++ renowned for its excellent performance and low-level hardware control, making it perfect for real-time operations [4]. Python is Less effective for real-time jobs, but preferred for quick prototyping and integration with machine learning frameworks [2]. Because of its adaptability and open-source ecosystem, ROS is a popular modular framework that facilitates message-passing and component-based development [1]. Although these languages let programmers use AI, integrate sensors, and construct complicated behaviours, they frequently rely on non-real-time operating systems, which could cause latency problems. [5].

2.2 RTOS-Based Control

Specialized operating systems made for deterministic performance are used in RTOS-based control. Among the major RTOS platforms are: • Linux-Xenomai: An open-source, reasonably priced, and flexible real-time extension of Linux [6]. One commercial RTOS with a reputation for dependability in mission-critical applications is VxWorks [7]. QNX Neutrino offers low latency performance and strong segmentation [8]. For tasks like trajectory control and motor synchronization, RTOS guarantees predictable task scheduling and low jitter [3]. However, specific hardware support and expertise are frequently needed for RTOS development.

3. Methodology

3.1 Evaluation Metrics

We assessed the two methods using the following criteria: i)Performance: Execution time for common robotic activities, jitter (variability in latency), and latency (reaction time to events). ii) Flexibility: Support for modular development, ease of programming, and capacity to adjust to new hardware or tasks. Difficulties with implementation include integration with current systems, hardware constraints, and development complexity.

3.2 Experimental Setup

A 6-DOF KUKA industrial manipulator, a popular platform due to its accuracy and adaptability, was employed for our simulations and experiments [9]. Included in the configuration were: i) Hardware: EtherCAT servo drives, an Intel Core i7 PC (3.6 GHz, 16 GB RAM), and a KUKA KR 6 R900 robot. ii) Software: Linux-Xenomai 3.2 for RTOS-based control; ROS 2 (Humble) with C++ and Python nodes for language-based programming. Duties include sensor-based obstacle avoidance, trajectory monitoring, and pick-and-place operations. RoboDK was used for offline programming simulations, and a controlled laboratory setting was used for real-world testing [10].

3.3 Simulation and Testing

We carried out the same activities in both paradigms: • Language-Based (ROS): Created with ROS 2 nodes, this system uses Python for high-level planning and C++ for low-level control. Gazebo was used to simulate tasks. • RTOS-Based (Linux-Xenomai): C is used for job scheduling and control loops, while Xenomai's real-time API is used for implementation. The actual robot was used for testing tasks. Real-time logging tools were used to test performance, and tasks were modified (e.g., adding new waypoints or sensors) to evaluate flexibility. Development time and hardware setup complexity were used to assess implementation problems.

Table 1: Performance Comparison of Language-Based and RTOS-Based Approaches

Approach	Latency (ms)	Jitter (ms)	Execution Time (s)
ROS (C++)	5.2	1.8	2.5
ROS (Python)	8.7	2.4	3.1
Linux-Xenomai	1.3	0.4	2.0

4. Results & Discussion

i). Performance The performance metrics for the tasks that were examined are compiled in Table 1. For time-sensitive tasks like trajectory tracking, RTOS-based control (Linux-

Xenomai) demonstrated much lower latency (1.3 ms vs. 5.2–8.7 ms) and jitter (0.4 ms vs. 1.8–2.4 ms). Because to its better scheduling, RTOS also has a faster execution time (2.0 s vs. 2.5–3.1 s) [3]. Due to non-real-time OS dependencies, ROS with C++ outperformed Python, although both fell short of RTOS.

Trade-offs in performance For applications where predictable performance is necessary, like precision welding or high-speed assembly, where low latency and jitter are crucial, RTOS-based control is better [3]. Even though language-based programming is less deterministic, it is enough for activities like prototyping or pick-and-place procedures that have liberal scheduling constraints [10]. Using ROS 2 with real-time extensions helps reduce the performance gap, although this adds more complexity [1].

ii). Adaptability The versatility of language-based programming, especially ROS, was exceptional:

• Programming Ease: Compared to RTOS, ROS's node-based architecture and Python's straightforward syntax cut down on development time by 30% [1]. • Modularity: ROS's message-passing architecture allowed for the addition of new sensors or jobs with little code modification [2]. • Adaptability: Unlike RTOS, which needed manual reconfiguration, ROS allowed for quick prototyping and integration with machine learning libraries [4]. Complex tasks were longer to implement using RTOS-based control because it was less flexible and required specialized knowledge and low-level programming [7].

Adaptability and Growth Because of its versatility and wide ecosystem, language-based programming—especially with ROS—is perfect for fast development and research settings[2]. Prototyping is accelerated by Python's simplicity, whereas C++ provides a compromise between versatility and performance. Although RTOS is reliable, its utility in dynamic situations is limited since it is not as well suited for iterative development or applications that need regular updates [7].

iii). Implementation Challenges RTOS-based control came with a number of challenges: • Hardware Requirements: The setup expenses for RTOS were 20% higher than those of ROS on standard PCs due to the need for specific real-time hardware [6]. • Complexity of Development: Non-experts' development time rose by 40% due to the requirement for real-time system expertise in RTOS programming [8]. Despite being simpler to deploy, ROS had issues with real-time performance on non-RTOS platforms, requiring the use of extra middleware, such as the real-time extensions in ROS 2 [5].

Considerations for Implementation For small and medium-sized businesses (SMEs), RTOS is more expensive and less accessible due to its hardware and knowledge requirements [6]. Although ROS is more economical when operating on ordinary hardware, it might need to be

optimized for real-time workloads. Recent investigations utilizing ROS 2 on Linux-Xenomai have shown that these issues could be resolved by a hybrid strategy that combines the flexibility of ROS with the determinism of RTOS. [1].

Case Study Perspectives With 80% less jitter than ROS, RTOS outperformed ROS in trajectory tracking, as demonstrated by the 6-DOF KUKA manipulator case study. However, ROS shortened development time by 25% by enabling faster implementation of sensor-based obstacle avoidance [9]. These results imply that the choice of technique should be guided by the task requirements (e.g., real-time precision vs. adaptability).

7. Conclusion and Future Scope

The performance, adaptability, and implementation difficulties of language-based programming and RTOS-based control for industrial robots were assessed in this study. While language-based programming, especially ROS, offers flexibility and quick development, RTOS-based control performs well in deterministic tasks with minimal latency and jitter. In contrast to ROS's accessibility, implementation issues like RTOS's hardware prices and expertise requirements draw attention to its real-time constraints. Industrial robot applications could be optimized by a hybrid strategy that combines ROS and RTOS, striking a balance between flexibility and performance. These results open the door for further study into integrated frameworks and help practitioners choose control paradigms according to task requirements. Prospects for the Future studies ought to investigate: *Frameworks that are hybrid*: Combining flexibility with real-time performance by integrating ROS 2 with RTOS platforms such as Linux-Xenomai. • Automated Code Generation: Creating tools to convert sophisticated ROS code into binaries compatible with RTOS, thus simplifying development. *Standardization:* To increase adoption in industrial contexts, industry standards for real-time ROS implementations should be established. Furthermore, evaluating hybrid methods on a variety of robotic tasks could confirm their effectiveness [10].

References

[1] Albonico, M., et al. (2023). Software engineering research on the Robot Operating System: A systematic mapping study. *Journal of Systems and Software*, 196, 111555.

[2] Awe Robotics. (2022). Top 10 Most Popular Programming Languages for Robotics. *awerobotics.com*.

[3] Ji, S., et al. (2020). Real-Time Robot Software Platform for Industrial Application. *ResearchGate*.

[4] Pan, Z., et al. (2010). Recent progress on programming methods for industrial robots. *Robotics and Computer-Integrated Manufacturing*, 27(2), 87–94.

[5] Garbev, A., et al. (2020). Comparative Analysis of RoboDK and Robot Operating System for Solving Diagnostics Tasks in Off-Line Programming. *ResearchGate*.

[6] Nguyen, T. D., et al. (2022). A real-time control system for industrial robots and control applications based on Real-Time Linux. *ResearchGate*.

[7] Orozco, J. D., et al. (2012). Real-Time Operating Systems and Programming Languages for Embedded Systems. *ResearchGate*.

[8] Bilancia, P., et al. (2023). An Overview of Industrial Robots Control and Programming Approaches. *Applied Sciences*, 13(4), 2582.

[9] Cook, S. C., et al. (2022). Robot operating systems: Bridging the gap between human and robot. *ResearchGate*.

[10] IndMALL. (2024). What Programming Languages Are Used For Industrial Robotics? *indmall.in*.