DOI: https://doi.org/10.15276/eltecs.42.118.2025.1
UDC 621.316.13

ROS-based manipulator robot control system

O. Streltsov, Y. Ornovetskyi, M. Katrichenko, V. Pozdnyakov, S. Konovalov
Odessa Polytechnic National University

Abstract. *In order to realize the control of the robot manipulator, the control system of the robot manipulator was studied. On the premise of ensuring the robustness and real-time of the system, it is proposed to build a control system based on Ubuntu system combined with Robot Operating System (ROS), and use Controller Area Network (CAN) communication to build a robot manipulator. Finally, through the simulation experiment and the physical robot control experiment, the application effect of the robot manipulator control system is verified. The experimental results show that the robot manipulator control system has the basic working ability to control the coordinated robot for path planning, and can well establish the communication between the upper and lower machines and control the robot manipulator. At the same time, the control system has the characteristics of modularity, high portability, clear frame, and low delay.*

Keywords: *robot manipulator, ROS system, control system design, algorithm RRT, communication interface.*

Introduction

The growing popularity of robotics and the increasing demands for precision in robotic operations have driven the development of new hardware and software solutions for control systems. The proliferation of diverse robot types has led to the emergence of a wide range of control systems, each tailored to specific robot categories.

Despite the active efforts of numerous companies engaged in research and development within the robotics field, existing control systems are predominantly designed for individual robot types and lack universality, which limits their applicability in the broader market. This limitation hinders the advancement of universal control systems that could meet market demands. Consequently, the development of a universal control system capable of supporting the majority of modern robots has become an urgent market need.

In the development of control systems for modern robots, several development platforms can be considered, including Microsoft Visual, Matlab, MRDS, among others. Microsoft Visual [1] is well-suited for development within the Windows environment and offers a comprehensive set of tools, though the development process can be complex and prone to errors. Matlab [2] is primarily used for simulation experiments, but it presents challenges when implementing control for physical robots.

Since these software platforms encapsulate certain algorithms, such as OpenCV, OMPL, and others, it is essential to identify a development platform that is compatible with various algorithms and is easy to use across multiple platforms.

To achieve this, a control system platform was developed based on the Robot Operating System (ROS) [3], and corresponding experiments were conducted for both simulation and physical control.

1. Configuring the ROS Environment

The Robot Operating System (ROS) is a framework for developing robotic control systems. It includes features such as 3D reconstruction, dynamic modeling, real-time environment simulation, physical control, and other functionalities that enhance the development requirements for robots [4]. ROS is available for all major operating systems, with Ubuntu being one of the most advanced in terms of ROS support. ROS does not handle system process management; instead, it provides various functional packages and establishes communication between them through a peer-to-peer mode to create a control platform for robot simulation and physical control. This framework introduces key concepts such as "node," "topic," "service," and "message." As illustrated in Fig. 1, these core concepts are used to deliver data to the control system through their interactions.

1.1 . Node

Nodes in ROS represent individual processes that interact with each other through topics and services,

contributing to the system's efficiency and maintainability. Each node is assigned a unique name within the system, preventing name duplication and ensuring the uniqueness of interactions between nodes.

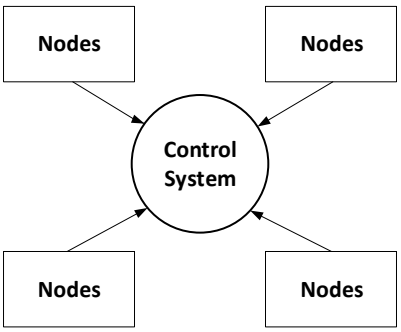


Fig. 1. ROS management structure

1.2 .Topic

A topic in ROS serves as the primary bus for data exchange between nodes. The publisher, who sends the data, and the subscriber, who receives it, do not need to be aware of each other's existence.

A topic can support multiple subscribers and multiple publishers; however, to avoid errors, different nodes must publish data under the same topics.

1.3 . Service

Services in ROS are used for direct interaction with nodes and for obtaining feedback from them. Each service consists of two messages: one that makes a request and another that processes and responds to it. Services are user-defined, stored in the srv directory, and are converted into source code during compilation.

1.4 . Message

Messages in ROS are used to transmit data between nodes by publishing them in predefined topics. All data in ROS can be represented as messages, which have a fixed type and structure, defined as the package name/.msg file name. The management of all processes in ROS can be implemented using nodes, topics, services, and messages.

2. Research methods and progress

2.1. The control system algorithm

Controlling robotic manipulators requires high-precision control over joints to ensure smooth motion, which in turn necessitates the use of specialized control algorithms. In this study, the primary algorithm used is robot trajectory planning. The system is designed based on the input of initial and final positions to guide the robot's movement. Given these characteristics, the control system is built upon the Rapidly-exploring Random Tree (RRT) algorithm [5].

The RRT algorithm is specifically developed for real-time trajectory planning of non-linear mechanical systems. Unlike other well-known sampling methods, whose effectiveness largely depends on numerous tunable parameters, RRT demonstrates versatility and is the most suitable approach for solving a wide range of motion planning problems.

RRT employs a discrete representation of the configuration space in the form of a tree, where the root node corresponds to the object's initial position. This tree expands in such a way that it gradually covers the entire permissible configuration space, thereby increasing the accuracy of the constructed path network.

At the core of the RRT algorithm is a random tree used to detect collisions of sample points in the motion space by randomly growing branches from the root node toward the target. This algorithm does not require 3D modeling but effectively solves path planning tasks in a three-dimensional space with multiple constraints. Due to its high efficiency, accuracy, and simplicity, RRT is well-suited for controlling the motion of robots with a large number of degrees of freedom.

The RRT algorithm takes the initial position as the root node and generates child nodes through random sampling, resulting in a tree that grows randomly. When this tree reaches the target node, a path from the root to the target node is constructed, thereby facilitating the robot's trajectory planning. The following algorithm outlines the process of robot trajectory planning using the RRT method:

Function RRTPlan: BOOL(env :environment, T :RRTTree, S goal: node)

1. *Var S target, S nearest, S new, node*

Initialization of variables: Variables *S target*, *S nearest* і *S new* which represent the nodes of the RRT tree, are declared, які представляють вузли дерева *RRT*.

2. *While(search time/space remaining) do*

Search cycle: Within the search time or space, the main cycle of the algorithm is performed.

3. *S target = ChooseTarget (S goal)*

Select target: The *ChooseTarget* function selects a target node *S target*, which can be a random point or a final target *S goal*.

4. *S nearest = Nearest (T , S target)*

Searching for the nearest node: The closest node of the tree *T* to the target node *S target* is found using the *Nearest* function.

5. *IF (Distance (S nearest, S goal) < Distance Threshold)*
6. *Return true*
Distance to target check: If the distance between the nearest node and the target node is less than a given threshold value, *true* is returned, indicating successful trajectory planning.
7. *S new = Extend (S nearest, S target)*
8. *IF (S new ≠ NULL) then*
9. *T.AddNODE(S new)*
10. *Return false*
Extending the tree: If the condition of steps 5.6 is not met, the tree is extended towards the target node using the *Extend* function. If a new node *S new* is created, it is added to tree *T*. If no new node is created, *false* is returned, indicating that no path was found.
11. *Var p : real*
12. *p = Random(0, 1.0)*
13. *IF o < p <aim then*
14. *Return goal*
15. *Else if aim< p 1<1.0 then*
16. *Return RandomNode();*
Random node selection: a random number *p* is generated. If it is in a certain range, the function returns the target node. Otherwise, a random node is selected.

An example of the RRT algorithm used in the movement of the manipulator is shown in Fig. 2.

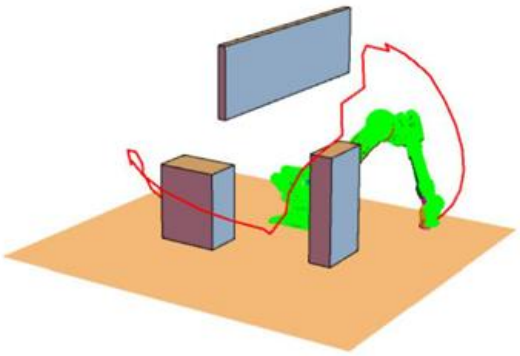


Fig. 2. Use of the RRT algorithm on the manipulator

To accelerate path planning, the random tree is optimized by initially calculating the midpoint between the starting and target positions in the space. The robot first moves toward this midpoint and then proceeds to the final position. This allows the random probability to determine whether the next point is the correct target or a random point. The parameter *aim* is predefined, and a random number *bp* is generated within the range of 0 to 1. When *0<bp<aim*, the random number directs movement toward the target point; when *aim<bp<1*, the random number facilitates growth in a random direction [6].

The RRT algorithm can be employed to compute and output the rotational speed and radians of each joint during the robot's movement, given the initial and target positions as input.

2.2. Communication interface design

The communication interface module is utilized to establish a connection between the main computer and the robotic manipulator. This module facilitates the exchange of control commands and feedback transmission between the two entities. Traditional ROS communication is based on ID connections, which are typically employed for a limited number of advanced robots, such as the UR series. However, Controller Area Network (CAN) communication is currently more widely adopted [7]. Thus, in this study, only the path planning algorithm and simulation environment in ROS are used, combined with CAN communication to develop a new communication method.

CAN communication is employed to transmit motor parameters—such as motor identifier, rotational angle, speed, etc.—calculated by the main computer to the control chips of each robot motor, allowing for direct control of the robot. This communication method does not rely on ROS communication and directly interfaces with the motors. This approach enhances existing characteristics of robustness and responsiveness while adding features such as simplicity in communication, broad applicability, and high portability, making it suitable for most robotic manipulators available on the market.

2.3. The main process of controlling a manipulator robot

The control process is divided into three parts: trajectory planning operations, communication connections, and motor control. Controlling a robotic manipulator essentially involves managing the rotational angles, speed, and switching time of the motors at each joint. Thus, the steps of the control system are as follows:
(1) Construct a 3D model of the robot, configure the ROS environment, and set the initial position of the end-effector.
(2) Launch the configured robot in the ROS 3D visualization environment (Rviz), input the end-effector coordinates via the editing interface, invoke the RRT algorithm to compute the necessary joint angles, velocities, accelerations, and positions required to reach the target position, and sequentially transmit the computed results.
(3) Establish communication between the main computer and the robotic manipulator using CAN communication. The calculated joint angles ando

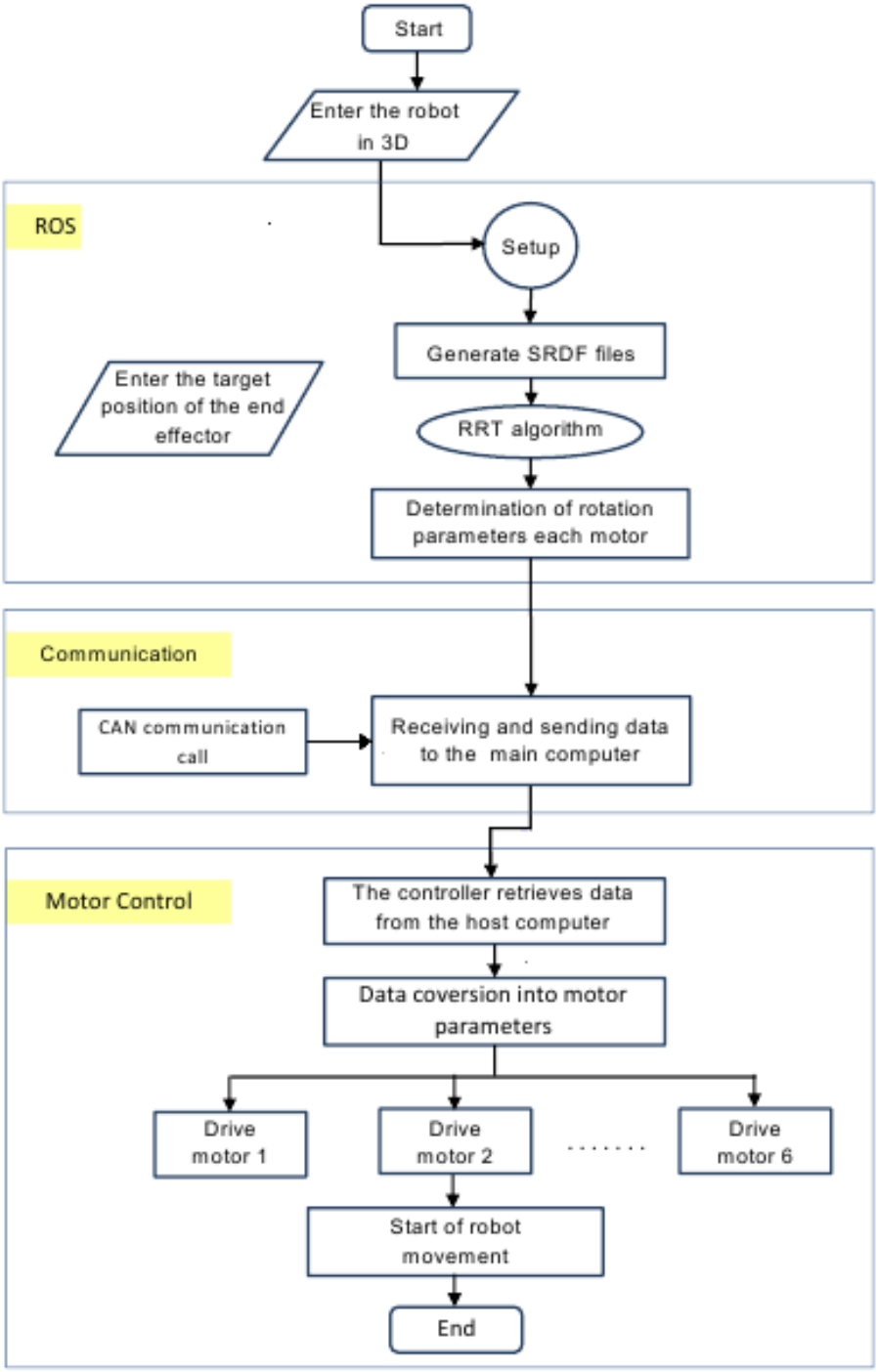


Fig. 3. Block diagram of the robot-manipulator control process

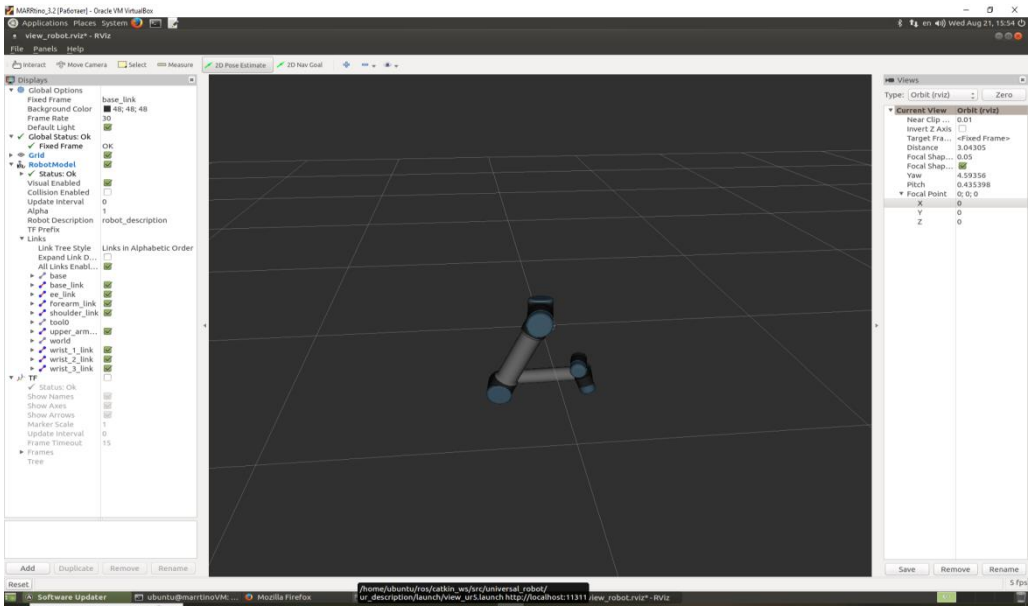
ther necessary parameters are sent to the robot through the communication interface.

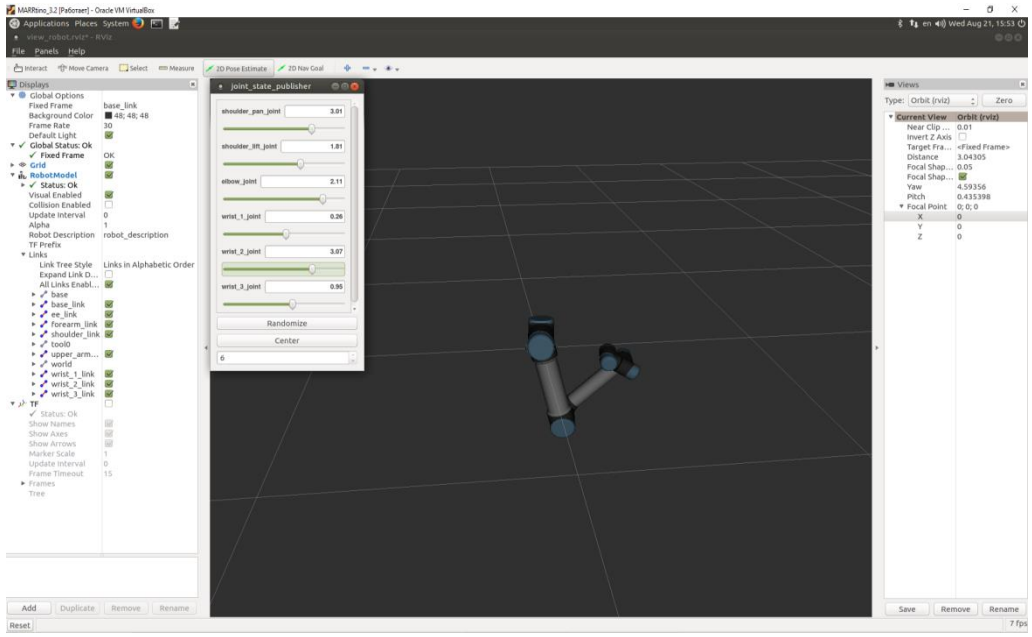
(4) The robot receives the data, converts it into motor motion parameters, and sends these parameters to the motors to control the robot's movement.

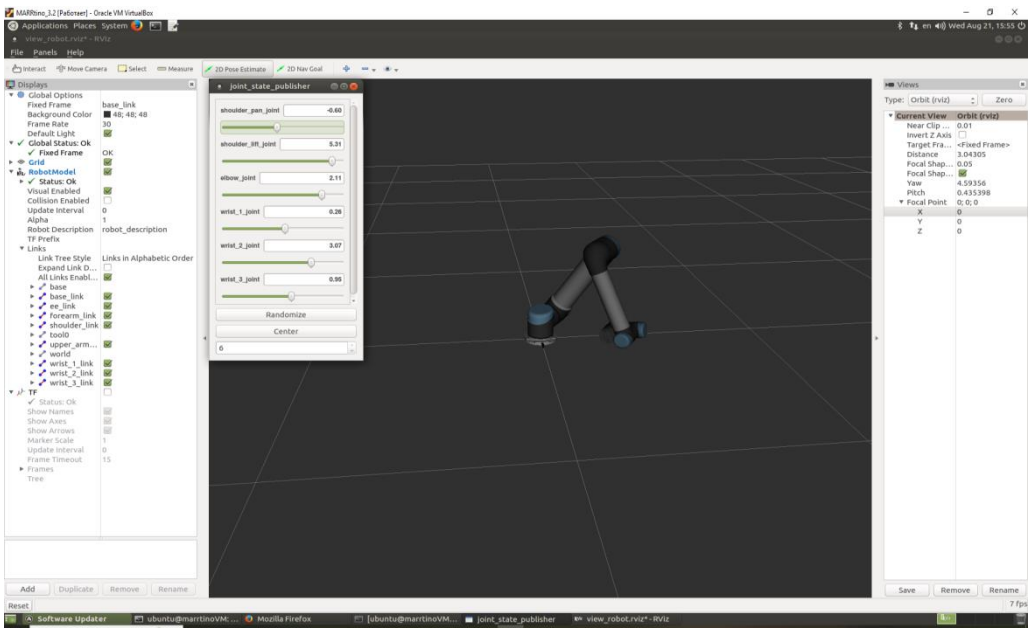
3. Experimental results and analysis

3.1 A simulation experiment

In the simulation, a control interface is created where the coordinates of the end-effector can be set and modified to control the simulated robot in Rviz. The initial position of the robot is observed horizontally through the simulation interface, while the target position can be adjusted by either dragging or entering the end-effector coordinates directly into the interface. First, the end-effector coordinates are reset to the initial position. Second, the coordinates of the end-effector are altered in the interface to direct the robot towards the target position. Target positions are assigned to move the robot in each direction, and an appropriate path is planned accordingly. The simulation is illustrated in Fig. 4.







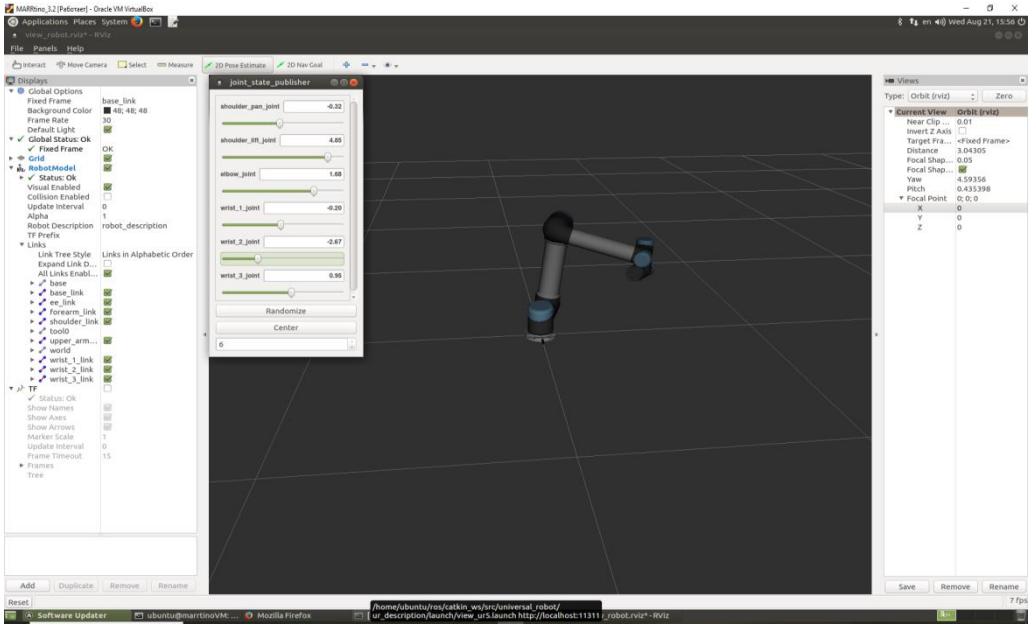


Fig. 4. Simulation of the robot control experiment

The simulation results demonstrate that by inputting the target coordinates of the end-effector, the RRT algorithm can be invoked within the simulation software to compute the rotation angle for each joint, which is then visualized in the simulation interface, thereby achieving control of the simulated robotic manipulator. The specific coordinates used in the experiment are provided in Table 1.

Table 1

Coordinates of the input end effector

Position points	The specified coordinates
Initial state	(0, 0, 0)
1	(0.3, 0, 0)
2	(0.3, -0.3, 0)
3	(0.3, -0.3, 0.3)

3.2 An experiment in controlling a manipulator robot

The UR5 robotic manipulator [8] was used as the physical robot for the experiment. The data was transmitted from the computer interface to a microcontroller located within the robot. The microcontroller processes the received data and converts it into parameters recognizable by the motor, thus controlling the speed and timing of motor rotations. Upon receiving a command from the main computer, the motor controller executes the corresponding action, completing the path planning, and the robot performs the intended motion.

Due to hardware inaccuracies and deviations in the ROS algorithm, the motion parameters sent were calibrated within the control system. The robot moved at a constant speed, with the absolute positions of the joints being read. The results indicated that the robot did not reach the specified position within a 2 mm margin. After adjusting the control parameters and refining the software, the robot's spatial movement could be precisely controlled via the main computer interface, ensuring high reliability and real-time performance.

By utilizing the ROS print function, detailed information regarding the speed, time, and acceleration for each joint of the robot can be effectively displayed on the output screen. This functionality allows for monitoring the dynamics of the robot's motion in real-time, ensuring that the movements align with the desired trajectory. Table 2 presents the time required for the robot to reach its specified positions after entering the end-effector coordinates provided in Table 1. These output times are critical for evaluating the performance and responsiveness of the system. The recorded times illustrate that the robot's motion is characterized by smooth transitions, exhibiting excellent stability and consistent real-time performance, which is essential for precise and efficient operation in dynamic environments.

Table 2

The time required to move the robot to the given positions

Position points	Duration (s)
Initial state	0
1	1.94
2	1.89
3	2.02

Conclusion

In this study, the control of a robotic manipulator was implemented on Ubuntu using ROS. The path planning algorithm was optimized, and the communication between the main computer and the motor controllers in ROS was enhanced to develop a communication method that is both applicable and easy to operate, while maintaining the original real-time performance and reliability. The experimental results clearly demonstrate that the control system can accomplish the primary tasks of a robotic manipulator, indicating its practical application value. However, the development of additional functionalities, such as the implementation of a machine vision system and the design of end-effectors, requires further research to achieve improved outcomes and future advancements.

References

1. Team Development with Visual Studio Team Foundation Server [Electronic resource]. - URL: https://visualstudio.microsoft.com/

2. MATLAB The Language of Technical Computing. [Electronic resource]. - URL: https://www.mathworks.com/help/matlab/

3. Martinez A. Learning ROS for Robotics Programming[M]. Packt Publishing, 2013.

4. Hart S, Dinh P, Hambuchen K. The Affordance template ROS package for robot task programming. Proceedings of the IEEE International Conference on Robotics and Automation. Seattle, WA, USA. 2015. 6227-6234.

5. Martin SR, Wright SE, Sheppard JW. Offline and online evolutionary bi-directional RRT algorithms for efficient replanning in dynamic environments. Proceedings of IEEE International Conference on Automation Science and Engineering. Scottsdale, AZ, USA. 2007. 1131-1136.

6. S. Li, D. Zhao, Y. Sun, "Path Planning Algorithm Based on the Improved RRT-Connect for Home Service Robot Arms," 2021 IEEE International Conference on Intelligence and Safety for Robotics (ISR), 2021, pp. 403-407, doi: 10.1109/ISR50024.2021.9419385.

7. A. Ghatak, A. A. Kumar, "Controller Area Network Bus based Communication system for Thermal Power Plant," 2020 Third International Conference on Advances in Electronics, Computers and Communications (ICAECC), 2020, pp. 1-6, doi: 10.1109/ICAECC50550.2020.9339516

8. UR5 Robot. [Electronic resource]. - URL: https://www.rarukautomation.com/collaborative-robots/ur-6-axis-collaborative-robots/ur5-robot/

Система управління роботом-маніпулятором на основі ROS

О. В. Стрельцов, Ю. В. Орновецький, М. О. Катріченко, В. Г. Поздняков, С. С. Коновалов

Національний університет «Одеська політехніка»

Анотація. *Для реалізації системи контролю роботом-маніпулятором досліджуються підходи та алгоритми управління. З метою забезпечення надійності та стабільності системи в реальному часі, пропонується створити систему керування роботом-маніпулятором на ПК з ОС Ubuntu, використовуючи ROS та CAN-зв'язок. Практичне тестування системи керування роботом-маніпулятором проведено за допомогою імітаційного експерименту. Результати показали, що система керування роботом-маніпулятором відповідає основним параметрам і вимогам для планування траєкторії руху робота-маніпулятора і здатна встановлювати зв'язок між головним комп'ютером та роботом для керування. Система керування є модульною, компактною, з чіткою структурою та низькою затримкою реагування.*

Ключові слова: *робот-маніпулятор, система ROS, проектування системи управління, алгоритм RRT, комунікаційний інтерфейс.*

Отримано 17.02.2025

About the authors

Oleh V. Streltsov, Ph.D., Associate Professor, Associate Professor of the Computer Systems Department, Odesa Polytechnic National University; 1, Shevchenko Avenue, Odessa, 65044, Ukraine. E-mail: streltsov.o.v@op.edu.ua; ph.: +38 048 705 8473

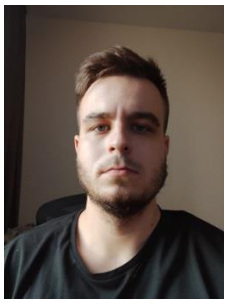
Стрельцов Олег Васильович, к. т. н., доцент, доцент кафедри комп`ютерних систем, Національний університет «Одеська політехніка»; проспект Шевченка, 1, Одеса, 65044, Україна. E-mail: streltsov.o.v@op.edu.ua; тел.: +38 048 705 8473

ORCID: https://orcid.org/0000-0002-4691-5703

Yuriy V. Ornovetsky, postgraduate student of the Department of Computer Systems, Odesa Polytechnic National University; 1, Shevchenko Avenue, Odessa, 65044, Ukraine.
E-mail: y.ornovetskyi@gmail.com; ph.: +38 063 050 4833

Орновецький Юрій Васильович, аспірант кафедри комп`ютерних систем, Національний університет «Одеська політехніка»; проспект Шевченка, 1, Одеса, 65044, Україна.
E-mail: y.ornovetskyi@gmail.com; тел.: +38 063 050 4833

ORCID: https://orcid.org/0009-0006-2470-1559



Mykhailo O. Katrichenko, postgraduate student of the Department of Computer Systems, Odesa Polytechnic National University; 1, Shevchenko Avenue, Odessa, 65044, Ukraine.
E-mail: m_katrichenko@ukr.net; ph.: +38 063 302 6906

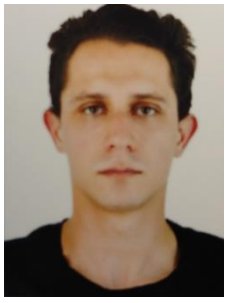
Катріченко Михайло Олегович, аспірант кафедри комп`ютерних систем, Національний університет «Одеська політехніка»; проспект Шевченка, 1, Одеса, 65044, Україна.
E-mail: m_katrichenko@ukr.net; тел. +38 063 302 6906

ORCID: https://orcid.org/0009-0004-2251-5600

Vladyslav G. Pozdnyakov, postgraduate student of the Department of Computer Systems, Odesa Polytechnic National University; 1, Shevchenko Avenue, Odessa, 65044, Ukraine.
E-mail: wladpozdnyakov676@gmail.com; ph.: +38 066 227 2373

Поздняков Владислав Геннадійович, аспірант кафедри комп`ютерних систем, Національний університет «Одеська політехніка»; проспект Шевченка, 1, Одеса, 65044, Україна.
E-mail: wladpozdnyakov676@gmail.com; тел.: +38 066 227 2373

ORCID: https://orcid.org/0009-0000-8393-1107



Serhiy S. Konovalov, postgraduate student of the Department of Computer Systems, Odesa Polytechnic National University; 1, Shevchenko Avenue, Odessa, 65044, Ukraine.
E-mail: rancor1997@ukr.net; ph.: +38 098 454 6233

Коновалов Сергій Сергійович», аспірант кафедри комп`ютерних систем, Національний університет «Одеська політехніка»; проспект Шевченка, 1, Одеса, 65044, Україна.
E-mail: rancor1997@ukr.net; тел.: +38 098 454 6233

ORCID: https://orcid.org/0009-0009-7647-9875