MOBILE ROBOT CONTROL USING ROBOT OPERATING SYSTEM

Uroš Pešović[1], Marko Marković[1], Velibor Trifunović[1], Slađana Đurašević[1], Željko Jovanović[1]
[1] *Faculty of technical sciences Čačak, University of Kragujevac, Serbia*

Abstract
Mobile robots can move freely throughout their working environment in which they can encounter known and unknown obstacles. The main tasks of mobile robots are terrain mapping, localization, path planning, and motion control could be significantly complex to implement on bare robot software. ROS represents a framework that enables the usage of a wide array of algorithms for mobile robots which can be easily used by various robot platforms. In this paper, we presented the implementation of ROS on a simple robot with a differential drive. This widely available robot platform can be used by a student to implement complex navigation algorithms which are supported by ROS.

Keywords: mobile robot, differential drive, ROS, Arduino.

INTRODUCTION

Mobile robots are a class of robots that can move freely through their environment. Robots must possess a certain level of autonomy and intelligence that allows them to perceive and react to obstacles. Fundamental problems in mobile robotics are terrain mapping, localization, path planning, and motion control [1, 2]. These tasks can be significantly complex to implement on the bare robot control software. Furthermore, such implementation will be applicable only for certain robot types, making poor code reusability.

Robot operating system (ROS) is a framework that represents a collection of tools, libraries, and conventions aimed at simplifying the process of programming complex robot behavior on a wide range of robotic platforms. The need for such a platform arose due to the constant progress of robotics, which is reflected in placing increasingly difficult tasks before robotic systems. For this reason, robot programming is a very difficult job for an individual and requires a lot of time and knowledge in various fields. In the classic way of programming robots without the use of ROS, the created programs were strictly related to hardware and could not be implemented on other robotic platforms without major crucial changes. The idea of ROS is to always provide an initial level in programming, so it is not necessary to always move from the beginning, but it is possible to apply ready-made standardized algorithms that can be implemented on different robot platforms, i.e. the possibility of reusing the created software is provided. ROS provides a set of tools to support many common robotic problems, such as path planning, collision detection, image processing, and many others, which is one of its main advantages, with the primary purpose of ROS being to create software for mobile robots. The need for such a robot programming system has led to ROS today being the most commonly used framework for mobile robot programming and more than 50% of mobile robots working using ROS. There are numerous ROS implementations on mobile robot platforms which are used as teaching aid [3, 4, 5]

In this paper, we presented the implementation of ROS on a simple mobile robot with a differential drive. This widely available robot platform can be used by students to implement complex navigation algorithms which are supported by ROS.

ROBOT OPERATING SYSTEM

The Robotic Operating System (ROS) is not a real operating system, but a framework for writing control programs for mobile robots.

The ROS project was started by Morgan Quigley in 2007 at Stanford University [6]. ROS requires a real operating system for its operation, such as officially supported Linux Ubuntu and Mac OS. Although ROS is not a true operating system, it offers standard operating system content, such as hardware abstraction, low-level device control, inter-process messaging, and packet management.

ROS creates a peer-to-peer network in which all processes are connected. Any process (node) in the system can access the network, communicate with other processes, and exchange data on the network. A robot control system usually contains many nodes, for example, one node controls the wheel motors, one node performs path planning, and so on. The ROS node is written in C, Python or Lisp using *roscpp*, *rospy* or *roslisp* ROS client compilers [7]. The ROS master controls communication between all connected nodes. Without it, nodes would not be able to find, exchange messages or connect services. Nodes communicate with each other by forwarding messages. A message is a simple data structure that contains specific fields. Messages are routed via the transport system on a subscription basis. The subject is the name used to identify the content of the message. When a node sends a message by posting on a given topic, a node that is interested in a particular type of data will subscribe to the appropriate topic where there may be multiple publishers and subscribers for the same topic. The publisher and the subscriber are not aware of each other's existence. The communication between the nodes is shown in Fig. 1.

A ROS service is a type of request/response interaction between processes. The topic is an asynchronous way of communication between nodes. The service serves to satisfy need for synchronous communication between nodes.

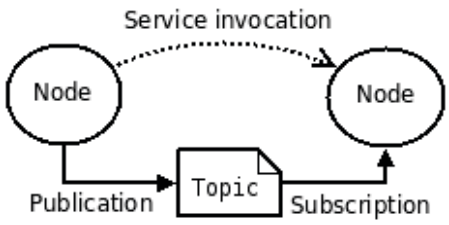

Fig. 1. *Communication between ROS nodes*

MOBILE ROBOT IMPLEMENTATION

The mobile robot is based on the Robot Smart Car Chassis Kit platform. This robot platform is designed as a platform for learning the initial steps from the world of mobile robotics, but it has all the elements necessary to perform a project task. The Smart Car Chassis Kit includes perforated Plexiglas chassis, two DC motors with reducers, two optical encoders, two plastic wheels and one pilot wheel. The mobile robot is controlled by ROS running on Raspberry PI4 computer which executes high-level algorithms for robot navigation, and by Arduino UNO microcontroller which performs low-level control of drive motors using driver L298N. Block diagram of mobile robot control system is shown in Fig. 2. The entire robot is powered by a rechargeable Li-po battery.

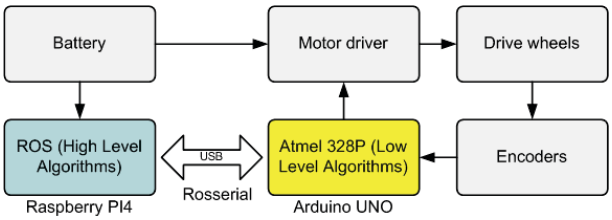

Fig. 2. *Architecture of mobile robot*

ROS is implemented on Raspberry PI4, a credit card format microcomputer, which is typically used for demanding embedded applications. Raspberry PI4 uses a quad-core ARM Cortex-A72 64-bit processor operating at 1.5GHz with 4 GB of RAM. It also features a wide range of wired and wireless interfaces, such as IEEE 802.11ac, Bluetooth 5.0 BLE, Gigabit Ethernet and several USB ports. Operating system Ubuntu 20.04 was installed on SD card, after which ROS Noetic framework is installed. In order to use ROS, it's first necessary to initialize the master node using *roscore* command, after which specific algorithms can be executed using *rosrun* command.

Rosserial is a protocol that enables the communication between ROS and serial devices, over a serial transmission line. In the Rosserial client-server implementation, the client is our case Arduino microcontroller while the server is a Raspberry PI4 computer on which ROS is installed. Rosserial packets are used to exchange via the serial link using the USB bus. The Arduino sends the data to

the ROS as messages that have a header and a tail, thus enabling subscription and publishing on multiple topics.

The structure of rosserial packet, shown in Fig. 3., consists of several fields. The first byte is used for synchronization and the second byte represents Protocol Version/Sync Flag. The following two bytes represent message length N and the fifth byte is a checksum of the message length. The sixth and seventh bytes are dedicated for Topic ID where values in the range from 0 to 100 are reserved for system functions. The remaining N+1 bytes are used for serial data and its checksum.

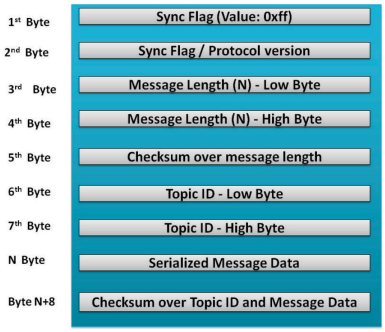

Fig. 3. *Structure of rosserial packet*

The packet checksum fields are calculated using the formula 1.

$$255 - (\Sigma \text{Bytes}) \% 256) \qquad (1)$$

The communication between the Arduino and Raspberry PI4 computer will start from the Raspberry side, which will send a query packet for getting the number of topics, names, and types of topics from the Arduino side. When the Arduino gets this query packet, it will reply to the Raspberry with a series of response packets. The mobile robot is controlled from ROS using wireless keyboard by executing commands depending on the pressed key. Based on the user input, ROS calculates linear and angular velocities which are published in form of the packet as shown in Fig. 4.

```
twist.linear.x = x * speed;
twist.linear.y = y * speed;
twist.linear.z = z * speed;

twist.angular.x = 0;
twist.angular.y = 0;
twist.angular.z = th * turn;

// Publish it and resolve any remaining callbacks
pub.publish(twist);
ros::spinOnce();
```

Fig. 4. *Generation of linear and angular velocities in ROS*

The Arduino microcontroller receives movement commands from ROS running on Raspberry PI4. Communication is carried using rosserial protocol via the USB interface. Based on the linear v_{lin} and angular speed ω_{ang} data, received from the ROS, the Arduino calculates the individual speed of rotation of the right and left drive wheels ω_r and ω_l. These angular drive wheel speeds are calculated using the diameter of drive wheels r=65 mm and wheel separation distance d=130 mm as shown by formula 2.

$$\omega_r = \frac{v_{lin}}{r} + \frac{\omega_{ang}}{r} \cdot \frac{d}{2} \qquad \omega_l = \frac{v_{lin}}{r} - \frac{\omega_{ang}}{r} \cdot \frac{d}{2} \qquad (2)$$

Since this robot has a differential drive, it is necessary to coordinate the movement of both drive motors to steer the robot as shown in Fig. 5. [8]. To drive a robot on a straight line it is necessary to rotate both wheels in the same direction with equal speed. For example, if the robot must turn to the right, the speed of the right wheel must be lower, in relation to the speed of the left wheel. To rotate the robot around its center, wheels need to rotate at the same speed but in different directions.

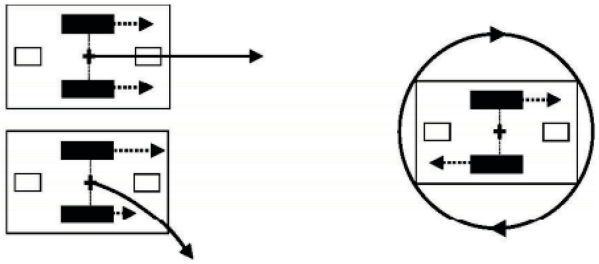

Fig. 5. *Types of movement for robot with differential drive*

For the robot to move at the required speed, it is necessary to regulate the speed of the rotation of each wheel. The regulation is performed by the PI regulator based on a comparison of the wheel setpoint value and the measured value of the wheel rotation speed. Wheel speed is measured by optical encoders connected to drive wheels, which generate 20 pulses per wheel rotation. Each encoder is connected to Arduino interrupt pin which triggers the appropriate interrupt routine for every pulse to update absolute wheel position.

To control the speed of each wheel, two PI controllers are implemented in the Arduino program. Their output is send to motor driver

in form of the PWM signal which controls the motor velocity, while the motor direction is controlled by motor driver H-bridge. The mobile robot chassis with both controllers is showed on Fig. 6, where Arduino Uno, motor driver and Li-po battery can be viewed from above, while Raspberry PI4 and motor drivers with encoders can be viewed from below.

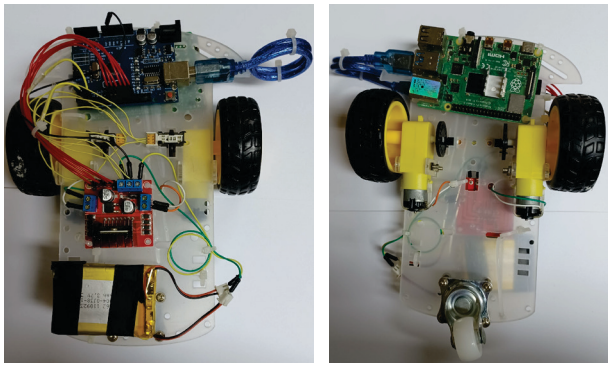

Fig. 6. *Mobile robot viewed from above and below*

CONCLUSION

Robotic systems have made rapid progress in recent years thanks to computing power and artificial intelligence. This progress of robotic systems is reflected in the increasing degree of independence and the possibility of creating a critical way of decision-making, which is characteristic of humans. For this progress to be possible, it was necessary to upgrade existing robot programming software or create new ones that would allow for rapid development and testing. Robotic Operating System (ROS) was created in response to all previous requirements, from its inception until today it has been constantly improved and refined so that a large number of robots base their programming on the application of ROS. The paper presented the implementation of ROS on a simple robot with a differential drive. Future work will be focused on the integration of robot sensor data into ROS to generate terrain maps and implement various obstacle avoidance algorithms.

ACKNOWLEDGEMENT

The research in this paper was supported by the Ministry of Education, Science and Technological Development of the Republic of Serbia, as part of the Project grant no. 451-03-9/2021-14/200132 with University of Kragujevac - Faculty of Technical Sciences Čačak.

REFERENCE

[1] Nourbakhsh I., Seigwart R., Introduction to Autonomous Mobile Robots, A Bradford Book The MIT Press, 2004, ISBN:978-0-262-19502-7

[2] Salichs M. A., Moreno L. E., Navigation of mobile robots: Open questions, Robotica, Volume 18, Issue 3, May 2000, pp. 227-234

[3] Zubrycki I., Granosik G., Introducing modern robotics with ROS and Arduino, Journal of Automation Mobile Robotics and Intelligent Systems, 2014, Vol. 8, No. 1, pp. 69-75, DOI: 10.14313/JAMRIS_1-2014/9

[4] West A., Arvin F., Martin H., Watson S., Lennox B., ROS Integration for Miniature Mobile Robots, Annual Conference Towards Autonomous Robotic Systems TAROS 2018, 25-27 July, Bristol, United Kingdom, pp 345-356, DOI: 10.1007/978-3-319-96728-8_29

[5] Araujo A., Portugal D., Couceiro M., Rocha R., Integrating Arduino-Based Educational Mobile Robots in ROS, IEEE 13th International Conference on Autonomous Robot Systems and Competitions, Lisbon, Portugal, 2013, DOI: 10.1109/Robotica.2013.6623520

[6]Quigley M., Gerkey B., Smart W. D., Programming Robots with ROS, O'Reilly Media, 2015, ISBN: 978-1449323899

[7] Martinez A., Fernandez E., Learning ROS for Robotics Programming, Packt Publishing, 2013, ISBN: 978-1783987580

[8] Djurasevic S., Milovanovic A., Correction of Systematic Errors in Odometry Model for Position Determination of Mobile Tracked Robot, 52nd International scientific conference on information, communication and energy systems and technologies (ICEST 2017), Niš, Serbia, June 28-30, 2017.