PROBLEMY MECHATRONIKI
UZBROJENIE, LOTNICTWO, INŻYNIERIA BEZPIECZEŃSTWA

ISSN 2081-5891
E-ISSN 2720-5266

13, 4 (50), 2022, 85-108

PROBLEMS OF MECHATRONICS
ARMAMENT, AVIATION, SAFETY ENGINEERING

Autonomous Robot Project Based on the Robot Operating System Platform

Szymon CHERUBIN, Wojciech KACZMAREK*, Natalia DANIEL

Military University of Technology,
Faculty of Mechatronics, Armament and Aerospace,
2 Sylwestra Kaliskiego Str., 00-908 Warsaw, Poland
**Corresponding author's e-mail address and ORCID*:
wojciech.kaczmarek@wat.edu.pl; https://orcid.org/0000-0003-3805-9510

Received: August 15, 2022 / Revised: September 1, 2022 / Accepted: December 9, 2022/ Published: December 30, 2022.

DOI 10.5604/01.3001.0016.1462

Abstract. This article presents the concept of an autonomous mobile robot running on the ROS system and using advanced algorithms for 2D map generation and autonomous navigation. The authors focused on presenting the hardware platform, the Linux-based software and the Robot Operating System (ROS) platform. This article also introduces an algorithm that provides the generation of a 2D map of the surroundings, autonomous robot driving and remote control of the device. Measurements of the temperature of the computer helped in the decision on which cooling system to use.
Keywords: robotics, mobile robot, map generation, ROS, Linux

1. INTRODUCTION

Dynamic technological development has been observed in the world for a long time, especially in the field of robotics. It already has a significant impact on people's everyday lives, especially in terms of replacing human work with activities carried out by machines. This phenomenon is particularly visible in industry [1-3], but it can also be found more often in the area of services. Most of the robots used in the world are fully autonomous constructions that are able to perform monotonous inspection works on their own and act as autonomous platforms for fleet management and control of intralogistic processes [4]. Nevertheless, such designs are also widely used in the military sphere. Using them provides the opportunity to carry out reconnaissance, support and combat operations in dangerous or inaccessible terrain. It is extremely important for humans, as not involving human resources in tasks that are dangerous to human life or health increases the chance of survival or at least minimising any damage to the health. The use of highly advanced sensory devices in modern robots allows the use of perceptual abilities that are much more accurate and reliable compared to human organs – especially senses such as vision and hearing [5]. The resistance of robots to an environment dangerous for humans is particularly visible in tasks related with space exploration, where unmanned mobile robots can successfully operate on other planets of our solar system: Sojourner, Spirit, Opportunity, Curiosity and Perseverance [6]. For mobile robots, the Robot Operating System (ROS) software platform is often used, as it is a free platform and can be installed on the free LINUX operating system [7-11].

This article presents the developed autonomous robot that uses advanced SLAM algorithms for its operation, which allows the robot to generate a 2D map of its surroundings and locate itself on the generated map. The robot developed here is a design that uses ROS software running on the Linux kernel-based operating system – Ubuntu Mate.

2. AUTONOMOUS ROBOT PLATFORM

A robot comprises a set of electronic components and peripheral devices transmitting information and control commands with others, allowing the robot to perform its assigned tasks according to the implemented workflow algorithm. In order to function properly, the vehicle required a properly selected hardware and software structure. The main tasks performed by the developed robot were:

- generation of a digital, 2D map of the surroundings,
- location in space using proprioceptive odometry,
- autonomous driving,
- operating in a remote-control mode.

2.1. Hardware structure

The hardware structure of the robot consisted of appropriately selected electronic and mechanical components (Fig. 2.1).

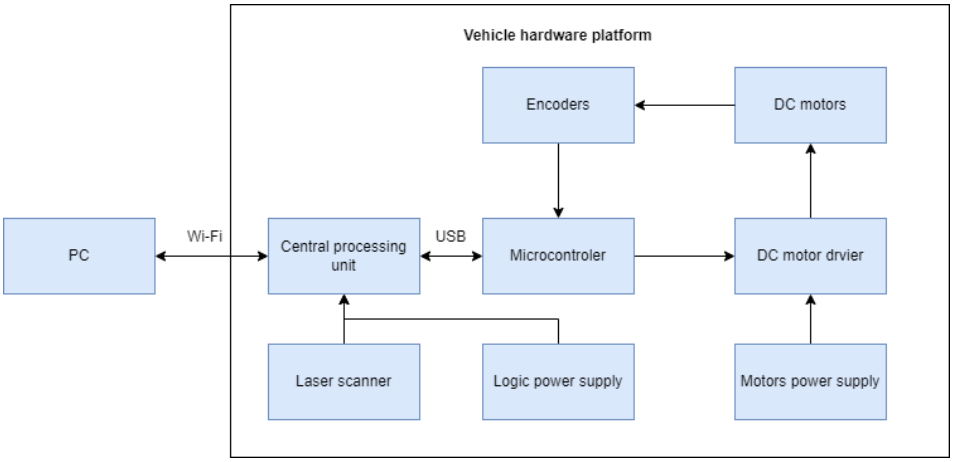

Fig. 2.1. Block diagram of the robot's hardware structure.

The heart of the robot was a single-board Raspberry Pi 4B computer (Table 2.1) with a Cortex-A72 64-bit processor acting as the robot's on-board computer and the supporting Arduino MEGA (ATmega2560) microcontroller responsible for motor control. The on-board computer was the unit responsible for processing any data coming from the sensors and for wireless data transmission to the operator's personal computer. The microcontroller, which was connected to the central unit via a USB connector, was responsible for receiving signals from the encoders and developing control commands transmitted directly to the motors via the GPIO pins. The purpose of using the Arduino MEGA microcontroller (Table 2.2) was to relieve the on-board computer in order to handle four system interrupts necessary for correct control (reading data from the encoders).

Table 2.1. Raspberry Pi 4B computer specifications [12]

Parameter	**Value**
Processor	Broadcom BCM2711, Cortex-A72 64-bit SoC, ARM8v-A, 4 x 1.5 GHz
RAM	4 GB LPDDR4-3200 SDRAM
Memory	Flash: micro SD card slot
Supply voltage	5.1 V, 3 A via USB C
Wi-Fi interface	2.4 GHz and 5.0 GHz IEEE 802.11ac
Communication interfaces	UART, SPI, I2C, GPIO, DSI, CSI, USB 2.0, USB 3.0

Table 2.2. Arduino MEGA microcontroller specifications [13]

Parameter	Value
Microcontroller	ATmega2560
Clock frequency	16 MHz
Flash memory	256 kB
SRAM	8 kB
EEPROM	4 kB
Analogue I pins	16
Digital I/O pins	54 (15 PWM outputs)
Supply voltage	7 – 12 V

The main source of information obtained from the surroundings was the RPLidar A3M1 laser scanner by Slamtec [14]. It was responsible for generating the data used to create a digital map of the surroundings. It was an allothetic source of information with high accuracy and high speed of work due to the use of a laser beam in the infrared range as a data carrier. Table 2.3. presents the basic operating parameters of the RPLidar A3M1.

Table 2.3. Basic operating parameters of the RPLidar A3M

Parameter	Indoor operating mode	Field operation mode
Range	White objects: 25 m Black objects: 10 m	White objects: 25 m Black objects: 8 m
Scan frequency	Nominal value: 15 Hz, changeable in range: 5-20 Hz	
Sampling frequency	16 kHz	16 / 10 kHz
Angular resolution	0.225 °	0.225 / 0.36 °
Communication interface	TTL UART	
Transmission speed	256 000 bps	

The ROVER 5-2 platform was used as the chassis in the developed project. It had a crawler track system to perform no holonomic motion. The platform was equipped with two TECO TFK280SC DC motors with a gear ratio of 87:1. They generated an output torque of 1 Nm and a rotational speed of 8804 rpm [15]. The sensors used to acquire data on the robot's position in space in the form of the number and direction of rotation of the motors were incremental encoders using the Hall effect with a resolution of 333.33 per revolution (Fig. 2.2).

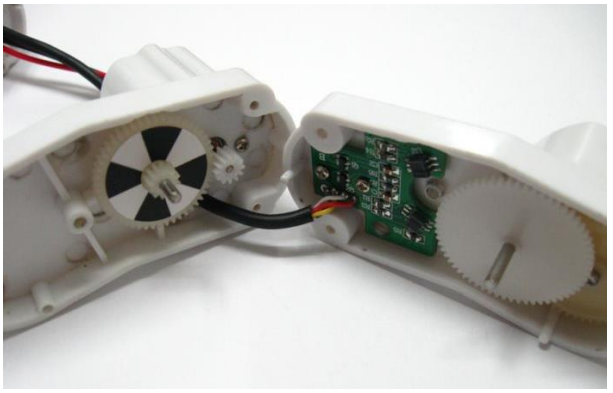

Fig. 2.2. Incremental encoder used in the robot design

The DC motors driving the platform were controlled by a two-channel L298N controller based on a double H-bridge. The controller achieved current efficiency on each of the channels equal to 2 [A]. Additionally, the maximum voltage supplying the motors was 35 [V]. This controller was protected against overheating by an added heat sink (Fig. 2.3).

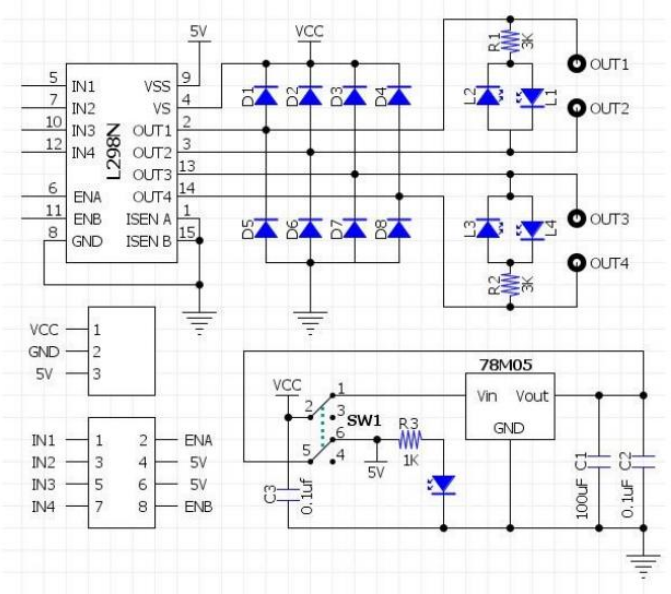

Fig. 2.3. Schematic diagram of the L298N DC motor controller [16]

The completed model of an autonomous robot is shown in Figure 2.4.

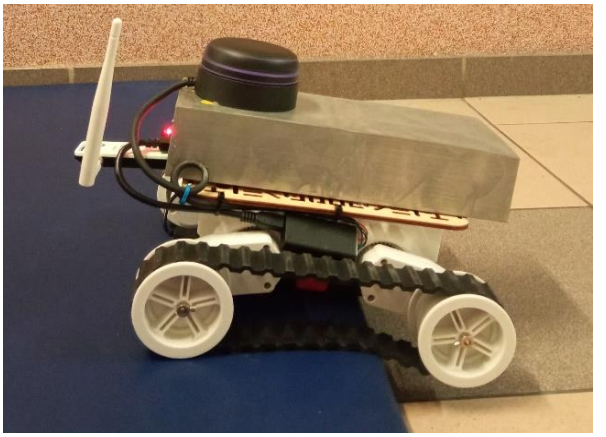

Fig. 2.4. Completed model of the autonomous robot

2.2. Software

The presented solution used the Ubuntu MATE operating system, one compatible with ROS software.

2.2.1. Ubuntu MATE

The Ubuntu MATE system (Figure 2.5) was a complete GNU/Linux operating system distribution using MATE as the desktop environment.

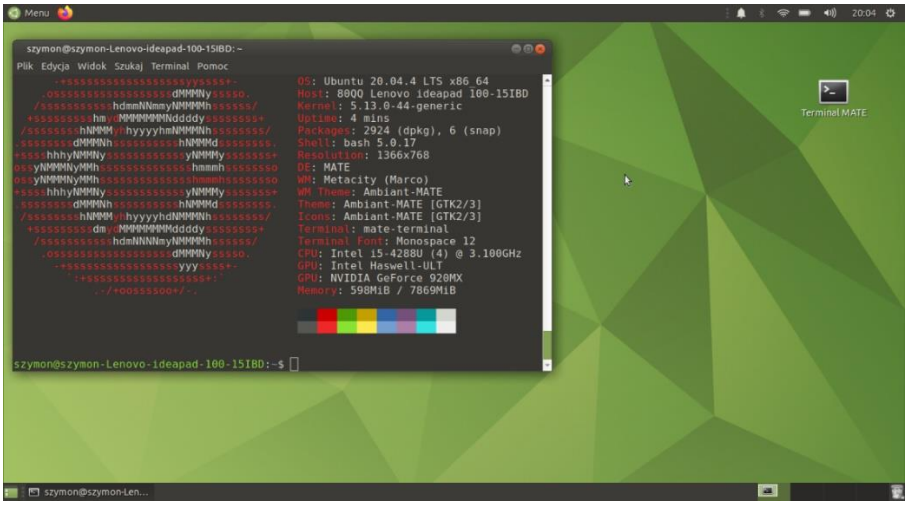

Fig. 2.5. Screen of a computer with Ubuntu MATE installed

This system provided great support for ROS software and was a version of the system that used much fewer computing resources of the CPU compared to the classic version of Ubuntu. The system version 20.04 Focal Fossa was installed on the robot's platform. It was the latest version of this operating system, with the longest technical support at that moment. The TCP/IP network was used for communication between the robot's central unit and the user's personal computer [17]. The communication was provided using the *ssh* communication protocol.

2.2.2. Robot Operating System

As previously mentioned, the software used to implement control algorithms and responsible for acquiring and processing of data from sensory systems was the ROS platform. It was a powerful programming platform running on a free software basis. Its main application in a robotic environment was system service responsible for control of humanoid, industrial and mobile robots. It was created in 2007 in California thanks to the cooperation between Stanford University and a robotics company – Willow Garage. Since its creation, the platform has been constantly developed by an international group of programmers dealing with robotics [18].

The operation of the ROS platform was based on the communication of processes, which were simultaneously working nodes responsible for various functions of the robot (Fig. 2.6). They formed a network in which the processes were interconnected. This ensured that each node had access to the network, cooperation between them and the ability to monitor the type of data sent to the network. ROS enabled the exchange of messages in the form of topics, services, parameters and actions [19] and [27].

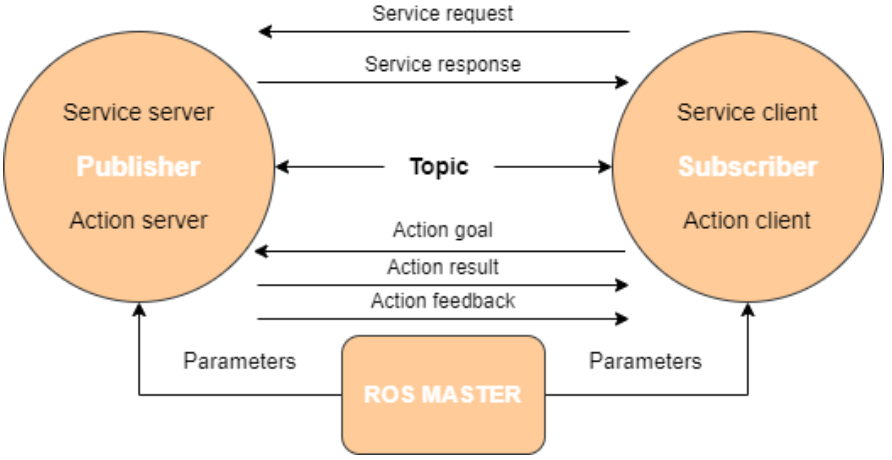

Fig. 2.6. Diagram of node cooperation in the ROS environment

The ROS environment supported programs developed in high-level languages: Python and C++. Catkin was responsible for building the structures. This was an official compiler made up of CMake macros and custom scripts developed in Python. Catkin was the successor to the basic ROS compiler – rosbuild. Compared to its predecessor, it featured significantly more functionality. As with CMake, catkin automatically managed the compilation processes, so that the programme was not compiled independently, but files with compilation rules for a specific environment were generated. For GNU/Linux, makefiles were generated by using scripts derived from Python, the user of the catkin compiler could automatically search for packages and develop multiple large format projects simultaneously [20].

The ROS environment also had many graphical tools that allowed control of the operation of individual vehicle components and software packages. In addition, it was equipped with tools that allowed one to visualize vehicle operation on the basis of data from sensory systems, as well as to simulate the vehicle operation. The most commonly used tools were:

- RQT package – a platform included in the ROS software that enabled the implementation of various GUI tools in the form of plug-ins. The package was used to manage all tools in a single window. The platform was the most commonly used software for the diagnostics of robots [21]. In addition, the platform allowed visualization of the structure of the nodes and the connections between them, which greatly facilitated the understanding of ROS operation. What is more, the RQT package enabled node analysis by mirroring the transformation tree launched by the basic node of the *rqt_tf_tree* system and invoking the node connection network – *rqt_graph*.
- Rviz (Fig. 2.7) – a package used for three-dimensional visualization of messages in ROS, which was developed at a Korean university [22].

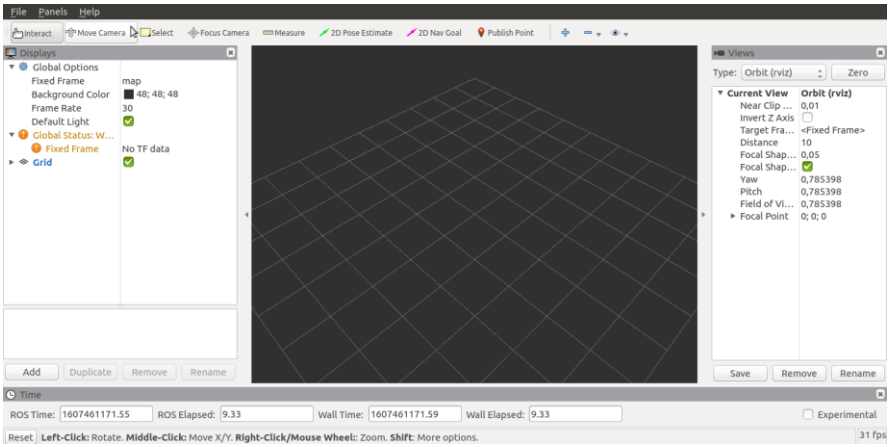

Fig. 2.7. Rviz graphic visualizer panel

It allows the visualization of data from sensors and depicts the robot's environment. In addition, it permits visualization of data from the perspective of a selected coordinate system based on data from the tf library. It also enabled graphical representation of the robot's URDF model and kinematic analysis of its motion. Additionally, the interface of the graphical tool enabled the creation of a digital map of the environment surrounding the robot based on sensor data.

- URDF (*Unified Robot Description Format*) – used to determine the kinematics and dynamics of the robot and was also used to represent the robot [23]. The format was used in the rviz visualizer and in the Gazebo simulator. URDF files were created using HTML. Based on the data contained in the unified format, it was possible to calculate the spatial pose of the robot and detect potential programming errors. URDF files with the appropriate plug-ins could be generated from CAD software. An example description of a humanoid robot using URDF is shown in Fig. 2.8.

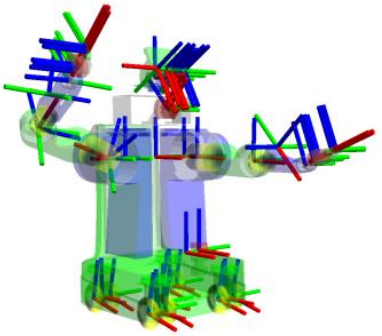

Fig. 2.8. Description of the humanoid robot representation using the URDF format

- Gazebo – a free ROS program for conducting 3D simulations of robot functioning, created in 2000 at the University of Southern California as a part of the Player project. It is one of the most popular robotics simulators that uses the OGRE (*Object-Oriented Graphics Rendering Engine*) graphics engine. This is because of its high efficiency and fully accurate mapping of reality and the laws of physics for which the ODE (*Open Dynamics Engine*) is responsible [24].

3. ALGORITHMS

A robot requires a finite sequence of predefined and consecutive actions in order to work correctly and complete its tasks.

A simplified block diagram of the robot's algorithm that generates a 2D map, autonomous driving and remote control is shown in Fig. 3.1.

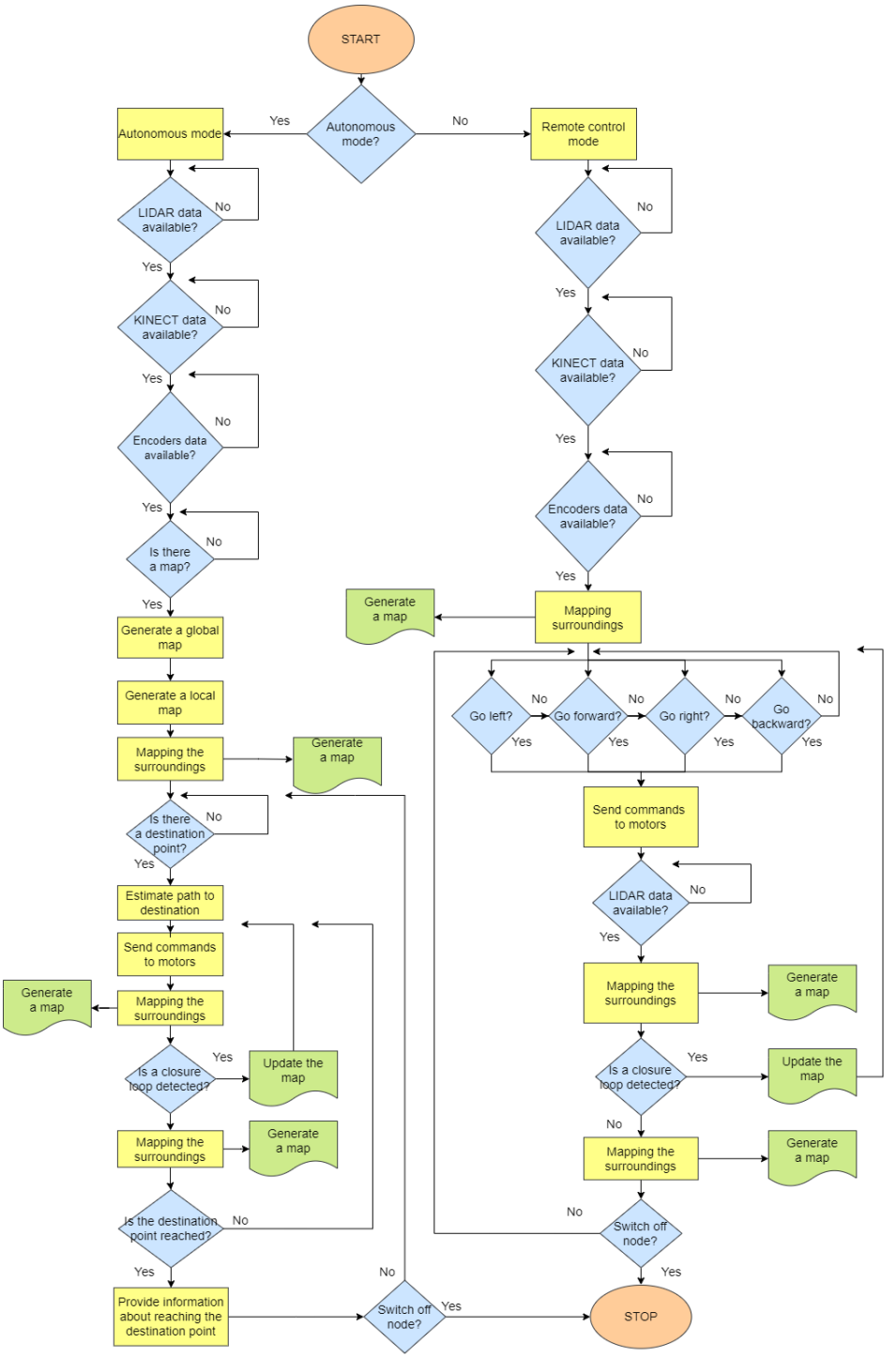

Fig. 3.1. Simplified block diagram of the robot algorithm

3.1. Control algorithms

The robot, apart from autonomous driving, can be remotely controlled. It was implemented through the ROS network and the *ssh* communication protocol. More specifically, the ROS network used a master-slave relationship in a duplex transmission. The central unit of the robot was ROS_MASTER_URI, which was responsible for the mutual location of the running nodes and the control of the transmitted data. The operator's computer unit was the host. From there, it was possible to visualize and control the robot's operations. The network of nodes responsible for remote control of the effectors is presented in Fig. 3.2.

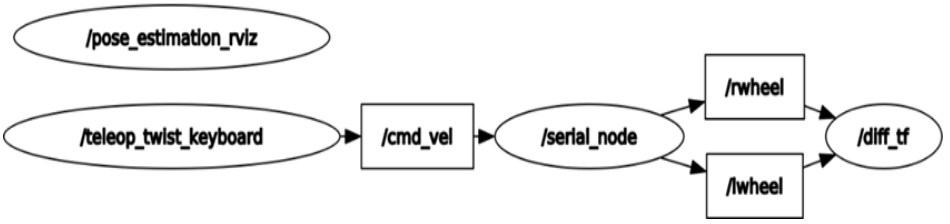

Fig. 3.2. Network of nodes responsible for remote control

The *teleop_twist_keyboard* node was responsible for handling the keys on the user's personal computer and converting the registered keys into an appropriate stream of commands in the form of *cmd_vel* messages. This message was a PWM signal with a length selected according to the vehicle speed received by the *serial_node*. This node was a representation of the *rosserial* packet, a communication protocol responsible for converting ROS messages, topics, and services for use in serial communication [25]. In this case, the node was responsible for communication between the robot's CPU and the Arduino MEGA connected with the USB connector. It then transmitted the control signals separately for each motor to the *diff_tf* node. It consisted of a differential drive package that transmitted the control signals to the motors. This node was also responsible for reading data from the encoders and transmitting odometry information in the form of *tf* transformation messages for the navigation stack. Figure 3.3 shows a diagram of the robot with a differential drive. Its mathematical development has been described in great detail by P. J. Alsin in [26].

The autonomous driving of the robot took place using the *navigation_stack*. This was an advanced algorithm responsible for estimating the route of the vehicle movement to a point set by the operator and transmitting appropriate control signals to the vehicle's motors to reach this point.

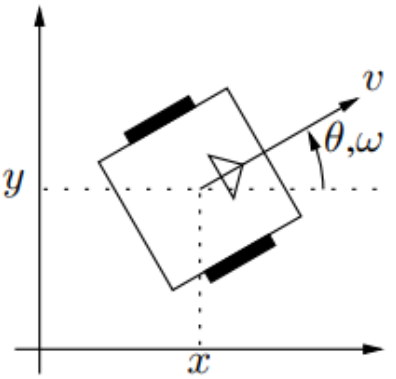

Fig. 3.3. Diagram of a robot with a differential drive [26]

The *move_base* package was the main node of the navigation stack that combined a global and local planner and supported two cost maps: local and global. These were the mandatory elements required to perform the navigation tasks. The global planner was responsible for planning the trajectory path from the starting point to the destination. It was based on Dijkstra's heuristic algorithm. This was a "greedy algorithm" for finding the shortest path on the map from the source vertex to the remaining vertices in the graph, characterized by a non-negative edge weight [28]. The role of the local planner was performed by the DWA algorithm – the dynamic windows method. It was proposed by D. Fox, W. Burgard and S. Thrun in 1997 [29]. The algorithm was responsible for generating control commands in the form of the linear and angular velocities (v, ω) that were optimal for the local conditions of the vehicle, aimed at avoiding obstacles on the path. The approach took into account a periodically short time interval when calculating the next control command. This avoided the enormous complexity of the traffic planning problem. The algorithm was implemented in the 5 steps proposed by K. Zheng [30]:

1. Sample discretization in the robot control space (dx, dy, dΘ).
2. For each of the velocity samples a simulation was made based on the current state of the robot, how its state would change if the sampled speed was applied to the effectors for a short period of time.
3. Assessing each simulation trajectory using metrics such as target proximity, global path proximity and speed. Rejection of trajectories causing a collision.
4. Selection of the most accurate trajectory and sending it to the vehicle's effectors.
5. Execute and repeat.

Figure 3.4 shows a block diagram of the navigation stack. It contained the previously mentioned *move_base* node, which was responsible for generating the map of the surroundings and retrieving the previously prepared map from the data memory.

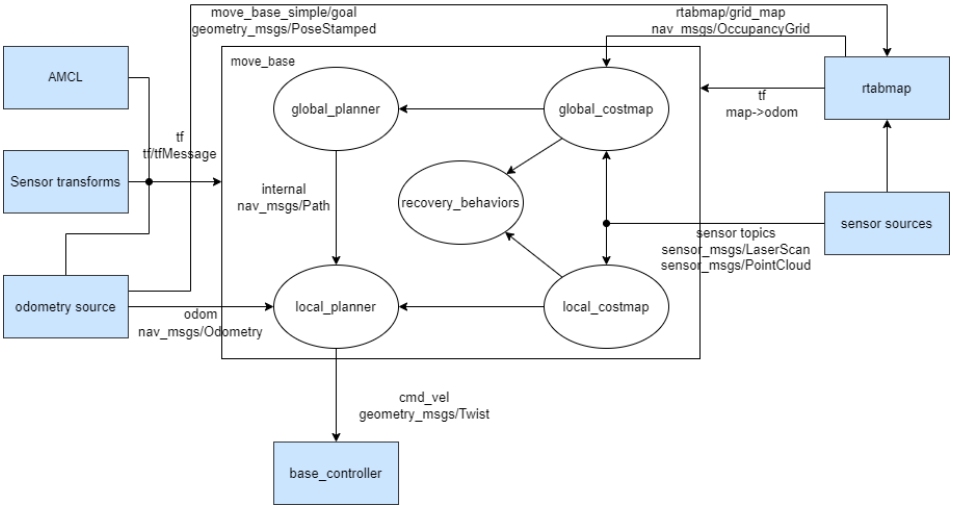

Fig. 3.4. Block diagram of the navigation stack.

Additionally, the diagram shows the relationship between the previously mentioned nodes with the components responsible for publishing data about the surroundings. The published data must be in the form of appropriate message types, which are also shown in the diagram. In order to work properly, the navigation stack required the use of the ROS environment. In addition to using the appropriate environment, it was also necessary to correctly configure it and deliver the required types of messages. A mandatory node in the navigation stack was the *amcl* node. It was a probabilistic system of locating a moving robot in two-dimensional space. The system was responsible for Monte Carlo adaptive localisation, which used a particle filter to track the robot's position on a known map. It was an algorithm presented by D. Fox in 1999 [31]. Equally important was the information on the relationships between the sensor coordinate frames and the odometry information published by the library, *tf*. The publication of the odometry information was performed by the *diff_tf* node. This was described in the discussion on remote control of the robot. Additionally, odometry information must be published in the form of the nav_msgs/Odometry message.

To move in the environment without collisions, the navigation stack used information from a laser scanner, which published information in the form of a sensor_msgs/LaserScan message. The following control messages were used to control the robot's motors: (*cmd_vel.linear.x*, *cmd_vel.linear.y*, *cmd_vel.angular.z*).

They were sent in the form of a geometry_msgs/Twist message and converted into commands that properly controlled the operation of the motors.

Figure 3.5 shows the network of operating nodes responsible for the correct implementation of autonomous navigation.

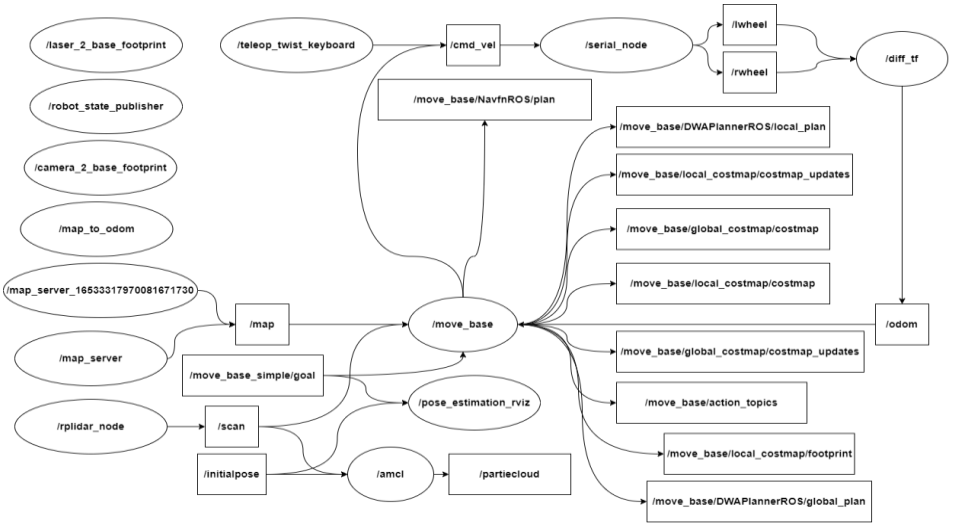

Fig. 3.5. Diagram of nodes responsible for autonomous navigation

Figure 3.6 shows the map of the room, with the global and local maps superimposed on it. They were used for the implementation of autonomous navigation tasks. In addition, the estimated path of the vehicle to reach the set target position was marked on the map.

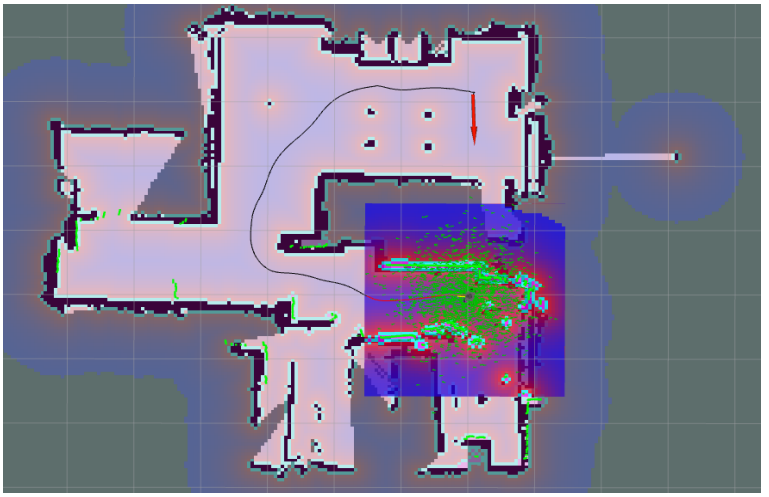

Fig. 3.6. Map showing the implementation of autonomous navigation by a robot

3.2. Map generating algorithm

The advanced SLAM algorithm was responsible for generating a two-dimensional map of the robot's surroundings. The algorithm used in the project is gmapping, which was an open access package developed by the team of G. Grisetti, C. Stachniss and W. Burgard [32]. It has been the most widely used algorithm for SLAM problems. It used an Extended Kalman Filter (EKF) to locate the robot, which, in comparison to the Monte Carlo methods, was less computationally expensive for each mean-dimensional state space. The comparison of both algorithms was conducted by D. Yuen and A. MacDonald in 2002 [33]. Gmapping was a planar algorithm responsible for solving SLAM problems based on proprioceptive odometry and measurements from a laser scanner, while the hector_slam package used only data from a laser scanner.

The gmapping package was based on the presence a grid representation, which represented the robot's environment map as binary fields of random variables, each of which represented the presence of an obstacle. Presence grid maps were algorithms used in the probability of generating a map assuming a known robot position. According to the well-known Bayesian approach [4], used to combine transitions between robot positions and sensor measurements, the algorithm solved the SLAM problem using the Rao-Blackwellized particle filter. Its key idea was a posteriori estimation $p(x_{1:t},m|z_{1:t},u_{1:t-1})$ of the m map, and the robot's trajectory $x_{1:t} = x_1, \ldots, x_t$. Estimation was based on observations $z_{1:t} = z_1, \ldots, z_t$ and odometry measurements $u_{1:t-1} = u_1, \ldots, u_{1:t-1}$.

The Rao-Blackwellized particle filter for the SLAM problem used the following factorization [32]:

$$p(x_{1:t}, m|z_{1:t}, u_{1:t-1}) = p(m|x_{1:t}, z_{1:t}) \cdot p(x_{1:t}|z_{1:t}, u_{1:t-1}) \tag{3.1}$$

At first, this factorization allowed an estimation of the robot's trajectory, and then on its basis calculated a map of its surroundings. Due to the strong dependence of the map on the trajectory, this approach provided high computational efficiency. This technique is often referred to as Rao-Blackwellization [32].

Matching of scans, observation model and scoring method of the motion model was done using a particle-based scan matching technique – "vasco", which was part of *Carnegie Mellon Robot Navigation Toolkit* (CARMEN) [35]. This technique performed a gradient search based on the reliability function of the current observation, taking into account the current grid map. It aimed to find the most likely pose by matching the current observations with the map produced so far. The equation describing this technique has the following form [32]:

$$\hat{x}_t^{(i)} = argmax\ p(x|m_{t-1}^{(i)}, z_t, x_t'^{(i)}) \tag{3.2}$$

where: $x_t'^{(i)}$ - this is the first supposition, z_t - current scan, $m_{t-1}^{(i)}$ - reference map.

Bayes' rule is used to solve equation (3.2) – the pose with the highest observation probability was sought $p(z_t|m,x)$. In order to calculate this probability, the "*beam endpoint mode*" was used [36]. In this model, the individual beams within the scan were considered independent. The beam probability was calculated based on the distance between the beam end point and the nearest obstacle from that point. In order to obtain quick calculations a tangled, local grid map was used [32].

The ROS software for generating a digital map, apart from an advanced mapping algorithm, also required an appropriate transformation of the coordinate systems representing each of the robot members. This was carried out by the tf library, which was responsible for the scene graph concept, reflected in the form of a hierarchy tree. The tree root was a representation of the environment map, and each of its vertices was a geometric transformation, translation, or rotation between the systems of each member [37]. Figure 3.7 shows the transformation tree of the robot's coordinate systems generated by the *rqt_tf_tree* package.

The laser subtree reflected the coordinate system in a polar form represented by a laser scanner, while the *base_link* was a representation of the coordinate system of the robot platform. Odom was the frame responsible for locating the robot based on the data provided by the *diff_tf* node, which was responsible for acquiring data from encoders and transforming them into odometry information. The root of the tree was the map frame.

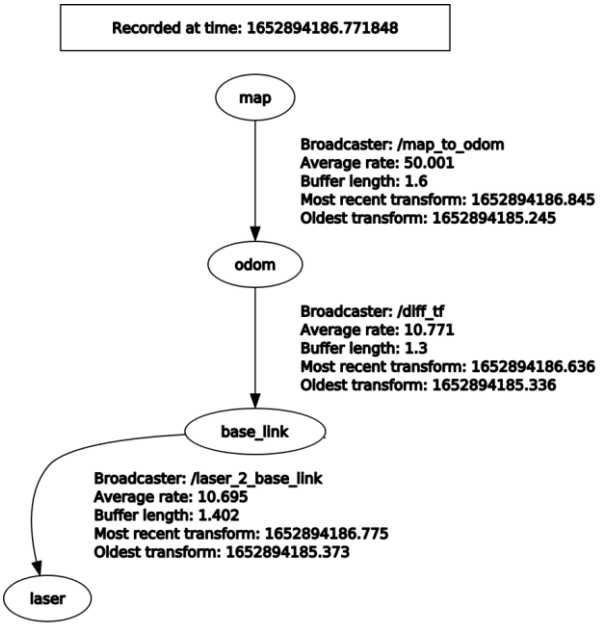

Fig. 3.7. Transformation tree of robot coordinate systems

Figure 3.8 shows the generated digital, two-dimensional map of the room generated using the *gmapping* package.

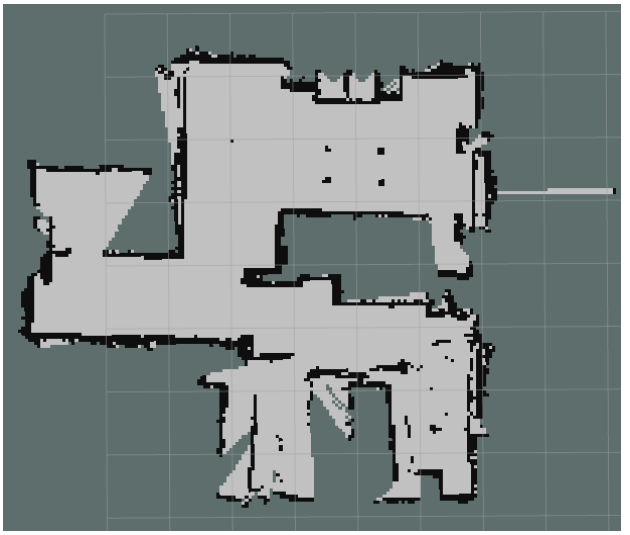

Fig. 3.8. Two-dimensional map of the room generated using the gmapping algorithm

4. VEHICLE PLATFORM TESTS

Due to the type of computer and its cooling method, the completed robot was tested in terms of the processor operating temperature and the entire central unit. Measurements were made by reading the processor temperature by triggering a command in the system shell and measuring the temperature of the on-board computer with a thermal imaging camera. They were performed at the moment of starting the CPU and the software responsible for the nominal operation of the vehicle, and then after 5, 10 and 15 minutes of operation. Measurements were taken in the following configurations:

- unit without cooling,
- unit with passive cooling in the form of a housing,
- unit with passive and active cooling.

Figure 4.1 shows an exemplary photo of the operating vehicle central unit with cooling taken with a thermal imaging camera.

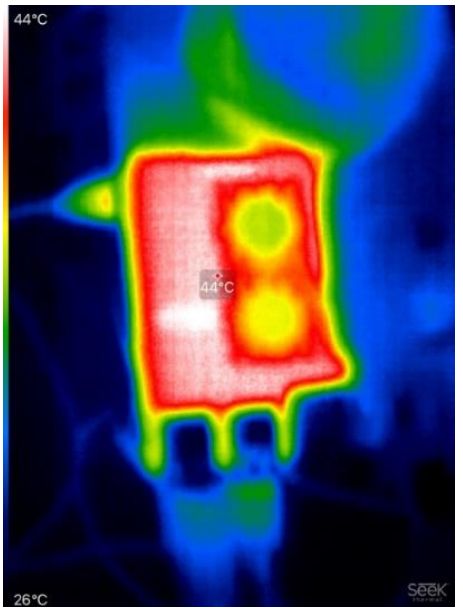

Fig. 4.1. Working central unit with passive cooling after 5 minutes of operation

The results of the temperatures obtained are shown as graphs. Figure 4.2 shows the results of temperature measurements of the CPU with a thermal imaging camera, while Figure 4.3 shows a graph of the CPU temperature.

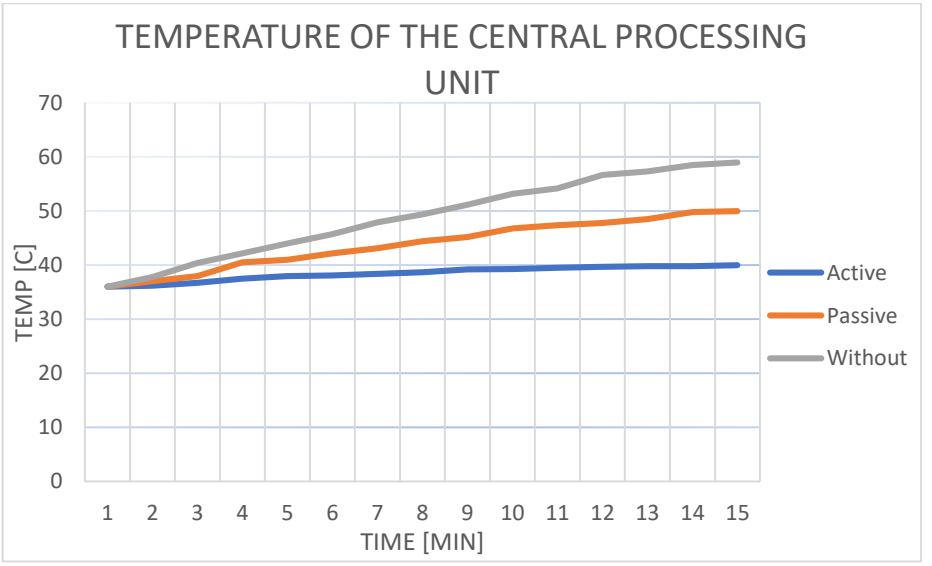

Fig. 4.2. Temperature graph of the CPU using different cooling methods

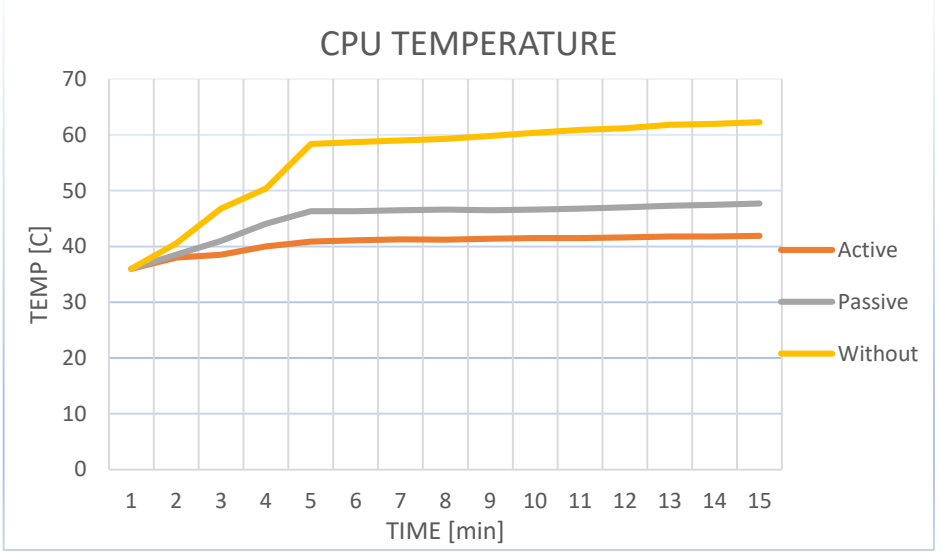

Fig. 4.3. CPU temperature graph using different cooling methods

The measurement of the temperature reached by the vehicle's CPU and its processor during rated operation was aimed at selecting the most favourable method of cooling the robot's CPU. Cooling is an extremely important aspect of an autonomous robot design due to the protection of its components against overheating and damage.

When the RPi 4B temperature is above 80°C the processor is throttled, resulting in an unwanted reduction in the performance of the computing unit. Considering the results of the tests, the lowest temperatures were achieved using the active cooling method, which provided a 34% reduction in temperature compared to the unit without cooling. However, due to the use of the vehicle in various conditions and the additional energy consumption of the fans, the passive cooling method, which reduced the temperature of the CPU by 17%, is the most advantageous. For the temperature reached by the CPU, the difference between the active and passive methods is about 8%.

5. CONCLUSIONS

The development of robotics has an increasing impact on people's daily lives. Engineers are increasingly trying to replace people in labour-intensive, monotonous and dangerous jobs by developing new types of robots. This phenomenon is particularly visible in industry but increasing numbers of robots can also be found in the services area or even in private life. Alongside the emerging designs, where powerful computers are required, increasingly sophisticated software is being developed, without which there would be no autonomous machines.

An example here are the advanced algorithms used to solve SLAM (Simultaneous Localization and Mapping) problems, which make robots more autonomous and increasingly safe for humans.

The autonomous robot presented in the article is equipped with advanced sensory devices (including the RPLidar A3M1 laser scanner by Slamtec), and the algorithms used to control the robot can be used in other designs of mobile robots. Thanks to the use of the LINUX OS and the Robot Operating System programming platform, all operations performed by the robot were executed in real time. The proposed design and software met the requirements, and the use of a laser scanner allowed for the implementation of innovative and advanced algorithms for autonomous navigation and generation of two-dimensional maps. This allowed the robot to move independently in the field. Due to the design of the developed unit, it is rather intended for use in indoor work. A series of tests was performed to generate a two-dimensional room map using the gmapping algorithm. On its basis, the robot correctly travelled between the entered trajectory points. In search of optimal cooling for the computer used in the robot, measurements were made with a thermal imaging camera. It was verified whether passive cooling would be sufficient and by how much such cooling is less efficient than active cooling.

We are currently working on extending the capabilities of the robot with an RGB-D video module and developing software to perform a fusion of data from the module and the laser scanner to generate a 3D map of the surroundings. This development is an extremely challenging undertaking that requires the use of state-of-the-art algorithms.

FUNDING

The study was co-financed by the Military University of Technology (Warsaw, Poland) under Research project UGB 893/2021.

REFERENCES

[1] Borys, Szymon, Wojciech Kaczmarek, and Dariusz Laskowski. 2020. “Selection and optimization of the parameters of the robotized packaging process of one type of product”. *Sensors* 20 (18) : 5378-1-21.

[2] Kaczmarek, Wojciech, Bartłomiej Lotys, Szymon Borys, Dariusz Laskowski, and Piotr Lubkowski. 2021. "Controlling an industrial robot using a graphic tablet in offline and online mode”. *Sensors* 21 (7) : 2439-1-20.

[3] Panasiuk, Jarosław, Wojciech Kaczmarek, Michał Siwek, and Szymon Borys. 2022. “Test Bench Concept for Testing of Gripper Properties in a Robotic Palletizing Process”. *Problemy mechatroniki. Uzbrojenie, lotnictwo, inżynieria bezpieczeństwa / Problems of Mechatronics. Armament, Aviation, Safety Engineering* . 13 (2) : 51-64.

[4] Płaczek, Ewa, Kornelia Osieczko. 2020. „Zastosowanie robotów AGV w intralogistyce”. *Zarządzanie Innowacyjne w Gospodarce i Biznesie* 1 (30) : 165-176.

[5] Matthews, Kayla. 2018. "How Robot Precision Has Evolved, Enabling More Uses”. *Robotics Business Review* . (https://www.roboticsbusinessreview.com/manufacturing/robot-precision-evolves/)

[6] Jayawardana, J.K. Rahul, and T. Sameera Bandaranayake. 2021. "A review of unmanned planetary exploration on Mars”. *International Research Journal of Modernization in Engineering Technology and Science* 3 (2) : 451-462.

[7] Besseghieur, Khadir Lakhdar, Radosław Trębiński, Wojciech Kaczmarek, and Jarosław Panasiuk. 2020. “From Trajectory Tracking Control to Leader–Follower Formation Control”. *Cybernetics and Systems* 5 (27) : 339-356.

[8] Prusaczyk, Piotr, Wojciech Kaczmarek, Jarosław Panasiuk, and Khadir Besseghieur. 2019. “Integration of robotic arm and vision system with processing software using TCP/IP protocol in industrial sorting application”. *AIP Conference Proceedings* 2078 : 020032-1-8.

[9] Siwek, Michał, Leszek Baranowski, Jarosław Panasiuk, and Wojciech Kaczmarek. 2019. “Modeling and simulation of movement of dispersed group of mobile robots using Simscape multibody software”. *AIP Conference Proceedings* 2078 : 020045-1-5.

[10] Besseghieur, Khadir Lakhdar, Radosław Trębiński, Wojciech Kaczmarek, and Jarosław Panasiuk. 2018. “Trajectory tracking control for a nonholonomic mobile robot under ROS”. *Journal of Physics: Conf. Series* 1016 : 012008-1-5.

[11] Besseghieur, Khadir Lakhdar, Wojciech Kaczmarek, and Jarosław Panasiuk. 2017. “Multi-robot Control via Smart Phone and Navigation in Robot Operating System”. *Problemy mechatroniki. Uzbrojenie, lotnictwo, inżynieria bezpieczeństwa / Problems of Mechatronics. Armament, Aviation, Safety Engineering* 8 (4) : 37-46.

[12] Ghael, Hirak Dipak, L. Solanki, and Gaurav Sahu. 2021. "A Review Papier on Raspberry Pi and its Applications”. *International Journal of Advances In Engineering and Management (IJAEM)* 2 (12) : 225-227.

[13] Gil De La Iglesia, Didac. 2012. *Tangible User Interfaces. Laboratory Guide*. Master in Social Media and Web Development, Linnaeus University.

[14] Slamtec. 2019. *RPLIDAR 360 Degree Laser Range Scanner Interface Protocol and Application Notes*. Shanghain Slamtec. Co.
[15] TECO ELECTRIC CO.: *Motor Specification TFK280SC-21138-45*. http://cdn.sparkfun.com/ datasheets/Robotics/RP6%20motor%20TFK280SC-21138-45.pdf (access on 1.02.2022).
[16] *L298N Motor Driver Controller Board*. Instructables. 2015. https://www.makerfabs.com/l298n-motor-driver-board.html (access on 1.02.2022).
[17] Docter, Quentin, and Jon Buhagiar. 2019. *Introduction to TCP / IP* (from https://www.researchgate.net/publication/332460567_Introduction_to_TCPIP)
[18] Dudek, Wojciech. 2013. *Wykorzystanie czujnika Kinect i systemu ROS do sterowania ruchem robota mobilnego* (praca dyplomowa). Warszawa: Politechnika Warszawska.
[19] Quigley, Morgan, Brian Gerkey, Ken Conley, Josh Faust, Tully Foote, Jeremy Leibs, Eric Berger, Rob Wheeler, and Andrew Ng. 2009. ROS: an open-source Robot Operating System. In *Proceedings of the ICRA workshop on open source software* 3 (3.2).
[20] Quigley, Morgan, Brian Gerkey, and Wiliam D. Smart. 2015. *Programming Robots with ROS*. O'Reilly.
[21] Borkowski, Mateusz, Krystian Ługas. 2017. "Model robota szeregowego typu SCARA w środowisku ROS". *Autobusy : technika, eksploatacja, systemy transportowe* 18 (6) : 551-554.
[22] Kam, Hyeong Ryeol, Sung Ho Lee, Taejung Park, and Chang Hun Ki. 2015. "RViz: a toolkit for real domain data visualization". *Telecomunication Systems* 60 (2) : 1-9.
[23] Kang, Yeon, Donghan Kim, and Kwangjin Kim. 2019. URDF Generator for Manipulated Robot. In *Proceedings of the Third IEEE International Conference on Robotic Computing (IRC)*.
[24] Peake, Ian, Joseph La Delfa, Ronal Bejarno, and Jan Olaf Blech. 2021. Simulation Components in Gazebo. In *Proceedings of the 22nd IEEE International Conference on Industrial Technology (ICIT)* pp. 1169-1175.
[25] Kuzin, Sergei, and Gabor Sziebig. 2020. SROS: Educational, Low-cost Autonomous Mobile Robot Design Based on ROS. In *Proceedings of the 2020 IEEE/SICE International Symposium on System Integration (SII)*.
[26] Guerra, N. Patricia, Pablo Javier Alsina, Adelardo A.D. Medeiros, and Antônio P. Araújo Jr. 2004. Linear Modelling and Identification of a Mobile Robot With Differential Drive. In *Proceedings of the First International Conference on Informatics in Control, Automation and Robotics.* Setúbal, Portugal, August 25-28, 2004.

[27] Michał Siwek, Jarosław Panasiuk, Leszek Baranowski, Wojciech Kaczmarek, and Szymon Borys. 2022. "Trajectory Tracking Control of a Mobile Robot with the ROS System". *Problemy mechatroniki. Uzbrojenie, lotnictwo, inżynieria bezpieczeństwa / Problems of Mechatronics. Armament, Aviation, Safety Engineering* 4 (50) : 67-84.

[28] Marin-Plaza, Pablo, Ahmed Hussein, David Martin, and Arturo de la Escalera. 2018. "Global and Local Path Planning Study in ROS-Based Research Platform for Autonomous Vehicles". *Journal of Advanced Transportation* 2018 : 6392697.

[29] Fox, Dieter, Wolfram Burgard, and Sebastian Thrun. 1997. "The Dynamic Window Approach to Collision Avoidance". *IEEE Robotics & Automation Magazine* 4 (1) : 23-33.

[30] Zheng, Kaiyu. 2017. ROS Navigation Tuning Guide. In *Robot Operating System (ROS) - The Complete Reference* (Volume 6). Springer International Publishing.

[31] Fox, Dieter, Wolfram Burgard, Frank Dellaert, and Sebastian Thrun. 1999. "Monte Carlo Localization: Efficient Position Estimation for Mobile Robots", In *Proceedings of the Sixteenth National Conference on Artificial Intelligence and Eleventh Conference on Innovative Applications of Artificial Intelligence*, July 18-22, 1999, Orlando, Florida, USA.

[32] Grisseti, Giorgio, Cyrill Stachniss, Wolfram Burgard. 2005. "Improved Techniques for Grid Mapping with Rao-Blackwellized Particle Filters", *IEEE Transactions on Robotics*. 23 (1) : 34-46.

[33] Yuen, D.C.K., and B. A. MacDonald. 2002.A comparison between Extended Kalman Filtering and Sequential Monte Carlo techniques for simultaneous localisation and map-building. In *Proceedings of the 2002 Australasian Conference on Robotics and Automation*, Auckland, Australia, pp. 111–116.

[34] Thrun, Sebastian, Wolfram Burgard, and Dieter Fox. 2005. *Probabilistic Robotics (Intelligent Robotics and Autonomous Agent).* The MIT Press.

[35] Montemerlo, M., N. Roy, S. Thrun, D. Hähnel, C. Stachniss, and J. Glover. *CARMEN—The Carnegie Mellon Robot Navigation Toolkit*. (Available online: http://carmen.sourceforge.net/).

[36] Thrun, Sebastian, Yufeng Liu, Daphne Koller, Andrew Y. Ng, Zoubin Ghahramani, and Hugh Durrant-Whyte: "Simultaneous localization and mapping with sparse extended information filters". *Int. Journal of Robotics* 23 (7-8).

[37] Behl, Madhur. *ROS Transformations and frames*. University of Virginia. (from: https://linklab-uva.github.io/autonomousracing/assets/files/L11-compressed.pdf).

Projekt robota autonomicznego bazującego na platformie Robot Operating System

Szymon CHERUBIN, Wojciech KACZMAREK, Natalia DANIEL

Wojskowa Akademia Techniczna,
ul. Sylwestra Kaliskiego 2, Warszawa,

Streszczenie. W artykule przedstawiono koncepcję autonomicznego robota mobilnego pracującego w systemie ROS i wykorzystującego zaawansowane algorytmy do generacji dwuwymiarowej mapy oraz autonomicznej nawigacji. Autorzy skupili się na przedstawieniu platformy sprzętowej urządzenia oraz na zaimplementowanym oprogramowaniu opartym na systemie LINUX oraz platformie Robot Operating System (ROS). W artykule zaprezentowano również algorytm realizujący generację dwuwymiarowej mapy otoczenia, autonomiczną jazdę robota oraz zdalne sterowanie urządzeniem. Wykonane pomiary temperatury komputera pozwoliły na podjęcie decyzji dotyczącej zastosowanego układu chłodzenia.
Słowa kluczowe: robotyka, robot mobilny, generacja mapy, ROS, Linux