PROBLEMY MECHATRONIKI
UZBROJENIE, LOTNICTWO, INŻYNIERIA BEZPIECZEŃSTWA

PROBLEMS OF MECHATRONICS
ARMAMENT, AVIATION, SAFETY ENGINEERING

ISSN 2081-5891; E-ISSN 2720-5266 **https://promechjournal.pl/**

Research paper

Evaluation of Robot Motion Trajectory Based on Selected Mapping Algorithms

Wojciech KACZMAREK* (wojciech.kaczmarek@wat.edu.pl)
Natalia DANIEL (natalia.daniel@wat.edu.pl)
Szymon CHERUBIN (szymon.cherubin27@gmail.com)

*Corresponding author
ORCID: https://orcid.org/0000-0003-3805-9510

Military University of Technology,
2 Sylwestra Kaliskiego Str., 00-908 Warsaw, Poland

Received: August 3, 2022 / Revised: August 16, 2022 / Accepted: August 18, 2022 / Published: March 31, 2024.

2024, 15 (1), 65-82; https://doi.org/10.5604/01.3001.0054.4489

Cite: Chicago Style

Kaczmarek, Wojciech, Natalia Daniel, and Szymon Cherubin. 2024. "Evaluation of Robot Motion Trajectory Based on Selected Mapping Algorithms". *Problemy mechatroniki. Uzbrojenie, lotnictwo, inżynieria bezpieczeństwa / Probl. Mechatronics. Armament Aviat. Saf. Eng.* 15 (1) : 65-82. https://doi.org/10.5604/01.3001.0054.4489

Abstract. This paper presents the concept of a remotely controlled mobile robot that generates a two-dimensional map of its surroundings. The hardware platform developed relies on the LINUX operating system with the Robot Operating System (ROS) to function properly. The authors focused on discussing the robot's hardware and presenting the software used. In line with the assumptions made, the robot is capable of generating a two-dimensional digital map of its surroundings, as well as of recording images of those surroundings. The robot relies on lidar odometry for identifying its position, meaning that the developed algorithm calculates the vehicle's location based on data from a laser scanner. The main sources of environment-related information acquired by the robot include the following: RPLidar A3M1 laser scanner by Slamtec (generating a digital map) and a 5mpx HD OV5647 camera (capturing images of the surroundings). These devices are coupled with the Raspberry Pi 3B on-board computer via a CSI interface.
Keywords: mobile robot, 2D map generation, ROS, Linux, Lidar

1. INTRODUCTION

Robotics is an interdisciplinary field that combines other fields of science. Therefore, the process of designing, constructing and programming robots is a difficult and demanding task. In addition to industrial robots which are usually stationary or move along fixed paths (the so-called tracks), mobile robots exist as well, with the latter being either autonomous devices or relying on operator’s control inputs. Many solutions used in industrial applications (e.g.: vision systems, recognition algorithms, or robot programming and control methods) are used in mobile robotics, and vice versa [1, 2, 3].

The increasing robot adoption rate proves that robots are not only used in the civilian sector (e.g. for performing inspections or working in warehouses), but also in the military[4], where their main task is to perform reconnaissance and combat missions. This ensures that the risk to human life and health is reduced. In addition, sensors mounted on robots are less prone to make errors, as their parameters surpass human perceptual capabilities. The ongoing conflict in Ukraine demonstrates the important role of unmanned vehicles in collecting information and securing an advantage over the enemy under combat conditions. The UAVs used in Ukraine are a good example here, e.g. the Bayraktar TB2, whose effectiveness was presented by N. Kumar [5]. Unmanned vehicles capable of inspecting enclosed spaces and visualizing their surroundings with the help of vision modules play a particularly important role when operating in urban environments.

Technological advances allow members of the public to build and program mobile robots themselves. This stems from the rapid development of software solutions relied upon when programming mobile robots, including ROS (Robot Operating System), increasing availability of a wide range of sensors needed to acquire data concerning the robot’s surroundings, and the abundance of universal communication protocols. Nevertheless, due to the complex nature of the matter at hand, the process of building robots still remains an extremely challenging task.

This stems, in particular, from the time-consuming nature of the process and the need for extensive knowledge in such fields as electronics, mechanics and computer science. When used in mobile robots, ROS offers a wide range of opportunities - from controlling robot to designing applications that allow them to collaborate in a group [6 - 9]. Hence, this software has become a standard used in mobile robotics. This is very much justified, because the platform is available free of charge and is widely accessible. Consequently, it is used by millions of engineers around the world, which boosts its development even further.

In most cases, autonomous robots rely on on-board power sources, meaning that their movement is not restricted by wires. One may also state that an autonomous mobile robot is a device that does not require any external infrastructure to navigate. It avoids collisions when moving along a preset route without any direct operator intervention, based on the uploaded movement trajectory. Thanks to their sensors, robots are able to move safely, avoiding people and obstacles and adapting their path to on-site conditions. In order for a robot to move autonomously, a map of its surroundings must be created first. This task is usually accomplished by deploying a special mechanism to map the space around the robot. As the robot is moving, its laser scanners detect walls and pieces of machinery or equipment, allowing it to generate a map of its surroundings.

2. MOBILE ROBOTS - CONCEPT AND CONSTRUCTION

A properly selected hardware and software structure of the robot guarantees that it will function as intended and that it will perform the tasks for which it has been designed. In the case described in this paper, it was assumed that the robot would be capable of performing the following tasks:

- generating a digital, two-dimensional map of its surroundings;
- capturing images of the surroundings;
- locating itself in space using lidar odometry;
- operating in the remote control mode.

2.1. Hardware platform

In order to operate properly, the robot requires the right configuration of its electrical, electronic and mechanical components which must work closely together. Figure 2.1 shows a simplified block diagram of the hardware of the mobile robot designed.

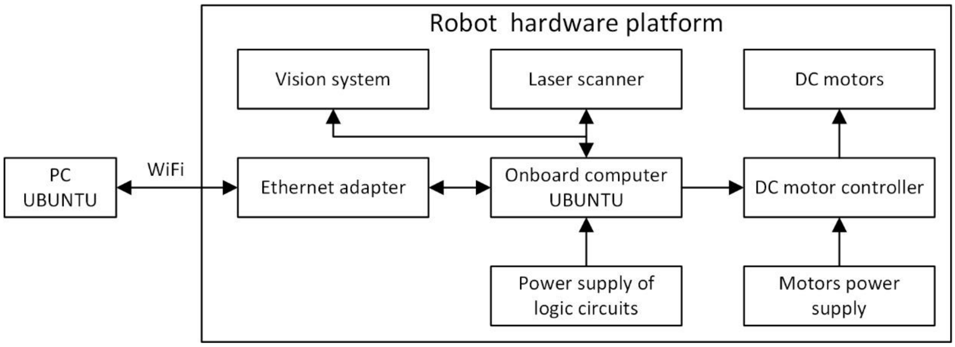


Fig. 2.1. Block diagram of the robot's hardware structure

The on-board computer responsible for all computations is the element that determines the robot's capabilities. It is tasked with processing data from individual sensors and generating relevant control commands in the form of electrical signals via its GPIO output. Additionally, the module must support a wireless transmission protocol to communicate with the user's computer. In the case at hand, the robot’s on-board computer is of the Raspberry Pi 3B single board variety [10]. It is equipped with a quad-core, 64-bit Broadcom BCM2837 processor relying on the ARMv8-A architecture and is clocked at 1.2 GHz. The computer offers numerous outputs, including: 4x USB, GPIO and CSI, allowing it to communicate with the robot's sensors.

The RPLidar A3M1 laser scanner by Slamtec is the main source of information about the surroundings [11]. This sensor is responsible for generating the data used to create a digital map of the surroundings and the establish robot's position in space. It is an allothetic source of information which is characterized by a high degree of accuracy and good operating speed, as it uses an infrared laser beam as a data medium.

A 5-megapixel HD OV5647 camera was used as the robot’s vision module for capturing images of the surroundings. The camera is connected to the on-board computer via the CSI interface. The connector uses the MIPI CSI-2 protocol, being the latest communication interface standard used in cameras. The protocol’s features include improved radio interference mitigation, higher compression rates and reduced latency (when compared with its predecessors). Before fabricating the actual platform, a CAD model was developed in the Solid Edge environment (Fig. 2.2).

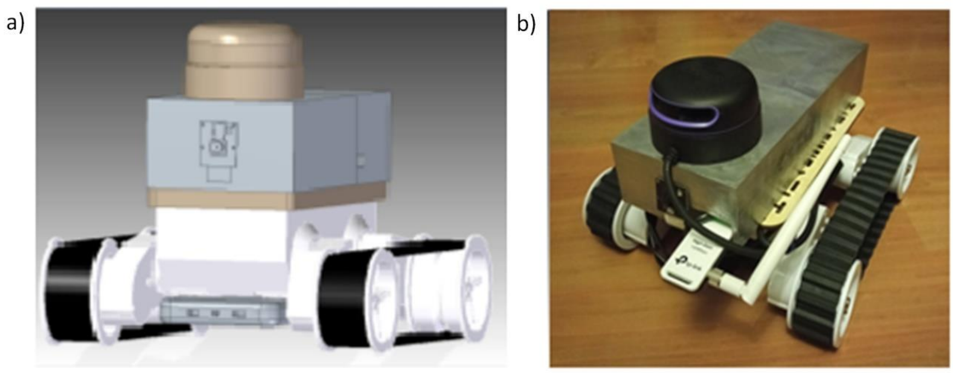


Fig. 2.2. General views of the robot platform: a) CAD model of the robot, b) actual robot

The robot’s chassis is the ROVER 5-2 platform. It is a tracked drive system relying on nonholonomic motion. The platform is equipped with two DC motors with a gear ratio of 87:1. The motors generate 1 Nm of torque and achieve 8,804 rpm. A two-channel TB6612FNG controller [12]was used to control the platform’s motors. Mentioned controller is based on an H-bridge relying on a MOSFET transistor. Such an approach allows to reduce voltage drops at the output, thus resulting in more efficient operation of the robot's motors.

2.2. Robot software

As the RPi 3B single-board computer was used, the decision was made to equip the robot with an operating system based on the Linux Ubuntu MATE kernel. It is compatible with ROS software - the system chosen as the robot's programming environment.

The Ubuntu MATE system used in the robot is a complete distribution of the GNU/Linux operating system. The system efficiently supports the ROS software and is the version deployed significantly reduces the use of CPU computing resources compared to the classic version of Ubuntu. Xenial Xerus version 16.04 was installed on the robot’s platform. The decision to choose and install the version published in 2016 was dictated by the fact that it allows the ROS environment to work with the Kinetic version of ROS, with the latter being one of the most effective releases of software for controlling mobile robots. Communication of the robot's CPU with the user's personal computer relies on the TCP/IP protocol [13]. An SSH connection was used for this purpose, allowing operating systems running on the same wireless network to interact.

The ROS (Robot Operating System) environment was software relied upon to implement the control algorithms, as well as acquire and process data from the individual sensor systems. It is a robust, open-source software platform.

It is applied in robotics primarily as a system responsible for controlling humanoid robots, industrial robots, as well as mobile robots - operating not only on the ground, but also on water and in the air. The platform was created in 2007 in California as a result of a collaboration between Stanford University and Willow Garage. Since its creation, the platform has been continuously developed by an international group of robotics developers [8, 14, 15]. The ROS platform relies on the communication between individual processes (Fig. 2.3) which simultaneously play the role of operational nodes responsible for the selected functions of the robot. The manner in which this software operates is described in detail by M. Quigley [15]. The nodes create a network interconnecting the individual processes. This ensures that each node has access to the entire network, that they cooperate with each other, and that the type of data sent to the network can be monitored. ROS allows to exchange messages in the form of: topics, services, parameters and actions.

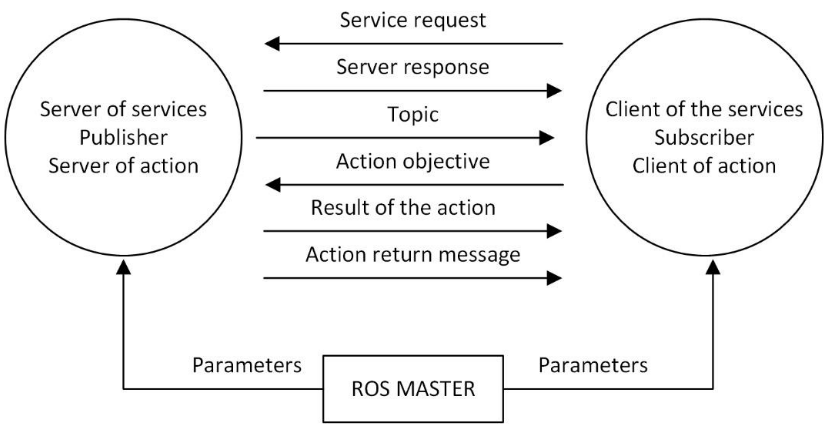


Fig. 2.3. Diagram showing the cooperation of nodes in a ROS environment

The ROS environment supports software developed with the use of high-level languages: Python and C++. The task of building structures, in turn, is the responsibility of *catkin*. It is an official compiler consisting of *CMake* macros and custom scripts developed in Python. *Catkin* is the successor of *rosbuild* - a basic compiler used in the ROS environment. Compared to its predecessor, it offers much improved functionalities. Similarly to *CMake*, *catkin* automatically manages the compilation processes, meaning that the program is not compiled on its own, but compilation rule files are generated for a specific environment. In the case of GNU/Linux, *Makefile* files are generated. By using Python-derived scripts, a user using the *catkin* compiler has the ability to automatically search for packages and develop multiple large-format projects simultaneously [16]. The Robot Operating System also has a number of graphical tools to control the operation of individual vehicle components and software packages.

In addition, ROS is equipped with tools to visualize vehicle operation based on data from sensory systems, as well as to simulate vehicle operation. The most commonly used tools include the following:

- **RQT package** – a platform included in the ROS software that allows to implement various GUI-type tools in the form of plug-ins [17]. In addition, the platform allows to visualize the structure of the nodes and the connections between them, which greatly facilitates understanding of the manner in which the ROS system operates. Furthermore, the *rqt* package allows to analyze nodes by mirroring the transformation tree, a process launched by the basic node of the *rqt_tf_tree* system, and calling the node connections network – *rqt_graph*.
- **Rviz** – a package used for 3-dimensional visualization of messages in ROS, developed at a Korean university [18]. It allows to visualize sensor data and depicts the robot's environment. In addition, it allows to visualize data from the perspective of the selected coordinate system, based on data from the *tf* library. It also enables graphical representation of the robot's URDF model and kinematic analysis of its movement. In addition, the graphical tool interface provides a digital map of the robot’s surrounding, based on sensor data.
- **URDF (Unified Robot Description Format)** – a format used to determine the robot's kinematics and dynamics. It is also relied upon to represent the robot [19]. The format is used in the rviz visualizer and the Gazebo simulator. URDF files are created using HTML. Based on data contained in the unified format, it is possible to calculate the robot's spatial position and detect potential programming errors.
- **Gazebo** – a free ROS program used to perform 3D simulations of the robot’s functioning. It is one of the most popular simulators used in the field of robotics and relies on the OGRE (Object-Oriented Graphics Rendering Engine) graphics engine. Such an approach stems from its good performance, very accurate representation of reality and the laws of physics for which the ODE (Open Dynamics Engine) engine is responsible [20].

3. ROBOT’S OPERATING ALGORITHMS

3.1. Robot's control algorithm

The robot’s control algorithm shown in Figure 3.1 is implemented using the procedures responsible for remote control of the vehicle [21]. As stated earlier, this feature is realized in the presented solution by using a wireless network based on the *ssh* protocol.

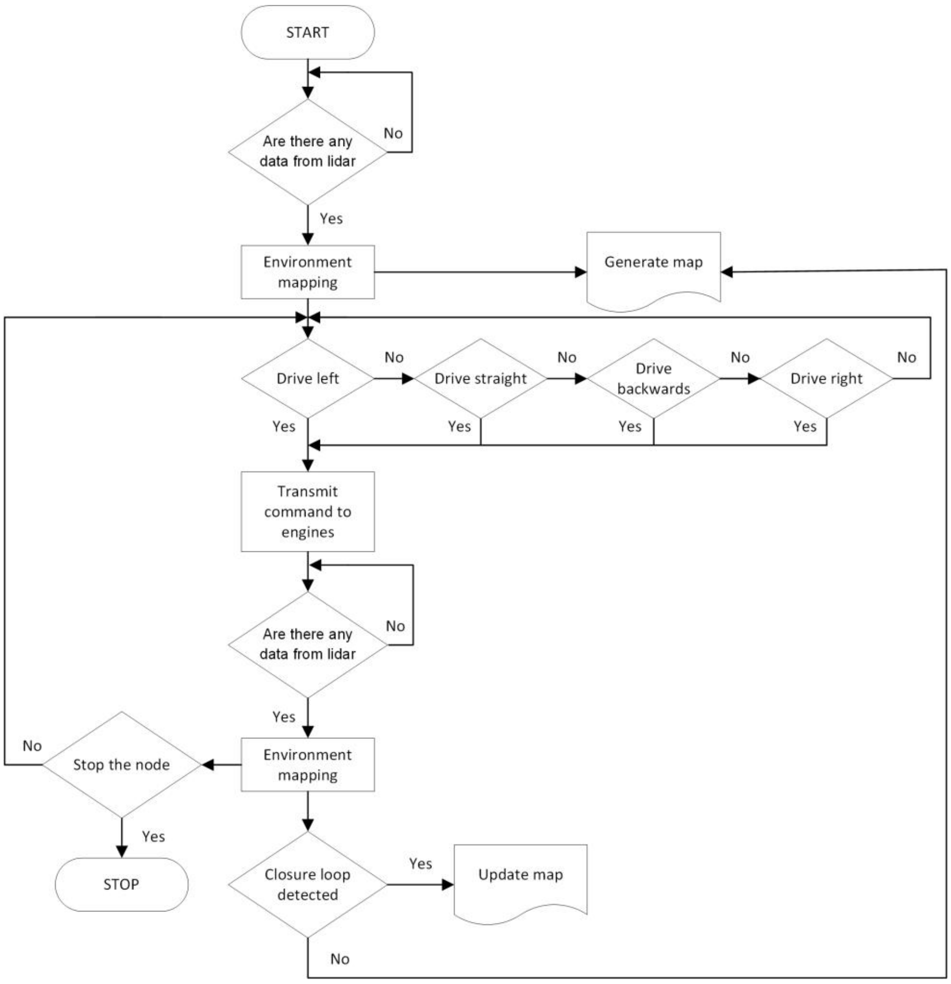


Fig. 3.1. Robot's control algorithm

The node network responsible for controlling effectors is shown in Figure 3.2. The *keyboard_driver* node is responsible for handling the keys on the user's personal computer and sending the relevant message to the *keys_to_twist* node. It converts the key strikes to corresponding command streams in the form of *cmd_vel* messages. Such messages have the form of a PWM signal with its length adapted to the vehicle's speed received by the *motors* node. This node represents those pins of the GPIO connector to which inputs of the DC motor controller are connected.

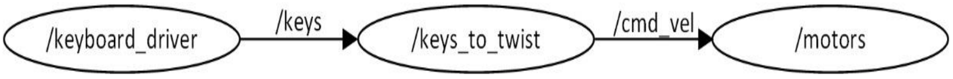
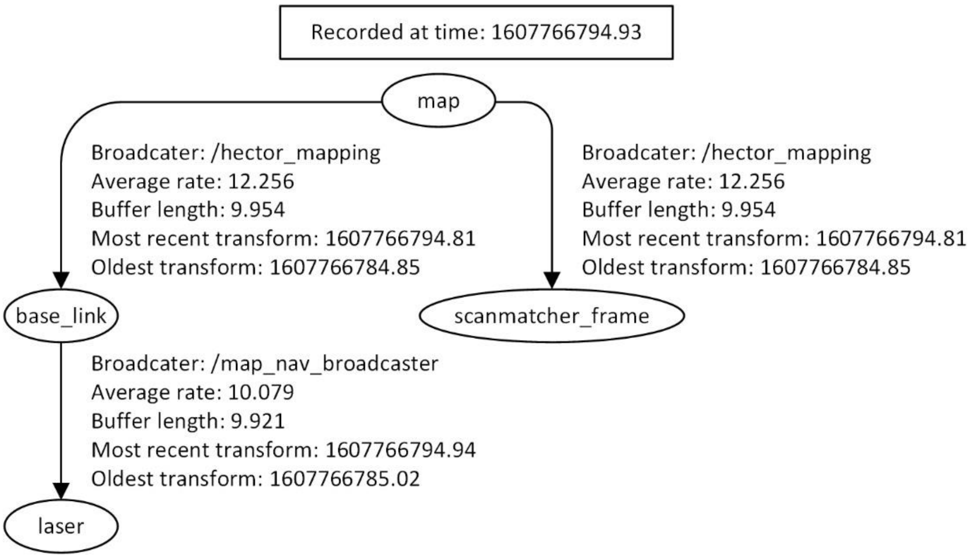


Fig. 3.2. Diagram of nodes responsible controlling the robot remotely

3.2. Map generating algorithm

One of the vehicle's most important algorithms is the advanced SLAM algorithm, as it is responsible for the simultaneous generation of a two-dimensional map of the surroundings and for locating the robot within those surroundings. The *hector_slam* algorithm is used in the project [22]. It is responsible for solving SLAM problems based solely on laser scanner data.

In order to generate a digital map, the ROS software requires an appropriate transformation of the coordinate systems representing each of the robot's components. This is done with the use of the *tf* library which is responsible for the scene graph concept, reflected in the form of a hierarchy tree. The root of the tree represents the map of the surroundings, while each of its tops is a geometric transformation, a translation, or a rotation between the arrangements of each component. Figure 3.3 shows a transformation tree of the robot's coordinate system, generated using the *rqt_tf_tree* package.


Recorded at time: 1607766794.93

map → base_link:
Broadcater: /hector_mapping
Average rate: 12.256
Buffer length: 9.954
Most recent transform: 1607766794.81
Oldest transform: 1607766784.85

map → scanmatcher_frame:
Broadcater: /hector_mapping
Average rate: 12.256
Buffer length: 9.954
Most recent transform: 1607766794.81
Oldest transform: 1607766784.85

base_link → laser:
Broadcater: /map_nav_broadcaster
Average rate: 10.079
Buffer length: 9.921
Most recent transform: 1607766794.94
Oldest transform: 1607766785.02

Fig. 3.3. Transformation tree of the robot's coordinate systems

The *laser* subtree reflects the coordinate system in a polar form represented by the laser scanner, while *base_link* is a representation of the coordinate system of the robot's platform.

Scanmatcher_frame, on the other hand, is the frame responsible for locating the robot on the root of the tree (being the *map* frame), based solely on data from the laser scanner[23]. The structure of the node network is responsible for solving the SLAM problem in the ROS environment and was generated using the *rqt_graph* package (Fig. 3.4).

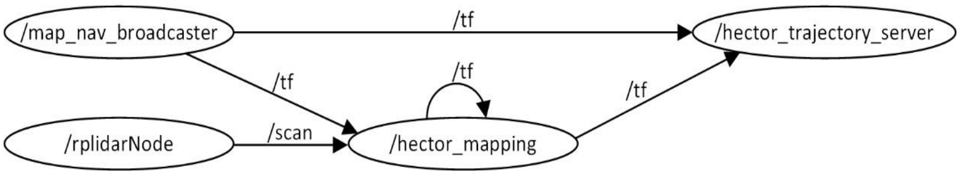


Fig. 3.4. Node network responsible for resolving SLAM problems

The *hector_slam* package is based on the representation of a presence mesh which consists in representing a map of the robot's surroundings as fields of binary random variables, with each of them representing the presence of an obstacle. Occupancy grid maps are algorithms used in probabilistics to generate a map, assuming that the position of the robot is known. They visualize the map as a fine-grained grid of cells in a continuous space of locations. They are responsible for estimating the probability of future occupancy of each of those cells, using a Bayes filter in the form of [24]:

$$p(m|z_{1:t}, x_{1:t}) = \sum_i p(m_1|z_{1:t}, x_{1:t}) \tag{3.1}$$

where: $x_{1:t}$ – map location,
$z_{1:t}$ – observer's position,
m_i – map slice.

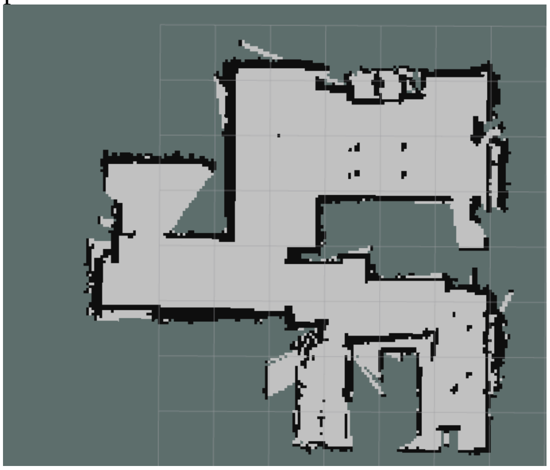


Fig. 3.5. General view of a two-dimensional map of a room

Figure 3.5 shows a digital, two-dimensional map of the vehicle's surroundings generated by the robot (using the *hector_slam* package).

4. TESTING THE ROBOT

The test aimed to compare the digital maps of the same environment created by different algorithms. The trajectories followed by the robot while generating the maps were compared as well. The following mapping algorithms were used in the study:

- Hector_slam – an algorithm using lidar odometry.
- Gmapping – an algorithm using proprioceptive odometry [24].
- RTAB-map – an algorithm using lidar odometry and the ICP algorithm [25].

Figures 4.1 – 4.3 present the maps generated with the use of the mapping algorithms tested. The drawings prove that all algorithms generated correct maps of the premises investigated. The map generated by the RTAB-map algorithm has rough surfaces, which is inconsistent with reality. However, this algorithm is responsible for generating a three-dimensional map of the surroundings and uses the two-dimensional form of the map solely for the purposes of guiding the autonomous robot. Other algorithms generate smooth obstacle surfaces.

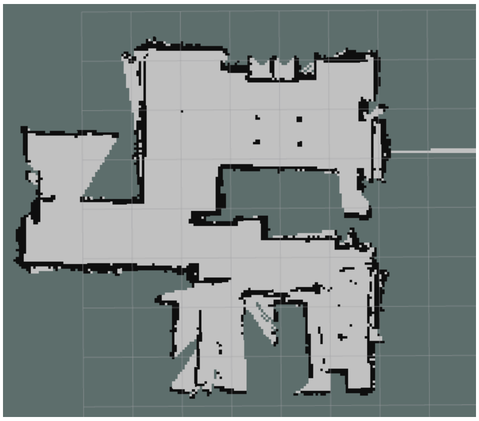


Fig. 4.1. Map generated using the gmapping algorithm

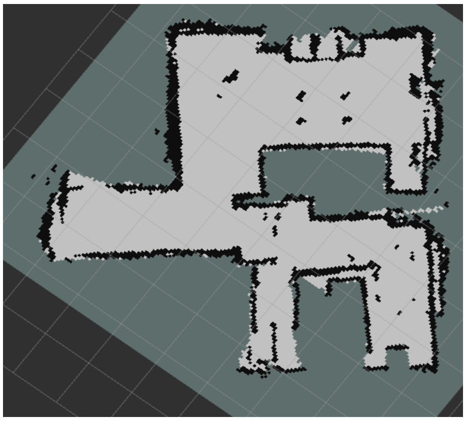


Fig. 4.2. Map generated using the RTAB-map algorithm

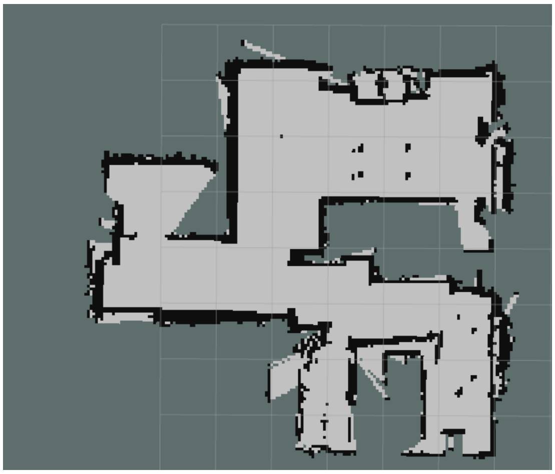


Fig. 4.3. Map generated using the hector_slam algorithm

The maps generated by RTAB-map and hector_slam algorithms are characterized by certain obstacle surfaces being marked in bold lines. This is the result of the algorithms' failure to detect the closure loop. The above means that the vehicle is treating the surface of an already known location as unknown, which results in a given obstacle being generated anew, instead of being updated. Upon encountering mirrors in the room, all algorithms represented them as free spaces. This error cannot be eliminated, as it stems from the physical properties of mirrors which reflect the scanner’s laser beam straight onto obstacles present opposite the mirror. Algorithms using lidar odometry, i.e. hector_slam and RTAB-map were used to evaluate the robot's motion trajectory. Figure 4.4 shows the trajectories of the robot's passages in a two-dimensional Cartesian system. Figure 4.5 shows changes in the robot's position relative to x and y axes.

When analyzing the diagrams of the robot's trajectory using mapping algorithms that rely on different localization methods, one may clearly see a deviation of the trajectory recorded by the RTAB-map algorithm compared to that recorded by hector_slam. Furthermore, one may clearly see, in the diagram showing changes in the robot's position relative to both axes, that the RTAB-map algorithm is less accurate than its counterpart and needs to rely on interpolation in determining the robot's location. There is also a noticeable difference in the number of position measurements taken by the two algorithms. RTAB-map took approximately 5,200 measurements, with hector_slam taking roughly 8,800.

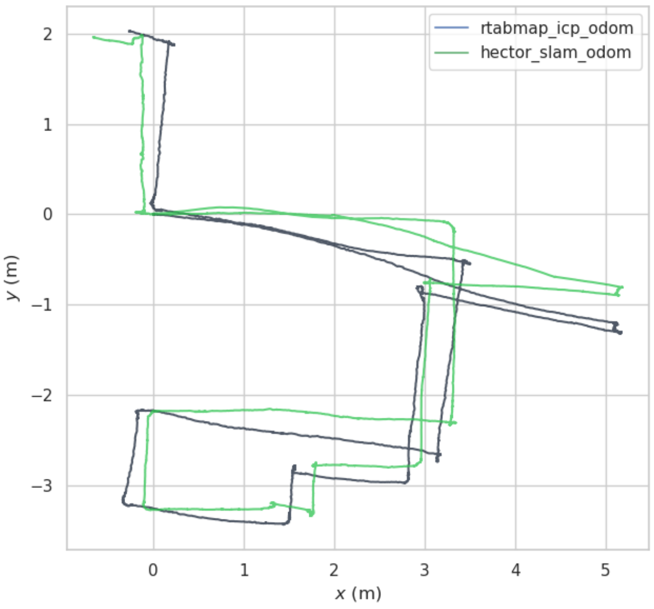


Fig. 4.4. Diagram of the robot’s trajectory in a two-dimensional Cartesian system

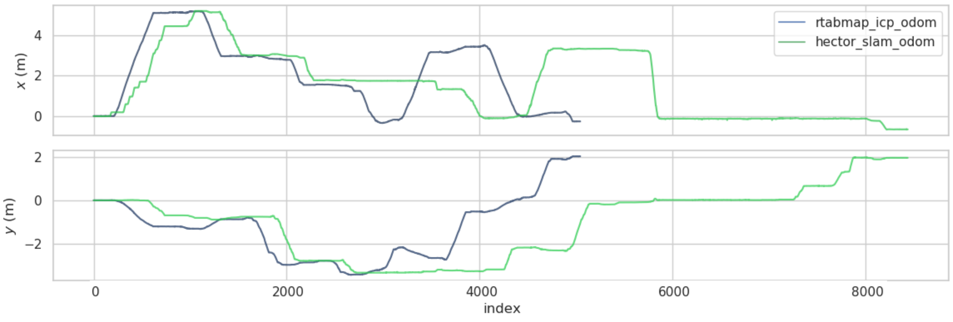


Fig. 4.5. Diagram of changes in the robot's position relative to x and y axes

In addition, the interpolation of successive points of the robot's location is clearly visible on the graph of changes in the robot's position relative to the x axis. This is due to the fact that the RTAB-map algorithm is used for simultaneous generation of a two-dimensional map and a three-dimensional map, meaning that the computational resources need to be divided and that more of them need to be allocated to the generation of the three-dimensional map. The problem of the presence of reflective surfaces in the room (i.e. mirrors) has been noted as well, as they may result in errors in the generated maps.

5. CONCLUSIONS

The robot developed and described in this paper proves that customized robot structures, relying on technological advances in the robotics industry, may be built almost by anyone. An unmanned inspection vehicle, such as the robot developed in the course of this project, offers a great development potential and may be used in many various applications. It may be relied upon, for instance, as a vehicle inspecting premises located in urban areas, to gain an information advantage and protect human life and health by accessing hard-to-reach or dangerous areas. The development potential consists, in particular, in the ability to increase the number of sensors mounted on the robot in order to enable it to acquire more information from the environment, as well as in developing autonomous navigation algorithms. Currently, work is underway to implement algorithms that allow to generate 3D maps of the surroundings, and to develop advanced autonomous navigation algorithms. The former of those modifications involves the use of an RGB-D vision module and requires that the sensor information be combined to develop a 3D map of the surroundings. The latter development consists in using a proprioceptive odometry sensor in the form of encoders that allow to obtain more accurate data on the robot's location in space. By transmitting such information to advanced autonomous navigation algorithms, the robot will be able to move autonomously to the designated location.

The selected tests presented in the paper have shown that the map generation algorithms used correctly generated a map of the premises in which the developed robot was moving. With that said, the map generated by the RTAB-map algorithm failed to accurately reflect smooth surfaces (the surfaces were generated as rough). Evaluation of the robot's trajectory generated by lidar odometry algorithms (hector_slam and RTAB-map) shows that the RTAB-map algorithm exhibits lower accuracy levels and requires to apply interpolation in determining the robot's location.

FUNDING

The work was supported by the Military University of Technology (Warsaw, Poland) under research project No. UGB 893/2021.

REFERENCES

[1] 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.

[2] Kaczmarek, Wojciech, Jarosław Panasiuk, Szymon Borys, and Patryk Banach. 2020. "Industrial robot control by means of gestures and voice commands in off-line and on-line mode". *Sensors* 20 (21) : 6358-1-15.

[3] Szymon Borys, 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.

[4] Trojnacki, Maciej, Piotr Szynkarczyk, Adam Andrzejuk. 2008. "Tendencje rozwoju mobilnych robotów lądowych (1) Przegląd robotów mobilnych do zastosowań specjalnych". *Pomiary Autom. Robot.*12 (6) : 11–14.

[5] Kumar, Brig Narender. 2022. *Russia Ukraine War Lessons for India in Conventional and Hybrid Warfare Domain*. https://www.researchgate.net/publication/361039389_Russia_Ukraine_War_Lessons_for_India_in_Conventional_and_Hybrid_Warfare_Domain

[6] Besseghieur, Lakhdar Khadir, Radosław Trębiński, Wojciech Kaczmarek, and Jarosław Panasiuk. 2018. "Trajectory tracking control for a nonholonomic mobile robot under ROS". *J. Phys. Conf. Ser.* 1016 : 012028-1-5.

[7] Besseghieur, Lakhdar Khadir, Wojciech Kaczmarek, and Jarosław Panasiuk. 2017. "Multi-robot Control via Smart Phone and Navigation in Robot Operating System". *Problemy mechatroniki. Uzbrojenie, lotnictwo, inżynieria bezpieczeństwa / Probl. Mechatronics Armament Aviat. Saf. Eng.* 8 (4) : 37-46.

[8] Besseghieur, Lakhdar Khadir, Radosław Trębiński, Wojciech Kaczmarek, and Jarosław Panasiuk. 2020. "From Trajectory Tracking Control to Leader–Follower Formation Control". *Cybern. Syst.* 51 (4) : 339–356.

[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 Conf. Proc.* 2078 (1) : 020045-1-5.

[10] Maksimović, Mirjana, Vladimir Vujović, Nikola Davidović, Vladimir Milošević, and Branko Perišić. 2014. "Raspberry Pi as Internet of Things hardware : Performances and Constraints," *Des. Issues,* 3 : 8.

[11] *RPLIDAR Interface Protocol and Application Notes*. 2018. SLAMTEC.

[12] Qureshi, M.F. 2020. “Double Self-Balancing Robot”. *Affil. with NED Univ. Eng. Technol., Karachi Dep. Electr. Eng.*, DOI: 10.13140/RG.2.2.14414.48965.

[13] Docter, Quentin, and Jon Buhagiar. 2019. *Introduction to TCP/IP*. DOI: 10.1002/9781119553588.ch7.

[14] Dudek, Wojciech. 2013. *Wykorzystanie czujnika Kinect i systemu ROS do sterowania ruchem robota mobilnego*. Praca dyplomowa. Warszawa: Wydawnictwo Politechniki Warszawskiej.

[15] Yoshida, Hideki, Hiroshi Fujimoto, Daisuke Kawano, Yuichi Goto, Misaki Tsuchimoto, and Koji Sato.2015. Range extension autonomous driving for electric vehicles based on optimal velocity trajectory and driving braking force distribution considering road gradient information. In *Proceedings of the IECON 2015 - 41st Annu. Conf. IEEE Ind. Electron. Soc.*, pp. 4754–4759, DOI: 10.1109/IECON.2015.7392843.

[16] Quigley, Morgan, Brian Gerkey, and William D. Smart. 2015. *Programming Robots with ROS*. USA: O’Reilly Media, Inc., 1005 Gravenstein Highway North, Sebastopol, CA 95472.

[17] Borkowski, Mateusz, Krystian. Łygas. 2017. “Model robota szeregowego typu Scara w środowisku ROS". *Autobusy Tech. Eksploat. Syst. Transp.* 18 (6) : 551–554.

[18] Kam, Hyeong Ryeol, Song-Ho Lee, Taejung Park, and Chang-Hun Kim. 2015. “RViz: a toolkit for real domain data visualization”. *Telecommun. Syst.* 60 (2) : 337–345.

[19] Kang, Y., D. Kim, and K. Kim. 2019. URDF Generator for Manipulator Robot. In *Proceedings of the 2019 Third IEEE Int. Conf. Robot. Comput.*, pp. 483–487.

[20] Peake, Ian, Jozeph La Delfa, Ronal Bejarano, and Jan Olaf Blech. 2021. Simulation Components in Gazebo. In *Proceedings of the 22nd IEEE International Conference on Industrial Technology (ICIT)* 1 : 1169–1175.

[21] Cheng, Yao-Yao, and Dian-Xi Shi. 2017. Ros Based Remote Robot Task Monitoring and Control System. In *Proceedings of the ITM Web Conf.* 12 (5) : 01023.

[22] Kohlbrecher, Stefan, Johannes Meyer, Thorsten Graber, Karen Petersen, Uwe Klingauf, and Oskar von Stryk. 2014. Hector open source modules for autonomous mapping and navigation with rescue robots. In: Behnke, S., Veloso, M., Visser, A., Xiong, R. (eds) *RoboCup 2013: Robot World Cup XVII. RoboCup 2013*. Lecture Notes in Computer Science(), vol 8371. Springer: Berlin, Heidelberg.

[23] Kohlbrecher, Stefan, Oskar von Stryk, Johannes Meyer, and Uwe Klingauf. 2011. A flexible and scalable SLAM system with full 3D motion estimation. In *Proceedings of the 2011 IEEE International Symposium on Safety, Security, and Rescue Robotics* pp. 155–160.

[24] da Silva M.F. Bruno, Rodrigo S. Xavier, and Luiz M.G. Gonçalves. 2019. "Mapping and Navigation for Indoor Robots under ROS : An Experimental Mapping and Navigation for Indoor Robots under ROS : An Experimental Analysis. *Preprints* 2019, 2019070035. https://doi.org/10.20944/preprints201907.0035.v1

[25] Labbé, M.and F. Michaud. 2019. "RTAB-Map as an open-source lidar and visual simultaneous localization and mapping library for large-scale and long-term online operation. *J. F. Robot.* 36 (2) : 416–446.

[26] Grupp, Michael. 2022."Python package for the evaluation of odometry and slam". *A library for handling, evaluating and comparing trajectory results of odometry and SLAM algorithms, https://github.com/MichaelGrupp/evo (access: January, 10.2022).*

Ocena trajektorii ruchu robota na podstawie wybranych algorytmów mapujących

Wojciech KACZMAREK, Natalia DANIEL, Szymon CHERUBIN

Wojskowa Akademia Techniczna
ul. gen. Sylwestra Kaliskiego 2, 00-908 Warszawa

Streszczenie. W artykule przedstawiono koncepcję zdalnie sterowanego robota mobilnego, generującego dwuwymiarową mapę otoczenia. Opracowana platforma sprzętowa do poprawnego funkcjonowania wykorzystuje system operacyjny LINUX z systemem Robot Operating System (ROS). Autorzy skupili się na omówieniu struktury sprzętowej robota oraz przedstawieniu zaimplementowanego oprogramowania. Zgodnie z przyjętymi założeniami wykonany robot umożliwia generowanie dwuwymiarowej cyfrowej mapy otoczenia i rejestrację obrazu otoczenia. Do lokalizacji w przestrzeni, robot wykorzystuje odometrię lidarową, co oznacza że opracowany algorytm wylicza położenie pojazdu na podstawie danych pochodzących ze skanera laserowego. Głównymi źródłami informacji pozyskiwanych przez robota z otoczenia są: skaner laserowy RPLidar A3M1 firmy Slamtec (generowanie mapy cyfrowej) oraz kamera HD OV5647 o rozdzielczości 5mpx (rejestracja obrazu otoczenia). Urządzenia te współpracują z komputerem pokładowym Raspberry Pi 3B za pomocą złącza CSI.
Słowa kluczowe: robot mobilny, generacja mapy 2D, ROS, Linux, Lidar.