DOI: 10.2507/33rd.daaam.proceedings.xxx

PROBLEMS OF APPLICATION OF METHODS OF DYNAMIC REPROGRAMMING OF CONTROL SYSTEMS OF MOBILE ROBOT MODULES

Pavel Pletenev & Victor Andreev

This Publication has to be referred as: Andreev, V[ictor] & Pletenev, P[avel] (2022). Problems of application of methods of dynamic reprogramming of control systems of mobile robot modules, Proceedings of the 33rd DAAAM International Symposium, pp.xxxx-xxxx, B. Katalinic (Ed.), Published by DAAAM International, ISBN 978-3-902734-xx-x, ISSN 1726-9679, Vienna, Austria
DOI: 10.2507/33rd.daaam.proceedings.xxx

Abstract

This paper presents an analysis of existing and promising methods of dynamic reprogramming of embedded systems suitable for use in a mobile robot with a modular control system architecture. The functional-modular architecture of the mobile robot control system implementing distributed computing is briefly described. This control system architecture makes it possible to run a mobile robot in real time using functional modules control systems implemented on embedded systems – low performance microcontrollers. Remote dynamic reprogramming of the software aspects of the control systems of individual robot modules without stopping the functioning of the robotic complex is necessary to ensure an operational change in the configuration of the modular robot – reconfiguring the robot structure in order to adjust its functionality to the task that has arisen. The features of using 4 methods of dynamic reprogramming according to 6 criteria are considered: applicability on different embedded microprocessor systems, the amount of RAM and program memory required, the speed of calculations, the theoretical complexity of implementing the method, the theoretical complexity of using the method by the end user (customizer), the flexibility of solutions obtained using the method. As a result of the study, usage recommendations for the considered methods are given.

Keywords: mobile robot; modular robot; control system; communications channel; dynamic reprogramming; embedded systems.

1. Introduction

The experience of using robotic complexes (RTC) in extreme conditions (for example, the accident at the Chernobyl nuclear power plant in 1986) has shown that in order to accurately match the RTC functionality to the task when working in aggressive environments for humans (space, zones of chemical or radioactive contamination), not only robots in special design (extreme robotics) are required, but also it may be necessary to promptly change the composition and/or structure of the RTC directly at the work site. The need for this functionality for extreme robotics was pointed out by such well-known Russian scientists as Academician of the Russian Academy of Sciences, Doctor of

Technical Sciences Evgeny Pavlovich Popov [1], Doctor of Technical Sciences Evgeny Ivanovich Yurevich [2] and Doctor of Ph.D. Platonov Alexander Konstantinovich [3]. In other words, reconfigurable robots are needed for such applications.

Reconfiguration - altering the configuration of a modular robot in order to achieve an intended change in the function of the modular robot [4].

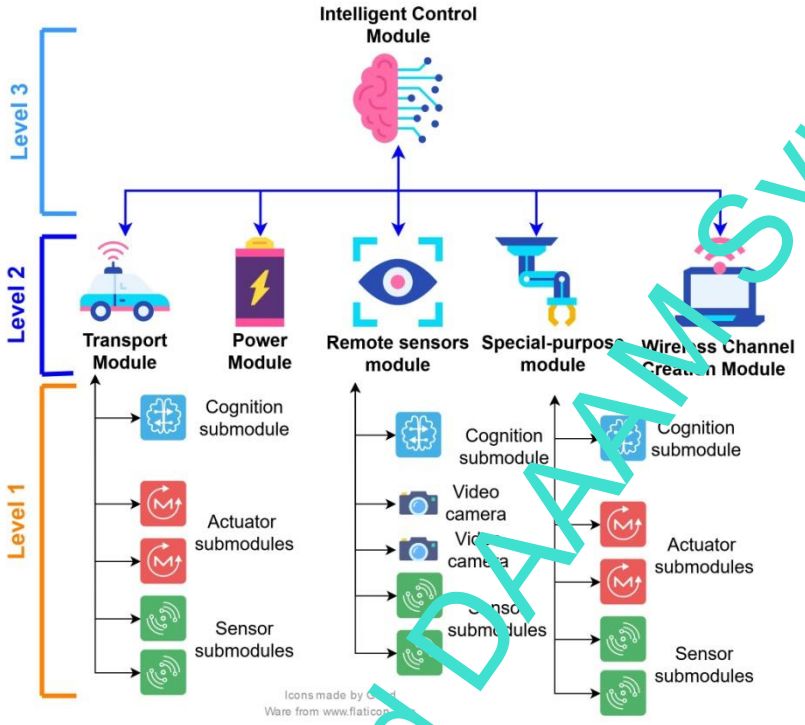

Fig. 1. Functional-modular architecture of a control system of a mobile robot

In fact, reconfiguration is a purposeful change in the configuration and/or technical characteristics of a mobile robot (MR) when changing the composition of its functional units without the need to reconfigure the software of its control system (CS). But such a reconfiguration is possible only with a modular solution of the architecture of the RTC control system. An example of such an architecture is shown in Fig.1 [5].

At the top of the architecture there is an intelligent control module responsible for planning and distributing tasks by function between the modules of the second level – system-wide management. In this architecture, modules at the third and second levels of the hierarchy are responsible for one enlarged function of the whole robot according to the principle of full functionality [6]: *each robot module should be able to perform its target function in any way convenient for it, using only its own means to execute commands from an external control system*. At the first level of the hierarchy there are submodules that perform the function of low-level control of actuators, for example, drives; in fact, this is a group of submodules (Worker group). The interaction of the second and first level modules is carried out according to the "Master-Worker group" principle.

The control systems of most modern MR are built on the basis of a single computing device on which all computing operations are performed (both high-level and low-level control), which does not allow for the operational reconfiguration of a robotic system; the implementation of reconfiguration of such robots in practice means the creation of a new robot with a given new functionality. The modular solution of the CS means splitting the RTC functionality into simpler subfunctions (see Fig.1), which are implemented by separate fully functional mechatronic modules with their own control system [5]. It should be particularly noted that in order to minimize the weight and size parameters and energy consumption of autonomous mobile robots, module control systems should be implemented on embedded systems – microcontrollers of low performance. To synchronize the operation of such a distributed structure, it is necessary to provide appropriate intermodule information interaction, which is implemented by combining control systems of all modules and submodules into a local area network (LAN) [7]. As a result, distributed computing is implemented by analogy with multi-agent systems [8].

In an MR with a modular architecture, the target task is solved due to the correct selection of functional nodes-modules and their automatic software integration into the robot's control system as a whole without processing the

software aspects of both the general control system and the control systems of the modules themselves. However, due to the uncertainty of the requirements for the parameters of the CS in cases where it is impossible to determine in advance the environmental conditions in which the robot will work (for example, reconnaissance MR on other planets or in areas of man–made disasters), there is a need to reconfigure the CS or even replace its software (software) - remote and externally controlled adaptation, settings and adjustments to the situation. There is a need for operational remote reprogramming of the software aspects of the control systems of individual robot modules without stopping the operation of the RTC, since under these conditions it turns out to be extremely costly or even impossible to deliver the robot to its developer or manufacturer. At the same time, if there is a connection with the robot, it is possible to remotely perform this work with the help of a software developer.

The purpose of this work is to analyze existing and promising methods of dynamic reprogramming suitable for use in a mobile robot with a modular control system architecture. The main problem of using dynamic reprogramming for a modular robot is the need to use embedded microcontroller system modules in the CS and update the software on the fly via the main data bus used by the microcontroller without disconnecting and dismantling the blocks. Since the control systems of modules and submodules are integrated into a LAN, the solution to this problem is based on the search for appropriate methods of intermodule communication (IMC) and network protocols that meet certain requirements.

Authors from different countries and scientific schools have proposed different methods, both for building modular robots [9], [10], [11] and methods of intermodular communication [12], [13], [14]. Our work [15] shows that the Cyphal specification [15] (formerly known as UAVCAN) and its OpenCyphal implementation are the most effective in terms of application as an IMC. Therefore, the various dynamic reprogramming methods discussed in this article will use messages from the Cyphal specification as IMC.

2. Analysis methods

In the framework of this work, we will consider an embedded hardware and software system implemented on a microcontroller that controls some physical process. The control is performed by the "main algorithm". Any changes to the main algorithm are made by a "side algorithm".

At the moment, there are many different and significantly different methods of dynamic reprogramming. Each of them has its own unique properties and differs in a different distribution of the complexity of tasks for changing the "main algorithm" between the human developer of the control system, the human adjuster (end user) of the control system, the software on the computer of the developer/adjuster of the embedded system and the direct control system of the embedded system. The differences also lie in the breadth of flexibility of solutions that can be created using each of these methods. The 4 main methods of dynamic reprogramming are discussed below:

M1. Parametric reprogramming.
M2. Replacing the microcontroller code.
M3. Bytecode replacement.
M4. Replacing a script in an interpreted programming language.

The analysis of each of the listed methods is performed according to the following criteria:

K1. Applicability on various embedded systems – microcontrollers of low performance.
K2. The amount of RAM and program memory required.
K3. Computing speed (CPU time).
K4. The theoretical complexity of creating an implementation of the method in the form of a basic control program.
K5. The theoretical complexity of using the method to adapt to the operational situation of the basic control program by the end user.
K6. Flexibility of the method – the depth and complexity of possible changes to the basic algorithm of the embedded system when using this method.

An important criterion of remote dynamic reprogramming methods is the possibility of ensuring information security, since remoteness is usually realized through the use of a radio channel. The methods discussed below enable Remote Code Execution, which is one of the largest vulnerabilities for any software package. But the authors are not specialists in the field of information security, so ensuring the information security of the methods presented below is a topic for separate consideration by relevant specialists. In this case, it is assumed that all interactions occur inside the noise-proof internal bus of the robot, inaccessible to external influences. Also, the software of the internal network node that exchanges information with external, potentially open networks is written reliably enough that it does not allow downloading and executing code from unreliable and potentially malicious sources.

3. Analysis results

3.1. *M1 – Parametric reprogramming*

One of the first and simplest methods of dynamic reprogramming is parametric reprogramming – that is, the replacement of immutable (from the point of view of the main control algorithm) internal quantities that play an essential role in the operation of this algorithm. An example is the adjustment of the values of the proportional, integral and differential parts in the motor speed controller in the motion module's control system. The values (for example, default ones) for such parameters can be stored in the non-volatile memory of the computing device. This is the method of reprogramming provided by data types with access service (request-response): uavcan.register.Access and uavcan.register.List from the Cyphal specification. Both of these services provide information about named firmware parameters – the so-called registers. Each register has a name and a value. A list of register names can be obtained using consecutive calls to the uavcan.register.List service; one can get existing value or set a new one to a named register using the uavcan.register.Access command. The OpenCyphal implementation provides the possibility of mass installation of these parameters using a configuration file, which facilitates the configuration of the entire interacting system as a whole. It should be noted that the use of registers allows not only to set the values of the parameters of a separate algorithm, but also provides an opportunity to change the main algorithm itself on the fly by selecting one of the possible algorithms.

Evaluating the method by criterion K1, it can be noted that this method is suitable for the vast majority of embedded systems, since it does not imply modification of non-volatile memory in general or modification of only a small part of it. Many modern embedded microcontrollers and embedded systems either have the ability to write data directly to the non-volatile program memory, or contain a separate small non-volatile built-in memory area, which is designed specifically for storing parameters. If the microcontroller does not have any of these features, then such memory can be installed from the outside.

From the point of view of RAM and program memory consumption (criterion K2), this method requires an initial "investment" in the volume of implementations of the method of receiving and transmitting the messages listed earlier, the method of accessing non-volatile memory, as well as additional costs for storing both parameter names and parameters themselves. The Cyphal specification assumes a maximum message length per read and/or modification of one parameter of 515 bytes. Of these, 256 bytes are allocated to the name of the register, and 259 bytes are allocated to its value.

An additional burden also falls on the computational complexity of the most basic algorithm (criterion K3). If the parameters were set in the program text as constants known when creating machine codes, then there is scope for optimizations that are difficult or simply impossible to implement if these parameters can change during operation. This load is difficult to predict and needs to be measured in advance. On modern microcontrollers with a wide (32-bit) data bus, when using a limited number of parameters and a good optimizing compiler, we can assume a performance degradation of no more than a few percent.

From the point of view of both the developer of the control program (criterion K4) and the user (criterion K5), the use of this method turns out to be quite simple. It is enough for the developer to add another function that processes the corresponding messages. The user can use ready-made interfaces, both command-line and graphical, to configure a ready-made device without opening its source code.

The main disadvantage of this method is the lack of flexibility. It is impossible to adapt the module to a completely new scenario that was not described in the source code. For this method to work, there is no need to stop the operation of the microcontroller.

M2 – Replacing the microcontroller code

The following method of dynamic reprogramming is based on the replacement of machine code written in the program memory of the embedded system. Examples of the implementation of such an approach are discussed in [17], [18].

For this method to work with the Cyphal specification, it is necessary to implement message processing for the service of generalized command execution uavcan.node.ExecuteCommand.1.0. One of the generalized commands is COMMAND_BEGIN_SOFTWARE_UPDATE (launching program updates on an embedded system). The name of the file with the new program is specified in the request parameter for this command. In response to this command, the network node first makes a uavcan.file.GetInfo request to get information about a file with new machine codes, and then uacan.file.Read request to get this file in parts.

There are several implementations of this method:

M2.1. A separate bootloader program.
M2.2. Update inside the program.

3.2. *A separate bootloader program (M2.1)*

The simplest and at the same time the most difficult way to update the machine code is to use a separate loader program. This program is called by means of the embedded system itself before loading the main program and can perform basic configuration of the microcontroller, and then call the main program for execution. At the same time, the loader program has more extensive control capabilities of the embedded system, in particular, the ability to get from the outside and overwrite the control program.

Not all embedded systems hardware support booting using a bootloader (criterion K1). On such devices, the work of the loader must either be replenished with software, or not to use the loader at all.

Using the loader reduces the maximum amount of program memory by the size of the loader itself (criterion K2).

According to the K3 criterion, the use of the loader does not affect the operation of the main algorithm. A properly written loader completely gives control to the main subroutine, leaving no traces in RAM or setting up the entire peripherals.

The main difficulty of loaders lies, from the point of view of the developer (criterion K4), in creating (a) a fairly reliable and (b) a small program. First, it is necessary that the embedded system remains online, despite errors during transmission and/or writing to the non–volatile part of the embedded system's program memory. Secondly, it is necessary that the loader can fit into a limited and small area of program memory.

From the user's point of view (criterion K5), the complexity of this method is quite high – you need to make edits directly to the source texts of programs for the embedded system, you need to know the programming language of the source texts, you need to have a programming environment and a sufficiently productive computer. One of the main problems here is creating a user–repeatable development environment with all the necessary tools.

This method provides maximum flexibility (criterion K6) in the modification and adaptation of software, but depends on the delivery of the source codes of the embedded system programs. Also a big disadvantage and advantage of this method is the need to stop and restart the microcontroller to update the program. The main algorithm should be able to respond adequately to requests for software updates.

3.3. *Update inside the program (M2.2)*

The method of updating inside the program and the next method – updating a part of the machine code – solve the problem of updating the software by merging the main and secondary algorithms. The update is performed in parallel with the operation of the main algorithm, the machine codes of the new program are written to a separate area of program memory, it is checked for errors and copied instead of the main program. After that, the control program is started again.

Unfortunately, not all embedded systems can use this method, since it requires the ability to change program memory during operation (criterion K1).

This method uses program memory somewhat more efficiently (criterion K2) by using the same communication mechanisms as the main program. However, the main disadvantage of this method is the maximum amount of program memory that the program can use – it is reduced by half so that at any time the update algorithm can write another piece of update to the program memory.

The operation of the update algorithm does not take additional processor time (criterion K3) in standby mode, and in update mode it can work with low priority without interfering with the real-time mode of the main program.

From the point of view of the developer (criterion K4), the implementation of this method is not particularly difficult – it is enough to create an additional command handler and a separate "thread" that performs the update. But, compared with a separate loader, the complexity is lower due to lower requirements for the amount of code and the ability to share the code of network interaction with the main program.

For the user (criterion K5), the use of this method is comparable in complexity to the complexity of the loader method – you also need to get a properly configured development environment, know and be able to use the programming language in which the source texts are written. However, due to the limitation on the size of the firmware, the user may also be forced to perform additional optimization of the program so that it can fit into the program memory area reduced by half.

The flexibility of application (criterion K6) of this method is higher than that of the method with a loader due to the shorter downtime of the microcontroller - long operations for obtaining new firmware are integrated into the main algorithm, the main downtime occurs at the time of copying and restarting the control program.

3.4. *M3 – Bytecode replacement*

The bytecode replacement method is based on the fact that on the embedded system, instead of the main program, a program interpreter of codes other than the machine code of the embedded system itself, i.e. a bytecode program processor, is launched. At the moment, there are many different bytecode software processors with their own features. An example of such a software processor is given in [19].

This method can use intermodule communication methods according to the Cyphal specification by analogy with the two previous methods. In the case of parametric reprogramming, the bytecode of the control program is written to a parameter (register) of the "unstructured array" type. If a file update is used (discussed in the last section), then instead of updating machine codes, the program memory area responsible for storing the bytecode is updated.

In terms of performance, the bytecode program processor will always be a multiple of slower than similar code in machine codes. The software processor itself can take up significant amounts of program and data memory. A small improvement in the speed of work can be achieved by transferring complex computational operations to the machine code of the software processor. Because of this, this method may not be applicable on embedded systems with a narrow (8 and 16 bit) data bus and small (up to 1 kB) amounts of program and data memory (criteria K1 to 3).

From the point of view of the software developer (criterion K4), in this method, the main difficulty is the selection and transfer of an existing software processor to an embedded system, as well as the development of additional mechanisms for the interaction of this software processor with the peripheral devices of the microcontroller. Also, the developer must provide the user with a translator program of a high-level language in the bytecode of the program processor. If the bytecode of an existing interpreted programming language is used, then this task is significantly simplified.

For the user (criterion K5), this method turns out to be easier than using machine code replacement (M2), but requires knowledge and use of some high-level language or even programming in bytecodes. However, the development cycle is significantly simplified, since a translator program from high-level languages can be much easier to set up and install than toolkits for working with the machine code of an embedded system.

The flexibility (criterion K6) of this method is significantly lower than that of the machine code replacement method due to the smaller volume of possible algorithms: a complex algorithm in bytecode may not meet the speed limit in real time, and the same algorithm converted into machine code and integrated into a software processor may not give that flexibility. adjustments that would give a complete replacement of machine codes. Because of this, the developer has to balance the algorithms of the main program (bytecode or machine code) at the place of implementation, and the user has to put up with this balance when adapting their algorithms.

3.5. *M4 - Replacing a script in an interpreted programming language*

This method is based on the bytecode replacement method and has similar characteristics. In this method, not a bytecode is sent to the embedded system, but the source code in a high-level language, which is then executed directly or using translation into bytecode. An example of such an interpreter is given in [19].

This method requires higher performance of the embedded system – it is necessary to perform the same or a larger amount of calculations compared to the bytecode program processor (criteria K1, K3). Due to the need to parse the text of the programming language for direct interpretation or conversion into bytecode, the volume of machine code and the amount of data increases. Also, due to the use of text as a program, the memory requirements for interpreted programs increase (criterion K2).

In comparison with the bytecode, the shoulders of the developer (criterion K4) also bear the transfer and optimization of the translator program on the embedded system. Also, this method increases the requirements for error

handling and reporting mechanisms – the user must receive messages if any syntactic or logical errors are made in his program.

From the user's point of view (criterion K5), this method is somewhat more complicated than parametric reprogramming, since it implies knowledge of a high-level programming language. However, this method is much simpler than the bytecode replacement method – there is no need to install a translator program.

In terms of flexibility (criterion K6), this method is comparable to the bytecode replacement method. However, due to the need to work with the textual representation of commands, the developer has to search even harder for a balance between algorithms embedded in machine code and algorithms provided by the user.

4. Conclusion

At the moment, there are many different ways of dynamic reprogramming. Among those presented in the article, the easiest to use is the M1 method – parametric reprogramming. The method is implemented in many embedded systems, unlike other methods that require either hardware support for some functions (replacement of machine code – M2), and/or a large amount of program and data memory (replacement of bytecode (M3), or programs in an interpreted programming language (M4)), but it is the least flexible. The simplest methods for the user are M1 and M4, since they do not require special software or additional knowledge. The M2 method is ideal for the use of embedded systems with the smallest amount of program and data memory. The M3 method is good if the flexibility of the M4 method is needed, but the use of an embedded system that does not have the appropriate parameters does not allow using M4.

In addition to the analysis of the listed methods according to six criteria, it is planned in the future to experimentally test the feasibility of each method on microcontrollers of the STM32F103, GD32VF103, ESP32 families and Raspberry Pi and Orange Pi microcomputers.

5. Acknowledgments

Research is supported by the Russian Foundation for Basic Research: Grant 19-07-00892a.

6. References

[1] Popov E.P. & Pismennyi G.V. (1990) Fundamentals of Robotics: Introduction to the specialty [Основы робототехники: введение в специальность], Higher School [Высшая школа] – 224 p.
[2] Lopota A.V. & Yurevich E.I. (2013) Stages and prospects of development of the modular principle of building robotic systems [Этапы и перспективы развития модульного принципа построения робототехнических систем], Scientific and technical bulletin of SPbSPU [Научно-технические ведомости СПбГПУ], Vol 1 2013 Computer science. Telecommunications. Management.[Информатика. Телекоммуникации. Управление.], pp. 98 - 103, ISSN 1994-2354.
[3] Platonov A.K. (2010) Robotics for a lunar base [Робототехника лунной базы], XXXIV Readings on cosmonautics KIAM RAS [XXXIV Чтения по космонавтике. ИПМ им. М.В. Келдыша РАН].
[4] ISO Central Secretary, (2021). Robotics – Modularity for service robots – Part 1: General requirements (Standard No. ISO/IEC TR 22166-1:2021), International Organization for Standardization, Geneva, CH.
[5] Andreev, V.; Kim, V. & Eprikov, S. ModRob: the Hardware-Software Framework for Modular Mobile Robots Prototyping // Proceedings of the 31st DAAAM International Symposium, B. Katalinic (Ed.), Published by DAAAM International, ISBN 978-3-902734-29-7, ISSN 1726-9679, Vienna, Austria, 2020, pp.0391-0402.
[6] Andreev, V.; Kim, V. & Plotnev, P. The principle of full functionality – the basis for rapid reconfiguration in heterogeneous modular mobile robots // Annals of DAAAM and Proceedings of the 28th International DAAAM Symposium on Intelligent Manufacturing and Automation, DAAAM 2017; Zadar; Croatia,B. Katalinic (Ed.), Published by DAAAM International, ISBN 9781510853270, ISSN 1726-9679, Vienna, Austria. Curran Associates, Inc. (Feb 2018), pp. 0023-0028.
[7] Andreev V. & Plotnev P. Organizing Intermodular Communication for Heterogeneous Modular Mobile Robot. Annals of DAAAM and Proceedings of the 28th International DAAAM Symposium on Intelligent Manufacturing and Automation, DAAAM 2017; Zadar; Croatia,B. Katalinic (Ed.), Published by DAAAM International, ISBN 978 1 510853 27 0, ISSN 1726-9679, Vienna, Austria. Curran Associates, Inc. (Feb 2018), pp. 0474-0480.
[8] Andreev V. Control System Mobile Robots with Modular Architecture as a Multi-Agent System with a Hierarchical Topology, Proceedings of the 30th DAAAM International Symposium, B. Katalinic (Ed.), Published by DAAAM International, ISBN 978-3-902734-22-8, ISSN 1726-9679, Vienna, Austria, 2019, pp.0010-0019.
[9] Herbrechtsmeier, S.; Korthals, T.; Schopping, T. & Ruckert, U. (2016). AMiRo: a modular & customizable open-source mini robot platform. 20th International Conference on System Theory, Control and Computing (ICSTCC), Sinaia – 2016. – pp.687-692.
[10] Guifang Qiao; Guangming Song; Jun Zhang; Hongtao Sun; Weiguo Wang & Aiguo Song (2012). Design of Transmote: a Modular Self-Reconfigurable Robot with Versatile Transformation Capabilities, Proceedings of the 2012 IEEE International Conference on Robotics and Biomimetics, 2012. – pp.1331-1336.
[11] Baca J., Ferre M. & Aracil R. (2012) A heterogeneous modular robotic design for fast response to a diversity of tasks, Robotics and Autonomous Systems. vol. 60. no. 4. pp. 522–531.

[12] Bonarini, A.; Matteucci, M.; Migliavacca, M .& Rizzi, D. (2014). R2P: An open source hardware and software modular approach to robot prototyping. Robotics and Autonomous Systems. – 2014. – No.62. – pp.1073-1084.
[13] Losada D. P., Fernández J. L., Paz E. & Sanz R. (2017) Distributed and modular CAN-based architecture for hardware control and sensor data integration, Sensors. No.17. – pp.1013-1030.
[14] Peng, L.; Guan, F.; Perneel, L.; Fayyad-Kazan, H. & Timmerman M. (2016) EmSBoT: A lightweight modular software framework for networked robotic systems. 2016 3rd International Conference on Advances in Computational Tools for Engineering Applications (ACTEA), Beirut, 2016, pp.216-221, doi: 10.1109/ACTEA.2016.7560142.
[15] Andreev, V. & Pletenev, P. (2021) Problems of Choosing an Intermodule Information Interaction Protocol for Mobile Robots with Modular Control System Architecture, Proceedings of the 32nd DAAAM International Symposium, pp.0151-0157, B. Katalinic (Ed.), Published by DAAAM International, ISBN 978-3-902734-33-4, ISSN 1726-9679, Vienna, Austria.
[16] Kirienko, P.; Dixon, S. & others. (2022) OpenCyphal: Open technology for real-time intravehicular distributed computing and communication based on modern networking standards. URL: https://opencyphal.org/specification/Cyphal_Specification.pdf (accessed 10.09.22).
[17] Lobdell M. (2012) Robust over-the-air firmware updates using program flash memory swap on kinetis microcontrollers, Freescale Application Note, p. AN4533. 2012.
[18] Jaouhari S. E. & Bouvet E. (2022) Secure firmware Over-The-Air updates for IoT: Survey, challenges, and discussions, Internet of Things. 2022. (18). C. 100508.
[19] Zandberg K. & Baccelli E. (2020) Minimal virtual machines on IoT microcontrollers: The case of berkeley packet filters with rbpf, arXiv preprint arXiv:2011.12047. 2020.