DOI: 10.2507/31st.daaam.proceedings.xxx

DEVELOPMENT OF A COMMUNICATION PROTOCOL SPECIFICATION FOR THE MODROB-1 MODULAR MOBILE ROBOT

Victor Andreev; Pavel Pletenev & Stanislav Eprikov

This Publication has to be referred as: Andreev, V[ictor]; Pletenev, P[avel] & Eprikov, S[tanislav] (2020). Development of a Communication Protocol Specification for The Modrob-1 Modular Mobile Robot, Proceedings of the 31st 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: 10.2507/31st.daaam.proceedings.xxx

Abstract

The features of modular mobile robots with a pyramidal (multi-level) structure of the information-measuring and control system (CS) are considered from the point of view of inter-module inter-level and intra-level network interaction. The modular architecture allows for rapid reconfiguration of robotic systems. A hierarchical robot's CS topology, when each module has its own IMCS with a separate computer, allows one to increase the system's performance by distributing the computational load between the modules' computing devices. The implementation of distributed computing in a multiprocessor system with a hierarchical topology imposes a number of restrictions on the organization of inter-module information interaction. The concept of intermodule interaction is formulated, on the basis of which logical intermodule connections are analysed and possible information flows in a system with distributed computing are evaluated. In the context of these flows, specifications are formed defining methods of implementation of inter-module interaction based on existing networks and information interaction protocols. Virtual experiments are conducted with the developed specifications to find out the place of each of the compared networks in the robot as a whole.

Keywords: modular robot; mobile robot; network technologies; informational interaction; reconfiguration; rfc.

1. Introduction

The increased use of mobile robots (MR), which is currently observed, leads to an expansion in both algorithmic and memory complexity of the robotic systems (RTS) control systems (CS). This in turn requires higher performance computers implementing RTS' CS, which is known to be not always possible. Another way is to create an RTS with a modular architecture [1], in which each module is equipped with its own CS, built on the basis of a low-performance systems – embedded computing devices (microcontrollers and microcomputers). Then RTS-wide control and decision-making is performed on a distributed computing cluster, lowering the necessary computing power overall. Modular implementation of RTS's CS requires such an intermodule communication organisation that synchronizes operation of all modules. This can be a real problem, which isn't solved to present day in robotics, especially in the context of use of small embedded computing devices.

The second problem is the need for reconfigurability of an RTS. This is due to the obvious need to use MRs where human stay is fraught with danger to their health or life, for example, for work in extreme conditions of the Arctic and Antarctic, space and in solving the problems of the Ministry of emergency situations. In these conditions, robots with the necessary functionality should be assembled in the place of the carried-out works and during them. Therefore, the reconfiguration and/or scaling process should be as simple as possible. It should be performed in the "plug and play" fashion. Therefore there is a need for the rapid and automatic reconfiguration of the CS. As shown in [2], this problem is solved by creating robots with modular architecture of CS. The CS should consist of modules, communicating using predefined rules and exposing an application programming interface (API). It is necessary to develop a well-defined specification on both rules and API to ensure interoperability of modules of different manufacturers.

Many teams in different parts of the world are conducting research on modular robots [3]. One of the most important principles of designing modular systems in our case is the principle of full functionality. The principle of full functionality of modules is formulated as follows [4]: each robot module must be able to perform its target function in any way convenient to it, using only its own means to execute commands from an external control system.

In the structure of a robot divided into modules based on the principle of full functionality, each module is responsible for only one function of the robotic system. Studies performed in the framework of the Grant RFBR 16-07-00811a "Development of functional-modular principle of building hardware / software intelligent mobile robotic systems" [5] showed that a two-level hierarchy of modular architecture on the principle of full functionality is not able to ensure the implementation of process management in real time embedded computing devices, as algorithms CS functional modules are quite complex. The amount of information processed in modules is still large, and the computing power of embedded systems is not always enough to provide real-time mode. Especially expensive is the process of processing and integrating sensory information and decision-making at the level of the intelligent system-wide management module. As the tasks solved by the MRs in both offline and supervisory mode become more complex, the algorithms of modules' CS will become more complex, and the use of microcomputers with higher performance will still lead to limited functionality of the module.

A number of experiments conducted with the transport module showed the possibility of further development of the modular architecture of MR by dividing full-featured modules into submodules and creating a multi-level hierarchical (pyramid) CS network topology that provides the use of embedded computing systems at each level of the hierarchy [6].

A robot built using such architecture wouldn't be able to function as a whole without all modules working together in a synchronized way. So, the modules need to communicate a lot of information between them. Developing intermodular communication protocols is a necessity. It is complicated by a variety of possible modules hardware. At the moment, there are many modular robots developed by different teams around the world – [7], [8], [9], [10], [11],[12] and [13]. All of these modular robots are incompatible with each other as they do not propose a communication protocol in a formal way. And this is the problem and the **goal** of this paper - develop a specification for methods of inter-module interaction in general and for different communication networks in particular to unify the development of modules based on different hardware platforms for different levels of subordination in a modular robot. Authors have previously attempted creating such a specification earlier – [14], but this work deepens that work with hierarchical control system structure and bases the new specification on the currently developing standard – ISO/DIS 22166-1:2019 [15].

2. Communication protocol specification ModRob-1

Note: the specification is still under development and is provided at the time of writing. However, the main provisions of the proposed specification require public discussion.

This section uses keywords to define the requirement levels – "**necessary**", "**must**", "**must not**", "**should**", "**recommended**", "**should not**", "**not recommended**", "**possible**", "**optional**". These words must be understood in accordance with [16].

This specification discusses and sets requirements for the features of modular robots that can be used together with it and the family of specifications based on it.

The structure of a modular mobile robot **must** meet the following requirements:

1. *Robot* as a whole **must** be represented as a single *composite module (CM)*, both from a mechanical and software point of view.
2. Each *CM* **must** contain one or more end devices – *modules* or *composite modules.*
3. Each *CM* **should** be part of a more complex system – the parent *CM*.
4. Subordinates of a *CM* **must** communicate through the *communication network (CN)* of this *CM*.
5. *CN* of a composite module **may be** part of *CN* of the parent *CM*.
7. Each *CM* **must** have one leader module – a module that controls all other modules included in the *CM*.

The *module* requirements are listed below:

1. Each *module*, in accordance with the functional modular principle, **must** perform one function and do it in the most efficient way (engine control, sensor operation, etc.).
2. A *module* **must** have a unique *module identifier (MI)* within the communication network of its *CM*.
3. A *module* **must** export one or more *variables* for exchange over CNs as *messages*, to control the implementation of its function.

A *variable* is defined as follows:

1. A *variable* **must** have a unique *variable identifier (VI)* within the module.

2. A *variable* **must** be a real number single-precision (float32).

3. A *variable* **must** have a unit of measurement. The units of measurement of *variables* **must** be included in the international system of units (SI), except for special dimensionless variables that can indicate the mode of operation of the module and/or be counters.

4. It **must** be possible to change any *variable* from outside the module using the inter-module communication protocol or from inside the module during the operation of the module's control system.

5. A *module* **must** publish any requested variables with the requested frequency on request.

6. All *module variables* can be implicitly divided into "input" and "output" according to a *module's* response to setting a new value from outside of it:

6.1. Setting an external value to the "input" *variable* **must** change the current and future state of the module.

6.2. Setting an external value in the "output" *variable* **must not** change the current and future state, and the value of such a variable **may** be further overwritten during the operation of the internal algorithms of the module's CS.

7. A *module* **should** filter out the values of variables that may cause the module to break or malfunction.

8. A *module* **must** be able to store the current value and/or settings for publishing and/or receiving *variables* in non-volatile memory.

For the robot to work as a whole, it is necessary that the modules are combined into one or more communication networks, each of which must meet the following requirements:

1. A *CN* **must** ensure that messages are transmitted between modules with sufficient bandwidth to support real-time control modes. Where necessary, the network **must** ensure that message delivery is deterministic.

2. It is **recommended** that the *CN* does not have an explicit leader node that provides message transmission, i.e. the *communication network* **should** ensure the creation of horizontal links.

3. A *CN* **should** allow the node to obtain information about its topology.

4. If multiple networks are selected for the same layer, there **must** be an unambiguous way to convert one network's message to another network's message.

5. It is **recommended** that the standard connectors for network communications, if they are specified by the standard of the physical layer of the network, have the smallest size.

6. For ease of implementation, it is **recommended** that as many communication network layers as possible be implemented at the hardware level in embedded systems.

7. It is **necessary** that for most embedded devices had a library (or several libraries) to work with the communication network. If there is no such library, it is **recommended** that creating such a library **should** be relatively easy.

8. Physical and/or channel layer of the *communication network* **must** be noise-proof and/or have error detection and elimination mechanisms for transmitting information at distances specific to the *modules*.

As mentioned above, modules should be able to exchange *messages*. *Messages* **must** meet the following requirements:

1. A *message* **may** contain a unique message identifier (MessId), such that:

1.1. A *MessId* **must** be continuously and monotonically increasing.

1.2. A *MessId* **should** be unique enough to avoid collisions between messages from different modules.

2. A *message* **should** be one of three types:

2.1. *Command* – a direct change in the state variable.

2.2. *Publication* – informing modules about the current status of the module.

2.3. *Piecewise transmission* – transmission in the form of an array of bytes of general information about the state or type of the module, intended to be shown to a person.

3. *Command* and *Publication* type *messages* **must** contain a unique *module ID (MI)* and *variable ID (VI)*. Together, in such messages, MI and VI form the *variable's address*.

3.1. The *variable's address* of a *command message* **must** point to the variable for which the *command* is intended.

3.2. The *variable's address* of a *publication message* **must** point to the variable whose value is included in this publication.

4. *Piecewise transmission message* **must** contain the MI of the module from which the transfer is being made.

5. A *command message* **must** be one of three types of commands: *setting the value*, *setting the publication frequency*, and *setting the subscription address*.

5.1. "*Setting the value" command* **must** contain one real number (float32).

5.2. *"Setting the publication frequency" command* **must** contain one real number (float32), which means the publishing frequency of this variable in Hertz.

5.3. "*Setting the subscription address*" *command* **must** contain the *variable's address* (*MI* and *VI*). When receiving a *publication* from this *address*, the *module* **must** set the value received in such a message to the variable to which this command was sent.

6. A *Publication message* **must** contain a single real number – the value of the *variable* at the time of publication.

7. *Piecewise transmission message* **must** contain:

7.1. The number of chunks in the transmission.

7.2. Chunk of the transmitted data.

7.3. The chunk sequence number in the whole transmitted data.

8. It is **recommended** that a *piecewise transmission message* also contain information about the length of the transmitted data, if this information cannot be obtained from information from the communication network.

The expected cycle of operation within a single *composite module* is as follows:

1. When powered up, the *CM's leader module* receives information about its subordinate modules.

2. The leader module **may** store information about the configuration of the subordinate modules in non-volatile memory. If there is a subordinate module that matches this information, the leader module **must** configure the found module by sending *command messages*, setting *variables' values*, *publication frequencies*, and *subscription addresses* according to the saved configuration.

3. Next, the *leader module* is in normal mode: receives publish *messages* from subordinate *modules*, processes them, and publishes new *variable values* for its subordinate *modules* and its parent *CM*.

The *CM's leader module* is a specialized border router that converts messages from the upstream network to messages from the downstream network.

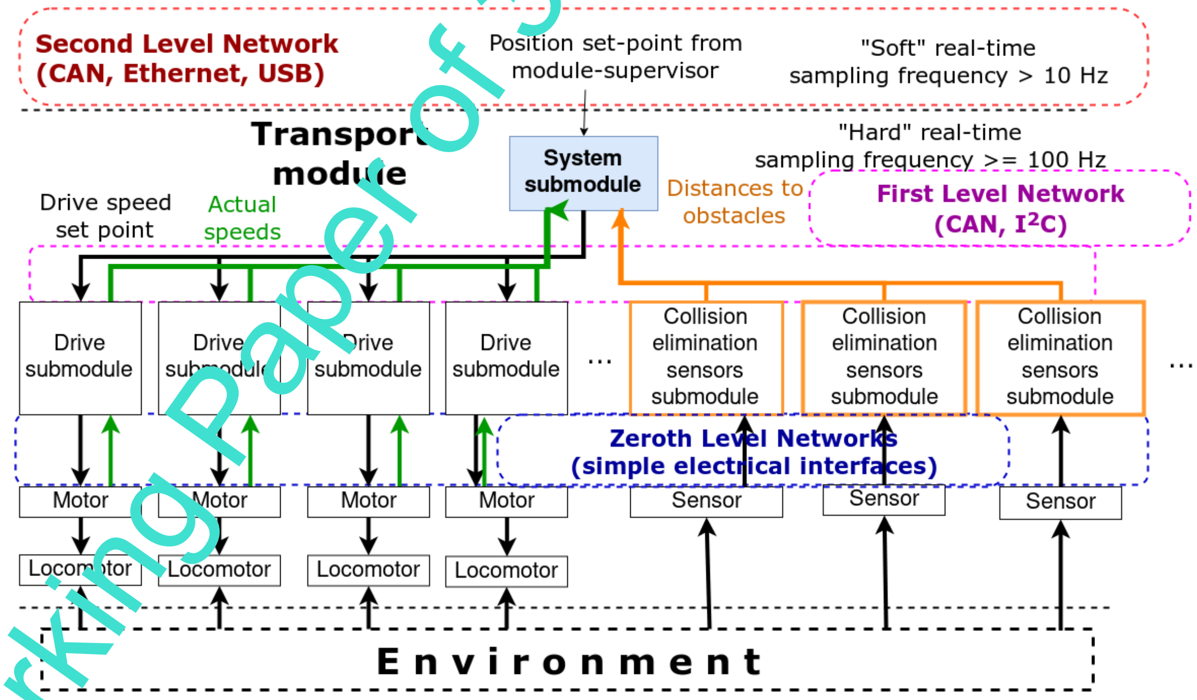
This specification is the base on which it is intended to build further specifications. It is necessary, for example, to determine how messages will be transmitted over a particular network, how data will be distributed within data packets transmitted over the network.

Let's look at how this specification can be applied to form the robot architecture.

3. An example architecture of a robot in the form of a composite module

Within the framework of the given concept, a multi-level subordination of modules and submodules is introduced so that the resulting system is (a) implemented on simple embedded devices and (b) the computational load of overall robot control is distributed among cluster of a variety of different computers. Let us consider this approach on the examples of the main functional modules of MR: functional blocks of the transport module [4], of the short-range and long-range sensor module [17], and of the power module. The proposed schemes of the modules are shown in fig. 1, 2 and 3. At least 3 levels of interaction and subordination can be distinguished in these modules:

- 0th (zeroth) level of interaction (and corresponding "networks" of zeroth level, see fig. 1-3):
  - primary analysis of sensory data, filtering and conversion to structural representation of information;
  - creating closed-loop drives and other control systems.
- 1st level of interaction (and corresponding First Level Network, see fig. 1, 2) – level of full-functional submodules. The following is implemented here:
  - secondary data analysis, aggregation of readings from several submodules;
  - development and distribution of tactical tasks for this module as a whole.



Second Level Network
(CAN, Ethernet, USB)
Position set-point from module-supervisor
"Soft" real-time sampling frequency > 10 Hz
Transport module
System submodule
"Hard" real-time sampling frequency >= 100 Hz
Drive speed set point
Actual speeds
Distances to obstacles
First Level Network (CAN, I²C)
Drive submodule
Drive submodule
Drive submodule
Drive submodule
...
Collision elimination sensors submodule
Collision elimination sensors submodule
Collision elimination sensors submodule
...
Zeroth Level Networks (simple electrical interfaces)
Motor
Motor
Motor
Motor
Sensor
Sensor
Sensor
Locomotor
Locomotor
Locomotor
Locomotor
Environment

Fig. 1. Transport module's structure [4].

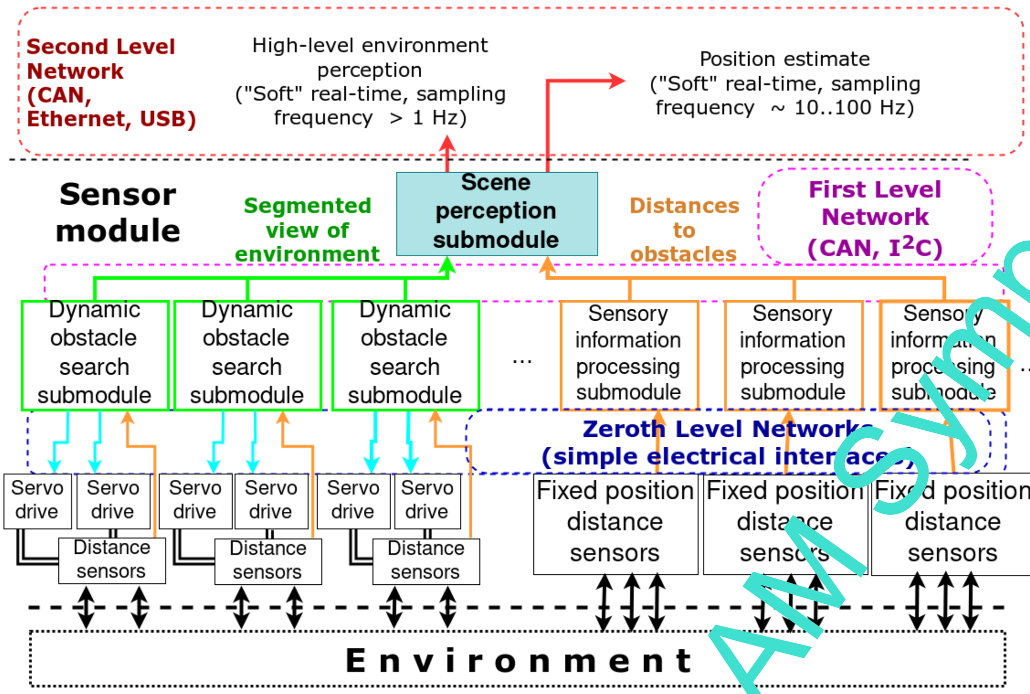

Second Level Network (CAN, Ethernet, USB)
High-level environment perception ("Soft" real-time, sampling frequency > 1 Hz)
Position estimate ("Soft" real-time, sampling frequency ~ 10..100 Hz)
Sensor module
Segmented view of environment
Scene perception submodule
Distances to obstacles
First Level Network (CAN, I2C)
Dynamic obstacle search submodule
Dynamic obstacle search submodule
Dynamic obstacle search submodule
...
Sensory information processing submodule
Sensory information processing submodule
Sensory information processing submodule
...
Zeroth Level Networks (simple electrical interfaces)
Servo drive
Servo drive
Distance sensors
Fixed position distance sensors
Environment

Fig. 2. Sensor module's structure.

- 2nd level of interaction (and corresponding Second Level Network, see. fig. 1-3) – level of full-functional modules:
  - tertiary data analysis, aggregation of data from same and different types of modules;
  - development and distribution of strategic tasks for the robot as a whole.

It is assumed that the interaction of modules with an external supervisor will also occur at the 2 level of interaction.

Another, specialized method of interaction is also possible – through the power line. Since it is assumed (see [6]) that all modules will be powered from one shared power line, it is possible to create a communication system between all modules and submodules included in the robot. However, such a communication channel will have low bandwidth.

As one can see in figures 1, 2 and 3, each of the modules and submodules is, in fact, a specialized border router, which converts network traffic of a higher level to a lower level, except when all modules and submodules are connected to one common bus. Also, figure 3 shows the case when the network of the 2nd level interacts with the networks of the 0-th level, i.e. the 1st level "collapses" inside one submodule, and the submodule of the generalized control of the power supply performs the tasks of two levels at once, which will require separation in time.

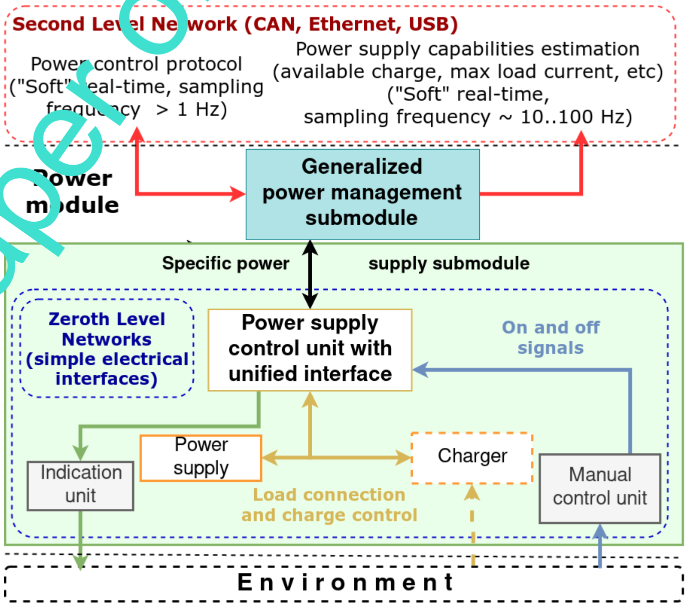

Second Level Network (CAN, Ethernet, USB)
Power control protocol ("Soft" real-time, sampling frequency > 1 Hz)
Power supply capabilities estimation (available charge, max load current, etc) ("Soft" real-time, sampling frequency ~ 10..100 Hz)
Power module
Generalized power management submodule
Specific power supply submodule
Zeroth Level Networks (simple electrical interfaces)
Power supply control unit with unified interface
On and off signals
Indication unit
Power supply
Charger
Manual control unit
Load connection and charge control
Environment

Fig. 3. Power module's structure.

This concept is complicated by the multiplicity of possible technical solutions. In principle, there are no universal solutions for zeroth level networks – each actuator or sensor has its own unique ways of connection and control. Many sensors and actuators can be grouped together, and the ways they interact at the 0-th level of the network are the details of the implementation of each specific submodule and should not be significant in the organization of intermodule interaction.

First and Second Level Networks are also a part of the implementation of each module, but they generally need to interact with each other, as well as with all possible modules according to general principles, and obey the same set of commands. This is only possible if these networks are standardized.

4. Estimating data flows in a modular mobile robot

As mentioned earlier, there have been developed many robots with modular architecture have been created. They use many different types of networks. Each type of network can be implemented in different topologies and include only a part of the standard model levels. To assess the applicability of each network, we estimate the volume of messages between the modules and the frequency of their transmission for the worst case. A special mathematical representation is often used for such estimates – network calculus, which is based on min-plus algebra.

In network calculus, all network nodes are represented as black boxes with multiple inputs and outputs. Each input is characterized by the so-called arrival function (arrival curve), and each output – departure function (departure curve). Both of these functions characterize the total amount of information received or sent by a particular host. Some of the networks are more amenable to such formalism than others. For example, when using networks with priority of delivery such as CAN, must make extra efforts to analyse traffic due to the fact that the message with higher priority to interrupt the transmission of a message with lower priority [18]. To evaluate the performance of the network as a whole, you must first evaluate the parameters of the arrival curves for each transmitter in each module and submodule. Let us use the Heaviside function (1) to set the step curve (2), which is the arrival curve for the submodule transmitter (PM), where f_{sm} , Hz, is the frequency, a V_{sm} , bytes, is the payload volume of the transmitted messages.

$$h(t) = \begin{cases} 0, x < 0 \\ 1, x \geq 0 \end{cases} \tag{1}$$

$$A_{sm}(t) = \sum_{n=0}^{\infty} V_{sm} \cdot h(\mathrm{t} - \frac{n}{f_{sm}}) \tag{2}$$

Let’s evaluate these parameters for various modules and submodules:

1. For transport module (see fig. 1):

1.1. For drive submodules: $f_{sm} \geq 100$ *Hz*, new speed set point $V_{sm} = 4$ *bytes*, readings on the current speed and position of the output shaft $V_{sm} = 8$ *bytes*. There can be several such submodules.

1.2. For collision elimination sensors submodules: $f_{sm} \geq 100$ *Hz*, sensors can be two kinds: (1) bumper with touch sensors $V_{sm} = 1$ *bytes*, (2) non-contact distance sensor $V_{sm} = 4$ *bytes*. There can be several such submodules.

1.3. For the motion control submodule: $f_{sm} \geq 100$ *Hz*, the submodule is given the desired linear and angular velocities $V_{sm} = 8$ *bytes*, the actual position of the module relative to a certain coordinate system $V_{sm} = 4 \cdot 6 = 24$ *bytes*.

2. For sensor module (see fig. 2):

2.1. For a dynamic search obstacles submodule: $f_{sm} \sim 1$ *Hz*, this submodule can transmit the segmented view of environment in either of *n* line segments in the plane $V_{sm} = n \cdot 16$ *bytes* or *n* line segments in space $V_{sm} = n \cdot 16 \cdot 6 = n \cdot 48$ *bytes*. There can be several such submodules.

2.2. For sensory information processing submodule: $f_{sm} \sim 1$ *Hz*, this sub-module processes signals from various sensors and generates the average estimate of the distance $V_{sm} = 8$ *bytes*.

3. For power module (see fig. 3):

3.1. For the generalized power management submodule: $f_{sm} \sim 1$ *Hz*, this submodule periodically publishes information about the current state of the battery, the maximum load current, etc. – $V_{sm} = 1..4 \cdot 1..4$ *bytes*.

In the above data, all frequency parameters f_{sm} depend on the speed of the robot. For other modules, the pattern will be approximately the same. Actuating modules, such as transport module, manipulator module or process module (capture module), have high frequencies of interaction inside the module with small sizes of transmitted messages. Sensor modules or data processor modules, on the other hand, have a much lower frequency of message transmission, which is compensated by the large volume of these messages. A separate case is the power module – on the one hand, this module should not very often publish sensor readings, and on the other hand – should quickly convey a message about the need to start a safe shutdown to all modules. Thus, it is important that messages in the communication network are transmitted with the least overhead (overhead), and that all packets have a roughly equal chance of being transmitted despite their size, i.e. that long packets do not occupy the common bus for too long.

It can also be noted that different topologies may be preferred for different module types. Topology "bus" is preferable to the modules of the actuators, especially if they are the creation of horizontal linkages. The star topology is preferable in the case of sensor modules, as it allows data to be transmitted directly to the supervisor. The definition of requirements for the use of certain topologies for different types of networks will be formulated in further research.

5. Specification ModRob.CAN-1

The CAN bus is significantly limited – each message **must** contain a 16-bit or 29-bit message identifier (depending on the mode) and up to 8 bytes of payload. Let's define the requirements for the interaction mechanism:

1. Due to network restrictions, *modules* connected to it **must** have a *module ID (MI)* of 16 bits long.

2. Each *module* **must** have no more than 255 variables – the *variable identifier (VI)* must be 8 bits long.

3. CAN bus message ID has a second meaning – it is the priority of sending the message. The smaller the ID, the lower the priority for this message. Only the extended CAN bus mode – 29 bits of the identifier **must** be used for module communication.

3.1. The first (most significant) 3 bits of the identifier **must** always be 0 – they are reserved for further development.

3.2. The following 2 bits are the message type code: 00 – *command*, 01 – *publication*, 10 – *piecewise transmission*, 11 – reserved for future use.

3.3. The following 16-bit and 8-bit are *MI* and *VI*, respectively.

Thus, the message type determines its priority.

1. In *command and publication messages* the CAN message length **must** be 6 bytes.

2. In *command messages*, the fifth byte of the message **must** encode the command type. Depending on its content, the first 4 bytes should be interpreted as follows:

2.1. A value 0 in the fifth byte means setting the value of the variable – the first 4 bytes **must** contain the value of the *variable*, type – float32 in order from the lowest to the highest value byte.

2.2. A value of 1 in the fifth byte means setting the publication frequency – the first 4 bytes **must** contain the *publication frequency*, type – float32 in order from the lowest to the highest byte.

2.3. A value of 2 in the fifth byte means setting the subscription address. The first byte must not be interpreted. The next 2 bytes **must** contain the *module ID* in order from the lowest to the highest byte. The 4th byte **must** contain the *variable ID*.

2.4. All remaining values are reserved for future use and **should not** be used.

3. In *publication messages* the first 4 bytes **must** contain the value of the variable, type – float32 in order from the lowest to the highest byte. The remaining two bytes are reserved for future use and **should not** be used.

4. In piecewise transfer messages the message **must** be 8 bytes long. The CAN bus message ID **must** contain the number of the transmitted chunk instead of the *variable ID*, and the number of parts **must** be specified in the 8th byte of the message. The length of the chunk **must not** change and **must** always be equal to 7 bytes. Bytes 1 through 7 in the message **must** contain the values of the transmitted part.

6. Specification ModRob.UART.B-1

UART – or universal transceiver – is one of the most popular data buses used in embedded systems. To use it with the ModRob-1 specification, 8-bit transmission mode **must** be used. Messages are encoded in accordance with the ModRob.CAN-1 specification, but then turn into an array of bytes to send as follows:

1. The first byte of the array **must** always have the value of the hexadecimal number *5A*.

2. The following 4 bytes **must** be interpreted as a 32-bit CAN message identifier, the lower 4 bits of which are the number of bytes in this message.

3. Following them **must** be 6 or 8 bytes of a CAN message.

4. The last byte **must** contain CRC8-sum of all the previous bytes in the current message.

7. Specification ModRob.SPI-1

SPI (Serial Peripheral Interface, SPI bus) is a serial synchronous full — duplex data transmission standard designed to provide simple and inexpensive high-speed interfacing of microcontrollers and peripherals.

Messages are encoded in accordance with the ModRob.UART.B-1 specification. Mode 0 of the SPI interface must be used to transmit an encoded byte array.

8. Specification ModRob.Ethernet-1

Each module included in the Ethernet network **must** have a 128-bit identifier generated using the UUID scheme [19]. Each message transmitted over an Ethernet network **must** also have a 128-bit identifier generated using the ULID scheme [20], which meets the requirements for the message identifier from the ModRob-1 specification.

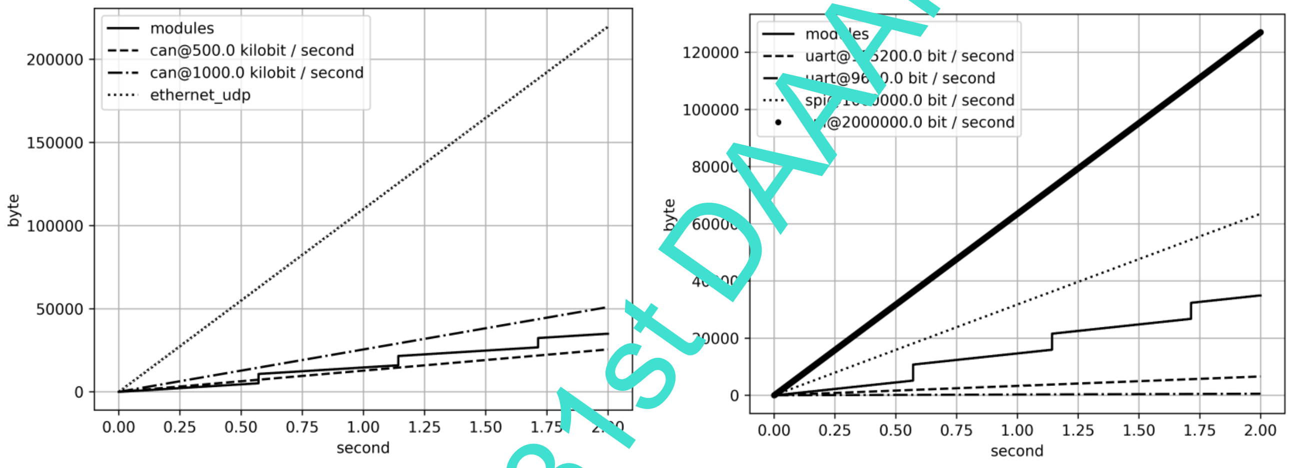
A packet **must** be transmitted over the network, where the first 32 bytes **must** be interpreted as the *message ID* and the *module ID*. After them, a 16-bit integer must be written, which encodes the number of messages in ModRob.UART.B-1 encoding included in this message followed by actual encoded messages.

Any module implementing this specification **must** be configured to maximize the number of messages transmitted at a time, minimizing the message transmission delay.

9. Modelling example

In network calculus, the worst-case bit and frame rates must be taken into account to estimate the dispatch function of each network. In the following example, all networks are treated as simple black boxes with a fixed data rate, without any consideration for their special collision rules. This method will help avoid some of the more complex modelling tasks for CAN, as discussed earlier. The CAN bus has a typical data transfer rate of 0.5 bits. 1 Mbit/sec. Ethernet networks can provide different bandwidths, but to assess the worst – case situation, we use the minimum data transfer rate of 10 Mbit/s.

For evaluation, we assume that only messages of the publish type will be transmitted, which is 119 bits per frame. It is also necessary to take into account bit staffing and inter-frame pauses in CAN transmission, which additionally increases the frame size by about 1.5 times, which is 182 bits per frame. This coefficient also takes into account any imperfections in the hardware and software of devices interacting with the CAN bus. The resulting departure curve (2) for CAN with data rates of 0.5 and 1 Mbit/s is shown in figure 4a.



a) b)

Fig. 4. Departure and arrival curves for CAN with various bitrates and Ethernet networks (a) alongside with UART and SPI networks with various bitrates (b).

Next, consider the Ethernet network. A maximum of 35 ModRob.UART.B-1 messages can be placed in a single UDP frame without frame fragmentation, making up a 420-byte data frame (almost the maximum data frame size in most Ethernet implementations), of which 140 bytes make up the payload. Adding a 46-byte header and 12-byte inter-frame silence further increases the frame size to 478 bytes. When considering the worst-case scenario, you should consider collisions that leave only about 1/3 of the bandwidth used, which turns out to be 784 frames per second on a 10 Mbit/s channel. The departure curve (2) for this installation is shown in figure 4a.

The UART interface has a set of transmission clock rates, but the most frequent and supported in microcontrollers are 115200 and 9600 bps. To transmit 1 byte of data, you must transmit at least 10 bits – 1 start bit, 1 stop bit, and 8 payload bits. 14 bytes of payload are transmitted in ModRob.UART-1 encoding. The resulting departure curve (2) for UART with data transfer rates of 115200 and 9600 bps is shown in Fig. 4b. SPI interface, unlike the UART, is not limited to specific frequencies of data transmission. Typical data transfer rates over this interface on modern microcontrollers are 1 Mbit/s. To separate bytes between them, a 1-bit pause is usually set. The sending curves (2) for SPI with data transfer rates of 1 and 2 Mbit/s are shown in figure 4b.

As an example, let's run a simulation of a simple modular mobile robot where all networks are connected on a single bus. This mobile robot consists of 4 drive modules, 4 remote collision avoidance modules, 1 bumper collision avoidance module, 1 motion control module, 2 flat dynamic obstacle search modules reporting up to 100 linear segments, 1 spatial dynamic obstacle search module reporting up to 100 linear segments, and 1 sensor information processing and location detection module. Each of them forms an arrival curve (2) for the corresponding networks, as discussed earlier. For both scenarios shown in Fig. 4a and 4b, we assume that the network is always as full of data as possible, even when it does not transmit module data. The module curve is the sum of all arrival functions (2) of all modules. The main slope is formed by drive modules and submodules. Large jumps are caused by large volumes of data transmission generated by submodules of dynamic obstacle search, which are manifested by relatively rare bursts.

In this example, the modules use almost all the bandwidth on the CAN and SPI buses, as well as a small portion of the bandwidth on the Ethernet network, even in the worst case. Unfortunately, the same cannot be said about the UART interface bandwidth, which is completely occupied by data transmission. But, as mentioned earlier, when designing a system, network bandwidth is the main, but not the only criterion that should be taken into account. This example also shows the need to divide networks into different composite modules, since this will increase the throughput of any lower-level network in proportion to the number of composite modules that make up the robot.

10. Conclusion

The representation of a CS robot with a modular architecture in the form of a multi-level hierarchical (pyramid) network topology will solve the problem of providing real-time mode; simplify reconfiguration and scaling of the robotic system. The presented research in the field of intermodule networks has addressed only a small part of this problem of such a complex structure.

It is shown that at the first division of CS into functional simple and composite modules, at least 3 levels of interaction and subordination can be distinguished, each of which defines the tasks of information interaction. In this case, each of the composite modules is essentially a specialized edge router that converts network traffic from a higher level to lower level network traffic. Therefore, in this structure, you can use well-known methods for both software and hardware implementation of information flow routing, as well as define new paths that are more suitable for robotic applications.

Since the multi-level topology implies the division of fully functional composite modules into simpler modules, the structure of inter-module information interaction should provide for the transfer of changes in the weight and size parameters of the modules in the case of robot reconfiguration.

Most embedded computing devices, which are mandatory in multi-level CS, have a CAN interface. This is why special attention has been paid to this standard in this work. However, since CAN has a fairly low bandwidth, the faster Ethernet standard is also considered. Both standards were compared with the actual needs in terms of information distribution of various modules.

In the future, it is necessary to conduct a more detailed analysis of the applicability of various networks, such as I2C, SPI, RS-232, RS-485, CAN, USB, Ethernet, EtherCAT, at different levels of the hierarchy. You need to define requirements for network resources at different levels of the hierarchy and restrictions on their use. The main goal of further research is to create new specifications for the method of inter-module interaction for other networks and protocols.

11. Acknowledgments

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

12. References

[1] Andreev, V. & Poduraev, Y. (2016). Network–Based Design of Heterogeneous Modular Mobile Robotic Systems, Proceedings of the 27th DAAAM International Symposium, pp.0004-0009, B. Katalinic (Ed.), Published by DAAAM International, ISBN 978-3-902734-08-2, ISSN 1726-9679, Vienna, Austria. DOI: 10.2507/27th.daaam.proceedings.001.

[2] Andreev, V.; Kim, V. & Pletenev, P. (2017). The Principle of Full Functionality – the Basis for Rapid Reconfiguration in Heterogeneous Modular Mobile Robots, Proceedings of the 28th DAAAM International Symposium, pp.0023-0028, B. Katalinic (Ed.), Published by DAAAM International, ISBN 978-3-902734-11-2, ISSN 1726-9679, Vienna, Austria. DOI: 10.2507/28th.daaam.proceedings.003.

[3] Andreev, V.; Kim, V. & Pryanichnikov, V. (2019). A Hierarchical Modular Architecture for Reconfigurable Mobile Robots, Proceedings of the 30th DAAAM International Symposium, pp.1152-1159, B. Katalinic (Ed.), Published by DAAAM International, ISBN 978-3-902734-22-8, ISSN 1726-9679, Vienna, Austria. DOI: 10.2507/30th.daaam.proceedings.162.

[4] Andreev, V. & Kim, V. (2016). Control System and Design of the Motion Module of a Heterogeneous Modular Mobile Robot, Proceedings of the 27th DAAAM International Symposium, pp.0586-0594, B. Katalinic (Ed.), Published by DAAAM International, ISBN 978-3-902734-08-2, ISSN 1726-9679, Vienna, Austria. DOI: 10.2507/27th.daaam.proceedings.086.

[5] Andreev V.P., Kim V.L., Pletenev P.F. (2018). Hardware & Software Solution for Rapid Reconfiguration of Heterogeneous Robots, *Mekhatronika, Avtomatizatsiya, Upravlenie*, 2018, vol.19, no.6, pp.387-395. DOI: 10.17587/mau.19.387-395.

[6] 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, pp.0010-0019, B. Katalinic (Ed.), Published by DAAAM International, ISBN 978-3-902734-22-8, ISSN 1726-9679, Vienna, Austria.

[7] Mayoral, V.; Hernandez, A.; Kojcev, R.; Muguruza, I.; et al. (2017). The shift in the robotics paradigm – the Hardware Robot Operating System (H-ROS); an infrastructure to create interoperable robot components, NASA/ESA Conference on Adaptive Hardware and Systems (AHS), Pasadena, CA. – 2017. – pp.229-236.

[8] 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.

[9] 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.

[10] Losada, D.P.; Fernández, J.L.; Paz, E. & Rafael Sanz (2017). Distributed and modular CAN-based architecture for hardware control and sensor data integration. Sensors. – 2017. – No.17. – pp.1013-1030.

[11] 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.

[12] 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.

[13] Baca, J.; Ferre, M. & Aracil R. (2012). A heterogeneous modular robotic design for fast response to a diversity of tasks, Robotics and Autonomous Systems. 2012. Vol.60. No.4. – pp.522-531.

[14] Pletenev P. F. (2017) 1/PIMI – Protocol of intermodular interaction Available at: https://asmfreak.github.io/modular_robots_rfc/1/ПММВ/. Accessed: 20.01.2017.

[15] Robotics — Modularity for service robots — Part 1: General requirements (2019). ISO/DIS 22166-1:2019(E).

[16] Bradner S., Key words for use in RFCs to Indicate Requirement Levels/ URL: https://tools.ietf.org/html/rfc2119 (accessed: 20.01.2019).

[17] Andreev, V. & Tarasova, V. (2018). Identification of the Obstacle Shape Using the Ultrasonic Sensors Module of Modular Mobile Robot, Proceedings of the 29th DAAAM International Symposium, pp.1001-1009, B. Katalinic (Ed.), Published by DAAAM International, ISBN 978-3-902734-20-4, ISSN 1726-9679, Vienna, Austria. DOI: 10.2507/29th.daaam.proceedings.143.

[18] Klehmet, U.; Herpel, T.; Hielscher, K. & German, R. (2008) Delay Bounds for CAN Communication in Automotive Applications, 14th GI/ITG Conference - Measurement, Modelling and Evaluation of Computer and Communication Systems, Dortmund, Germany, 2008, pp. 1-15.

[19] Leach P., Mealling M., Salz R. A Universally Unique IDentifier (UUID) URN Namespace URL: https://tools.ietf.org/html/rfc4122.

[20] Universally Unique Lexicographically Sortable Identifier (2017) URL: https://github.com/ulid/spec.