Software Models for Investigation of Turbo-Product-codes Decoding

Yaroslav Krainyk[1[0000-0002-7924-3878]], Ievgen Sidenko[1[0000-0001-6496-2469]] and Oleksandr Kylymovych[1]

[1] Petro Mohyla Black Sea National University, 10. 68 Desantnykiv str., Mykolaiv, Ukraine
yaroslav.krainyk@chmnu.edu.ua, ievgen.sidenko@chmnu.edu.ua, kilimovich.alexandr@gmail.com

Abstract. In the following paper we provide analysis on testing procedure and development of software models, and application architecture for investigation of error-correcting codes' parameters and decoding algorithms. The proposed models can be applied for the development of software for decoding Turbo-Product Codes (TPC). They allow simplifying development process and retrieve universal solution for TPC investigation. The models are described in the Unified Modeling Language (UML) and follow design pattern recommendations. They can be used for software implementation in various programming languages that support object-oriented model. Heatmap visualization tool is supposed to be the main part for visual investigation of the decoding process. In this work, we propose metrics for heatmap organization and explained behavior of the cells to deliver comprehensible presentation of the message state during decoding process. The combination of metric and heatmap component provides effective way to observe impact of each decoding algorithm element on the process and gives abundant information for comparative analysis about algorithm improvements.

Keywords: Turbo-Product code, software model, decoding, metric.

1 Introduction

Telecommunication systems are complex systems that includes various set of components on the different levels. One of the substantial parts is coding systems where error-correcting codes (ECC) are heavily utilized [1-4].

The problem of investigation of ECCs' parameters is one of the most important issues to achieve better performance of the system. Different codes reveal different characteristics under changeable conditions. Even slight optimization might be critical for the realization of the final system. It also should be mentioned that it requires much time to meticulously check combination of parameters of the code, compare different codes with each other, and select the most suitable code according to the expected properties.

The first testing stage is usually performed on software level as it takes minimum time to verify proof-of-concept. Testing on hardware level implies actual implementation of the device that can be insufficient. However, available software solutions either

provide only limited functionality without possibility to change coding parameters or requires payment for the software usage. Consequently, there is a lack of software that provide flexibility in configuration of the code and decoding algorithms.

Decoding algorithms are the cornerstone of the coding level in telecommunication systems as throughput as well as complexity are directly connected with them. Their implementation requires execution immense number of computational operations.

Testing of the ECC is concerned with processing tremendous amount of data. Input message, intermediate results, decoding results, computations performed according to the algorithm form the list of the most important information required to investigate decoding performance. It is important to observe transitions inside the message and conclude how each part of the algorithm makes impact on the decoding process and whether it can be improved.

In this paper, we propose architecture and models for development of software that assists in investigation of ECC and respective decoding methods. The proposed software models are supposed to be extensible in terms of adding new codes or decoding algorithms. The software is designed to work with block ECC but can be improved by adding support of convolutional codes.

2 Analysis of Scientific Sources and Software Solutions

Software decoding of ECC is represented by several well-known software packages and libraries [5-9]. In general, they contain several additional components besides decoding process itself. Typically, encoders, channel models, various modulation schemes are the part of the libraries to represent all stages of data transmission over channel.

Before considering ECC software itself, it is worth to mention that various ECCs are employed in the modern electronic applications. Two most popular fields are data transmission and data storage. However, some of the codes are preferable due to the characteristics they demonstrate. The most important parameter of ECC is error-correcting ability that strongly depends on the procedure of decoding and data representation. Another significant parameter is throughput. Once again, typically, it depends on the number of operations in the decoding algorithm, their complexity, and suitability for parallelization. The following group of the ECCs stands out among the others according to their involvement in different technologies and their main properties:

- Low-Density Parity Check (LDPC) codes (have high correction ability, suitable for complex systems) [2];
- Polar codes (claimed to be the best ECC, require further investigation);
- Hamming codes (simple ECC that is a basis for Turbo-Product codes);
- Reed-Solomon codes;
- Turbo-Product codes (TPC; constructed from Hamming codes or other ECC to achieve better performance) [3].

In this work, we are investigating software development for TPC-decoding system. Therefore, it is worth mentioning that many decoding algorithms can be applied to process TPC. Chase, Chase-Pyndiah [4], syndrome decoding, etc. The decision on the algorithm choice is actually a trade-off between computational complexity and correction ability. This is the reason why some of the algorithms are not feasible for concrete systems as they are hard to implement.

Matlab [5] system includes Communication toolbox that implements full stack of the technologies that work in the telecommunication field. It contains abundant amount of models, classes, and functions that greatly simplifies research on almost every topic connected with data transmission. However, Matlab is a product that distributes under commercial license. Another important drawback of the Matlab is the trade-off of ease of use and speed of the test execution, which can be significant factor in case of very large arrays of data to process on the test stage.

A Fast Forward Error Correction Toolbox or AFF3CT [6] is software library and command-line application for investigation properties of the most common codes and respective decoding methods. The project is open-source and available on GitHub. This toolset provide exhaustive information on the results of the decoding. It greatly simplifies investigation of the code and evaluation of its performance. However, AFF3CT have no options for analysis of the intermediate results during decoding.

Other software libraries for ECC investigation like [7-9] provide abundant set of tools for modeling decoding process. Each of the libraries has specific strong points and brings many advantages for the developer. However, decoding algorithm in this libraries does not suppose to return intermediate decoding results. Another drawback is complexity of bringing your own algorithm into work with the libraries.

Companies specialized in ECC-development provide their own software and hardware implementation [10]. They also provide comprehensive analysis on the results that can be achieved using proposed tools. However, this kind of support is limited within defined range of tools flexibility. Customization and adoption to the specific and changing demands falls short for these solutions. Therefore, software that provides higher level of flexibility for investigation and searching for optimal solution is actual from both scientific and industry point of view.

The codes can be applied in various industrial application [11] for improvement of communication part of the system and remote control of the device in case of custom communication protocol usage.

The vital point for decoding software is comprehensible visual presentation of data. Heatmap is a special type of diagram that is very convenient for visual representation of quantitative data in form of matrix. Because of that factor, we selected heatmap as main component for data presentation.

In the paper we investigate modelling of the decoding process for implementation in software, provide software models for the further software realization. We also provide detail on data presentation in software in the form of heatmap. The established software models are intended to be used in the software development process for investigation of TPC-decoding and other ECCs. We apply experience and results attained in the work [12-14] for the following investigation.

3 Main Part

Typically, the following stages are an obligatory to model signal transmission over network:

1. Encoding.
2. Transmission over the noise channel.
3. Decoding received message.

Depending on the detalization level of a system, a few additional stages (scrambling, interleaving, modulation, manipulation, etc.). However, those abovementioned three stages are the basis for investigation.

At the encoding stage, the input message is processed with encoding algorithm (multiplication of each component on corresponding generation matrix in case of TPC). Consequently, the message is altered with new information that assists in error correction at the decoding stage. The message for transmission at this stage is considered as a set of binary data.

Data transmission stage is intended to model influence of noise to the signal. As a result of this stage, errors appear in the transmitted signal. Moreover, due to performance reasons, each bit in the data is presented as soft value with limited quantization (for frugal memory consumption), e.g. it can be range [-3; 3] for case of 3-bit representation.

Decoding stage plays utmost important role in the system. It performs processing on the message according to the decoding algorithm and outputs actual data for further usage once again in the form of binary message without redundancy data. In the most cases, decoding algorithm is applied several time to the message. Number of iteration is restricted. The results of each iteration are featured by the message state and decoding algorithm's parameters. Additionally, knowledge of the initial message implies that we can gain an advantage of directly comparing input and output of the system.

After the performed analysis, we can identify main entities for the software decoding system:

- message;
- decoding results;
- decoding algorithm;
- decoding system.

First, we propose the following workflow for researcher who works with software. The workflow is illustrated in Fig. 1 in form of Unified Modelling Language (UML) activity diagram.

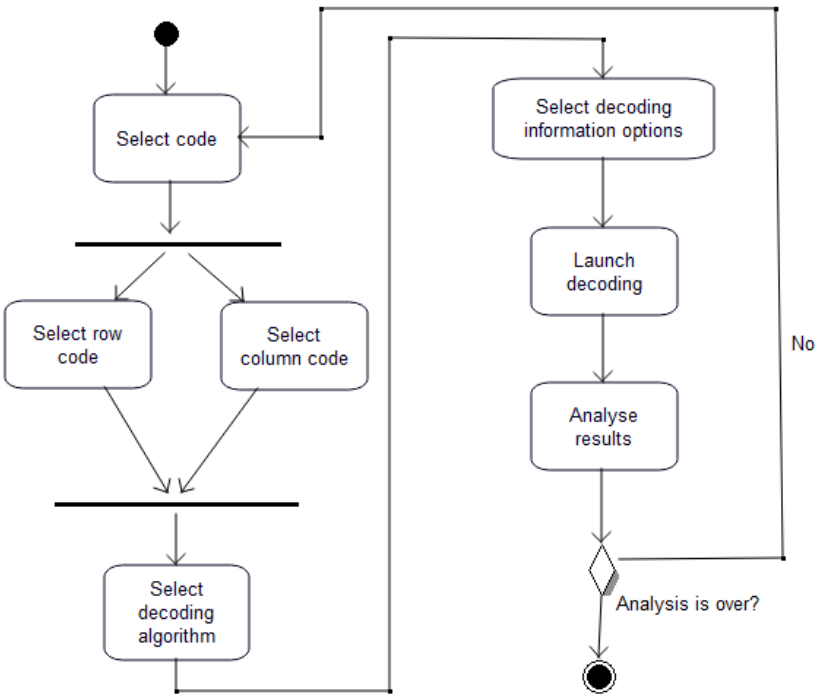

Fig. 1. Activity diagram for user actions

The user sets decoding parameters and launches decoding process. When the results of the decoding are available, analysis stage begins. That actually means work with user interface and finish of the decoding process itself. However, single run of the main cycle gives results only for one code. Comparison of several algorithms is more effective procedure for investigation and selection of better approach. One of peculiarities of TPC is that single decoding iteration is divided on two half-iterations. First, the message is processed in one dimension (e.g. rows) and then takes place processing in other dimension. Row decoding affects column decoding and column decoding affects row decoding. Thus, it is necessary to grant user option to store results of both half-iterations (one of them matches final iteration results).

Let us further introduce class infrastructure for the software that is illustrated in Fig. 2. The class with corresponding name represents decoding algorithm. It is an abstract class that serves as a parent class for other classes that actually provide algorithm implementation. Hence, presence of multiple algorithm is supposed by the system design. It follows instruction for the Strategy design pattern which means that user can easily switch from one decoding algorithm to another. Regarding benefits of class parametrization, decoding algorithm classes may also be parameterized. In this case, only single decode method should have type parameter. However, due to the extension reasons, new method or even whole class may become parameterized.

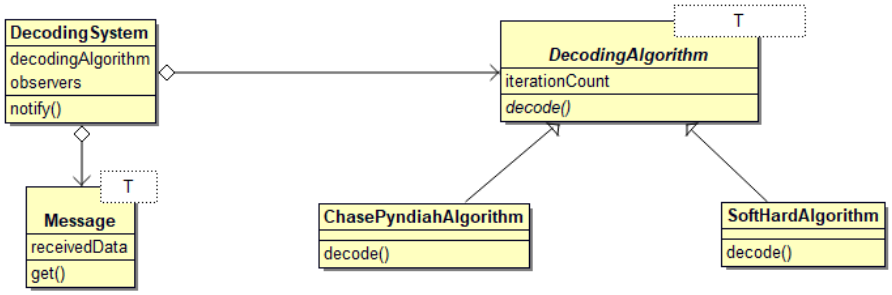

Fig. 2. Main classes of the system under development

Another important peculiarity of the TPC-decoding is that hardware implementation is performed for the algorithms that make use of limited precision in number representation. Typically, operations with integer operands are more preferable. Thus, it is important to observe results yielded by algorithm with different precision in number representation. For that purpose, decoding classes may be parameterized with type for number representation.

From this point of view, we can deduce that several instances of the same class can be present with different type parameter. Creation of the corresponding class instances is a task instance of class that implements Factory design pattern. As the user selects new decoding algorithm to explore, the appropriate call is directed to factory objected. However, it will not be efficient usage of resources if the requested instances has been created previously. Therefore, the factory object should also manage internal algorithm pool object where they are stored. If the requested object is available in the pool, it is passed back to the caller. Otherwise, factory creates it and stores it into the pool immediately. This part of the software model follows instructions of the Object pool design pattern. The corresponding class diagram is demonstrated in Fig. 3.

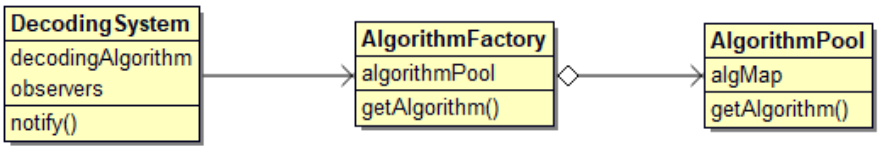

Fig. 3. Classes related to the creation of algorithm objects

Several options are available to retrieve data for testing:

1. Actual channel data stored in the file. This is the simplest variant, as it only requires reading data from the file. It is also a convenient way to check performance two different decoders on the same data and compare their performance with each other.
2. Simulate transmission process over the noisy channel. This option is much closer to the real world functioning of the system. It begins with reading data to encode from file or generating random data for this purpose. The data are altered “on the fly”

according to noisy channel's parameters. More computational operations are required for the simulation then simple reading from file.

Thus, we can identify more artifacts for the application. The first one is data loader that executes load of data from the specified source. The loader is aware that there are two cases for reading data (raw not encoded data, transmitted data). The second artifact is a channel simulator. Main responsibility of the simulator is to apply noise or other channel impact to the encoded data. The relations between mentioned classes are depicted in Fig. 4.

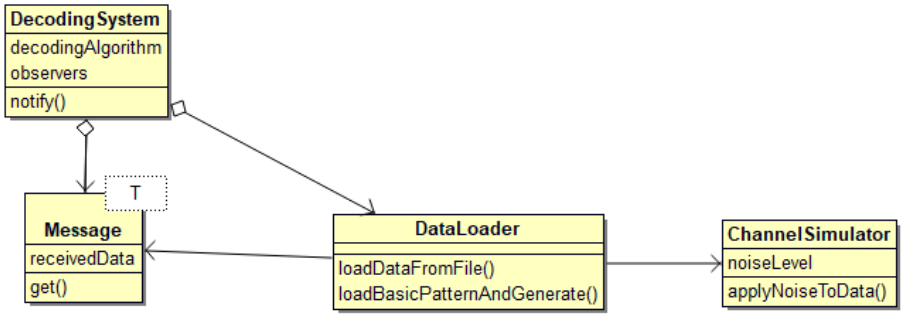

Fig. 4. Data loading infrastructure

One of the most significant features of decoding software is possibility to analyze decoding results and compare outputs of different algorithms. To collect information on the decoding results we propose usage of an Observer pattern. Classes connected with decoding should be able to issue notification about the end of each half-iteration and expose necessary information to the interested observing instances. As notification is sent, observer retrieves information and stores it into the list. Each data chunk can be wrapped into instance of class for storing data to distinguish this information from the original message. Additionally, it provides comprehensible way to control change history of the message under processing. This option is very helpful for user interface implementation because demonstration of changes requires simple traversing over the list with stored instances. The classes for implementation of result storage and review functions are shown in Fig. 5.

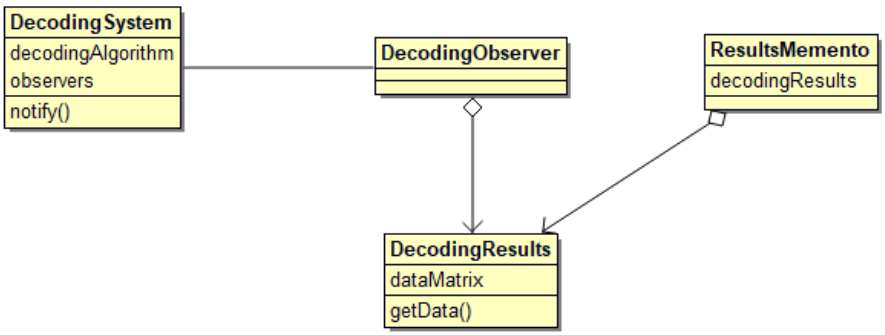

Fig. 5. Classes concerned with decoding results storage

All the models proposed in this paper have been developed using Bouml modeling software [15].

Visual data representation of the decoding results is an actual feature for the decoding software. In opposite to well-known metrics for ECC system efficiency, it gives more details on how decoding algorithm deals with message information. Such low-level information also provides an opportunity to optimize algorithm by tuning parameters or specific decoding operations. As encoded message for TPC is a matrix of numbers, it is handful to represent it in form of heatmap. Each cell in the heatmap corresponds to the actual value in the message. Because heatmap applies color to the values, it is easier for the user to identify error patterns in the message, overview general message state, etc. Heatmap is also an effective tool to observe changes happened on the different iterations. As has been stated before, heatmap assigns specific color to each value. Straightforward usage of the numeric values to form color map gives no additional information and is hard to analyze. This implies that additional metric is necessary to form heatmap. Providing that initial message is known, we can infer message state at the very start of transmission. We assume that this is the desired state for output message. The numerical value $s_{i,j}$ corresponding to each bit takes either maximum value $\max_s$ or minimum value $\min_s$ in the interval [$\max_s$, $\min_s$] in the start message. Value $d_{i,j}$ in the decoded message takes value from the very same interval [$\max_s$, $\min_s$] but it does not necessarily need to be equal to $\max_s$ or $\min_s$. Being aware of two values the for bit representation, we can devise metric that expresses difference between desired and actual values

$$m_{i,j} = | s_{i,j} - d_{i,j} | . \quad (1)$$

The metric can have only non-negative values. The lower metric, the better value $d_{i,j}$ matches to the desired value. In fact, when $m_{i,j} = 0$ it means that decoding converged i, j bit to the correct value. The maximum value that can be assigned to the metric is

$$m_{\max} = |\max_s| + |\min_s|. \quad (2)$$

If some bit position gets $m_{\max}$ metric value, that identifies decoding for bit finished with absolutely wrong value. Let us assume that final binary decoded value is identified according to the sign of the associated numerical value, e.g.

$$h_{i,j} = 1^{1-sign(d_{i,j})} \quad (3)$$

where *sign* – function that returns -1 or +1 depending on sign of input parameter. Small bias from the target value does not end up with wrong decision about binary value in this case. Thus, on a heatmap we should clearly distinguish erroneous positions from correct ones. In most cases, we can identify error if the following condition is satisfied

$$b_{i,j} > \frac{m_{\max}}{2} \quad (4)$$

if $m_{\max}$ is even and

$$b_{i,j} > \frac{m_{\max} + 1}{2} \quad (5)$$

if $m_{\max}$ is odd.

It clearly states that the sign in the decoded value $d_{i,j}$ has been changed due to large bias value $b_{i,j}$. Sign transition in the heatmap should be marked by color for ease of visual interpretation.

Besides metric for each bit in the message, it is also convenient to have an aggregate metric that represents state of row or column of the message. Let us use T_{ri} to denote aggregate metric for i row and T_{cj} to denote metric for j column. Their values are calculated as

$$T_{ri} = \sum_j b_{i,j}, \quad (6)$$

Obviously, minimum value for the metrics is zero. The maximum value reaches $m_{\max}$ multiplied by the number of elements in the corresponding dimension. Therefore, range of possible values for the metrics is much larger than for individual bit. The color scale for aggregate metrics presentation should not contradict with general style of heatmap.

Regarding colors selection for heatmap, we may conclude with next statement. Two colors have to be selected to denote absence of bias and maximum level of bias (e.g. green and red are an essential choice). Errors in the decoded message should be easily recognizable. That means that transition from acceptable bias value (no error) to erroneous one in context of color presentation should not be smooth but rather stiff.

In the Fig. 6 illustration of the proposed approach to visual organization of heatmap is shown for TPC constructed of (7, 4) Hamming codes with $d_{i,j}$ takes value within range [-2, 1].

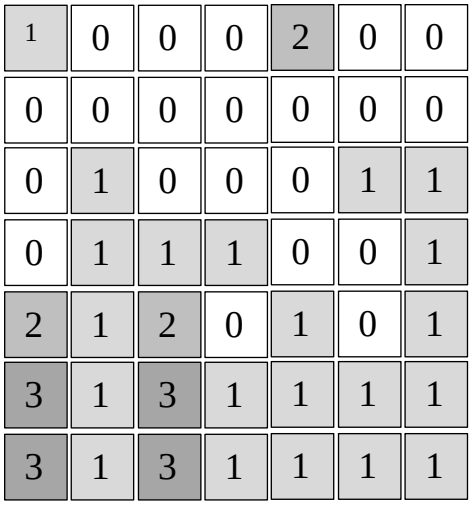

Fig. 6. Sample heatmap representation

Each cell contains its metric $m_{i,j}$ in the center. The higher value of metric, the darker background color. We can easily identify erroneous bits by the background. This example uses short codes. In case of long codes, it is feasible to make optional metrics display inside the cell.

The system has been implemented using JavaScript programming language and React library [16]. The developed application can run on desktop computers in browser as well as on mobile device. The application uses heatmap component compatible with React library. We selected sample message for the code with 16 rows and 16 columns and with quantization in range [-7, 7] In the Fig. 7, results of decoding for first and last iterations are presented.

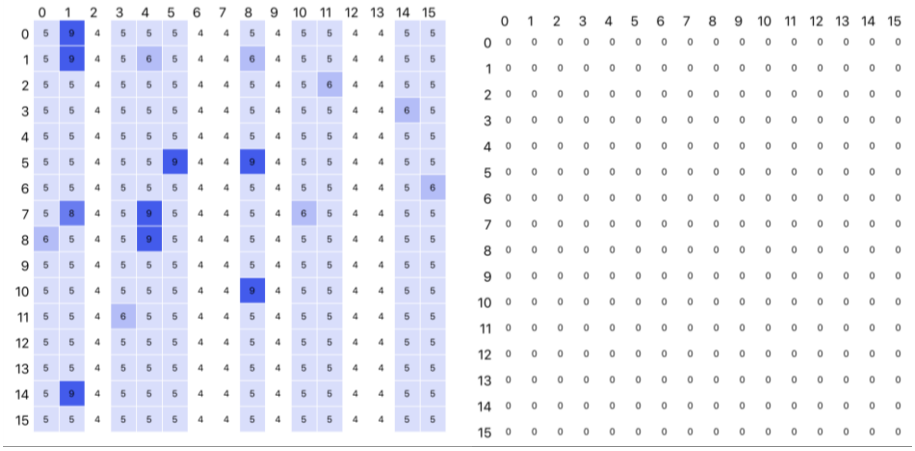

Fig. 7. The state of the message after first and last decoding iteration

The presented samples demonstrates that after the first iteration message contained errors (positions with metric values equal to 9). However, the result contains only zeros in all positions which means that all errors have been corrected. The heatmap component displays color information as shades of blue color. Hence, the more intense color, the bigger distance to the correct value.

The authors are looking forward to cooperation with other researchers and developers who are interested in the topic. We encourage everyone to contact us so the access to the code repository can be provided freely. Moreover, after investigation of the applicable software licenses for open-source projects and finalization of the application code, the decision about moving to the public domain is going to be made to disseminate findings of the presented paper and to facilitate further developments.

4 Conclusions

In the current paper, we present developed software models for ECC software design, specifically for software that deals with TPC. The software models leverage best practices of object-oriented programming to provide flexible solution for TPC investigation. The main advantage of the proposed model is that they take into consideration peculiarities of TPC-decoding process and support different option for accessing data on every stage of the decoding, comparing different algorithms, and different numerical data representation. The models represent workflow of the user actions, main classes that forms the whole decoding system, and supportive classes that are responsible for data loading, presentation, communication between classes, etc. The models can be used as a fundamental part for decoding software and have capability for further extension.

References

1. Tomlinson, M., Tjhai, C.J., Ambroze, M., Ahmed, M., Jibril, M.: rror-Correction Coding and Decoding: Bounds, Codes, Decoders, Analysis and Applications. Springer, (2017).
2. Gallager, R.: Low Density Parity Check Codes. Cambridge, Mass (1963).
3. Berrou, C., Glavieux, A., Thitimajshima, P.: Near Shannon limit error-correcting coding and decoding: Turbo-codes. 1. In Proceedings of ICC ’93 - IEEE International Conference on Communications, pp. 1064–1070, vol. 2, IEEE, Geneva, Switzerland (1993).
4. Pyndiah, R.: Near-optimum decoding of product codes: block turbo codes. IEEE Transactions on Communications, 46(8), pp. 1003–1010 (1998).
5. Matlab – MathWorks – Matlab & Simulink, https://www.mathworks.com/products/matlab.html, last accessed 2019/02/02.
6. AFF3CT – A Fast Forward Error Correction Toolbox, https://aff3ct.github.io/, last accessed 2019/02/04.
7. Forward Error Correction (fec), http://liquidsdr.org/doc/fec/, last accessed 2019/02/05.
8. Github, https://github.com/simonyipeter/Arduino-FEC, last accessed 2019/02/03.
9. Forward Error Correcting Codes, http://www.ka9q.net/code/fec/, last accessed 2019/02/06.
10. AHA Products Group, http://www.aha.com/, last accessed 2019/02/06.
11. Kondratenko, Y., Gerasin, O., Topalov, A.: A simulation model for robot's slip displacement sensors. International Journal of Computing, Vol.15, Issue 4, pp. 224-236 (2016).
12. Krainyk, Y., Perov, V., Musiyenko, M., Davydenko, Y.: Hardware-oriented turbo-product codes decoder architecture. In Proceedings of 2017 9th IEEE International Conference on Intelligent Data Acquisition and Advanced Computing Systems: Technology and Applications (IDAACS), pp. 151–154, vol. 1, Bucharest, Romania (2017). DOI: 10.1109/IDAACS.2017.8095067
13. Krainyk, Y., Perov, V., Musiyenko, M.: Low-complexity high-speed soft-hard decoding for turbo-product codes. In Proceedings of 2017 IEEE 37th International Conference on Electronics and Nanotechnology (ELNANO), pp. 471–474, Kyiv, Ukraine (2017). DOI: 10.1109/ELNANO.2017.7939798
14. Musiyenko, M., Krainyk, Y., Denysov, O.: Reconfigurable decoder for irregular random low density parity check matrix based on FPGA. In Proceedings of 2015 IEEE 35th International Conference on Electronics and Nanotechnology (ELNANO), pp. 498–503, Kyiv, Ukraine (2015). DOI: 10.1109/ELNANO.2015.7146937
15. BOUML – a free UML tool box, https://www.bouml.fr/, last accessed 2019/02/09.
16. React – A JavaScript library for building user interfaces, https://reactjs.org/, last accessed 2019/02/07.