DOI: 10.13009/EUCASS2017-399

Programming technology for autonomous adaptive control using on-board many-core platforms

Alexey Syschikov, Sergey Pakharev, Boris Sedov and Yuriy Sheynin
Saint-Petersburg State University of Aerospace Instrumentation
67, Bolshaya Morskaya str., Saint-Petersburg, 190000, Russia

Abstract

The paper considers the complex technology that consistently covers multiple aspects of the development of autonomous adaptive control in aeronautics systems. The major challenges faced by developers of modern on-board systems are advanced parallel and distributed computing technologies required for heterogeneous multicore and many-core architectures of embedded hardware platforms. Presented programming technology aimed to create a software for on-board data-processing, targeting distributed autonomous adaptive control systems.

1. Introduction

Nowadays unmanned autonomous systems become more and more popular: vehicles, aircrafts, robots etc. At the same time we can see an increasing of number of functions which this systems shall execute. Today autonomous systems are not just collecting and processing some simple datasets from external sensors, but they also perform video processing, produce terrain map and made decisions in real-time without any manual control. To ensure that all these functions were carried out in real time it's necessary to use complex computing systems, consisting of many computing units, merged into a single structure. For this purpose developers often use MPSoC [1], which contains many different cores – DSP, CPU and various co-processors. But with an increase of architectural complexity of such systems, the complexity of their programming also grows up.

One of the main fields of domain application of many-core systems is autonomous systems. These systems can be found in all spheres of life, from the popular "smart home" concept to military applications. Evolution of autonomous systems can be divided into 5 main stages:

1) Fully manual systems.
2) Implementation of the simplest devices to interact with the external environment. In addition, these devices are able to have a very narrow set of output parameters. For example, in case of the development of self-driving cars, individual and simple driver assistant features, like anti-lock-braking systems (ABS), which are widely used at most modern cars.
3) Systems, combining assistant features leading to complex behavior. These systems are dynamic and adaptive – they must adapt to a control system with variable parameters. The difference between simple and combining assistant features is that now behavior depends on environment.
4) Semi-autonomous systems for a specific tasks. These systems can make their own decisions that do not require human attention, but only for "template" tasks. Such systems are able to make independent decisions that do not require human attention for any specific, template situations. For example, there are now autopilot systems on the aircraft, which allows landing aircraft without aircraft crew participation, based on the values from sensors.
5) Fully autonomous systems – the systems that can make decisions in all possible situations. Such systems are self-learning, which allows them to predict all possible outcomes of the events and choose the most suitable variants. In addition, such systems absolutely don't require any manual control and perform all the functions independently, interacting between themselves.

Evolution of semi-autonomous systems has produced the need to increase the performance of computing systems that underlie stand-alone devices. To perform most of tasks associated with changing the environment it is necessary to perform the task of pattern recognition and detection of environment changes. Today this task is computationally complex and requires high performance hardware. At the same time, time of decision making depends on critically level. This problem is crucial for avionics, in particular for the development of software for unmanned aerial vehicles (UAVs): the recognition of objects often should be done in real-time. With the increase in number of tasks that UAVs must perform, the need for complex process units, like many-core systems, also grows. So, modern on-board systems has begun to use a hardware with more than one core (often 2 or 4 cores) in order to effectively parallelize existing

DOI: 10.13009/EUCASS2017-399

recognition algorithms, as well as perform distributed calculations to ensure flight safety. However, effective use of the full potential of many-core systems is not an easy task. In addition, the same recognition algorithms are often applied simultaneously in various autonomous systems. In this regard, it is important to have the means to «tune» the software being developed and take into account bottlenecks of hardware.

2. Programming technology

The presented programming technology is implemented in integrated tool flow with various design and analysis tools. All of them can be divided into three groups: frontend, middle-end and back-end (Figure 1). At every stage the developer has the corresponding instruments with a strong feedback and feedforward connections to the rest of the tool flow. This approach was implemented in VIPE IDE [2] – visual environment for coarse-grained parallel programming.

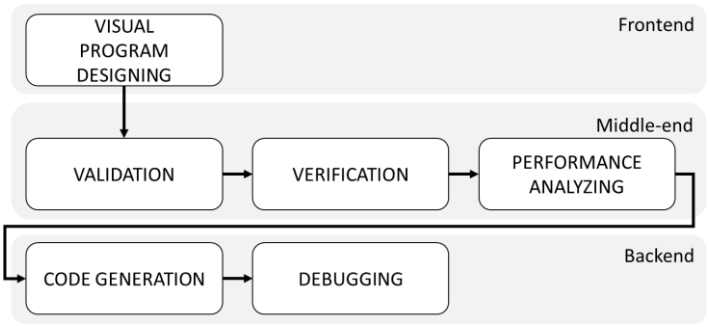

Figure 1 Programming technology tool flow

Programming technology, implemented in VIPE, can be easily used for creating software for various domains from data processing to aerospace industry. But the main area of this technology is a software developing for heterogeneous many-core embedded systems, like vehicles, spacecraft's and UAVs.

2.1 Frontend

As a frontend of presented technology, the toolchain provides visual design approach for creating software. Using hierarchical diagram as a program structure (Figure 2) for creating on-board many-core systems, users can divide software into several fragments and program them collectively. Each fragment can be implemented by separate programmer or domain expert. Among such fragments, there can be image or video processing, decision-making or SAR, which are involved to the different knowledge areas and should be implemented by different programmers or domain experts. This approach is well known for creation of software for mixed-critical systems [3] [4] and become more popular in designing of complicated systems.

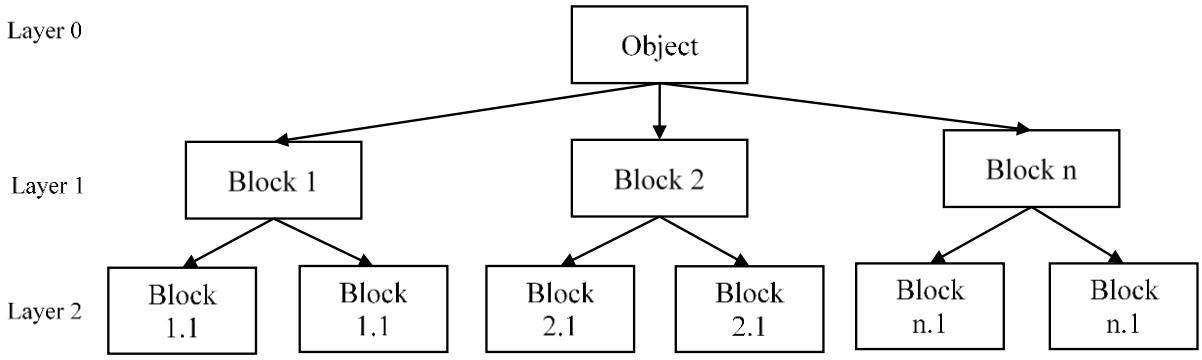

Figure 2 Block-hierarchical programming approach

Visual programming [5] has a great usability for creating systems with strong hierarchy and great modularity. There are a lot of VPL languages, based on different models. The most commonly used models are ones based on control flow and dataflow graphs. Control flow graph (CFG) [6] is a directed graph whose nodes are the basic blocks of a program, it also has two additional nodes, «entry» and «exit». Edges between nodes represent transfers of control (jumps) between basic blocks. In data flow graph (DFG) [7] nodes are computational units that send and receive data

DOI: 10.13009/EUCASS2017-399

messages. Some nodes can only send messages, others can only receive messages, and others may send messages in response to messages they receive. Edges in DFG represent data dependencies between nodes: outgoing edge from node equals to sending a message and incoming edge – to receiving a message.
But in our programming technology we are using different model, which is named Asynchronous Growing Processes (AGP) [8]. In AGP-model control of computation is distributed, parallel and asynchronous. Main difference between AGP-model and CFG or DFG is that AGP is dynamic parallel computation model that covers static parallel computations as its particular cases. AGP contains approaches of both models and contains data flow dependencies and control flow dependencies at the same time, but uses dynamic change of the executable graph. It is based on fundamental research in parallel computations domain and is well suitable for using in heterogeneous or many-core systems. One of the main syntax differences between AGP-model and popular dataflow model is that AGP-model contains two type of nodes: data nodes and operation nodes (an analogue of basic block in CFG and DFG). It has several basic elements to store and share data between processes, like arrays in traditional text-based programming languages. This is very common functionality for most of software developers, who are developing software in languages like C/C++, Java etc. Let's consider the main syntax elements of the language.
VPL language used in VIPE IDE, based on AGP-model, is algorithmic complete, and provides capabilities to organize computations control depending on data values on the level of parallel program scheme. VPL contains three groups of language operators (blocks): terminal operators, data blocks and control operators. Terminal operators are atomic tasks, which can be performed as soon as ready at the same time, in parallel. Terminal operators are implemented by text-based programming languages. For now we are using C/C++ implementation, because it's the most popular language for embedded platforms and for many-core systems [9]. Data blocks are data store objects, which provide to user applying habitual terms of programming like shared variables or arrays. In general, data blocks are untyped storages for data, which can be accessed by any task. Control operators are used to organize hierarchy and modularity of the program and to create program loops or conditional branches. Communication between nodes is organized, as it was said above, by two types of edges: data and control edge. The most frequent form of communication between nodes is data dependency. But if users need to organize order of execution between two independent nodes, they can add control dependency and one node can execute only when the other node will finish executing. They can use it to organize dependencies between different threads in parallel programs.
The main benefit from using VPL for creation of complicated software is that users can separate creating program architecture from tasks implementation with one of the text-based programming languages (Figure 3). Domain expert or user can create visual program representation and he doesn't have to be a professional programmer. The best approach is to attract as many domain experts as possible. For example, one domain expert for image processing, one data-processing expert and one control systems expert. The domain expert designs a set of program blocks and defines their relationships. At the same time, a team of programmers writes code for each block independently knowing nothing about the program structure. Moreover, domain experts can apply optimization tools at the earlier stages of developing by adding approximate time estimation of tasks, which may depend on input data size for each task.

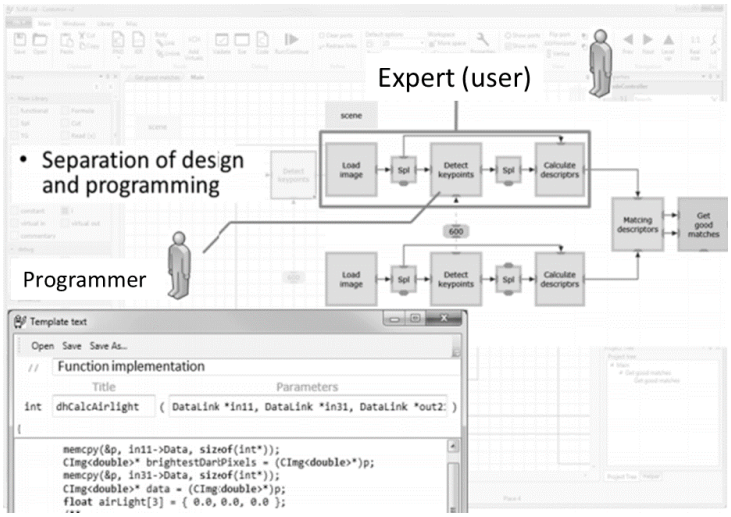

Figure 3 Separation of design and programming in VIPE

At the beginning of software developing domain experts, for example, can begin to create main logic of program from large blocks and then use decomposing program blocks before the best granularity will not achieve. The best granularity for VPL is coarse-grained granularity (about 10-20 code lines per atomic task at high-level language) because at this situation interconnection overhead between tasks is very small on program scale.

DOI: 10.13009/EUCASS2017-399

Users can create program by combining tasks from pre-defined libraries, which are included at VIPE, or create their own libraries or single tasks. This is very flexible mechanism, that provide to user easily create their own tasks and integrate it with existing program if needed, unlike more complicated IDE's.

2.2 Middle-end

Middle-end of the programming technology contains set of tools, which provides to users common and platform-specific program analysis. First of them doesn't depend on target platform specification and can be used for all programs. Other tools can help users to find some platform-specific issues, and these tools are being used for late program analyzing.
Early estimation tools are: validation, verification and static code analyzation tools.
Validation tool ensures that a software system meets language specification (syntax correctness) and that it fulfills its intended purpose. Validation of VPL-program is a high-level check and can be applied only for program scheme. If validation detects fragments built against VPL rules, following optimization tools will not work appropriately because resulting program is not syntactically correct.
Verification tool can be used for parallel programs to find bugs like deadlocks, livelocks, data races and other typical errors, that can be reached by any programmers. Verification is a very useful tool, because it can identify subtle bugs. We use formal verification methods to detect the most common software bugs. As it is known, there are a lot of situations, when missing this step at program analysis stage leads to lethal outcomes or economic collapses.
Static code analysis tool (static performance analyzer) of parallel program in VPL approximately estimates parallelism level and potential acceleration of the program on various hardware configurations. It's a useful tool for quick early estimation of program characteristics. Among these characteristics are performance, parallelism, maximal processors occupation etc. Estimating program logic on early stages provides minimal effort the performance of program on potential hardware many-core platforms. At any stage of the development of an on-board system, a program designer has an opportunity to evaluate the program and identify a further way of its design. Of course, the static analyzer is not designed for getting accurate time measurement of the program. There are too many factors, which depend on hardware platform, runtime and parallel execution model, and all of them affect the program performance. More detailed analysis can be done by using other tools from the performance analysis complex, for example, by using virtual simulator and platform simulator.
A virtual simulator bridges the gap between the hardware and software domains. It's used after static performance analysis and provides an analysis of platform-independent characteristics, such as overall computation time, amount and intensity of data exchange, maximal possible parallelism level, memory usage and other. From this values users can learn maximum possibilities of theirs program on unlimited hardware (with zero latency and infinite number of cores).
For estimating possibility performance of software on real many-core platform users can apply platform simulator. It operates with a virtual hardware platform which is a high-level abstraction of hardware. Platform simulator is a starting point for detailed designed work. By using this tool, software and hardware engineer can gain knowledge about:

- effectiveness of hardware loading;
- requirements to embedded system cores memory and it's usage;
- requirements to embedded system cores performance and it's usage;
- bottlenecks of hardware platform and problems of tasks allocation.

Hardware platform is described by minimal characteristics such as types and characteristics of computation cores and structure and characteristics of their interconnections. When required, users can describe memory model of a platform: maximal storage size and its location (associated with cores or individual memory bank). As one of the examples of program analysis, users can get information about cores load and refactor theirs program in respect to requirements. For example, we have a program for data processing based on radar-detection algorithm. After implementation of this algorithm in VPL, users attach it to the platform simulation tool, run it on target octa-core system and get visual representation of hardware load (Figure 4). This graphic shows us that about a half of the working time only one of the cores was working. At the same time other three cores of the target platform were idle. After that, about 40% of all time system was half idle. And latest 10% was fully-loaded. As a result of analysis, user should refactor his program to advance its performance.

DOI: 10.13009/EUCASS2017-399

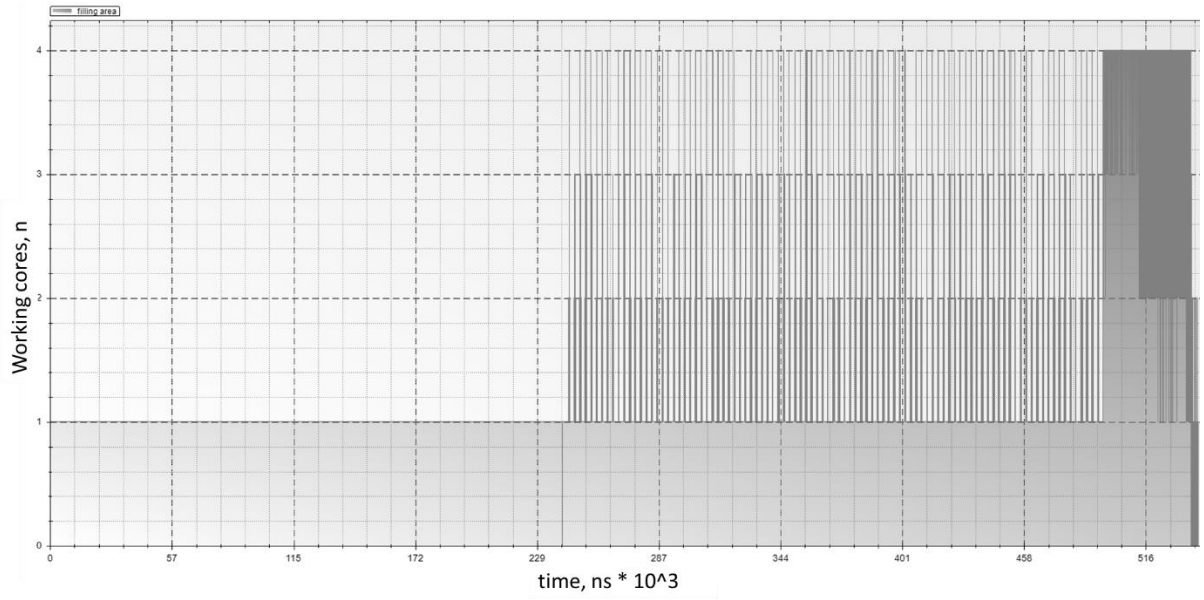

Figure 4 Cores load in data processing

2.3 Backend

The backend of presented programming technology is template-based code generation tools, tools for debugging and deploying software at real hardware. It's necessary to integrate development environment with a target platforms. Using code generation instead of code compiling is one of the crucial ideas of the proposed technology. The reason to use code generation is that code generation is much easier to implement than full compilers. Otherwise, users can apply native compiler to compile resulting code for target platform. Nearly all target platforms have at least one ready compiler to some language.

The technology provides rather simple tools of code generation into a language supported by target platform compiler. If required, users can implement their own code generation tool if this approach can attach some benefits. From the standard package we provide code generators to C/C++ languages with OpenMP 4.0 support, because this is the most popular language for embedded systems. After code generation and compiling, toolchain deploys development results to the set of many-core heterogeneous target platforms with various run-time models.

Debugging is no less important process in software developing. Debugging works in close cooperation with code generation tools. Debugging process in VIPE IDE becomes possible by adding some directives at generated code and creating interconnection between IDE and debugging program. By using this approach, users can add breakpoints into VPL program and observe values of data objects or values transmitted between programming blocks at every step of a program. For debugging we can use physical channel between the developer's PC and target board consisting of a USB connection or wireless connection by SSH, if debugging platform support this type of connection. Debugging on-board platforms can be used to analyze the flight characteristics and behavior of a system under development, which is very important task for on-board systems. Users can attach to hardware platform, run the UAV or aircraft and perform a ground analysis of its behavior in a real-time.

Using of multi target debugging is very comfortable, because users can create software that are easily ported to many hardware platforms without rewriting big parts of program. Using code generation tool, which can produce a parallel code based on a program scheme level parallelism also allows users to port software to another platforms.

3. Autonomous adaptive control model in VPL

There are a lot of types of control model systems. For example, gain scheduling [10] – is an adaptive control strategy, where the gain of the system is determined and based on its value the controllers parameters are changed. This system is very easy to apply, but there is no real learning or intelligence.

DOI: 10.13009/EUCASS2017-399

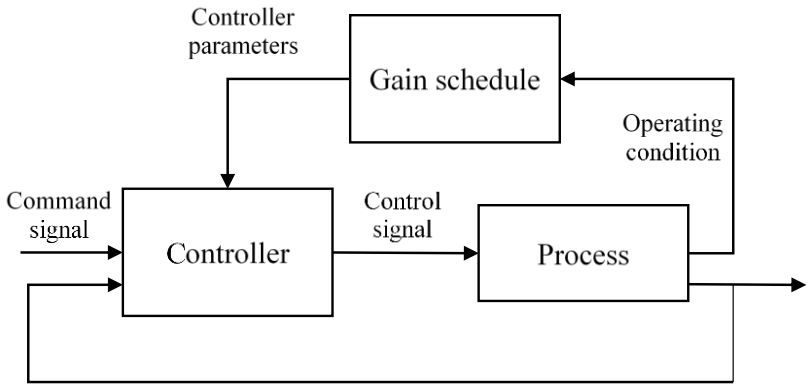

Figure 5 Gain scheduling model

More popular system for flight control is Model-Reference Adaptive Control (MRAC) [11]. It's used to solve a problem in which the performance specification are given in terms of a reference model. This type of models are more "intelligent" then gain scheduling. But to create MRAC-based system we should describe system in terms of theory of guidance and control. And if we create software for various types of hardware, we cannot plan it. We should create system, which will be more "flexible" and mustn't not depend on system hardware model.

Autonomous system should perform many tasks, but main tasks are survival and adaptive learning, like for every life form. Other tasks are minor tasks and they are occur from main tasks: cargo or cargo delivery by autonomous vehicles, detecting and eliminating of targets for military uses, creating terrain map for UAVs, etc. Before describing structure of autonomous system we should denote principal purposes of system:

1) *Image formation task*. The main purpose of this task is to find consistent patterns in the input data coming from external devices and sensors. This regularities should reflect the process taking place in the environment. In mathematics this problem is close to the problem of «automatic classification». As a result of this task, a set of grouped images is obtained, which will be recognized later.
2) *Patterns recognition task*. The task of this stage is to recognize the objects among the formed images and to identify those on the basis of which further work will be done.
3) *Formation of knowledge base*. Formation of the knowledge base about the environment is needed to assess the current state, as well as forecasting based on some models. The knowledge base of the environment contains all classified and recognized objects and their properties.
4) *Decision-making task*. Decision are made on the basis of the knowledge base of the environment, as well as the goals set for the system. For this task, an important factor is the holistic view of knowledge about environment.
5) *Multilevel control task*. This task consists of the implementation of a certain actions, which is determined on the basis of data after the decision-making stage. The executing of actions must be performed any specific time of actions with possibility of parallel execution of some actions.

Based on this five tasks the most suitable model of the system was selected. This model (Figure 6) based on the method of adaptive autonomous control (AAC) [12] [14] – an approach to the design of intelligent systems. AAC was based on structure and functions of the human nervous system and brain. The main idea is to create system which can adapt to environment conditions and can make decisions by itself, while being fully autonomous.

Each part of this system performs its specific function and is also implemented as an independent module. The autonomy of this system is that it can decide to perform a particular action without the intervention of an operator. In addition, the system should be able to self-learning. Adaptability of the model means that changes in the environment lead to changes in behavioral algorithm of the system. At the same time, the system must independently build priorities in solving specific problems, identifying the most important for each moment.

DOI: 10.13009/EUCASS2017-399

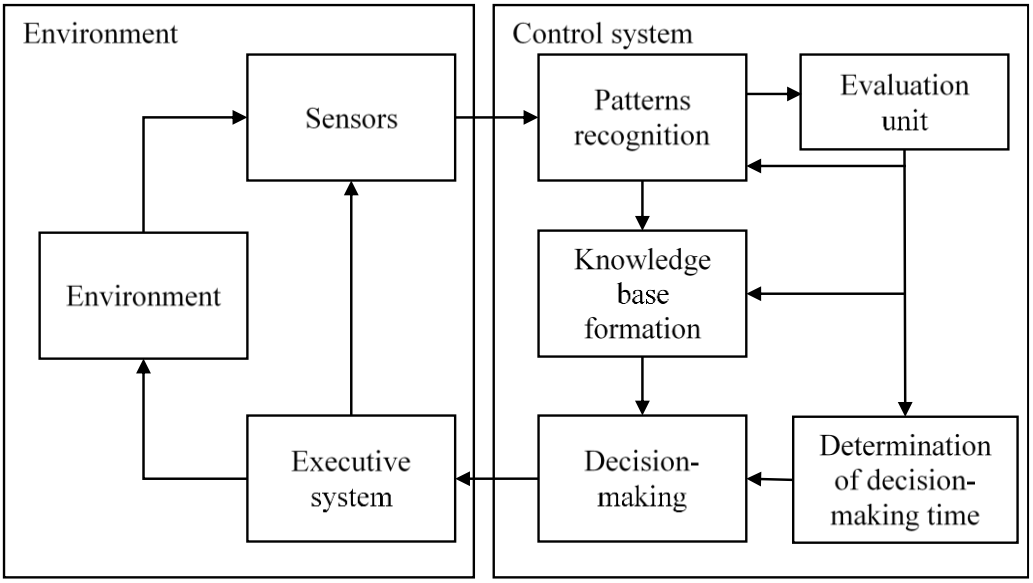

Figure 6 Autonomous Adaptive Control system

The interaction of the system with the environment occurs through sensors blocks to obtain information about the properties of the external environment as well as the actuators that affect this environment. In general, the sensor units and the executive system should be located outside the control system since they do not affect the operation of the system itself, being only actuators. Among the sensors can be both digital and analog sensors and more complex devices that the system can control to perform some action. In addition, when one or more actuators or sensors will fall out the system must correct theirs behavior taking into account information about the failure of these devices.
The control system itself contains four basic elements that correspond to the main purposes of the system:

- Pattern recognition. This unit is focused on the recognition of patterns and regularities in input data. It generally aims to provide a reasonable answer for all possible inputs and perform «most likely» matching of the inputs, taking into account their statistic variation. Algorithms like visual object detection can be implemented in this unit.
- Evaluation unit. It is required for setting some definitions for current state. This unit is based on the results from previous iterations and it sets some characteristics for recognized images, passed by pattern recognition unit.
- Knowledge base formation. At this point system stores current values, obtained from environment in a special format, understandable to the system. Knowledge base contains information and characteristic of external environment from all previous state. This information is needed for evaluation and adjustment.
- Decision making. This element of the system uses prepared information to make a decision about any specific action, based on all system knowledges and evaluations. Among them may be a change in the trajectory of flying, or transmitting some date to the monitoring center.

There is a fifth unit named «Determination of decision making». Its additional unit for time evaluation, in which system should make some decisions. It's a very important value, because for mixed-critical systems there should be implemented real-time work and at the decision-making time system should continue to perform its tasks, control motors and prevent crashes.
The sequence of these elements allows pipelining information with a possibility to parallelize individual stages. For example, the formation of a knowledge base can take place at the same time with the evaluation of the decision-making time. Based on the external environment data, images are formed and every iteration adjusts to changing conditions.
As an example of the implementation of the autonomous adaptive control model we have decided to implement base AAC model for UAVs, contains main functions like images recognition and decision making (Figure 7). This model can be applied for UAVs identifying targets in a video stream, obtained from an on-board camera. Every frame from camera is subject to image processing and detecting three main types of objects: obstacles, targets and others important objects, which may be helpful for environmental assessment.

DOI: 10.13009/EUCASS2017-399

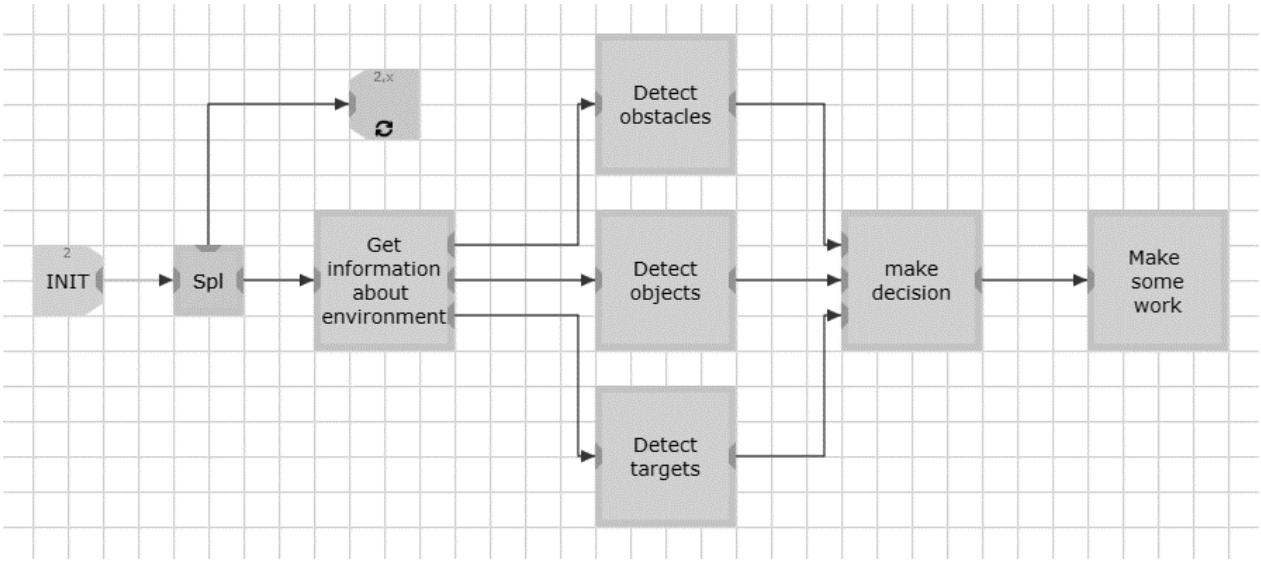

Figure 7 Main part of AAC model in VIPE

Detecting obstacles is a very important task for any autonomous system, because system should predict all critical paths and mustn't crash. It must be performed in a real time, robustly and accurately, without any false alarm and with a low detection failure. Firstly, obstacles must be localized in the open space on the way of flying UAV. In addition, information about obstacles can be used to predict their dynamic evolution. Stereovision is often used for this purpose and the most popular algorithm to do is «V-Disparity» algorithm [13]. But for detecting obstacles in the nearest environment we can use, for example, ultrasonic sensors or LIDARs. The key advantage of using this distance measure based on lasers or sound-effect is that it doesn't need powerful on-board system to process video stream, unlike image-based recognition.

Detected objects and targets can be classified by type of properties of required objects: how we can classify difference between them. To determine objects a lot of tag types can be used: radio-frequency, electro-mechanical or image-based tags. But image-based recognition is the most popular today because it doesn't depend on distance and interferences. To detect objects in image we should implement classification of objects and make algorithms, which we can use in our on-board system.

The best case for all recognition tasks is to compute trajectory for all dynamic objects and predict it for the nearest future. For this we need a large database with full history of changed parameters of the recognized objects and complex predictable system. This is very complicated task and for good precision it's need a lot of estimated parameters of an objects and quality system of automatic objects classification.

At the moment we have implemented the basic algorithms of simple objects recognition like people, faces and pattern recognition. One of the algorithms is SURF – local featured detector and descriptor, which can be used for object recognition, image registration or 3D reconstruction. SURF is significantly faster than SIFT (Scale-Invariant Feature Transform algorithm). The second algorithm is object detection with Haar feature-based cascade classifiers. It's an effective object detection method proposed by Viola-Jones [15]. It is machine learning based approach where a cascade function is trained from a lot of descriptions of a target object (positive and negative). Third algorithm is a stereo vision algorithm for creating a disparity map from environment (Figure 8) [16].This algorithm can be used for detecting obstacles and planning trajectory of UAVs. At the rate of 30 frames per second (fps), an UAV moving at 10 m/s will move forward on one-third of a meter between 2 frames, whereas at 120 fps one acquires new data every 8 cm. So fast detection of obstacles for on-board systems is essential to successful flight.

In addition, our library for UAVs has primitives to control some of them. It contains functions for controlling engine speed, controlling some additional equipment (servomotor for camera control and digital/analog inputs/outputs of hardware platform) and communication between other hardware units by popular protocols like I2C.

Therefore users can combine implemented algorithms, primitive library elements and implemented model of autonomous adaptive control to create their own software for on-board systems. After a creation of visual program they can apply some analyzing tools for obtaining information about performance of the created software and deploy the resulting software, for example, in quadcopter's hardware platform.

DOI: 10.13009/EUCASS2017-399

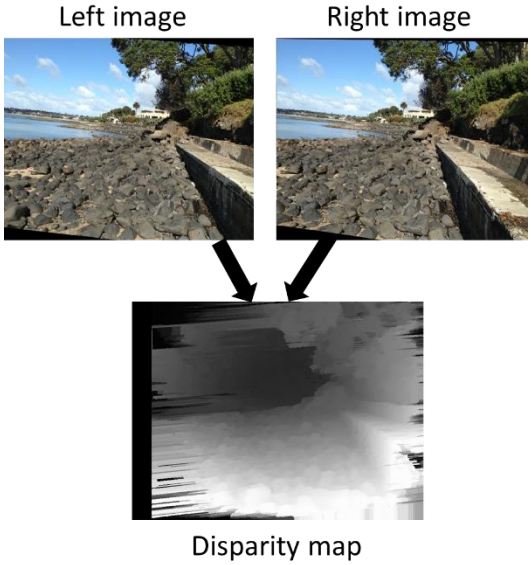

Figure 8 Example of disparity map produced by stereo algorithm

4. Conclusion remarks

In this paper, a new programming technology for autonomous adaptive control using on-board many-core platforms was suggested. The programming technology can also be used in the context of programming UAVs with real-time processing. The key advantage of presented programming technology is the ability to programming on-board many-core platforms easily without unnecessary complications and with a production distributed among domain experts and programmers.

Presented programming technology contains the set of tools that can help developers to create complicated systems with required characteristics. However, there are several certified tools for programming many-core aircrafts. But theirs major disadvantages are high price for small companies or single users and fussiness for general purpose aircrafts, that do not need certified software. Our technology was created for general consumption, because today autonomous systems enter our lives, but there are a lot of issues connected with a programming of such systems. Visual approach for creating autonomous system's software is very useful for people, who are not professional programmers and who do not understand how to tune software manually to avoid low efficiency.

These results are very encouraging and we are looking forward to improve our programming technology and upgrade adaptive autonomous model to take new trends in UAVs into account. We hope that visual programming will spread widely among the people who want to create their own autonomous embedded systems.

Acknowledgements

The research leading to these results has received funding from the Ministry of Education and Science of the Russian Federation according to the project part of the state funding assignment No 8.4048.2017/PP, May, 31st, 2017.

References

[1] Wolf W., Jerraya A. A., Martin G. Multiprocessor system-on-chip (MPSoC) technology //IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems. – 2008. – Т. 27. – №. 10. – С. 1701-1713.

[2] A. Syschikov, Y. Sheynin, B. Sedov, V. Ivanova. "Domain-Specific Programming Environment for Heterogeneous Multicore Embedded Systems". International Journal of Embedded and Real-Time Communication Systems (IJERTCS), vol. 5, ט. 4, 2014. pp. 1-23

[3] Joshi A., Heimdahl M. P. E. Model-based safety analysis of simulink models using SCADE design verifier //International Conference on Computer Safety, Reliability, and Security. – Springer Berlin Heidelberg, 2005. – С. 122-135.

[4] Garza F. R., Morelli E. A. A collection of nonlinear aircraft simulations in matlab. – 2003.

[5] Shu, Nan C. Visual programming. New York: Van Nostrand Reinhold, 1988.

DOI: 10.13009/EUCASS2017-399

[6] Cytron R. et al. Efficiently computing static single assignment form and the control dependence graph //ACM Transactions on Programming Languages and Systems (TOPLAS). – 1991. – Т. 13. – №. 4. – С. 451-490.
[7] Gilles Kahn. The semantics of a simple language for parallel programming //In Information Processing. – 1974. – Т. 74. – С. 471-475.
[8] Ivanov, V., Y. Sheynin, and A. Syschikov. "Programming model for coarse-grained distributed heterogeneous architecture." XI International Symposium on Problems of Redundancy in Information and Control Systems: Proceedings, SUAI. 2007
[9] Nick Diakopoulos and Stephen Cass, "The Top Programming Languages 2016", IEEE Spectrum magazine, 2016
[10] Åström K. J., Wittenmark B. Adaptive control. – Courier Corporation, 2013.
[11] Monopoli R. V. Model reference adaptive control with an augmented error signal //IEEE Transactions on Automatic Control. – 1974. – Т. 19. – №. 5. – С. 474-484.
[12] Zhdanov A. A. The Autonomous Adaptive Control Method //JOURNAL OF COMPUTER AND SYSTEMS SCIENCES INTERNATIONAL C/C OF TEKHNICHESKAIA KIBERNETIKA. – 1999. – Т. 38. – С. 792-798.
[13] Labayrade, Raphael, Didier Aubert, and J-P. Tarel. "Real time obstacle detection in stereovision on non flat road geometry through" v-disparity" representation." *Intelligent Vehicle Symposium, 2002. IEEE*. Vol. 2. IEEE, 2002.
[14] Kamagata S., Kobori T. Autonomous adaptive control of active variable stiffness system for seismic ground motion //Proc., First World Com". on Struct. Control, TA4. – 1994. – С. 33-42.
[15] Viola P., Jones M. Rapid object detection using a boosted cascade of simple features //Computer Vision and Pattern Recognition, 2001. CVPR 2001. Proceedings of the 2001 IEEE Computer Society Conference on. – IEEE, 2001. – Т. 1. – С. I-I.
[16] Scharstein D., Szeliski R., Zabih R. A taxonomy and evaluation of dense two-frame stereo correspondence algorithms //Stereo and Multi-Baseline Vision, 2001.(SMBV 2001). Proceedings. IEEE Workshop on. – IEEE, 2001. – С. 131-140.