Baltic J. Modern Computing, Vol. 1 (2013), No. 3-4, pp. 139-160

Testing of RUAV and UGV Robots' Collaboration in the Simulink Environment

Janis BICEVSKIS, Artis GAUJENS, Janis KALNINS

Institute of Mathematics and Computer Science, University of Latvia,
29 Raina Blvd., Riga, Latvia, LV-1459

Janis.Bicevskis@lu.lv, Artis.Gaujens@lumii.lv,
Janis.Kalnins@lumii.lv

Abstract. The research has been made on modelling and testing cooperation of autonomous robots in order to verify correctness of the whole system operation. The Extended Finite State Machine (EFSM) has been choose as modelling language that enables describing of operation of each separate robot or its parts and ensures simulation in MATLAB/Simulink environment. The collaboration between processes is implemented by events and using common variables in programs. In this research "the complete test set" of robot cooperation has been defined as feasibility of all possible states of the cooperation model. It is certain, that there is no solution for EFSM model feasibility problem in general. The research defines conditions for use of modelling language in case when complete test set problem has been solved, as well as offers technologies/algorithms for creation of complete test set.

Keywords: Autonomous Robots, Model checking, Testing, Validation, Verification.

Introduction

Autonomous robots are intelligent machines capable of performing tasks in the world by themselves, without explicit human control (Bekey, 2005). Use of autonomous robots is increasingly reaches various areas of human activities, often even taking over execution and supervision of activities formerly performed exceptionally by humans. Thus drawing attention to different methods of development and testing of autonomous robot systems (ARS) that would enable verifying correctness of ARS operations in situations influenced by hardly predictable external environment factors.

Unlike development of data processing systems, development of robot system is affected by the fact that system operation tests under real circumstances are difficult to perform. For example, data bases can be copied and developed system tested in test environments before putting it in production, however there are only few situations when tests of robot systems can be run using real hardware (HW) under real circumstances. Therefore intensive efforts are devoted to development of methods and tools/environments that would enable testing of robot systems by operational simulation. One of the most popular approaches offered by MATLAB/Simulink (WEB, (1)) intends, - creation and testing of ARS model in the first step followed by fully or partly automated transfer of the model onto real HW. In this case a significant part of robot system development and test activities is concentrated in MATLAB/Simulink

environment that enables creating/designing, executing/simulating, testing/validating etc. system model without risk creating damages to real hardware. This approach is widely spread in theoretical research and practical applications. This paper proposes an original solution which bases on the symbolic execution of process descriptions and the checking of feasibility of various scenarios. The theoretical base of the approach is worked out according to software testing ideas (Barzdins et al. 1977, Bičevskis et al. 1979, Auziņš et al. 1991). At first, an ARS model consistent with MATLAB/Simulink environment is created. This can be done by using options provided by Finite State Machine (FSM) and programming language C, as well as software that imitate HW behaviour thus substituting real HW. The modelling language that provides extended FSM features and enables to include program code in the model is denoted as EFSM – Extended Finite State Machine. The created model is feasible in MATLAB/Simulink environment, thus enabling implementing significant share of development and testing activities to in simulation mode without real HW employment.

Scientific novelty of this research is defined by symbolic execution of ARS model proving feasibility of all possible states of the model. Conventionally, state of the model is determined by the concept – model state incorporates values of the model attributes. However, since software and software variables are allowed, this approach is not feasible due to huge number of states. The research offers more rational concept of model state including, for example, only relations among model attributes. In this case the number of model states is finite and it allows constructing model's reachability tree practically representing all various scenarios of the model operation that provide feasibility of all possible model states. The reachability tree ensure easy generation of "complete" test set based on which all possible model states are proved feasible and "complete" model test is ensured.

The paper has the following structure: chapter 1 examines ARS development methodology focusing on features provided by MATLAB/Simulink and system tests; chapter 2 provides review of activities performed by different researchers in development of ARS model and description of cooperation among different processes and hardware; chapter 3 provides basic usability example of the proposed methodology thus demonstrating main concepts of methodology; chapter 4 is devoted to analysis of applicability the proposed methodology. The conclusions include research results and chart further research courses of method's applicability.

1. MATLAB/Simulink as ARS development environment

The research examines experience acquired in ARTEMIS, R3-COP project (WEB, (3)), focusing on one type of autonomous robot systems (ARS) in particular – cooperation of autonomous robots RUAV and UGV, where RUAV - a quad rotor unmanned air vehicle has four rotors and requires no cyclic or collective pitch. A quad rotor RUAV can be highly manoeuvrable, has the potential to hover and to take off, fly, and land in small areas, and can have simple control mechanisms (Pounds, 2004). UGV - vehicles with computer-controlled, unmanned driving capabilities. In this particular case UGV is used as mobile energy source in order to recharge RUAV batteries.

The main project goal was development of robot system design methodology, including sub-goal of modelling and testing of autonomous robot cooperation. Further

activities included examination of autonomous systems - RUAV and UGV cooperation modelling and testing phases.

1.1. Choosing/creating simulation environment

In this research MATLAB/Simulink was chosen as simulation environment that provides a number of possibilities needed for creation of simulation model and execution of simulation processes. Developers of embedded systems have created a number of simulation systems but not all of them could be used because of the commercial and/ or functional requirements. The MATLAB/Simulink environment was chosen due to a) a possibility to simulate the collaboration of ARSs in language EFSM that is close to implementation of robots, b) a possibility to translate a simulation model to executable application for many hardware platforms (WEB (2)). The key criterion for choice of the simulation environment was its possibility to model the collaboration of robots by a finite state machine using embedded C-programs for many robot specific operations that can be directly transformed to hardware without any changes.

The environment ensures functionality of model designing/graphical editing, syntactic control, model storage and other conventional functions of graphical editors. However, possibility to combine wide range of model creation and simulation features inclusion of original software into model were the main MATLAB/Simulink feature that determined choice of this particular simulation environment.

1.2. Development of RUAV and UGV cooperation model

The next step after choosing the modelling environment included development of particular ARS cooperation model. Using conventional approach this included a set of modelling language, procedures and cooperation protocols where real HW was substituted by SW procedures (plant). The features provided by the modelling language played a significant role. When model enables specifying simulation process activities relating them to specific HW or SW component of their cooperation protocols, this model can be simulated only in such environment that supports features used to develop the model. In many cases specialized simulation environments are created, however creation of such specialized environments require huge resources available only to large organisations or projects. In this particular case the choice of modelling language came down to MATLAB /Simulink features that incorporate Extended Finite State Machine (EFSM) possibilities substituting HW with SW procedures (plant). The chosen modelling language is sufficient to describe different ways of RUAV and UGV cooperation and various operational scenarios.

1.3. RUAV and UGV cooperation model checking

RUAV and UGV cooperation model checking that follows the model development is necessary in order to validate model quality, e.g. probability of „dead lock" situation or objects involved in cooperation are not allowed to reach states that cause system failure, etc. It is well-known that model checking methods (Karpov, J.G., 2010) are evolving rapidly though facing difficulties with formalization of sophisticated systems. If a model has been created using low level programming language, then model correctness checking is not algorithmically feasible (Auziņš et al., 1991). This research compromises

about EFSM features used in model and model correctness checking, offering an original method to check ARS object cooperation. This method is based on symbolic execution of object activities description including analysis of all possible model states. The proposed approach allows complete check of RUAV and UGV cooperation model including test generation for all cooperation scenarios, in other words the complete test set. This constitutes the main part of the research elaborated in detail in the following chapters.

1.4. RUAV and UGV cooperation model testing

Simulation and testing of ARS processes is ensured by the chosen MATLAB /Simulink environment. ARS model is executed in short time intervals - tick, after each tick calculating variables of model application and performing transition of model processes from one state to another. Thus, operational check of model objects can be performed before HW realisation. This check can be done by threading changes in model attribute values and visualizing them for example on display. By executing the above mentioned test set it is possible to ascertain correctness of ARS all possible variety of operational scenarios. The main novelty of this research is defined by the model checking method based on symbolic model execution in MATLAB/Simulink environment with the following generation of the complete test set and testing of model for this test set.

1.5. Model migration to real HW

Migration to real HW is ensured fully or partially by the MATLAB/Simulink environment, and this served as a key criterion for the selection of simulation environment in the project. The environment enables to translate set of State Chart and C program to application that can be stored in memory and executed in real HW. Certainly, there might be discrepancies between simulation application and real time application caused by time limits in real HW and simulation environment. However proposed approach is favoured in ARS development and allows checking correctness of the ARS with a help of simulation in proper time.

2. Related papers

The approach that includes the development of an initial model with succeeding analysis of its features in the simulation environment (test bed) is discussed in many research works. The most significant disparity is related to features of the modelling language regarding the hardware and the transfer of models to their hardware. For instance the (Saad, E. et al., 2009) contains the experience analysis of Boeing where the special simulating environment for the ARS, the Boeing VSTL was created: “In 2007 Boeing demonstrated the first known twelve vehicle autonomous flight with a single operator. This behaviour includes vehicle and navigation state-based adaptation which increases flight safety. Safety bounds result in controlled actions, including normal landing, open-loop position graceful landing or thrust termination when things go wrong with a vehicle.”

The test bed also includes a recharge station for autonomous recharging of batteries, such as is required for persistent missions. The autonomous recharge station consists of a

landing pad, a battery charger, and control circuitry to be able to remotely control battery recharge capability. The design of the landing pad guides the vehicles into place as it lands to ensure reliable electrical contact is made between the vehicle and the charger. Only minor modifications are required to make the helicopter or ground vehicles compatible with the recharge station.

When a vehicle is low on battery or is disabled for any reason, the mission planner re-plans the area search mission by reallocating the remaining search area among the remaining good vehicles in order to complete the area coverage and guarantee the required probability of detection.

An indoor, multi-vehicle, flight test bed, such as the Boeing VSTL, provides a cost effective means for rapid prototyping of multi-vehicle mission planning, control, vehicle hardware and health-adaptive systems. This intermediate step between simulation and real-world operational testing provides further risk reduction of new technology development compared with only simulation validation. Health-based adaptation increases the level of autonomy in system of system operations under system and environmental uncertainty in order to achieve required mission assurance. Health-based adaptation extends to include adaptation to vehicle conditions and capabilities.

Boeing is continuing the development of the VSTL test bed to support experimentation and assessment of autonomous and collaborative system concepts for a wide variety of applications.

A similar approach is discussed in (Dong et al., 2011) regarding the collaboration of helicopter groups: the leading helicopter is controlled remotely but others cling to the leading one in the distance of 15 meters. The proposed solution confirms a validity of our approach. An efficient and flexible framework is adopted for cooperative control law design, on which we build up a software platform consisting of an on-board real-time software system for UAVs and a ground control station (GCS). Pre-flight simulation is important for evaluating the overall behaviours of both on-board software and GCS.

With a built-in UAV model, simulation environment carry out the hardware-in-the-loop simulation (HILS) in which the interactions among multiple UAVs and GCS and precautions under different failure situations are tested. This paper proposes to use the MATLAB/Simulink based simulation environment for the collaboration of several UATs like in the (Niland, 2006).

A wide potential research branch would be a creating of real UAV test beds to ensure flights of one or several UAVs and the continuous receiving of flight data after changes of external conditions (environment). For example the approach described in (Mutter, 2011) allows perform a functional testing of autonomic UAVs using range sensors via virtual integration, enabling efficient simulation of the system and complex environments. It extends (Bean, 2008) by providing mechanisms to systematically test functional safety aspects with respect to reliability and robustness, using modular scenarios. Authors are specifically interested in a modular testing approach, i.e., an approach that provides 'building blocks' for test scenarios with respect to those two aspects as well as their combination into complex test cases. Authors approach provides a dedicated Software-in-the-Loop (SIL) test environment offering modelling support for dynamic environments, sensors/actuators, and classes of test cases, as well as support for module and integration tests.

In the virtual integration and simulation of UAVs, the focus is generally put mainly on the modelling of the aerodynamics. Virtual test beds like (Johnson et al., 2004) or (Garcia, 2008) use simulation engines provided for flight simulators, but provide no facilities to simulate sensors other than gyroscopes, accelerometers, altimeters, GPS, etc.

The construction of simulation models of sensors like the ultrasonic sensors is generally only found in the field of sensor and/or filter development.

In addition, tests in the analysed sources were not delivered from any formal procedures or functional descriptions of UAVs. Even more: there are no indications about the formal model that would be used to describe the ARS (whether as a finite state machine or some other kind of diagrams). The papers (Johnson et al., 2004) and (Garcia, 2008) describe virtual test beds that contain a SIL model (in fact controlling software in an executable form). The SIL model allows vary indications of sensors (also to create errors) and to check if the real reactions of UAVs correspond to the foreseen reactions in missions. There could not be found any papers with descriptions about creating of full test set from the formalized descriptions of collaborative ARS.

Although this paper is primary devoted to modelling, simulating and testing of collaboration between RUAV and UGV there should be remarked that simulating of process has been studied for a long time. For example, Webots is a three dimensional mobile robot simulator (Hohl et al., 2006), that provides a rapid prototyping environment for modelling, programming and simulating mobile robots. With Webots the user can design complex robotic setups, with one or several, similar or different robots, in a shared environment. Robots can have different locomotion schemes: wheeled robots, legged robots or flying robots. They may be equipped with a number of sensor and actuator devices, such as distance sensors, servos, touch sensors, cameras, emitters, receivers etc.

The simulations by OpenSim are used for motion analysis of sport or daily living activities and diagnosis (Delp et al., 2007) patient specific musculoskeletal models are widely used by medical doctors and physiotherapists (Lim et al., 2003).

An extra approach for the investigation of robots is offered by (Dorigo, 2012), treating robot swarms as multi-agent systems where the collaboration of robots is differs from the traditional one. Swarm robotics systems are characterised by decentralised control, limited communication between robots, use of local information and emergence of global behaviour. Such systems have shown their potential for flexibility and robustness. However, existing swarm robotics systems are by in large still limited to displaying simple proof-of-concept behaviours under laboratory conditions.

This paper is based on the approach used in the ARTEMIS, R3-COP project, therefore the collaboration between RUAV and UGV is described for concrete implementation for the concrete hardware and the approach of Dorigo is not analysed.

3. RUAV and UGV collaboration model

3.1. Software-in-the-Loop (SIL) testing

This chapter demonstrates the offered approach based on one concrete example of RUAV and UGV collaboration. The MATLAB/Simulink simulation environment ensures the simulation process in ticks. Every tick contains a processing of the complete model – beginning with execution of the steps and concluding with the FSM state transition, if it is relevant. This Software-in-the-Loop (SIL) approach differs from the traditional approach in data processing a system that does not take into account the time component.

In the SIL testing, for the System under Test (SUT) is used real software. Usually, this is a C-code. Plant also can be transformed in C. For a platform can be used

Windows, Linux, Real-Time Linux. The strength of SIL is when it is used in combination with Hardware-in-the-Loop (HIL), then, for example, Plant model can be reused. This cannot be considered true. Real Time simulation as data are not coming from real I/O as in HIL, but it is close enough to test SUT with close to real environments using real code.

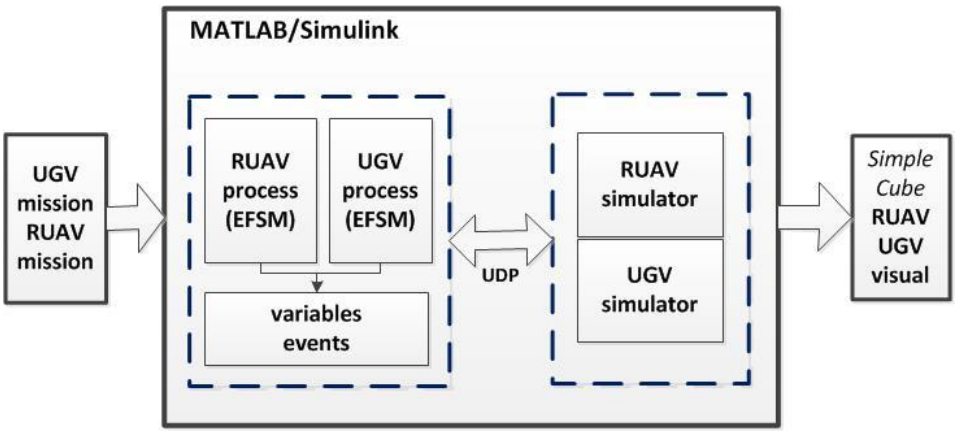

Figure 1. Collaboration model of RUAV and UGV

Toolset for SIL cardinally differs from HIL methods. There our SUT is an executable code (as a rule in C) and we execute it. There is no need for an expensive real time OS. For example, one of the methods is to use MATLAB/Simulink for creating model and RT-Workshop (WEB (2)) to generate the C code for desktop or PC platform with Windows or Linux.

3.2. Autonomous systems modelling language

For description of processes authors proposes to use the EFSM (Extended Finite State Machine) features that are available in the MATLAB/Simulink/Stateflow environment. The collaboration between processes is implemented by events and using common variables in programs. The collaboration model for ARS is executable in the MATLAB/Simulink environment, and it consists of the following components (see **Figure 2.**):

- Control Software model consisting of ARS process diagrams.
- Hardware simulator executing two independent simulations for all ARS. Hardware simulators are an independent part of the model.
- Module for visualization purposes Simple/Cube. It works as an autonomous application and shows movements of ARS on the screen. The visualization's component lets to monitor movements of ARS objects in the conditional space and timeframe.

3.3. The collaboration model of RUAV and UGV

The collaboration model of RUAV and UGV includes processes:

- the mission of RUAV (hereinafter - process A)
- the mission of UGV (hereinafter - process B)

The missions are described by the scenario consisting of executable statements. The following section contains a simplified model for description of collaboration between RUAV and UGV. The simplification was introduced due to the following considerations:

- The collaboration model in the continous evolution as the common model is developed in accordance with the available hardware technologies and features of simulators.
- The simplified description allows to understand the core idea of the solution; the general description of the algorithm is given in the next chapter of this paper.

RUAV has the following statements:

1. Start_RUAV – the RUAV checks the own readiness to start the mission, then it turns the engine on and rises from the coordinates (x0,y0) at a certain height.
2. Up_RUAV(z) – the RUAV rises at the height z.
3. Down_RUAV(z) – the RUAV lands at the height z.
4. Go_RUAV(x,y,z) – the RUAV moves to coordinates (x,y,z). Note: the reachability of the coordinates is not checked in this case.
5. Foto_RUAV – the RUAV stops and „takes a picture" of the object, it means it saves into memory the picture from the video camera attached at the bottom of the RUAV.
6. Stop_RUAV – the RUAV lands to the (x0,y0) and turns the engine off.

UGV has the following statements:

1. Start_ UGV – the UGV checks the own readiness to start the mission, then it turns the engine on and starts the movement from the actual coordinate.
2. Go_UGV(x,y) – the UGV moves to the coordinate (x,y).
3. Stop_UGV – the UGV moves to the coordinates (x0,y0), stops the movement and turns the engine off.

In the **Figure 2** the mission of UGV is represented by a solid line. The UGV moves along a closed rectangular path with the coordinates ((0,0), (3,1), (3,3), (1,3), (0,0)). Statements of the UGV mission are as follows:

Start_UGV
Go_UGV(3,1)
Go_UGV(3,3)
Go_UGV(1,3)
Go_UGV(0,0)
Stop_UGV

In the **Figure 2.** the mission of RUAV is represented by a dashed/ pink line. The RUAV moves in the height z=1 along a closed path pentagon with the coordinates ((1,1), (4,2), (4,3), (3,4), (2,4),(1,1)). The "taking of pictures" is performed in a certain coordinates ((2.5,1.5),(4,2),(2.5,4)).

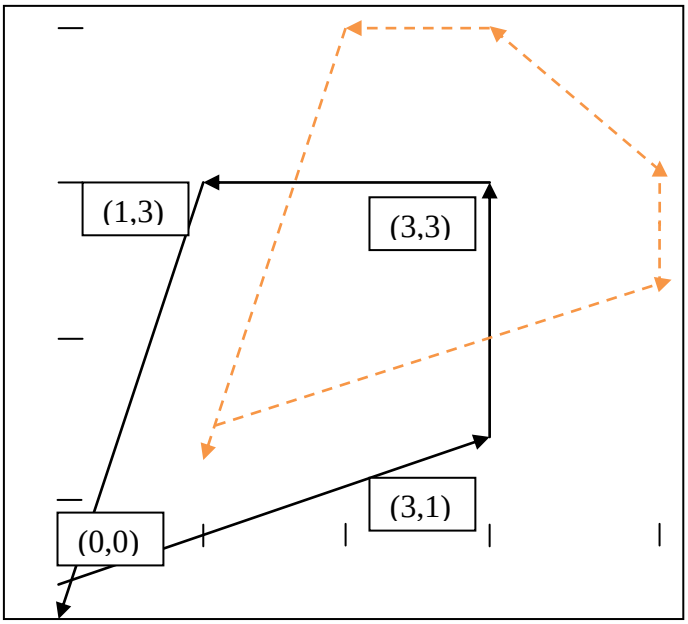

Figure 2. Trajectory of RUAV and UGV movements

Statements of the RUAV mission are as follows:

```
Start_RUAV
Up_RUAV(1)
Go_RUAV(2.5,1.5)
Foto_RUAV
Go_RUAV(4,2)
Foto_RUAV
Go_RUAV(4,3)
Go_RUAV(2.5,4)
Foto_RUAV
Go_RUAV(2,4)
Go_RUAV(1,1)
Down_RUAV(0)
Stop_RUAV
```

3.4. Execution of RUAV and UGV processes

The features MATLAB/Simulink are used to provide collaboration between UGV and RUAV. The collaboration between UGV and RUAV, hereinafter software model, is described in UGV and RUAV processes using events and common variables. The software model sends and receives information to/from hardware simulators through variables. Values are assigned to the variables using UDP links.

The MATLAB/Simulink executes the model in so called ticks. Every tick is executed according to the EFSM process description. Statements in C and process state transitions are executed according to the MATLAB/Simulink semantics of process execution.

An execution of a tick can result with transition to the next state (one step) or staying in the current state. During the execution of a tick processes send information to other components of the model including to the hardware simulators. The simulators perform activities similar to that would be performed by real time hardware and returns new values of common variables. The loop can be run for longer time period to simulate continuous actions of RUAV and UGV.

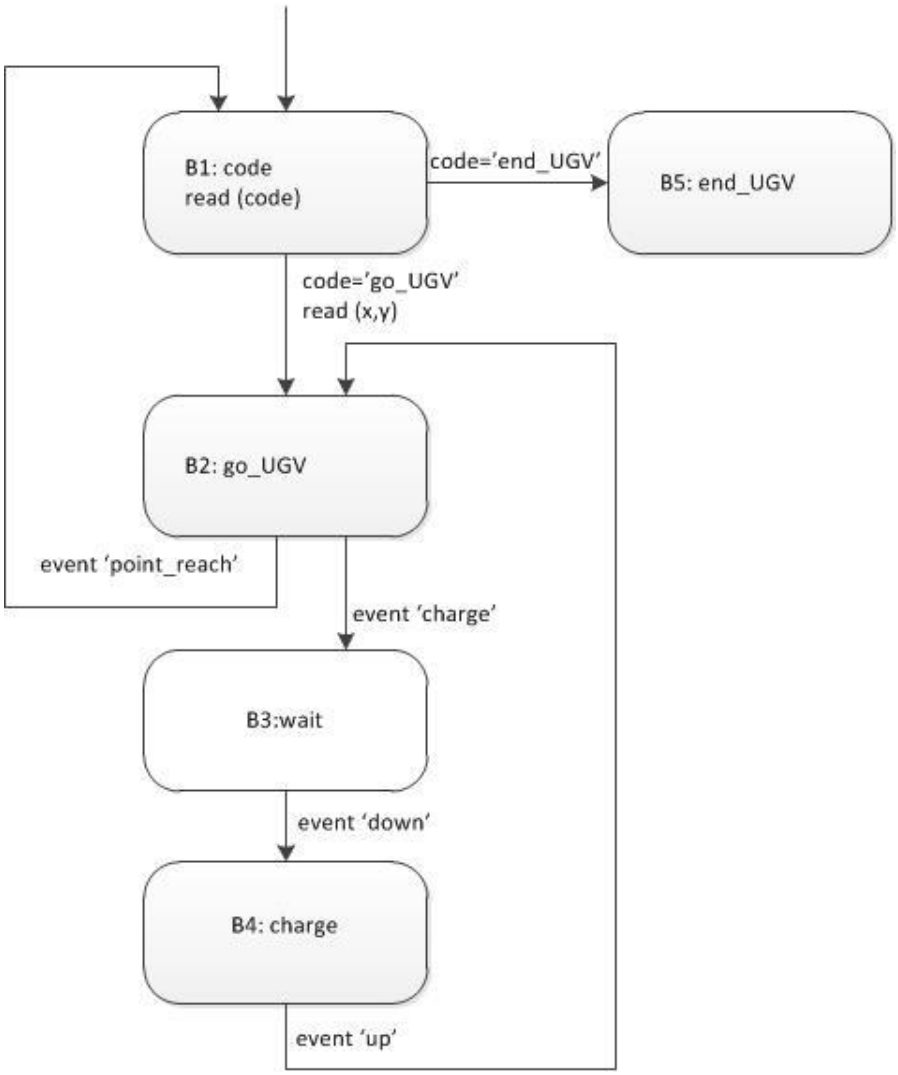

Figure 3. Stateflow diagram of UGV activities

Let's look at an example of UGV functioning process shown in the **Figure 3.** The MATLAB/Simulink starts the UGV mission with the first tick, and it means the transition in the UGV process form the initial state to B1. The UGV is prepared for functioning in the state B1.

In the next tick the code of the second statement Go_UGV (3,1) is read and the UGV process transits from the state B1 to the state B2. The transition includes also the

assigning of values to variables x=3 and y=1. These values will be assigned to the hardware module that exercises the movement of UGV to the coordinates (3,1). This operation can be repeated an arbitrary number of times. The UGV process will not change the state B2 until the event „point_reach" is received and the process transits to the state B1.

In the following tick the UGV reads the next statement of the mission (Go_UGV(3,3), obtains the values of x and y and repeats the procedure described above providing the transition of the UGV to the coordinates (3,3). The number of ticks necessary for moving of the UGV to the next point in the mission depends on the features of the hardware module – the speed of UGV, distance and other obstacles.

The RUAV transits to the state A4 „go_UGV" after receiving of the event „charge". This event is generated by the software of RUAV simulators in the result of analysis of the remaining amount of energy, the distance to the UGV and other parameters. If the UGV process being in the state B2 receives the event „charge" it transits to the state B3 („wait") and stays within it until it receives the event „down". It means the RUAV has landed on the UGV platform and is ready to start recharging. The recharging takes place in the state B4. After receiving of the event "up" the UGV process returns to the state B2 and the movement to the point of mission is continued.

The UGV process ends after fulfilment of all mission's statements by receiving of the statement "End_UGV". This statement turns all devices off, so the UGV is not able to send or to receive any statements. The RUAV process is similar to the UGV process; it is shown in **Figure 4.** The process contains addition states A3, A4, A5, A6 and A7.

The RUAV reaches the state A5 if it has found the UGV and is landing on it. The landing process includes the engine power control, the image recognition and the connection to charging devices. These activities are separate process and they are not further analysed in this example. When the RUAV has been landed on the UGV and is ready to start the recharging the event "down" is generated by the RUAV.

In the next tick the RUAV process is transmitted to the state A6:RUAV_charge and the UGV process to the state B4:charge. When the recharging is finished the UGV simulator generates the event: charged, and it lets the RUAV process to transit to the state A7: RUAV_up(z). It causes the RUAV raises to the height z. The RUAV simulator ensures the engine power adjustments within arbitrary number of ticks.

Once the height z is achieved the RUAV simulator generates the event: up. It transits the RUAV process from the state A7: RUAV_up to the state A2: go_RUAV and the UGV process from the state B4: charge to the state B2: go. The functioning of the RUAV and the UGV in these states is already described above. In addition the RUAV process has the state A3: foto which ensures the capturing of images by video camera to collect pictures which could be analysed by interested parties after the completion of the mission.

This model described above is feasible in MATLAB/Simulink environment. However, the event implementation, which is used to describe RUAV and UGV cooperation, must be specified. It can be supposed that each event is coded in a separate variable, which takes the value 0, if an event has not enrolled, and takes the value 1, if an event has enrolled. In this case, at process arrows that represent transitions between states is not specified event names, but the condition, for example, charge = 1. RUAV and UGV process models, which events are pictured as conditions are very similar to process diagrams given before.

Thus thanks to SW model cooperation of RUAV and UGV in simulation model in MATLAB / Simulink environment can be analysed. It is possible to make sure that the

RUAV and UGV cooperation model is correct, as well as to formulate testing criteria when RUAV and UGV collaborative testing can be considered sufficient.

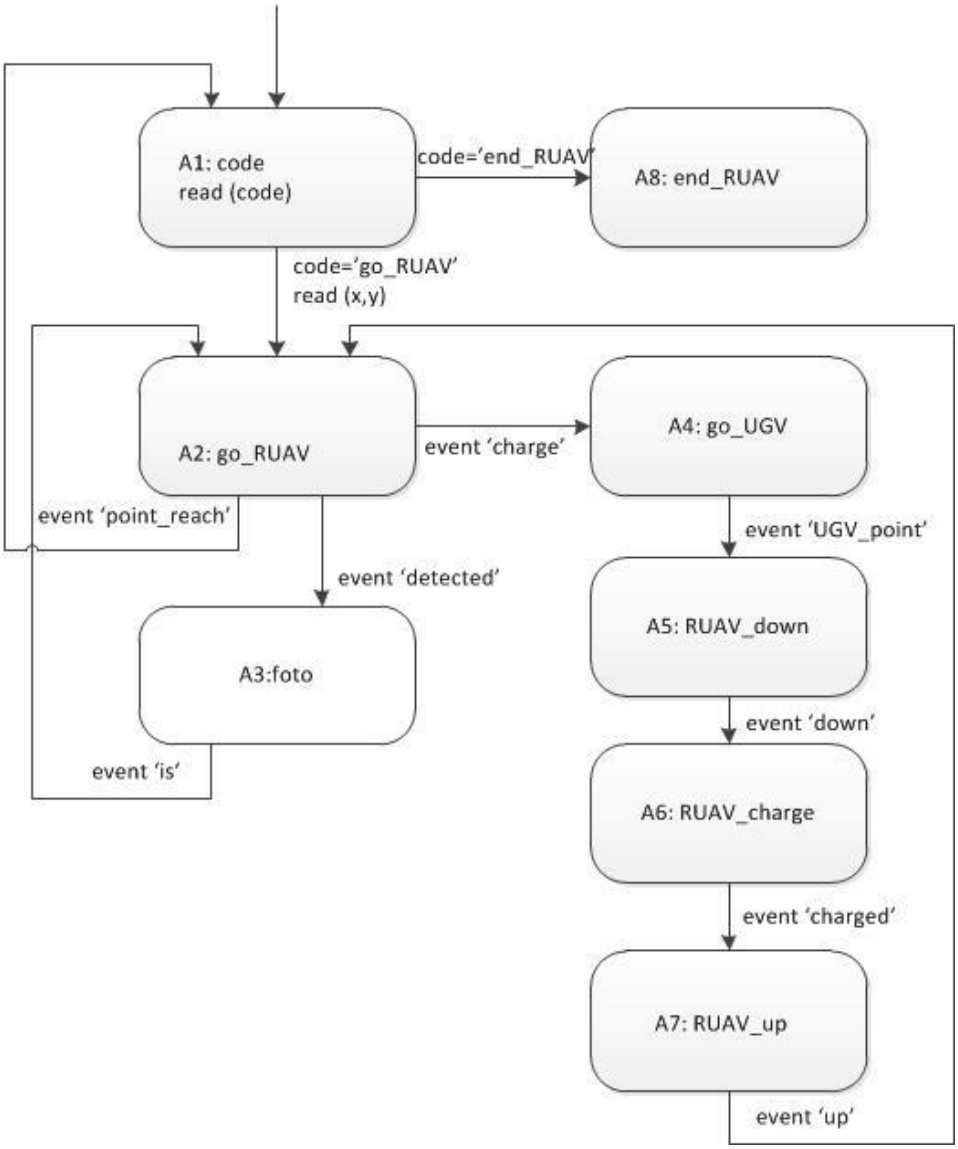

Figure 4. Stateflow diagram of RUAV activities

3.5. Reachability tree construction

The Reachability tree (abbr. RT; see **Figure 5.**) is constructed as follows. RUAV and UGV process state pairs are pictured as top nodes of the tree. They are achieved as process outcomes. Branches describe conditional state transitions and events. The

transition between process state pairs is performed when the condition takes place. Each branch is continued until at the same branch:

- process state pairs repeat
- continuation is not possible (impossible state)
- the end state pair is achieved

The RT of RUAV and UGV cooperation model is as follows. The root is pair (A1;B1). It pictures that processes are led to states A1 and B1 by initiation of process execution at the first tick in MATLAB. Process state pairs are called Model states. If _code="go_RUAV" and UGV_code="go_UGV" the transition to Model state (A2;B2) takes place. Formally it is possible that in the very first step of one or both missions they are discontinued by the command 'end' and it would cause Model states (A1;B5), (A8;B1) and (A8,B5). However a planning of such kind of mission can be considered as 'abnormal'. The similar situations will be described later on.

In case UGV process runs the event: UGV_point_reach the model transits from (A2;B2) into the state (A2;B1). It means UGV has reached the point indicated in a mission and transits into the state B1. The next mission command is given. If the next mission command is UGV_go(x,y) the model transits into the state (A2;B2) which repeats on this branch. Thereby this RT branch is completed. If the next mission command is UGV_end the model transits into the state (A2;B5). If RUAV process in this model state gets event *charge* the model transits into the state (A4;B5). The semantic meaning of this state is „to get RUAV charged when UGV is powered off/its action has completed. Apparently the construction of RT has discovered incorrectness cooperating RUAV with UGV.

The RT branch (A2;B2) => (A4;B3) => (A5;B3) => (A6;B4) => (A7;B4) => (A2;B2) will be described in more detail. This branch pictures the part of a mission when UGV having event *charge* transits into a state 'wait' (stops), but RUAV goes to UGV location to charge batteries. Then RUAV lands on UGV, charges batteries, after that rises up to the specified altitude. The missions of RUAV and UGV can be continued. According to the semantic of a process execution described above transitions (A2;B2) => (A4;B2) and (A2;B2) => (A2;B3) cannot be implemented. It is pictured as an elimination of a branch in the RT.

Summarizing the previous algorithm description it should be noted that RT is constructed to execute all possible continuations of each process and to make sure:

- such continuation is possible
- processes are not completed by the command 'end'
- the model has not returned into the state found before

In this case the model state is the pair of UGV and RUAV process states. Apparently a number of such states is finite because a number of every single process state in finite. Thereby a construction of RT always will be completed (at the appropriate resources). The situation would not change using C logic conditions of a programming language instead of an event. Unfortunately, if the usage of the complete range of C language statements and constructions is allowed, there is no possibility to design a finite RT in general case. This is discussed in the last section of this chapter.

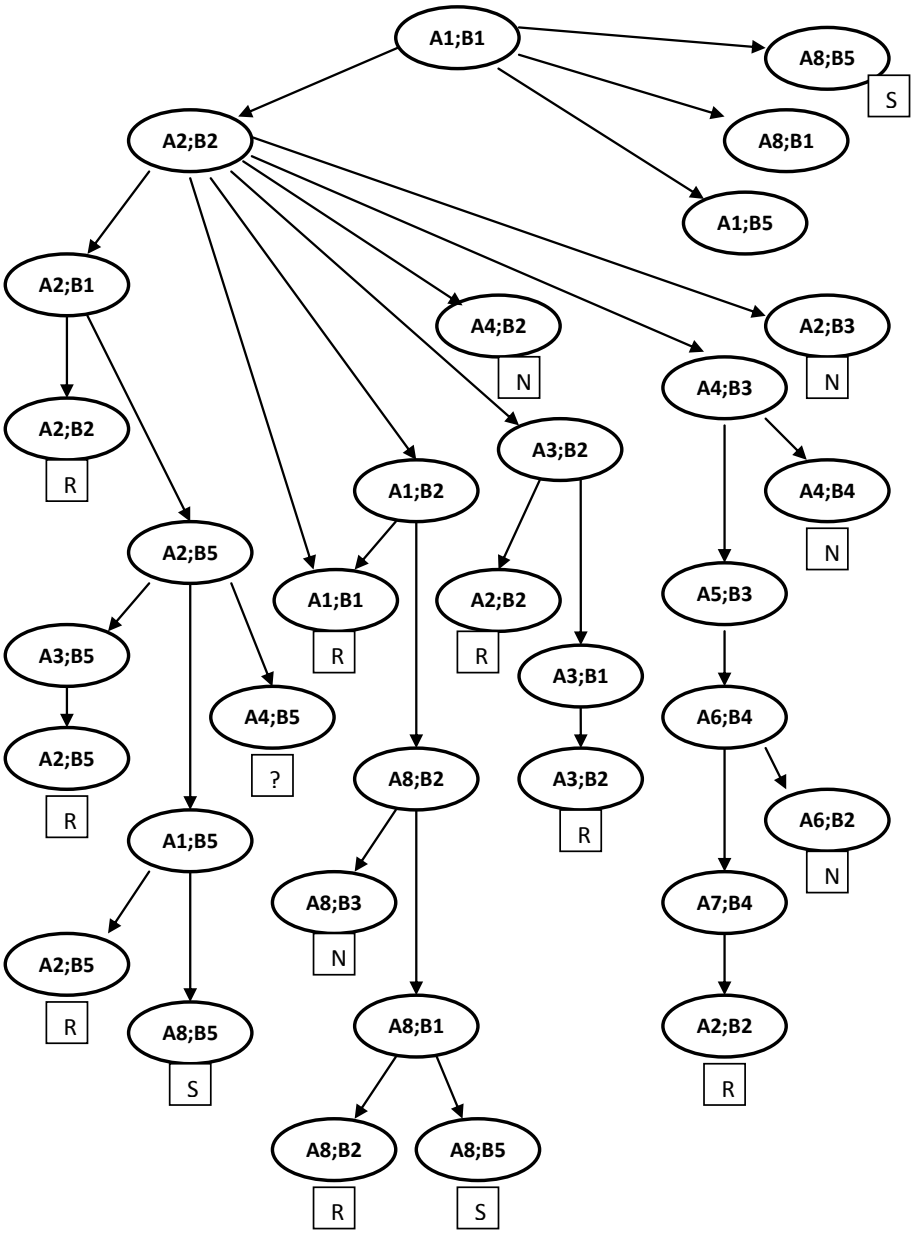

Figure 5. Reachability tree

3.6. Test coverage

The main feature of RT which is used for an examination of a model is as follows. Each model state which can be reached in a mission receiving some event combination is represented in RT. In other words – RT contains all existing conditions of a model.

There is an algorithm which finds steps of RUAV and UGV process execution leading a model to this state.

It is obvious that the execution of the set of all executable missions with all achievable states of the model should serve as the criterion of the testing completeness. It means all available states in the model should be reached that are represented in the RT.

In the simplified example the states of the model are represented by the pairs of process states of RUAV and UGV. The complete test set will contain the missions that achieve all achievable pairs of states in both processes. If the RT exists the designing of the test set is not a complicated task.

3.7. Test design

In the example below three scenarios of the model execution contain all of model states to be achieved.

1. (A1;B1) => (A2;B2) => (A1;B2) => (A2;B2) => (A1;B2) => (A2;B2) => (A3;B2) => (A2;B2) => (A1;B2) => (A2;B2) => (A3;B2) => (A3;B1) => (A3;B2) => (A2;B2) => (A1;B2) => (A8;B2) => (A8;B1) => (A8;B2) => (A8;B1) => (A8;B5)

   Test 1:

UGV:	Start_UGV	
	Go_UGV(3,1)	(event:point_reach)
	Go_UGV(3,3)	(event:point_reach)
	Go_UGV(1,3)	(event:point_reach)
	Go_UGV(0,0)	(event:point_reach)
	Stop_UGV	
RUAV:	Start_RUAV	
	Up_RUAV(1)	
	Go_RUAV(2,1)	(event:point_reach)
	Go_RUAV(3,2)	(event:point_reach)
	Go_RUAV(4,3)	(event:is_detected)
	Foto_RUAV	(event:is)
	Go_RUAV(2,3)	(event:is_detected)
	Foto_RUAV	(event:is)
	Stop_RUAV	

2. (A1;B1) => (A2;B2) => (A4;B3) => (A5;B3) => (A6;B4) => (A7;B4) => (A2;B2) => (A4;B3) => (A5;B3) => (A6;B4) => (A7;B4) => (A2;B2) => (A1;B1) => (A8;B5)

   Test 2:

UGV:	Start_UGV	
	Go_UGV(3,1)	(event:charge; event down; event:up)
	Stop_UGV	
RUAV:	Start_RUAV	
	Up_RUAV(1)	
	Go_RUAV(2,1)(event:charge; event UGV_point; event:down event:charged; event:up; event:point reach)	
	Stop_RUAV	

3. (A1;B1) => (A2;B2) => (A2;B1) => (A2;B2) => (A2;B1) => (A2;B5) => (A3;B5) => (A2;B5) => (A1;B5) => (A2;B5) => (A4;B5) =>???
   Test 3:

UGV:	Start_UGV	
	Go_UGV(3,1)	(event:point_reach)
	Go_UGV(3,3)	(event:point_reach)
	Stop_UGV	
RUAV:	Start_RUAV	
	Up_RUAV(1)	
	Go_RUAV(2,1)	(event:is_detected)
	Foto_RUAV	(event:is; event:point_reach)
	Go_RUAV(3,2)	(event:charge)
	Stop_RUAV	

Prepare three RUAV and UGV missions that will implement the above scenarios, it is possible to execute scenarios repeatedly and prepare state transition conditions.

3.8. Test evaluation

Considered achievable set of model states can be seen in the Table. The symbol "Y" in the table's cell refers to pairs of the states of RUAV (A1-A8) and UGV (B1-B4), which can be reached during the tree missions constructed and described in the previous section.

The symbol "N" indicates the pairs of states, which cannot be reached by any mission. The symbol "Y-?!" refers to the pairs of the states which can be reached but their semantic requires returning to the requirement analysis and decision. One example of such pairs is switching off of the UGV in the moment while the RUAV is moving to the UGV to recharge. This is a typical problem should be solved during the validation, and it's solution will be proposed in the further steps of the model designing.

	B1	B2	B3	B4	B5
A1	Y	Y	N	N	Y
A2	Y	Y	N	N	Y
A3	Y	Y	N	N	Y
A4	N	N	Y	N	Y-?!
A5	N	N	Y	N	N
A6	N	N	N	Y	N
A7	N	N	N	Y	N
A8	Y	Y	N	N	Y

In general case such a simple model state concept cannot be used in RT establishment. The problem is that the C variable values of programming language can affect the execution of the individual processes. Though the RT-construction can be processed of the C-programs are simple and values of variables are not used in conditional statements.

4. Algorithm generalizations

The collaboration model of RUAV and UGV in the previous sections served as an example to demonstrate the main ideas for testing of collaboration models of ARS. This chapter contains the generalization of the proposed approach and analysis of the possibilities to apply them practically.

As a modelling language will be used EFSM as it includes state flows in their traditional notion and possibilities of the programming language C. The MATLAB/Simulink environment will be used to create models for collaboration of ARS as it supports graphical editing and in-the-loop execution of the models according to the semantics prescribed by the MATLAB/Simulink environment. The model's complete test set (MCTS) is the set with test cases covering all ARS states that can be reached.

The theoretical researches show that chosen modelling language offers the same means of expression as a Turing machine. It means there is no possibility to generate a MCTS for every model described in this modelling language (Auziņš et al., 1991). Therefore the following applications of the proposed approach can be identified to solve the problem of the complete testing:

- The model is limited to FSM processes; programs are not used.
- The model allows very limited means of programming.
- The usage of programming means is not limited but the model is knowingly constructed to ensure the creation of the finite RT.

4.1. Collaboration models for FSM processes

If functioning of every autonomous robot can be described as a FSM (finite state machine) not using programming means, the constructing of the MCTS can be replaced by constructing of multiplication of the separate FSMs. Of course the construction of RTs, described in the previous chapter, is significantly more effective for obtaining of MCTS than the construction of multiplications! But there should be mentioned that the programming means used in the previous chapter are trivial, and they allow obtain the MCTS for the collaboration model among RUAV and UGV in an effective way.

The related researches about model checking use to describe the functioning of several processes as finite state machines and to represent the collaboration of processes by mutual messages between the processes. The problem regarding the creating of multiplications of machine states is outlined in papers as well as the resulting "explosion" of number of states. It makes the method practically inapplicable for creating of MCTS. The algorithm for RT constructing described above may significantly simplify the creation of MCTS as it analysis the feasibility conditions and includes only feasible activities/steps. It lets to solve (at least partially) the problem with "the exploding number of states" during the testing of ARS.

The simplified model described above uses only limited features of the language C. It lets to represent the status of the model by states' pairs of separate processes. Therefore there is possible to design a finite RT, and in this case it is possible to create a complete test set for the model.

4.2. Collaboration models for EFSM processes with limited programming means

Researches about creating of complete test sets (CTS) for programs (Auziņš et al., 1991) prove that the problem to create the CTS is solvable for every program if only a limited set of statement's type is used. This set of statements is called base statement set, and it may contain following statements: reading of data into program's variables from unlimited long one-direction string files, allocating of values to variables, comparing of variable values (including comparing with constants) and writing of data into one-direction string files. The algorithm is proposed to create CTS for every program that contains only statements from the base statement set. The proposed system with base statements is applicable for the cases if the data is stored in one-direction string files (f.i. on magnetic tapes).

The continuous researches formulate several extensions of base statement sets as well as conditions for solving of the CTS building problems. The usage of several one-direction counters and of one two-directions counter as well as the reading from direct-access devices and other enhancements are introduced. Additionally the theorems are proven showing the cases when the problem of CTS construction is not algorithmically solvable. Some examples of such cases are repeating reading of one-direction string files in one program even from two files (statement OPEN) and usage of two two-directions counters in one program. The practical implementation of CTS building for data processing systems (Bičevskis, 1979) has shown the possibility to apply the proposed algorithm for analysis of programs and creating of CTS.

Algorithms proposed for creating of CTS in the above mentioned researches can be also used to create MCTS for models describing the collaboration of autonomous robots and their components. The algorithms are based on the creating of RT and on the concept of the model's state. By symbolic execution the RT for feasible execution scenarios of the model is constructed, every path is continued until repeating states are reached. If the number of model's states is finite, the created RT is also finite, and it can be used to build the MCTS. An efficient concept of the state allows describe the functioning of the model using a small number of states. It assures that the algorithms are applicable in real projects. A very short description of the algorithm for creating of MCTS is given.

First of all the notion of path execution conditions (PEC) should be introduced. PEC is designed by symbolic execution of the model's path. After every step symbolic values of the variables according to the semantic of statements in language C are stored. PEC conditions are added to conditional statements using variables and logical expressions allowed in C.

For the above discussed example from chapter 3 the actual statement in the model program path would be:

- READ(x,y) – symbolic values x(i) and y(i) are assigned to the variables x and y, where i is the serial number of the READ statement in the mission
- x:=y – if value of y is y(i), the value y(i) is assigned to the variable x
- x<y – if the value of x is x(i) and the value of y is y(j), the inequality x(i)<y(j) is added to the PEC

Statement 1. The path in the model is feasible if and only if there is a solution for path execution conditions. The solution determines the test case for execution of this path in model.

The model's state is built after the execution of the appropriate path. It can be obtained from the eliminating all information that cannot be used in the succeeding steps of execution. For example values of variables not stored in memory can be excluded from the PEC.

Statement 2. If an execution of two paths in the model leads to the same model's state, the sets of continuing paths coincided.

The theoretical researches mentioned before offer concepts of model states ensuring a finite count of states in any model. Using the concept of the state the following algorithm for the creating of RT is proposed:

- execution of the model is represented as a tree where every branch shows exactly one feasible path; it can be done by designing of PEC and checking it's consistency
- The model's state after passing of branches is indicated on the vertices of the tree
- The branches of the tree are continued until vertices of the identical branches contain identical model states

If the model has a finite count of states, the RT can be constructed and it lets to choose from RT the set of executable paths containing all possible transitions in the model, thereby the according MCTS is created.

Nevertheless the handling is more complicated if complex C programmes are used which influence the model's behaviour in the next steps. For these cases is the notion of the state for other goals introduced. The model's state contains all information characterising the further behaviour of the model in the concrete time frame. In the example given above the model's state was characterised by a pair of the both processes' state. If C programmes would enable activities influencing transitions between different states in the model the state's notion should also include all values of variables and the PECs affecting the further behaviour of the model.

4.3. Collaboration models for EFSM processes with unlimited programming means

If the chosen state concept does not assure a limited count of the states, the proposed construction would never be finished. Theoretical research has shown that the usage of all accessible means of the C programming language do not allow to construct a finite RT for the proposed state concept. It means the proposed solution is only partly applicable for the analysis of autonomous systems.

In some cases a partly constructed RT (incomplete due to time or states number restrictions) can also be analysed to find defects in the model. Of course the usage of incomplete RTs do not guarantee the discovering of all possible defects. There is another approach possible to create MCTS. It is based on the wise choice of the ARS model, i.e., only those means are used during the construction of the ARS model that allows build a RT.

Choosing the set of missions that let to achieve the existing states of the model gives the possibility to create a complete test set for modelled processes. Since the model determines the process state achievable in the result of the path's execution the result of

the test's execution is also known, and it is the state where the path ends. It ensures the compliance of test execution's results with the model can be automatically checked.

Conclusions

A new approach/ method for validation and verification of autonomous processes' collaboration is proposed in the paper. Initially the model of processes collaboration is created using the EFSM modelling means of MATLAB/Simulink. The environment MATLAB/Simulink provides model simulating features, generating of software and transferring of software to hardware. It means that analysis of processes collaboration can be performed without usage of real hardware equipment.

1. The model of processes collaboration is analysed by symbolic execution of model. A finite tree with all feasible paths (a feasibility tree) allowing to reach all feasible states of the model is obtained as the result.
2. The proposed new method differs from the traditional approaches because it ensures to create the ARS collaboration model in the environment that is similar to development environment, f.i., to the programming language C. Therefore the traditional analysis of FSM collaboration can be replaced by the analysis of EFSM processes' collaboration. It offers a wider range of accessible analysis features. The traditional FSM models for analysis of processes collaboration lead to the "explosion of states' number" because new finite states machines are created by multiplication of machines and analysed. The proposed approach substitutes the traditional approach by the symbolic execution of processes, and it reduces the number of alternatives should be analysed.
3. Analysis of the feasibility tree allows discover all reachable states of the collaboration model including the unacceptable (for example stopping of UGV although the charging of RUAV is necessary). If unacceptable states can be reached, it means the collaboration model is designed inaccurately or inappropriately. It could lead to redesign of the model, and thereby the main aim of model checking is achieved. A new method for model checking is proposed to prove a correctness of the model or to construct a scenario leading to unacceptable state of the model.
4. Usage of proposed approach in the ARTEMIS, R3-COP project for describing of collaboration between RUAV and UGV shows that the approach is effective for modelling and analysis of high level processes. The eventual usage of EFSM for description of technical details could be a topic for further research.

Application of the proposed approach showed the method can be used also for testing of the generated software. MCTSs can be generated from the collaboration model to test all possible scenarios to reach all possible states. If the generated software is executed on the MCTS there could be checked if the software functions according to the model. When the tested software is transferred to hardware, it will function identically with the processes described by the model.

References

Auziņš, A., Bārzdiņš, J., Bičevskis, J., Kalniņš, A., Čerāns, K. (1991). Automatic construction of test sets: theoretical approach. *Lecture Notes in Computer Science.* Springer Verlag, Vol. 502, pp. 286-359.

Barzdins, J., Bicevskis, J., Kalnins, A. (1977). Automatic construction of complete sample systems for testing. *IFIP"77, Congress Proceedings.* North Holland.

Bean, T., Beidler, G., Canning, J., Odell, D., Wall, R., O'Rourke, M., Anderson, M., Edwards, D. (2008). Language and Logic to Enable Collaborative Behavior among Multiple Autonomous Underwater Vehicles. *International Journal of Intelligent Control and Systems*, Vol. 13, No. 1, March, pp. 67-80.

Bekey, A. (2005). Autonomos Robots. *Massachusetts Institute of Tehnol.*, ISBN 0-262-02578-7.

Bičevskis, J., Borzovs, J., Straujums, U., Zariņš, A., Miller, E.F.jr. (1979). SMOTL - A System to Construct Samples for Data Processing Program Debugging. *IEEE Transactions on Software Engineering*, Vol. SE-5, No.1, pp. 60-66.

Delp, S.L., Anderson, F.C., Arnold, A.S., Loan, P., Habib, A., John, C.T., Guendelman, E., Thelen, D.G. (2007). OpenSim: Open-source software to create and analyze dynamic simulations of movement. *IEEE Transactions on Biomedical Engineering*, Vol. 55, pp. 1940-1950.

Dong, X., Chen, B. M., Cai, G., Lin, H., Lee, T. H. (2011). Development of a Comprehensive Software System for Implementing Cooperative Control of Multiple Unmanned Aerial Vehicles. *International Journal of Robotics and Automation*, Vol. 26, No. 1.

Dorigo, M. et al. (2012). Swarmanoid: a novel concept for the study of heterogeneous robotic swarms. *IEEE Robotics & Automation Magazine.*

Garcia, R. D. (2008). Designing an Autonomous Helicopter Testbed: From Conception Through Implementation. *Ph.D. dissertation*, University of South Florida.

Hohl, L., Tellez, R., Michel, O., Ijspeert, A.J. (2006). Aibo and Webots: Simulation, wireless remote control and controller transfer. *Robotics and Autonomous Systems*, Vol. 54, pp. 472-485.

Johnson, E. N., Schrage, D. P., Prasad, J., Vachtsevanos, G. J. (2004). UAV Flight Test Programs at Georgia. *Georgia Institute of Technology*, Tech. Rep.

Karpov, J.G. (2010). Model Checking *Sankt-Peterburg*, ISBN 978-5-9775-0404-1.

Lim, C.L., Jones, N.B., Spurgeon, S.K., Scott, J.J.A. (2003). Modelling of knee joint muscles during the swing phase of gait—a forward dynamics approach using MATLAB/Simulink. *Simulation Modelling Practice and Theory*, Vol.11, pp. 91–107.

Mutter, F., Gareis, S., Sch¨atz, B., Bayha, A., Gr¨uneis, F., Kanis, M., Koss, D. (2011). Model-Driven In-the-Loop Validation: Simulation-Based Testing of UAV Software Using Virtual Environments, *Engineering of Computer-Based Systems*, ECBS 2011.

Niland, W. (2006). The Migration of a Collaborative UAV Testbed into the Flames Simulation Environment. *Proceedings of the 2006 Winter Simulation Conference.*

Pounds, P., Mahony, R., Gresham, J., Corke, P., Roberts, J. (2004). Towards Dynamically-Favourable Quad-Rotor Aerial Robots. *Proceedings of the 2004 Australasian Conference on Robotics and Automation*, Canberra, Australia.

Saad, E., Vian, J., Clark, G. J., Bieniawski, S. (2009). Vehicle Swarm Rapid Prototyping Testbed. *The Boeing Company*, Seattle, WA, 98124.

WEB (1) Simulation and Model-Based Design. *www.mathworks.com/products/simulink/*

WEB (2) Generate C and C++ code from Simulink and Stateflow models. *http://www.mathworks.com/products/rtw/*

WEB (3) ARTEMIS, R3-COP. *http://www.artemis-ia.eu/project/index/view%3Fproject%3D14*

Authors' information

Janis Bicevskis, born in 1944, graduated from the University of Latvia in 1970 in mathematics, Candidate of science from 1979, Doctor of computer science from 1992, professor from 2002, published more than 92 scientific papers and took part in development of number of software projects (design, implementation, programming, testing, management) from 1968. He was head of the Department of Computer Science at University of Latvia, one of the authors of national program "Informatics", project leader of Latvia's Education Information system since 1999 and he was leader of project "Integration of National information systems (Megasystem)" at 1998. His main current scientific interests are software engineering and software testing.

Artis Gaujens, born in 1959, received mathematician diploma from University of Latvia, Faculty of Physics and Mathematics in 1982, master degree in computer science from the University of Latvia in 1993 and currently is researcher in the Institute of Mathematics and Computer Science of the University of Latvia. His research interests include software engineering, particularly real time system development and software testing using in-the-loop methods. He has been software developer of different systems such as switching system for rural telephony, real time and simulation systems based on SDL language.

Janis Kalnins, born in 1942, graduated from the University of Latvia in 1965 in mathematics, Candidate of science from 1974, Doctor of computer science from 1992, published more than 60 scientific papers and took part in development of number of software projects (design, programming, testing) from 1967 and currently is senior research associate in the Institute of Mathematics and Computer Science of the University of Latvia. His research interests include software engineering, particularly real time system development.

Received October 3, 2013, revised November 24, 2013, accepted November 25, 2013