ISSN 1064-2307, Journal of Computer and Systems Sciences International, 2015, Vol. 54, No. 6, pp. 853–865. © Pleiades Publishing, Ltd., 2015.
Original Russian Text © I.P. Buzhinsky, S.V. Kazakov, V.I. Ulyantsev, F.N. Tsarev, A.A. Shalyto, 2015, published in Izvestiya Akademii Nauk. Teoriya i Sistemy Upravleniya, 2015, No. 6, pp. 17–30.

DISCRETE SYSTEMS

Modification of the Method of Generation of Control Finite-State Machines with Continuous Actions Based on Training Examples

I. P. Buzhinsky, S. V. Kazakov, V. I. Ulyantsev, F. N. Tsarev, and A. A. Shalyto

ITMO University, St. Petersburg, Russia

e-mail: buzhinsky@rain.ifmo.ru

Received February 4, 2014; in final form, January 26, 2015

Abstract—Control finite-state machines can be used in the development of reliable control systems due to their clarity and because it is possible to formally verify them. The paper deals with resolving the problem of the generation of machines that control plants with complex behavior based on training examples. The input and output actions of the machines are given by real numbers. A method for the generation of machines is proposed. It is a modification of the previously proposed approaches based on the genetic and ant colony optimization algorithms. Changes include a new way of representing machines and improving the fitness function. The method makes it possible to generate machines whose behavior is more consistent with training examples than the behavior of machines generated by the known approaches.

DOI: 10.1134/S1064230715050044

INTRODUCTION

In this paper, we consider the problem of generation of control finite-state machines (hereinafter, machines) based on a set of training examples that describe the behavior of a plant with complex behavior; i.e., a plant capable of producing output actions based not only on the current input actions but also on their history. An unmanned airplane model is selected as an example of such a plant.

Machines are widely used in the development of reactive systems [1–3] and can be formally verified by the *Model Checking* method, which makes it possible to use them as components of reliable software. In addition, the machines can serve as models of existing software systems. For example, in [4, 5] methods for obtaining such machines by program tracing are described.

It is often difficult or even impossible to construct machines manually. For example, for the Artificial Ant problem an optimal machine was found only with the automated construction of machines [6] using genetic algorithms [7–10]. When using these algorithms, as well as other methods of search optimization (for example, evolutionary strategies), it is necessary to define the quality criterion of the optimization problem solution, i.e., the fitness function (FF). There are two approaches to set up FF of machines. The first is based on the comparison of the ideal model of the behavior of a plant (for example, a predetermined route for the airplane model) with the behavior controlled by the machine [11]. The second approach uses prerecorded examples of the desired behavior of the machine [12] such that the verification of the compliance of the machine with these examples does not require simulation of the plant.

In [11] the genetic algorithm was used to solve the problem of generation of a machine controlling an airplane model. Fitness functions were calculated based on the behavior of the model in the flight simulator, which resulted in a large computational cost. In [13–15] a fundamentally different FF based on training examples was used to improve the performance. A core element of the approach proposed in [13] was the automatic arrangement of output actions on machine transitions. This approach can be applied not only to control the airplane model but also other plants, including those with a complex or unknown behavior model. This approach is evolved in this paper. In it, it is proposed to maintain the scheme of the approach but to change the way machines are represented. It should lead to a higher correspondence of the behavior of the generated machines to training examples without the overfitting problem (a situation in which proper machine behavior cannot be generalized to unknown cases during optimization on training examples) and make machines clearer. The ant colony optimization algorithm previously used in [14, 15] was applied for the generation of machines.

The formal description of the problem is given in Section 1 of this paper. Section 2 presents detailed proposed improvements of the method of the generation of machines. Their feasibility is demonstrated by the results of the experiments described in Section 3.

1. PROBLEM STATEMENT

To begin with, let us give an informal definition of the control finite-state machine used in this study, which will be further refined. The machine is a control system that works in cycles. In this paper, we consider synchronous machines, i.e., machines with cycles at regular intervals (0.1 s). The machine has a finite set of states S and the initial state $s_0 \in S$. At any time, the machine is in one of these states. In each cycle, based on the input actions, the machine makes a transition (changes its state) and generates an output action. Transitions are defined by the transition function δ and the output actions are determined by the output function λ.

In this study, the machine's input actions are defined by the state of the plant and are sets of real numbers that describe the plant's sensor readings. Output actions are also represented as sets of real numbers and are changes in the plant's control parameters. In the next section, a mechanism for obtaining input and generating output actions will be described in detail. The continuity of the input and output actions (setting them by real numbers) leads to nontrivial implementations of the transition and output functions compared to the classic machines that use only discrete actions. The proposed implementation of these features will be described in Section 2.1.

Let a finite set of training examples be given. They describe some of the tasks to be performed by the plant. For the airplane model we will use certain aerobatic maneuvers as these tasks. Each training example describes values of the plant's input parameters (for an airplane these are flight parameters: speed, altitude, bank angle, etc.) and control parameters at different points in time. The control parameters specify the position of the controls (for an airplane these are the elevator, ailerons, etc.). In order to record training examples and the launch of the airplane controlled by the machine the *FlightGear* flight simulator was used [16]. The considered problem consists in the generation of a machine with a behavior close to that described in the training examples.

1.1. Plant Control

We will call an ordered set of $p \in \mathbb{N}$ real numbers corresponding to the input parameters of the plant the *tuple of input actions*. Let us assume that the plant has $c \in \mathbb{N}$ control parameters corresponding to the *controls*. These controls are of two types, i.e., discrete controls with a finite set of states and continuous controls, the position of which can be specified by the number from a specific segment of the real axis. A starter is an example of discrete control (it has two positions: on and off). The rudder is an example of continuous control (the numbers between -1 and 1 correspond to its positions from the extreme left to the extreme right). For simplicity, we assume that all the controls are continuous, but the described method can be generalized to the case of both types of controls. The *tuple of control parameters*, the set of c real numbers, is the snapshot of the values of all control parameters at a certain moment in time. Let us assume that the jth ($j = \overline{1,c}$) element of the tuple of control parameters is always limited by the closed interval $[m_j, M_j]$, corresponding to the possible values of the control parameter j.

As mentioned above, the machine operates in cycles. At the beginning of each cycle, it takes a tuple of input actions of the plant. Each machine transition is labeled by a protected condition, a Boolean formula of *predicates* that are also Boolean values. A transition is carried out for which the calculation of the corresponding formula using the current predicate values gives the correct answer (there should be only one such transition). In general, the predicate's values depend on all the tuples of the input actions which have been transferred to the machine by the current clock cycle. The statements "the altitude is decreasing" and "the pitch angle is positive" are examples of predicates.

Next, let us introduce the concept of *components of output actions*. These are real values computed similarly to predicates, i.e., based on all input actions transmitted to the machine. Input actions, their powers, and their rates of change, calculated as the difference between the last two values of the corresponding input action, can be components of output actions. The bank angle, pitch angle deviation of 5°, and the vertical velocity square are examples of components. The name of these values is associated with the fact that in the present work they directly determine output actions, whereas in [13–15] the latter depended on the values of the predicates.

The control process diagram is shown in Fig. 1. The machine is shown schematically in the left panel of the figure (in the form of a transition graph). The transition graph vertices correspond to machine states

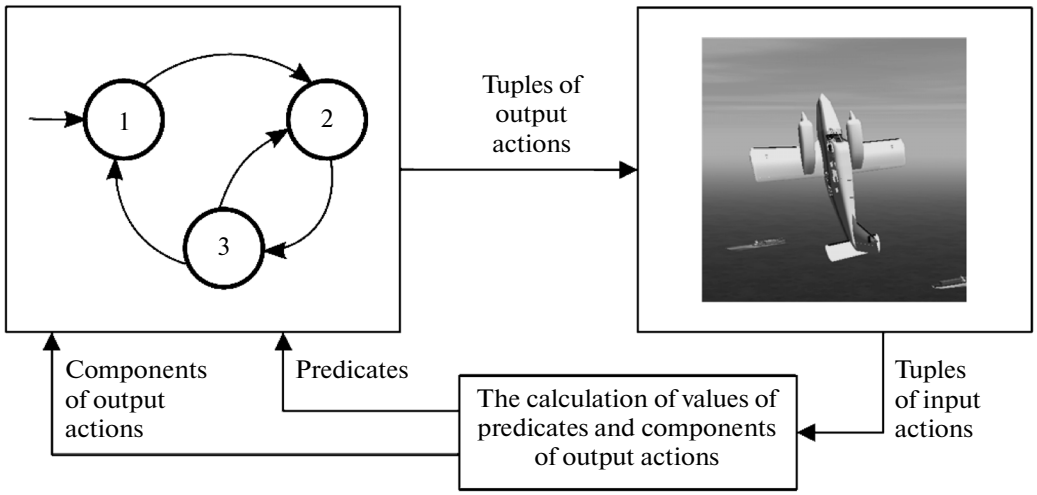

Fig. 1. The interaction of the machine and the airplane model.

and the figures inside the vertices represent their numbers. The arcs of the graph set the machine transitions. The initial state $s_0 = 1$ is denoted by the arc included in it that does not begin in any of the states.

1.2. Training Examples

Let us formalize the concept of training examples. A training example consists of two sequences of tuples of input actions and control parameters, each of which is indexed by time points. Since the machine is synchronous, the difference between neighboring time points equals the interval between cycles. The length of these sequences for the ith training example is L_i ($i = \overline{1, N}$, where N is the number of training examples). Let us call these numbers the lengths of the training examples. The sequence of tuples of input actions I_i consists of numbers $I_{i,t,j}$, where $t = \overline{1, L_i}$ is the tuple number in the sequence, and $j = \overline{1, p}$ is the number of the input action in the tuple. It corresponds to a sequence of tuples of the control parameters O_i that consists of the numbers $O_{i,t,j}$, where $j = \overline{1, c}$ is the number of the control parameter. A section of the training example for the case $p = 4, c = 3, L_i = 235$ is given in Table 1.

2. THE PROPOSED METHOD

This section describes the proposed approach as a modification of the method proposed in [13] and accelerated in [14, 15]. The general scheme of the method from [13] is preserved: the search for machines is carried out using the search optimization algorithm, whose individuals are *frames* of machines, i.e., machines with unknown output functions. The method of representation of machines and two FF modifications are new. One of the modifications is designed to increase the clarity of machines, and the second makes it possible to improve the quality of aerobatic maneuvers that require that the airplane maintained a certain constant value (for example, of the bank angle). The ant colony optimization algorithm [17] pro-

Table 1. Fragment of a training example

Values	Description	$t = 1$	...	$t = 10$	...	$t = 20$	...	$t = 235$
$I_{i,t,1}$	Pitch angle, deg	3.078	...	3.544	...	4.112	...	2.412
$I_{i,t,2}$	Bank angle, deg	−0.076	...	0.351	...	3.413	...	1.759
$I_{i,t,3}$	Heading angle, deg	198.03	...	198.11	...	198.41	...	205.64
$I_{i,t,4}$	Speed, knots	251.42	...	252.29	...	253.20	...	289.40
$O_{i,t,1}$	Position of ailerons (a number from −1 to 1)	0.000	...	0.032	...	0.073	...	−0.003
$O_{i,t,2}$	Position of the rudder (a number from −1 to 1)	0.000	...	0.016	...	0.037	...	−0.001
$O_{i,t,3}$	Position of the elevator (a number from −1 to 1)	−0.035	...	−0.039	...	−0.037	...	−0.011

posed in [18] was used as the search optimization algorithm. It showed a slightly better performance [14, 15] for the problem under consideration than in [13] and requires no crossover operator. In this algorithm, the search space is represented as a directed graph, whose vertices are machines (in this study, their frames) and whose arcs are their mutations (minor changes). Next, let us consider the details of the implementation of the proposed method of machine generation.

2.1. Machine Representation Method

The basic approach [13] to the solution of the considered problem makes it possible to generate machines whose output actions depend only on predicate values. This imposes some undesirable restrictions on the output actions. For example, when using only the predicates the machine cannot produce arbitrarily small outputs (changes in values of the control parameters) that could be useful to stabilize a parameter of the plant, for example, the pitch angle of the airplane. This is required in one of the aerobatic maneuvers discussed in this paper, in the case of the plane turning 180° in the horizontal plane. In addition, the machines given in [13] carried out several transitions per cycle that made the transition function difficult to analyze.

In order to eliminate these shortcomings, we propose another way to represent machines that makes it possible to use both predicates and components of output actions. Let us consider a set of predicates $x_1, \ldots, x_m$. In each machine state only their subset is used—the *values* in this state of the predicate. For each state $s \in S$ and for each set of values of predicates significant in s a transition from this state s and for these predicate values is defined. The mask of the significance of the predicates and the transition table, whose elements are the final states of outgoing transitions from s, are stored in the program representation of such a machine in each of its states s. In [11], this approach was called the method of reduced tables. It makes it possible to reduce the size of the machine description at high n compared with the simpler method of full tables [3] (the use of the latter leads to the information about all the 2^m transitions being stored in each state of the machine). Note that the number of states of the machine $|S|$ and the number of significant predicates are considered fixed during optimization. The number of significant predicates is the same for all states.

Components of output actions are used to generate output actions of the machine. For each control parameter j there is a set of constituting output actions $v_{j,1}, \ldots, v_{j,n_j}$. The reason for using several (possibly overlapping) sets of components is that for each of the control parameters its own components are important. For example, for the control of ailerons the bank angle and its rate of change are important, and for the elevator, the components associated with the pitch angle.

On each cycle the output action of the machine is formed as a linear combination of values of components of output actions with coefficients determined only by the machine state. In this sense, the machine is the Moore automaton [3]. More formally, the value of jth control parameter u_j at each cycle varies by

$$\Delta u_j = \sum_{i=1}^{n_j} r_{s,i,j} v_{j,i}, \tag{2.1}$$

where $r_{s,i,j}$ is the coefficient corresponding to state s of the machine before the transition and component $v_{j,i}$. Linearity (2.1) will make it possible to further automatically determine the coefficients so that machine FF reaches the maximum at the given frame.

Similarly to the predicates, not all the components of output actions are significant, and for each of the control parameters in each machine state an additional mask of the significance of the components of the output actions is stored. Let us assume that numbers $r_{s,i,j}$ corresponding to insignificant components are known to be zero. The remaining numbers will be determined automatically. The use of masks of significance of components of output actions, firstly, simplifies the generation of machines at high $n_1, \ldots, n_c$, since the time of the operation of the algorithm of the arrangement of the output actions, which will be described in Section 2.3, is proportional to the cube of the number of numbers determined by it. The appearance of numbers $r_{s,i,j}$ known to be zero reduces this time. Secondly, masks of the significance of the components of output actions can be used to avoid overfitting, which can occur if the number of optimized parameters is large.

An example of the proposed representation of the machine state is shown in Fig. 2. In the upper left corner of the figure there is a mask of the significance of the predicates (there are four of them). Significant predicates are labeled unity while insignificant are marked zero. To the right there is a table of transitions

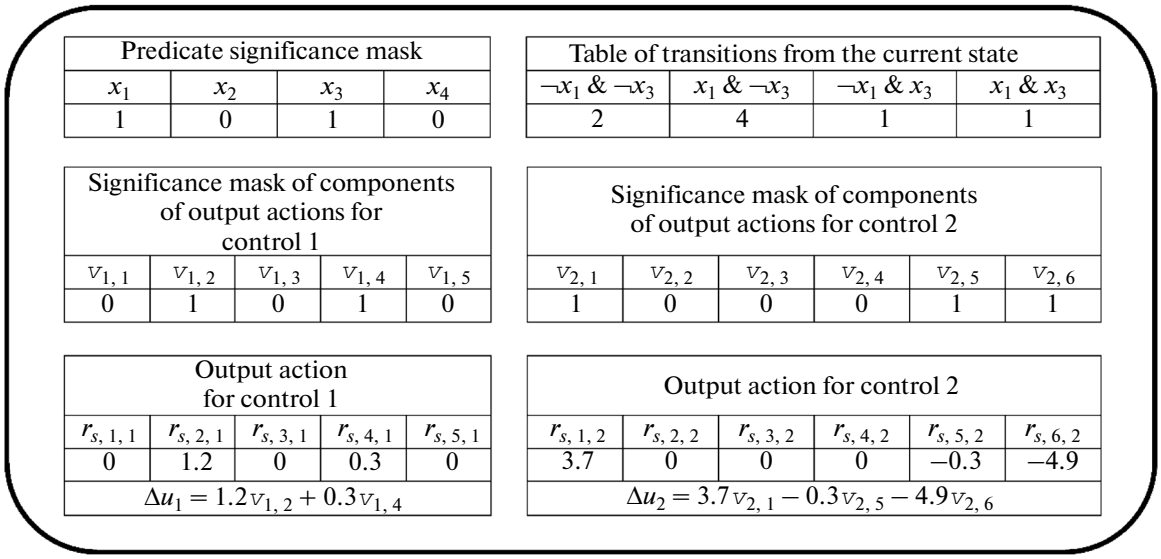

Fig. 2. An example of the program representation of the machine state.

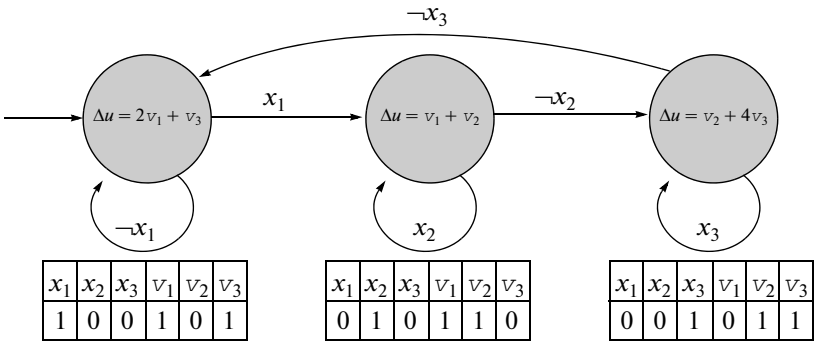

Fig. 3. An example of a machine with three predicates and three components of output actions.

from the current state. Below, masks of the significance of components of output actions, numbers $r_{s,i,j}$, and output actions of the form (2.1) given by these numbers are provided for two controls. Despite the rather complex representation of the machine states, the transition graphs of the latter appear to be relatively simple. An example of such a graph is shown in Fig. 3. In it for the sake of simplicity, the case of a single control is considered. Masks of significance of both predicates and components of output actions are placed under the state, and within states there are output actions of the form (2.1) set by them.

When using the ant colony optimization algorithm [18], it is necessary to set the mutation operator for the machine frame. In this paper we consider the following mutations: the change of the initial state of the machine to a random one, the change of the final state of a transition to a random one, the exchange of two unequal elements of a mask of significance of predicates or components of output actions.

2.2. Fitness Functions

Let $\tilde{O}_i$ be a sequence of tuples of control parameters produced by a fixed machine in response to I_i. We assume that $\tilde{O}_{i,1,j} = O_{i,1,j}$ (the initial output actions of the machine equal to the initial output actions recorded in the training example), but the result of the machine cycle with the number t is the production of actions $\tilde{O}_{i,t+1,j}$. In order to measure the proximity of the behavior of the machine in a specific training

example to the behavior recorded in the same training example, we use the following distance-penalty between the output sequences

$$\rho(\tilde{O}_i, O_i) = \sqrt{\frac{1}{L_i c}\sum_{t=2}^{L_i}\sum_{j=1}^{c}\left(\frac{\tilde{O}_{i,t,j} - O_{i,t,j}}{M_j - m_j}\right)^2}. \quad (2.2)$$

The following FF was used in [14, 15] based on this penalty:

$$\tilde{f} = 1 - \sqrt{\frac{1}{N}\sum_{i=1}^{N}\rho^2(\tilde{O}_i, O_i)}.$$

Let us introduce the first FF modification. Note that the machines which often change their state are less clear and more difficult to analyze. Therefore, it is reasonable to minimize values of $\max(0, \tau_i - |S| + 1)$, where τ_i is the number of machine states when passing the ith training example. We will take into account these values as penalties when calculating the FF:

$$P_\tau = K\sqrt{\frac{1}{N}\sum_{i=1}^{N}(\max(0, \tau_i - |S| + 1))^2};$$

$$f = 1 - \sqrt{\frac{1}{N}\sum_{i=1}^{N}\rho^2(\tilde{O}_i, O_i)} - P_\tau, \quad (2.3)$$

where K is a small number (in this paper, $K = 0.00015$). In earlier works [13–15], the term P_τ was not taken into account. Since in the earlier approach, the machine made several transitions per cycle, the minimization of P_τ would not be sufficient to establish the purpose of individual states of the machine.

Let us consider the second FF modification. Similar to function f we define function f_Δ:

$$f_\Delta = 1 - \sqrt{\frac{1}{N}\sum_{i=1}^{N}\rho^2(\Delta\tilde{O}_i, \Delta O_i)} - P_\tau,$$

where $\Delta\tilde{O}_i$ and ΔO_i are difference sequences of the adjacent elements of sequences $\tilde{O}_i$ and O_i corresponding to them. More formally, $\Delta\tilde{O}_{i,1,j} = 0$ and $\Delta\tilde{O}_{i,t,j} = \tilde{O}_{i,t,j} - \tilde{O}_{i,t-1,j}$ for $t > 1$. The same is true for ΔO_i.

The need to use f_Δ arises when training examples describe ramp changes of the control parameters, the purpose of which is to maintain the parameter of the plant. The authors established experimentally that in this case the optimization of f leads to excessively smooth output actions. Such output actions respond too slowly to changes in the maintained parameter, which degrades the quality of the parameter control. An example of the described case is shown in Fig. 4, which shows the curve that gives the position of the elevator for one of the training examples and the curve showing the behavior of one of the generated machines on this training example. The first curve describes values from the sequence O_i at different times, the second curve shows the values from the sequence $\tilde{O}_i$. These problems are not observed when optimizing f_Δ.

2.3. Determination of Output Actions

In [13–15] output actions were automatically placed on machine transitions. In this paper, this approach is also used, but the placement procedure has its own specificity because of the used method of machine representation. According to (2.1), in order to determine output actions, it is sufficient to set the numbers $r_{s,i,j}$. Recall that numbers $r_{s,i,j}$ corresponding to insignificant components of the output actions are known to be equal to zero. The rest of the numbers can be determined so that on a given machine frame the FF (f or f_Δ) reached the maximum. For this purpose it is necessary to solve a system of linear equations for each control parameter. The construction of this system is described in the Appendix. At the end of the Appendix the optimization is also given that makes it possible to combine the calculations of the systems for different control parameters. The determination of output actions is carried out for each machine

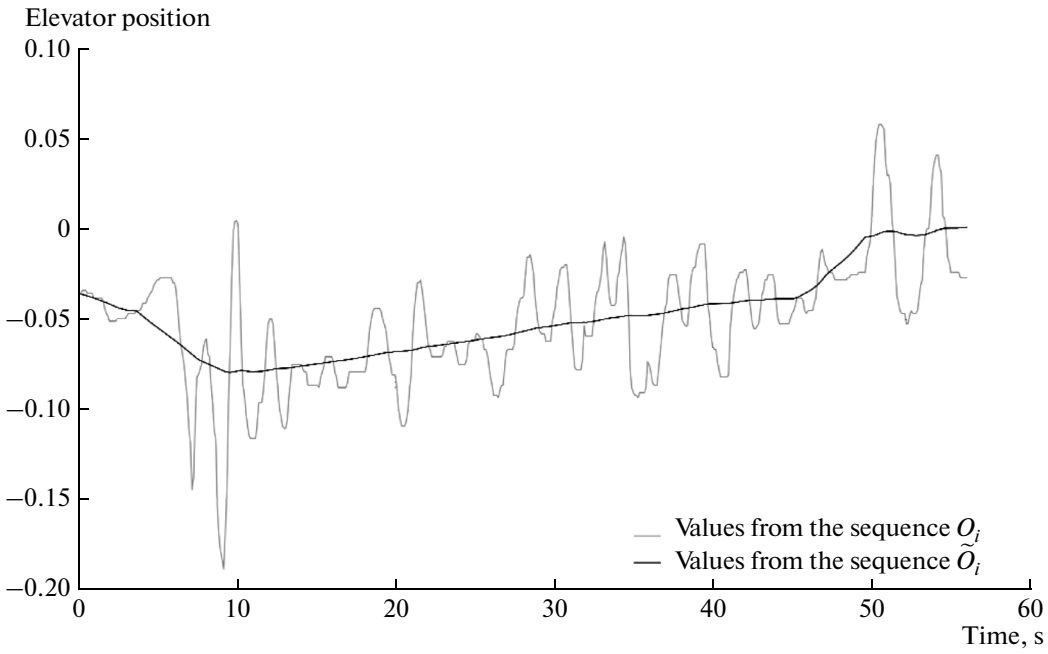

Fig. 4. An example of sawtooth output actions in the training example and output actions of a machine on the same training example.

frame generated in the process of the search optimization and is the most computationally intensive step of the method.

3. EXPERIMENTAL EVALUATION

The experiments included the search for machines with two methods of representation (with the method used in [13–15] and the one proposed in this paper), and the determination of the quality of machines generated in the flight simulator *FlightGear*. We will further call the two mentioned representation methods *previous* and *proposed* methods, respectively. Three sets of training examples that describe the making of loops, rolls, and a half turn in a horizontal plane at an angle of 60° were preliminary manually recorded in the simulator. Some properties of the sets of training examples are given in Table 2. When generating machines for making the loop and roll, f was used as the FF, but when dealing with machines using the previous representation method penalties P_τ were not used ($K = 0$): in the case of the complex structure of the machine cycle the purpose of the introduction of penalties is not achieved. On the third set of training examples that describes the specified turn the effectiveness of the two FFs is compared: f and f_Δ. The function f_Δ was also tested on the first two training sets, but experiments showed that its use on them is not justified.

For each set of training examples a set of predicates for the generation of machines using the previous representation method and two sets (of predicates and components of output actions) for the generation of machines with the proposed representation method were manually selected. The selection of these values is iterative: initially the intuitively selected set is corrected until the behavior of the automatically generated machines becomes acceptable. However, a good set of predicates for the execution of the turn using

Table 2. Properties of sets of training examples

Set of training examples	Airplane model	Number of training examples	Average length of a training example in the set, s
Loop	PA-34 Seneca II	33	35.4
Roll	Gloster Meteor	28	27.2
U-turn	Gloster Meteor	19	56.3

Table 3. Used parameters of the ant colony optimization algorithm

Machine representation method	N_{ants}	N_{mut}	n_{stag}	p_{new}
Proposed	5	13	55	0.1428
Previous	2	29	8	0.2471

Table 4. Median FF values obtained at different instances of the problem

$\|S\|$	Machine representation method	Loop, f	Roll, f	U-turn, f	U-turn, f_Δ
3	Proposed	0.9856	0.9854	0.9892	0.99748
	Previous	0.9812	0.9832	0.9894	0.99731
4	Proposed	0.9866	0.9863	0.9898	0.99749
	Previous	0.9836	0.9856	0.9901	0.99734
5	Proposed	0.9873	0.9868	0.9901	0.99751
	Previous	0.9842	0.9858	0.9902	0.99734

machines with the previous representation method was not selected, which affected the results of the modeling of the behavior of the machines, which will be given in Section 4.2. Note that the manual selection of the predicates is a disadvantage of the basic method, which in this paper is maintained.

3.1. Computational Experiments

For each of the four considered combinations of the set of training examples and FF (the combination of these two parameters will be called a *problem instance*), for the number of states $|S| = 3,4,5$, and for the two representation methods of the machines 50 runs of the ant colony optimization algorithm were carried out [18]. The algorithm proposed in [18] involves the use of a pheromone when the ant selects the following arc of its path. However, in this study a simpler strategy was selected that proved to be no less effective, i.e., equiprobable selection of the next arc.

The number of ants N_{ants}, as well as other nonpheromone algorithm parameters N_{mut}, n_{stag}, and p_{new} (a description of the parameters was given in [18]) were set using the software tool *irace* [19]. The use of the special tool to adjust the parameters of the algorithm is preferable than their manual adjustment, since the latter is influenced by the human factor. Table 3 lists the values of the parameters found by means of *irace*. When setting the parameters, the behavior of different sets of values on three of the four considered problem instances was taken into account. In this case, the instance with f_Δ as the FF was excluded, since it turned out to be the easiest for search optimization, and its use to set the algorithm parameters most likely would reduce the effectiveness of the derived parameters in complex instances.

The algorithm's stop criterion was stagnation during 5000 calculations of the FF. Table 4 shows the median values of the FF obtained using the ant colony optimization algorithm on different instances of the problem. The given values of the FF cannot be compared between different problem instances, but they can be compared between different representations of machines. Note that the penalty for changing machine states in the FF of machines with the proposed representation method reached low values because of optimization and did not have a significant impact on the FF value, which made such a comparison possible. Table 4 shows that the use of the proposed machine representation method increases the FF value in the loop and roll.

In addition, it is reasonable to take into account the time required to search for machines using one or another representation method. Table 5 shows the median numbers of FF calculations for different instances of the problem. On the one hand, in order to generate machines using the previous representation method more computations of the FF are usually required. On the other hand, the arrangement of output actions for machines using the proposed method required up to 60% more CPU time. In any case, the machine generation time did not usually exceed 10 minutes on a personal computer with a quad-core processor *Intel Core i*7–2670*QM* with the parallel computing of the FF of different machines.

Table 5. Median numbers of FF calculations at different problem instances

$\|S\|$	Machine representation method	Loop, f	Roll, f	U-turn, f	U-turn, f_Δ
3	Proposed	8556	8472	8339	7556
	Previous	9979	10071	10195	11400
4	Proposed	9234	8934	9119	9261
	Previous	12565	14883	15256	15456
5	Proposed	9008	10048	8982	10068
	Previous	17850	18431	17436	15441

Table 6. The median deviation of the bank angle (deg) for machines generated for different problem instances

$\|S\|$	Machine representation method	Loop, f	Roll, f	U-turn, f	U-turn, f_Δ
3	Proposed	1.71	16.52	4.80	2.07
	Previous	6.37	18.56	50.29	21.60
4	Proposed	2.41	15.35	4.10	1.99
	Previous	6.32	21.86	57.04	22.88
5	Proposed	3.21	14.74	4.07	1.99
	Previous	9.54	22.99	45.83	24.32

Table 7. The median deviation of the pitch angle (deg) for machines generated for different problem instances

$\|S\|$	Machine representation method	Loop, f	Roll, f	U-turn, f	U-turn, f_Δ
3	Proposed	17.21	3.20	1.95	0.50
	Previous	20.54	4.44	7.58	3.85
4	Proposed	23.04	2.51	1.42	0.48
	Previous	22.11	4.08	6.79	4.61
5	Proposed	25.27	2.43	1.36	0.47
	Previous	24.44	4.68	7.83	5.31

3.2. Experiment Using Computer Simulation

The machines generated by the ant colony optimization algorithm were tested in the flight simulator. The tests of each machine consisted in starting it ten times in conditions similar to the ones during the recording of training examples. In order to measure the quality of the machines two criteria were used that showed the mean deviation angles of the pitch and bank when testing from the respective angles recorded on training examples. More precisely, for the pitch angle the arithmetic mean of $\left|\alpha_{i,t}^{\text{test}} - \alpha_{j,t}^{\text{run}}\right|$ $(i = \overline{1,N}, t = \overline{1,L_i}, j = \overline{1,10})$ was used as the criterion of quality, where $\alpha_{i,t}^{\text{test}}$ is the pitch angle at time t of training example i, and $\alpha_{j,t}^{\text{run}}$ is the pitch angle at time t of the launch of machine j in the flight simulator. A similar criterion was used for the roll angle.

The selection of machines for testing in simulation was carried out as follows. For each pair of sample problems and the number of states the best (according to the maximum reached value of the FF) machines for each of the 50 runs of the ant colony optimization algorithm were considered. These machines were then tested in the flight simulator. Tables 6 and 7 contain the medians of the quality criteria for the described groups of 50 machines.

The differences of the values of the quality criteria for the same aerobatic maneuvers and number of states but for different machine representation methods were verified using the Mann–Whitney U test

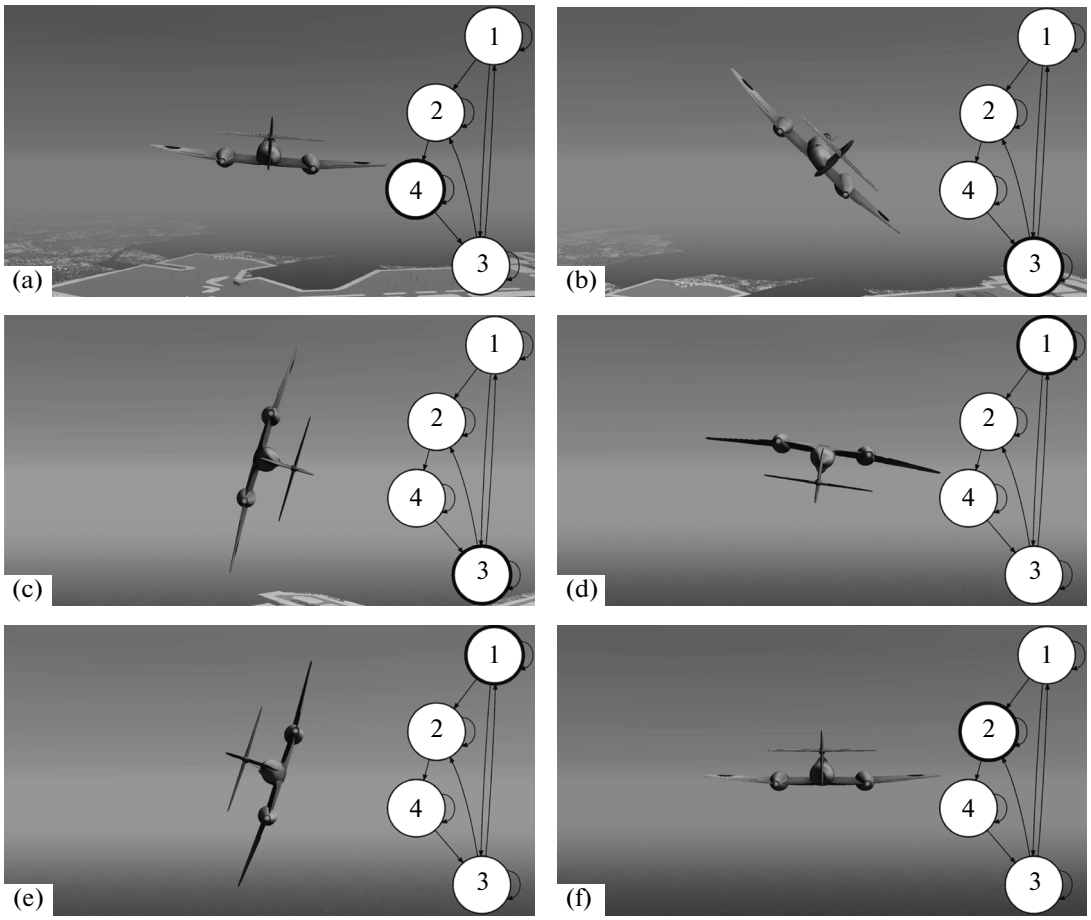

Fig. 5. Execution of the U-turn under the control of the generated machine (screenshots).

(for each of the two quality criteria, a total of 12 statistical tests were conducted, in each of which two groups of 50 values were compared). The differences for all pairs of samples but two (the values for the pitch angle when conducting a loop with four machines and five states) proved to be statistically significant ($p < 0.001$, the p-values were adjusted by the Holm method). The following conclusions can be made based on the tables:

(1) The values of quality criteria are generally better (less) for machines with the proposed representation method.

(2) The use of f_Δ improves the quality of the turn for the machines regardless of their representation method.

Note that differences between the values of the quality criteria actually correspond to qualitative differences in the behavior of machines in the simulation. Machines with the proposed representation method better align the position of the airplane at the end of a loop and roll, as well as make the roll without jerks that are sometimes characteristic of machines with the previous representation method. Considering the turn, the median values of the quality criteria for this aerobatic maneuver using machines with the previous representation method are sufficiently large. The reason is that none of these machines is able to cope with the turn from the beginning to the end. At the same time, most of the considered machines with the proposed representation method performed the turn and differed only in the quality of maintaining the angle. As it turned out, constant output actions on machine transitions from the previous representation method were not sufficient to maintain the bank angle near the desired values. For the proposed machine representation method, the values of the quality criteria values are much smaller, and the discussed machines are able to perform a U-turn, while machines constructed using f_Δ better maintained the bank angle at 60°.

Thus, the conducted experimental study demonstrates the successful implementation of the representation method of machines on different sets of training examples and fitness functions f_Δ on one of the sets of training examples, as well as the relatively high performance of the method.

Screenshots with the fighter *Gloster Meteor* performing a roll under the control of one of the machines generated using the approach proposed in the paper are shown in Figs. 5a–5f. The images show the 2nd, 5th, 9th, 14th, 19th, and 23rd seconds of the aerobatic maneuver. The machine with the current state highlighted in bold circle is superimposed on the right part of each screenshot. In addition, the video recordings of various aerobatic maneuvers with machines with the proposed representation method are available at the following links:

(1) Loop: http://www.youtube.com/watch?v=vnJ_-gMLjx8.

(2) Roll: http://www.youtube.com/watch?v=2GxzcPx0XNw.

(3) Turn: http://www.youtube.com/watch?v=n9q5FmCYs6M.

The different states of the machines correspond to different segments of aerobatic maneuvers. For example, the machine shown in the video of the roll successively changes states 4 (the beginning of the turn), 1 (the first half of the turn), 2 (the second half of the turn), and 3 (alignment of the airplane) during the execution of the maneuver.

CONCLUSIONS

In this paper, we propose a modification of the method for the generation of control finite-state machines with continuous actions. By the example of the problem of controlling an airplane model the benefits of the proposed approaches compared to the methods described in [13–15] are demonstrated. A new approach to the representation of machines improved the quality of the implementation of the aerobatic maneuvers discussed in the paper. In addition, the implementation of machines in the form of Moore automata makes them easier to interpret and analyze. The proposed approach made it also possible to construct a machine that performs a U-turn in a horizontal plane, which was not possible with the previously developed methods.

The results can be applied to the development of control systems for unmanned aerial vehicles (UAVs). For example, Transas [20] has developed UAVs and control systems for them. UAV models are tested in the *FlightGear* simulator. In addition, one of the possible ways of developing this study is the application of the approach to the control of mobile robots, for which there are also simulators. This direction is attractive due to the ease of use of control finite-state machine systems on real models.

APPENDIX

Reduction of the problem of the arrangement of output actions to the solution of the set of systems of linear algebraic equations. Let us construct a system of equations whose solution is required to determine the output machine's actions that maximize the FF on a given frame. Let us consider the case of the use of f (the case with f_Δ is similar). Recall that the optimized values are $r_{s,i,j} \in \mathbb{R}$, for which the component of output actions $v_{j,i}$ (the ith component for the jth control parameter) is significant in state s (the other numbers are known to be equal to zero). Let us reindex the optimized values and denote them as $\hat{r}_{l,j}$ ($j = \overline{1,c}$, $l = \overline{1,M}$, where $M = k_j|S|$, and k_j is the number of significant components for the control parameter j). When the index of the control parameter is fixed, we will omit it and denote the optimized values as $\hat{r}_1, \ldots, \hat{r}_M$. State $\hat{s}_l$ and some component of output actions $\hat{v}_l$ will correspond to each number $\hat{r}_l$, $l = \overline{1,M}$. We denoted the value of $\hat{v}_l$ calculated by $I_{i,t,j}$ ($j = \overline{1,p}$) by $\hat{v}_{i,t,l}$.

Now let us explain the possibility of solving the optimization problem for each control parameter individually. For a fixed frame the penalty P_τ is independent of the output actions. Thus, the maximization problem (2.3) is reduced to the following one:

$$1 - \sqrt{\frac{1}{N}\sum_{i=1}^{N}\rho^2\left(\tilde{O}_i, O_i\right)} \to \max_{\hat{r}_{l,j}, l=\overline{1,M}, j=\overline{1,c}}.$$

By expanding (2.2) and making elementary transformations, we obtain

$$\sum_{i=1}^{N}\frac{1}{L_i}\sum_{t=2}^{L_i}\sum_{j=1}^{c}\left(\frac{\tilde{O}_{i,t,j}-O_{i,t,j}}{M_j-m_j}\right)^2\to\min_{\hat{r}_l,j,l=\overline{1,M},j=\overline{1,c}}.$$

This shows that it is possible to minimize the amount for each of the control parameters j separately. For a fixed control parameter $j = j_0$ the problem takes the following form:

$$g_{j_0}=\sum_{i=1}^{N}\frac{1}{L_i}\sum_{t=2}^{L_i}\left(\tilde{O}_{i,t,j_0}-O_{i,t,j_0}\right)^2\to\min_{\hat{r}_1,\ldots,\hat{r}_M}. \tag{A.1}$$

Let us express $\tilde{O}_{i,t,j_0}$ through $\hat{r}_1, \ldots, \hat{r}_M$. For this purpose, we introduce an additional notation: $\beta_{i,t,l} = \hat{v}_{i,t,l}$, if $\hat{s}_l$ is the current state of the machine when it passes the ith training examples before cycle t, otherwise $\beta_{i,t,l} = 0$. By rewriting (2.1) in the new notation, we obtain

$$\tilde{O}_{i,t,j}=\tilde{O}_{i,t-1,j}+\sum_{l=1}^{M}\beta_{i,t-1,l}\hat{r}_l.$$

Using equality $\tilde{O}_{i,1,j} = O_{i,1,j}$, the formula can be closed:

$$\tilde{O}_{i,t,j}=O_{i,1,j}+\sum_{l=1}^{M}\sum_{t'=1}^{t-1}\beta_{i,t',l}\hat{r}_l.$$

Using the notation

$$\alpha_{i,t,l}=\sum_{t'=1}^{t-1}\beta_{i,t',l}$$

it is rewritten as follows:

$$\tilde{O}_{i,t,j}=O_{i,1,j}+\sum_{l=1}^{M}\alpha_{i,t,l}\hat{r}_l. \tag{A.2}$$

Now let us substitute (A.2) in (A.1) and equate partial derivatives g_{j_0} to zero:

$$\frac{\partial g_{j_0}}{\partial\hat{r}_{l_0}}=2\sum_{i=1}^{N}\frac{1}{L_i}\sum_{t=2}^{L_i}\alpha_{i,t,l_0}\left(O_{i,1,j_0}+\sum_{l=1}^{M}\alpha_{i,t,l}\hat{r}_l-O_{i,t,j_0}\right)=0,\quad l_0=\overline{1,M}.$$

This expression can be written in a more convenient form:

$$\sum_{l=1}^{M}\left(\sum_{i=1}^{N}\frac{1}{L_i}\sum_{t=2}^{L_i}\alpha_{i,t,l_0}\alpha_{i,t,l}\right)\hat{r}_l=\sum_{i=1}^{N}\frac{1}{L_i}\sum_{t=1}^{L_i}\alpha_{i,t,l_0}\left(O_{i,t,j_0}-O_{i,1,j_0}\right),\quad l_0=\overline{1,M}. \tag{A.3}$$

It follows from the above that the optimum output actions for a given output control parameter j_0 can be found by solving the set of equations (A.3), which are a system of linear algebraic equations M. In order to find its matrix, $\mathbb{O}(M^2(L_1+\ldots+L_N))$ time is required, and to solve it by the Gauss method, $\mathbb{O}(M^3)$ is needed. The system must be solved for each control parameter. However, the computation can be reduced in the following way.

If the sets of components of output actions for some of the control parameters coincide and the same masks of the significance of the components of the output actions are used for these parameters, the left-hand sides of systems (A.3) for these control parameters are also the same. It makes it possible not to recalculate them. In the present study, we decided to apply this optimization to ailerons and rudder, since it proved to be reasonable for them to use the same sets of components of the output actions: both controls are mainly used to control the bank angle and the corresponding components depend on this angle. For the remaining control, the elevator, the use of the same components is bad, because it controls the

pitch angle. The use of the combined set of components of output actions based on both the bank angle and the pitch angle for all three controls would lead to the fact that the components based on one angle would be used for all controls, which is undesirable. This would occur because of the importance of the combination of masks of significance, which are also necessary for optimization.

ACKNOWLEDGMENTS

This work was financially supported by the Government Assignment No. 2.1239.2014/K, by the Government of Russian Federation, Grant 074-U01, and also by the Russian Foundation for Basic Research, project no. 14-07-31244 mol_a.

REFERENCES

1. D. Harel and A. Pnueli, "On the development of reactive systems," in *Logic and Models of Concurrent Systems*, Ed. by K. Apt, NATO Advanced Study Institute on Logic and Models for Verification and Specification of Concurrent Systems Series (Springer-Verlag, 1985), pp. 477–498.
2. D. Harel and M. Politi, *Modeling Reactive Systems with Statechart. The Statemate Approach* (McGraw-Hill, New York, 1998).
3. N. I. Polikarpova and A. A. Shalyto, *Automata-Based Programming* (Piter, St.-Petersburg, 2011) [in Russian]. http://is.ifmo.ru/books/_book.pdf
4. N. Walkinshaw, K. Bogdanov, M. Holcombe, et al., "Reverse engineering state machines by interactive grammar inference," in *Proceedings of the 14th Working Conference on Reverse Engineering, WCRE'07* (IEEE Computer Society Press, Vancouver, 2007), pp. 209–218.
5. N. Walkinshaw, R. Taylor, and J. Derrick, "Inferring extended finite state machine models from software executions," in *Proceedings of the 20th Working Conference on Reverse Engineering WCRE'13* (IEEE Computer Society Press, Koblenz, 2013), pp. 301–310.
6. F. N. Tsarev and A. A. Shalyto, "Application of genetic programming for automata generation in the 'artificial ant' problem," in *Proceedings of the 4th International Scientific-Practical Conference on Integrated Models and Soft Computing in Artificial Intelligence* (Fizmatlit, Moscow, 2007), Vol. 2, pp. 590–597. http://is.ifmo.ru/genalg/_ant_ga.pdf
7. J. R. Koza, *Genetic Programming: on the Programming of Computers by Means of Natural Selection* (MIT Press, Cambridge, 1992).
8. V. M. Kureichik, "Genetic algorithms: state of the art, problems, and perspectives," J. Comput. Syst. Sci. Int. **38**, 137 (1999).
9. V. M. Kureichik and S. I. Rodzin, "Evolutionary algorithms: genetic programming," J. Comput. Syst. Sci. Int. **41**, 123 (2002).
10. L. A. Gladkov, V. V. Kureichik, and V. M. Kureichik, *Genetic Algorithms* (Fizmatlit, Moscow, 2006) [in Russian].
11. N. I. Polikarpova, V. N. Tochilin, and A. A. Shalyto, "Method of reduced tables for generation of automata with a large number of input variables based on genetic programming," J. Comput. Syst. Sci. Int. **49**, 265 (2010).
12. F. N. Tsarev, "Induction of finite state machines using genetic programming with fitness based on testing," Inform.-Upravl. Sist., No. 5, 31–36 (2010).
13. A. V. Aleksandrov, S. V. Kazakov, A. A. Sergushichev, et al., "The use of evolutionary programming based on training examples for the generation of finite state machines for controlling objects with complex behavior," J. Comput. Syst. Sci. Int. **52**, 410 (2013).
14. I. Buzhinsky, V. Ulyantsev, and A. Shalyto, "Test-based induction of finite-state machines with continuous output actions," in *Proceedings of the 7th IFAC Conference on Manufacturing Modelling, Management, and Control MIM'13* (IFAC, St.-Petersburg, 2013), pp. 1049–1054.
15. I. P. Buzhinsky, V. I. Ulyantsev, D. S. Chivilikhin, and A. A. Shalyto, "Inducing finite state machines from training samples using ant colony optimization," J. Comput. Syst. Sci. Int. **53**, 256 (2014).
16. FlightGear. http://www.flightgear.org/. Cited September 29, 2014.
17. M. Dorigo and T. Stützle, *Ant Colony Optimization* (MIT Press, US, 2004).
18. D. Chivilikhin and V. Ulyantsev, "Learning finite-state machines with ant colony optimization," Lect. Notes Comp. Sci. **7461**, 268–275 (2012).
19. M. López-Ibáñez, J. Dubois-Lacoste, T. Stützle, et al., "The irace package, iterated race for automatic algorithm configuration," Tech. Report TR/IRIDIA/2011-004 (IRIDIA, Univ. libre de Bruxelles, Belgium, 2011).
20. Transas. http://www.transas.ru/. Cited September 29, 2014.

Translated by O. Pismenov