ISSN 1064-2307, Journal of Computer and Systems Sciences International, 2013, Vol. 52, No. 3, pp. 410–425. © Pleiades Publishing, Ltd., 2013.
Original Russian Text © A.V. Aleksandrov, S.V. Kazakov, A.A. Sergushichev, F.N. Tsarev, A.A. Shalyto, 2013, published in Izvestiya Akademii Nauk. Teoriya i Sistemy Upravleniya, 2013, No. 3, pp. 85–100.

COMPUTER METHODS

The Use of Evolutionary Programming Based on Training Examples for the Generation of Finite State Machines for Controlling Objects with Complex Behavior

A. V. Aleksandrov, S. V. Kazakov, A. A. Sergushichev, F. N. Tsarev, and A. A. Shalyto

St. Petersburg National Research University of Information Technologies, Mechanics, Optics, and Automation, St. Petersburg, Russia

e-mail: Fedor.tsarev@gmail.com

Received July 5, 2011; in final form, September 21, 2012

Abstract—It is proposed to use evolutionary programming to generate finite state machines (FSMs) for controlling objects with complex behavior. The well-know approach in which the FSM performance is evaluated by simulation, which is typically time consuming, is replaced with comparison of the object's behavior controlled by the FSM with the behavior of this object controlled by a human. A feature of the proposed approach is that it makes it possible to deal with objects that have not only discrete but also continuous parameters. The use of this approach is illustrated by designing an FSM controlling a model aircraft executing a loop-the-loop maneuver.

DOI: 10.1134/S1064230713020020

INTRODUCTION

In recent years, automata-based programming has been widely used for the development of software for systems with complex behavior. Within this approach, the behavior of computer programs is described in terms of deterministic finite state machines (FSMs) [1]. In automata-based programming, it is proposed to design programs as a set of automated control objects. Each object of this kind consists of a plant (object under control) and a control system (a system of controlling FSMs). The FSM system gets variables and events from the external environment and from the plant at its input. Based on these data, the control system produces outputs (controls) used control the plant. The control paradigm based on a similar approach is called automata-based control in [2]. For many problems, control FSMs can be designed heuristically; however, there are problems in which this is impossible or difficult. These are, for example, the problems *Smart Ant* [3–5], *Smart Ant-3* [6], and control of a model of an unmanned aerial vehicle (UAV) [7].

There are several approaches to the last problem. One of them is to select a *perfect* trajectory among the trajectories of several flights performed under human control, and then follow this trajectory. This approach was considered in [8]. Another approach is to use an FSM for controlling an UAV and to design such an FSM using genetic algorithms described in [9–13].

In [14], a genetic algorithm based on the use of the reduced table method for the representation of FSMs was employed to generate an upper-level FSM controlling a model of an UAV. The computation of the fitness function in this case is based on the simulation of the UAV behavior in the environment, which is time consuming.

The purpose of this study is to develop a method for designing FSMs controlling plants with complex behavior that is free from this drawback. This method is based on evolutionary programming. We propose to design FSMs for the control of such plants separately for each mode of its operation using evolutionary programming and *training examples* created for each mode. Specifying a large number of training examples, one can avoid, as in [8], errors committed by human operators in the course of control. This approach may be considered as an elaboration of the approach proposed in [15], where plants controlled only by discrete controls were considered.

In the present paper, this approach is generalized to the plants controlled not only by discrete but also by continuous controls. As an example of a plant with complex behavior, we consider an aircraft model executing a loop-the-loop maneuver. To combine the FSMs controlling the individual modes, an upper-level FSM is designed using the method of reduced tables [14] or a heuristic method. Each state of this FSM corresponds to a specific mode. As a result, a hierarchical system of interacting FSMs is formed. The problem of designing the upper-level FSM is out of the scope of this paper. In addition to high perfor-

Description of transitions of the best FSM

Arc	Predicates	Discrete output controls		Continuous output controls			
		magneto	starter	throttle	ailerons	elevation rudder	rudder
0 ⟶ 0	not x_0	0	0	1.00000	0.05239	0.01043	0.00000
	not x_2	0	0	0.00000	−0.01315	0.02117	0.00000
	not x_4	0	0	0.00000	0.00199	−0.09535	0.00000
	not x_6	0	0	0.00000	0.01336	−0.07833	0.00000
	not x_8	0	0	0.00000	0.00178	−0.10299	0.00000
	not x_{11}	3	0	0.00000	−0.00178	0.01466	0.00000
0 ⟶ 1	not x_5	0	0	0.00000	−0.01119	0.09089	0.00000
	x_9	0	0	0.00000	−0.01622	0.20543	0.00000
	not x_9	0	0	0.00000	0.00000	0.00000	0.00000
0 ⟶ 2	x_1	0	0	0.00000	0.00217	0.01965	0.00000
	x_3	0	0	0.00000	0.02432	0.01225	0.00000
	x_5	0	0	0.00000	0.00000	0.00000	0.00000
	x_6	0	0	0.00000	0.00000	0.00000	0.00000
	not x_{12}	3	0	0.00000	0.00018	−0.00087	0.00000
0 ⟶ 3	x_4	0	0	0.00000	−0.01352	−0.08765	0.00000
	not x_7	0	0	0.00000	−0.00394	0.06189	0.00000
	not x_{10}	0	0	0.00000	−0.00688	0.05064	0.00000
1 ⟶ 0	not x_0	0	0	1.00000	0.05239	0.01043	0.00000
	x_2	0	0	0.00000	−0.02552	−0.01522	0.00000
	x_4	0	0	0.00000	0.00000	0.00000	0.00000
	x_{11}	3	0	0.00000	0.00000	0.00000	0.00000
1 ⟶ 1	not x_1	0	0	0.00000	−0.00132	0.00866	0.00000
	not x_2	0	0	0.00000	0.00000	0.00000	0.00000
	x_5	0	0	0.00000	−0.02809	−0.01434	0.00000
	x_7	0	0	0.00000	−0.00054	−0.00506	0.00000
	not x_{12}	3	0	0.00000	−0.01224	−0.03468	0.00000
1 ⟶ 2	x_0	0	0	0.00000	−0.00161	−0.00478	0.00000
	x_3	0	0	0.00000	0.00000	0.00000	0.00000
	not x_3	0	0	0.00000	−0.00274	−0.00202	0.00000
	x_6	0	0	0.00000	−0.01154	−0.11022	0.00000
	x_8	0	0	0.00000	−0.01082	−0.06436	0.00000
	not x_9	0	0	0.00000	−0.00993	−0.06966	0.00000
1 ⟶ 3	not x_5	0	0	0.00000	−0.03879	−0.09682	0.00000
	not x_{10}	0	0	0.00000	0.00172	0.03226	0.00000
	x_{12}	3	0	0.00000	0.00497	−0.01437	0.00000
2 ⟶ 0	not x_1	0	0	0.00000	0.03000	0.06766	0.00000
	not x_3	0	0	0.00000	0.00000	0.00000	0.00000
	x_9	3	0	0.00000	0.01939	0.04064	0.00000
2 ⟶ 1	x_0	0	0	0.00000	0.02731	0.02049	0.00000
	not x_0	0	0	0.00000	0.02047	−0.02413	0.00000
	x_4	0	0	0.00000	0.00110	−0.00006	0.00000
	not x_4	0	0	0.00000	0.00000	0.00000	0.00000
	x_8	0	0	0.00000	0.01159	0.06208	0.00000

Table (Contd.)

Arc	Predicates	Discrete output controls		Continuous output controls			
		magneto	starter	throttle	ailerons	elevation rudder	rudder
2 → 2	x_2	0	0	0.00000	0.01880	0.00322	0.00000
	x_6	0	0	0.00000	−0.02062	−0.01551	0.00000
	not x_7	0	0	0.00000	0.00309	0.05025	0.00000
	not x_{10}	0	0	0.00000	−0.00054	−0.01195	0.00000
	not x_{11}	3	0	0.00000	0.01573	0.06745	0.00000
	not x_{12}	3	0	0.00000	−0.00140	−0.00006	0.00000
2 → 3	not x_3	0	0	0.00000	−0.00889	−0.04245	0.00000
	x_5	0	0	0.00000	0.00000	0.00000	0.00000
	x_7	0	0	0.00000	0.00239	0.03443	0.00000
	x_{11}	0	0	0.00000	0.00000	0.00000	0.00000
3 → 0	x_1	0	0	0.00000	0.00661	−0.00695	0.00000
	not x_{10}	0	0	0.00000	0.01817	0.04933	0.00000
3 → 1	x_3	0	0	0.00000	0.02187	0.03925	0.00000
	x_5	0	0	0.00000	0.00905	0.09217	0.00000
	not x_6	0	0	0.00000	0.00664	0.08602	0.00000
	not x_{12}	3	0	0.00000	0.00556	−0.07512	0.00000
3 → 2	x_4	0	0	0.00000	−0.01550	−0.09491	0.00000
	x_8	0	0	0.00000	−0.00498	0.01483	0.00000
	x_{10}	0	0	0.00000	0.00000	0.00000	0.00000
	not x_{11}	3	0	0.00000	0.00177	−0.03823	0.00000
3 → 3	not x_0	0	0	1.00000	0.04394	0.11148	0.00000
	not x_2	0	0	0.00000	−0.02054	−0.17037	0.00000
	not x_7	0	0	0.00000	−0.00315	0.00256	0.00000
	not x_9	0	0	0.00000	−0.01064	0.00960	0.00000
	x_{12}	3	0	0.00000	0.00757	0.00196	0.00000

mance, the proposed method has one more advantage over the simulation-based computation of the fitness function—this function does not need to be modified when the operation mode changes; neither is it modified when the plant itself is replaced.

1. PROBLEM STATEMENT

Figure 1 shows a block diagram of designing the control FSM.

The source data for designing the control FSM is the collection of training examples (tests); their structure is described below in more detail. The tests specifying the reference behavior are created by human operators. The aim of the evolutionary programming algorithm is to construct an FSM that controls the plant such that its behavior is as close to the reference behavior as possible. If a sufficiently large number of tests are used, it is additionally possible to get rid of small errors committed by human operators when controlling the plant.

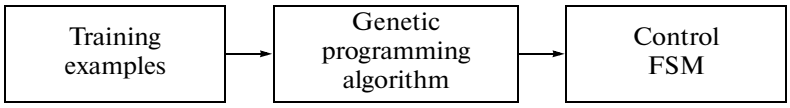

Fig. 1. Block diagram of the control FSM.

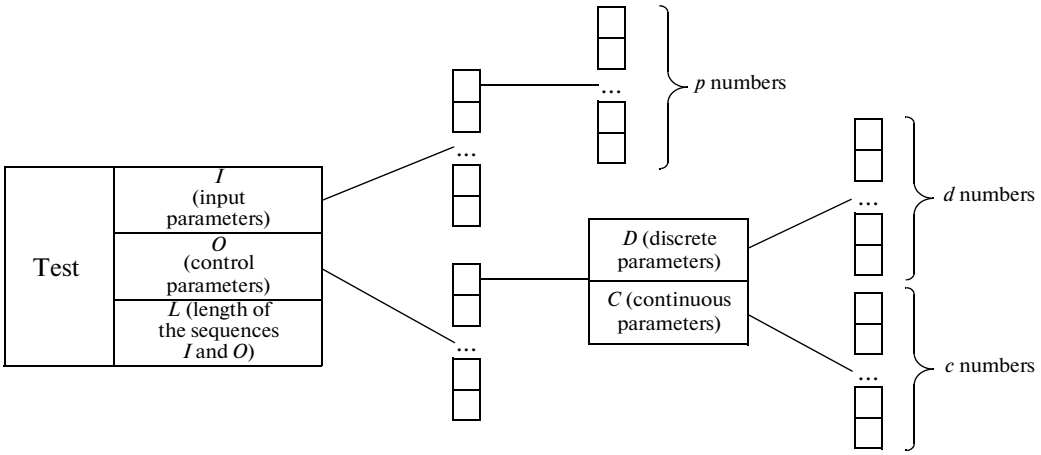

Fig. 2. Framework of a training sample.

1.1. Control Devices

The plant has a set of control devices, and one can control the plant by acting upon these devices. The parameters corresponding to the control devices are called control parameters. The parameters of some control devices can take only a finite set of values—such devices are said to be discrete. The parameters of other control devices take real values—such devices are said to be continuous. We also call the parameters acting upon continuous control devices continuous controls and the parameters acting upon discrete control devices, discrete controls.

The approach proposed in this paper suits for controlling plants with both discrete and continuous control devices. If all the control devices are discrete, it is recommended to use the approach described in [15]. A continuous control changes the corresponding control device parameter by a real magnitude; a discrete control sets the corresponding control device to a certain state. A sequence of controls applied to a control device is equivalent to the sum of these controls if the control is continuous and to the last control if it is discrete.

For example, the deflection of the aircraft steering wheel is a continuous control parameter. The continuous control of this device is the change of its deflection by a certain value. Then, the sequence of deflections by x and y degrees is equivalent to the wheel deflection by $x + y$ degrees. The starter is an example of a discrete control device. In this case, the discrete control is switching the starter on or off. The sequence of switchings of the starter on or off is equivalent to the last action.

1.2. Test Structure

The block diagram representing the test structure is shown in Fig. 2.

The test T_i consists of two parts— I_i and O_i. Each of them is a sequence of length L_i; the first part consists of the values of the input parameters, and the second part consists of the corresponding reference values of the control parameters that are registered in the course of experiments when the aircraft is controlled by a human operator. Each element $I_{i,t}$ of the sequence of input parameters consists of p numbers—the values of these parameters at the time t. Each element $O_{i,t}$ consists of two sets— $D_{i,t}$ and $C_{i,t}$. The sequence $D_{i,t}$ consists of d discrete parameters, and $C_{i,t}$ consists of c continuous parameters. Thus, $O_{i,t}$ consists of $d + c$ components. The elements of these sets are denoted by $D_{i,t,k}$ and $C_{i,t,k}$, respectively. We denote by $\tilde{O}_i$ the set of control parameters produced by the FSM on the test T_i. The structure of $\tilde{O}_i$ coincides with the structure of O_i. The corresponding elements are denoted by $\tilde{D}_{i,t}$, $\tilde{C}_{i,t}$, $\tilde{D}_{i,t,k}$, and $\tilde{C}_{i,t,k}$, respectively.

2. EVOLUTIONARY PROGRAMMING ALGORITHM

The classical evolutionary programming algorithm involves the following steps: (1) creation of the initial population, (2) evaluation of the fitness function, (3) selection of individuals for crossover, (4) crossover, (5) mutation. The generation obtained after step (5) becomes the current generation, and steps (2)–(5) are repeated until the stopping condition is fulfilled.

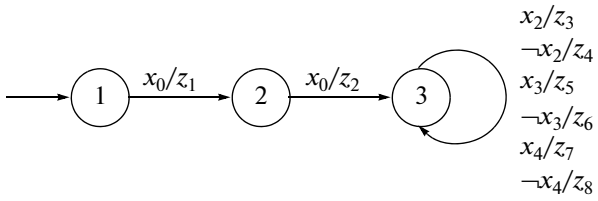

Fig. 3. Framework of the FSM.

The proposed algorithm differs from the classical one in an additional step executed before step (2) maximizing the fitness function; at the additional step, output controls are assigned to the transitions of the FSM framework. By the framework, we mean an FSM in which the values of the output controls will be determined later. The framework is specified by a complete table of transitions for each state and each truth and falsity conditions of the input variable. There are two types of transitions in the framework. The transitions of the first type have no output controls, and the transitions of the second type have output controls. Below, we describe the proposed algorithm in the order that facilitates its understanding.

2.1. Individual in the Evolutionary Programming Algorithm

In the proposed algorithm, the FSM is represented by an object that contains the descriptions of transitions from each state and the index of the initial state; the number of states is fixed, and it is a parameter of the algorithm. For each transition, a condition is specified under which this transition is performed. This condition has the form "x_i" or "$\neg x_i$", where x_i is the i-eth proposition (predicate) about an object state (for example, **the aircraft engine is on**). These conditions are composed manually before the evolutionary algorithm starts executing, and they do not change in the course of its operation. No values of the output controls z_j (where z_j is the jth tuple of changes in the control parameters) are specified for the transitions of the individuals. Thus, only the FSM framework is encoded by each individual, while the specific output controls produced at the transitions are determined by the output assignment algorithm.

2.2. Operation of the FSM

We assume that the FSM to be generated is synchronous; that is, all the steps of its operation are identical, and their duration is determined by the sluggishness of the plant. One step of the FSM operation is defined as its work on a single set of input parameters.

At each step, the control system gets the values of all input parameters. Then, the logical values of all the conditions in the order of increasing indexes are evaluated. Once the value of a condition is evaluated, it is sent to the FSM input. At each step, the FSM can perform several transitions, and the resultant output at each time t may consist of several sequentially executed controls at the individual transitions. Recall that each parameter of the resultant control is a sum of the controls if the parameter is continuous and is the last control if the parameter is discrete.

Figure 3 shows a possible FSM framework.

For this framework, x_0–x_4 are the given predicates, and the values of the output controls (tuples) $z_1, \ldots, z_8$ are determined later using the output assignment algorithm.

2.3. Fitness Function

To evaluate the fitness function, each of N sequences I_i —the input set of the ith test—is sent to the input of the FSM, and the sequence of output parameters $\tilde{O}_i$ generated by the FSM is determined. Then, the value of the fitness function is calculated. The fitness function reflects the closeness of the FSM behavior to the reference behavior; this function is defined as

$$Fitness = 1 - \sqrt{\frac{1}{N}\sum_{i=1}^{N}\frac{\rho^2(O_i,\tilde{O}_i)}{\rho^2(O_i,0)}}.$$

To simplify the further formulas, we denote by A_i the factor $\dfrac{1}{\rho^2(O_i,0)}$; then, the fitness function takes the form

$$Fitness = 1 - \sqrt{\frac{1}{N}\sum_{i=1}^{N} A_i \rho^2(O_i, \tilde{O}_i)}. \quad (2.1)$$

Here, by $\rho(O_i, \tilde{O}_i)$, we mean the "distance" between the output and reference sequences of the control parameters that is determined by the formula

$$\rho(O_i, \tilde{O}_i) = \sqrt{\sum_{t=1}^{L_i}\left(\sum_{k=1}^{d}\left[D_{i,t,k} \neq \tilde{D}_{i,t,k}\right] + \sum_{k=1}^{c}\left(C_{i,t,k} - \tilde{C}_{i,t,k}\right)^2\right)}.$$

As was indicated above, the control assignment algorithm is executed before the fitness function evaluation. This algorithm assigns controls to the transitions so as to maximize the fitness function for the given FSM framework. Below, when the maximization or minimization of a function is mentioned, it is assumed that its arguments are the values of the output controls at the transitions and their domain depends on the type of the control parameters (discrete or continuous) of the corresponding controls.

2.4. Control Assignment Algorithm as the Additional Step in the Evolutionary Programming Algorithm

When this algorithm is used for the transitions of the FSM framework, the set of input parameters of each test is fed at the FSM input. The transitions performed by the FSM are stored.

2.4.1. Single test. To facilitate the understanding of the proposed approach, we consider the case when the training example consists of a single test T ($N = 1$). Then, the fitness function has the form

$$Fitness = 1 - \sqrt{A\rho^2(O, \tilde{O})}.$$

Here and below in this section, we omit the first subscript 1 to simplify the formulas (we write A_1 instead of $A1$, O instead of O_1, and the like).

The fitness function attains its maximum when the distance squared $\rho^2(O, \tilde{O})$ is minimal. Upon the substitution and the change in the summation order, we obtain

$$\sum_{k=1}^{d}\sum_{t=1}^{L}\left[D_{t,k} \neq \tilde{D}_{t,k}\right] + \sum_{k=1}^{c}\sum_{t=1}^{L}\left(C_{t,k} - \tilde{C}_{t,k}\right)^2.$$

Notice that the sum for each parameter may be minimized independently. Therefore, it suffices to minimize the sum

$$\sum_{t=1}^{L}\left[D_{t,k} \neq \tilde{D}_{t,k}\right] \quad (2.2)$$

for each k in the range from 1 to d and the sum

$$\sum_{t=1}^{L}\left(C_{t,m} - \tilde{C}_{t,m}\right)^2 \quad (2.3)$$

for each m in the range from 1 to c. Below, the indexes k and m are assumed to be fixed.

2.4.1.1. Assignment of discrete controls. Let $\tilde{D}_{t,k}$ be the yet unknown value of the kth discrete control parameter at the last performed transition at the time t. Then, expression (2.2) can be rewritten as

$$\sum_{i=1}^{n}\sum_{t \in \Psi_i}\left[D_{t,k} \neq \tilde{D}_{t,k}\right]. \quad (2.4)$$

Here, Ψ_i is the set of times t at which the index of the last performed transition was i and n is the number of transitions in the FSM. To minimize sum (2.4), it is sufficient to minimize the sum

$$\sum_{t \in \Psi_i}\left[D_{t,k} \neq \tilde{D}_{t,k}\right]$$

for each i in the range from 1 to n. Let us fix i. For $t \in \Psi_i$, $\tilde{D}_{t,k}$ is equal to the value of the kth discrete parameter at the transition i, which depends on t. We denote this value by u ($u = \tilde{D}_{t,k}$ for $t \in \Psi_i$). Using this notation, we can write

$$\sum_{t \in \Psi_i} [D_{t,k} \neq u].$$

Let $\tilde{\Psi}_{i,h}$ be the set of instants of time t in Ψ_i at which $D_{t,k}$ is equal to v_h. Here, v_h is the h-eth possible value of the kth discrete parameter. Note that $\tilde{\Psi}_{i,h}$ are nonoverlapping subsets satisfying the condition

$$\bigcup_{h=1}^{G} \tilde{\Psi}_{i,h} = \Psi_i.$$

Taking this into account and denoting by G the number of possible values of the kth discrete parameter, we obtain

$$\sum_{h=1}^{G} \sum_{t \in \tilde{\Psi}_{i,h}} [u \neq D_{t,k}] = \sum_{h=1}^{G} \sum_{t \in \tilde{\Psi}_{i,h}} [u \neq v_h] = \sum_{h=1}^{G} [u \neq v_h] \sum_{t \in \tilde{\Psi}_{i,h}} 1 = \sum_{h=1}^{G} [u \neq v_h] \left|\tilde{\Psi}_{i,h}\right|. \quad (2.5)$$

Choosing $u = v_g$ as the value of the discrete parameter at the ith transition, we obtain the value of sum (2.5) equal to $|\Psi_i| - \left|\tilde{\Psi}_{i,g}\right|$ because all the coefficients $[[u \neq v_h]$ are equal to unity if and only if $g \neq h$ and

$$\sum_{h=1}^{G} \left|\tilde{\Psi}_{i,h}\right| = |\Psi_i|.$$

Therefore, in order to minimize sum (2.5), the value v_g must be chosen for which $\left|\tilde{\Psi}_{i,g}\right|$ is maximal, and this is the value among $D_{t,k}$ ($t \in \Psi_i$) that is encountered most often. If there are several such values, any of them may be chosen.

2.4.1.2. Assignment of continuous controls. As has been mentioned above, the change in the mth continuous parameter is equal to the sum of the changes of this parameter on all the transitions made. At each of these transitions, this parameter changes by a certain (constant for this transition) but unknown value. Denote by u_i the change in the parameter at the ith transition. Then, the final value of the parameter at the time t is equal to the sum of its initial value (which is zero) and all its changes at each transition:

$$\tilde{C}_{t,k} = \sum_{i=1}^{n} \alpha_i[t] u_i.$$

Here, $\alpha_i[t]$ is the number of executions of the ith transition by the time t. Therefore, sum (2.3) can be written as

$$S = \sum_{t=1}^{L} \left(C_{t,m} - \sum_{i=1}^{n} \alpha_i[t] u_i \right)^2.$$

To find the point at which S takes the minimal value, we should find the point at which the partial derivatives of S with respect to u_j for all j vanish. These derivatives have the form

$$\frac{\partial S}{\partial u_j} = -\sum_{t=1}^{L} 2\alpha_j[t] \left(C_{t,m} - \sum_{i=1}^{n} \alpha_i[t] u_i \right) = \sum_{t=1}^{L} 2\alpha_j[t] \left(\sum_{i=1}^{n} \alpha_i[t] u_i - C_{t,m} \right).$$

Upon some transformations, we obtain the system of linear equations

$$\begin{cases} \sum_{i=1}^{n}\left(\sum_{t=1}^{L}\alpha_1[t]\alpha_i[t]\right)u_i = \sum_{t=1}^{L}C_{t,m}\alpha_1[t]; \\ \sum_{i=1}^{n}\left(\sum_{t=1}^{L}\alpha_2[t]\alpha_i[t]\right)u_i = \sum_{t=1}^{L}C_{t,m}\alpha_2[t]; \\ \ldots \\ \sum_{i=1}^{n}\left(\sum_{t=1}^{L}\alpha_n[t]\alpha_i[t]\right)u_i = \sum_{t=1}^{L}C_{t,m}\alpha_n[t]. \end{cases}$$

In this paper, these c systems (for m in the range from 1 to c) are solved by Gaussian elimination for the unknowns u_j that indicate the change of the mth parameter at the transition j.

2.4.2. Several tests. In this subsection, we generalize the proposed method for the case of several tests. Function (2.1) attains its maximum when the sum

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

is minimal.

As before, the sums with respect to each parameter can be minimized independently of one another. Thus, we should minimize the sum

$$\sum_{i=1}^{N} A_i \sum_{t=1}^{L_i}\left[D_{i,t,k} \neq \tilde{D}_{i,t,k}\right] \tag{2.6}$$

for each k in the range from 1 to d and the sum

$$\sum_{i=1}^{N} A_i \sum_{t=1}^{L_i}\left(C_{i,t,m} - \tilde{C}_{i,t,m}\right)^2 \tag{2.7}$$

for each m in the range from 1 to c. Below, we assume that the indexes k and m are fixed.

2.4.2.1. Assignment of discrete controls. Let us single out in sum (2.6) the terms corresponding to each transition:

$$\sum_{i=1}^{N} A_i \sum_{j=1}^{n}\sum_{t\in\Psi_{i,j}}\left[D_{i,t,k} \neq \tilde{D}_{i,t,k}\right] = \sum_{j=1}^{n}\sum_{i=1}^{N} A_i \sum_{t\in\Psi_{i,j}}\left[D_{i,t,k} \neq \tilde{D}_{i,t,k}\right].$$

Here, $\Psi_{i,j}$ is the set of the instants of time at which the index of the last executed transition of the FSM run on test t is equal to i, and j, a n is the number of transitions in the FSM. Therefore, for each j in the range from 1 to n, it is required to minimize the function

$$\sum_{i=1}^{N} A_i \sum_{t\in\Psi_{i,j}}\left[D_{i,t,k} \neq \tilde{D}_{i,t,k}\right].$$

Let us fix j. As in the case when the training example consists of a single test, $\tilde{D}_{i,t,k}$ at $t \in \Psi_{i,j}$ equals the value of the kth discrete parameter at the transition j, which is independent of t. Again, we denote it by u.

Let $\tilde{\Psi}_{i,j,h}$ be the set of instants of time t in $\Psi_{i,j}$ at which $D_{i,t,k}$ is equal to v_h. As before, v_h is the hth possible value of the kth discrete parameter. The subsets $\tilde{\Psi}_{i,j,h}$ do not overlap and satisfy the relation

$$\bigcup_{h=1}^{G}\tilde{\Psi}_{i,j,h} = \Psi_{i,j}.$$

Denoting by G the number of possible values of this parameter, we obtain

$$\sum_{i=1}^{N} A_i \sum_{t \in \Psi_{i,j}} [D_{i,t,k} \neq u] = \sum_{i=1}^{N} A_i \sum_{h=1}^{G} \sum_{t \in \tilde{\Psi}_{i,j,h}} [D_{i,t,k} \neq u]$$
$$= \sum_{i=1}^{N} A_i \sum_{h=1}^{G} [v_h \neq u] \sum_{t \in \tilde{\Psi}_{i,j,h}} 1 = \sum_{i=1}^{N} A_i \sum_{h=1}^{G} [u \neq v_h] \left| \tilde{\Psi}_{i,j,h} \right|.$$

Choosing v_g as the value of the discrete parameter at the jth transition ($u = v_g$) and substituting it into the preceding sum, we can write it as

$$\sum_{i=1}^{N} A_i \sum_{h=1}^{G} [v_g \neq v_h] \left| \tilde{\Psi}_{i,j,h} \right| = \sum_{i=1}^{N} A_i \sum_{h=1}^{G} [g \neq h] \left| \tilde{\Psi}_{i,j,h} \right| = \sum_{i=1}^{N} A_i \left(\sum_{h=1}^{G} \left| \tilde{\Psi}_{i,j,h} \right| - \left| \tilde{\Psi}_{i,j,g} \right| \right)$$
$$= \sum_{i=1}^{N} A_i \left(\left| \Psi_{i,j} \right| - \left| \tilde{\Psi}_{i,j,g} \right| \right) = \sum_{i=1}^{N} A_i \left| \Psi_{i,j} \right| - \sum_{i=1}^{N} A_i \left| \tilde{\Psi}_{i,j,g} \right|. \quad (2.8)$$

To minimize sum (2.8), we should find the value of v_g for which

$$\sum_{i=1}^{N} A_i \left| \tilde{\Psi}_{i,j,g} \right|$$

is maximal. Therefore, we should assign v_g as the value of the kth discrete parameter at the jth transition; here,

$$g = \arg \max_{g} \sum_{i=1}^{N} A_i \left| \tilde{\Psi}_{i,j,g} \right|.$$

2.4.2.2. Assignment of continuous controls. Since the change in the mth continuous parameter at the time t ($\tilde{C}_{i,t,m}$) is equal to the sum of changes of this parameter on all the transitions made by this time, we obtain from formula (2.7) that

$$S = \sum_{i=1}^{N} A_i \sum_{t=1}^{L_i} \left(C_{i,t,m} - \sum_{j=1}^{n} \alpha_{i,j}[t] u_j \right)^2.$$

Here, as before, u_j is the unknown change in the mth continuous parameter at the jth transition and $\alpha_{i,j}[t]$ is the number of executions of the jth transition of the FSM run on test i by the time t. The derivative of S with respect to u_h is

$$\frac{\partial S}{\partial u_h} = \sum_{i=1}^{N} A_i \sum_{t=1}^{L_i} 2\alpha_{i,h}[t] \left(\sum_{j=1}^{n} \alpha_{i,j}[t] u_j - C_{i,t,m} \right).$$

Equating all these derivatives to zero, we obtain the following system of n equations for each mth parameter:

$$\begin{cases} \sum_{j=1}^{n} \left(\sum_{i=1}^{N} A_i \sum_{t=1}^{L_i} \alpha_{i,1}[t] \alpha_{i,j}[t] \right) u_j = \sum_{i=1}^{N} A_i \sum_{t=1}^{L_i} C_{i,t,m} \alpha_{i,1}[t]; \\ \sum_{j=1}^{n} \left(\sum_{i=1}^{N} A_i \sum_{t=1}^{L_i} \alpha_{i,2}[t] \alpha_{i,j}[t] \right) u_j = \sum_{i=1}^{N} A_i \sum_{t=1}^{L_i} C_{i,t,m} \alpha_{i,2}[t]; \\ \ldots \\ \sum_{j=1}^{n} \left(\sum_{i=1}^{N} A_i \sum_{t=1}^{L_i} \alpha_{i,n}[t] \alpha_{i,j}[t] \right) u_j = \sum_{i=1}^{N} A_i \sum_{t=1}^{L_i} C_{i,t,m} \alpha_{i,n}[t]. \end{cases}$$

Each of these systems is solved by Gaussian elimination. The unknowns u_j found from this system indicate the change in the mth continuous parameter at the transition j. Note that the linearity of this system depends on the fitness function. First, we used the fitness function

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

which gave a system of nonlinear equations and considerably complicated the solution of the problem.

2.5. Theoretical Estimation of the Time Complexity of the Fitness Function Evaluation

In this section, we estimate the execution time of the control assignment algorithm for the case of several tests. Denote by L the sum $\sum_{i=1}^{N} L_i$. For the assignment of discrete controls, $O(n + L)$ operations for each k in the range from 1 to d must be performed. The total number of operations is $O(dn + dL) = O(n + L)$ if d is a constant. In the case of the assignment of continuous controls, the matrix on the left-hand side of the system has the size $n \times n$. In the case of the fitness function used in this paper, this system can be solved by Gaussian elimination, which requires $O(n^3)$ operations. Since the matrix of the system is the same for every k, all the systems can be solved using $O(n^2(n + c))$ operations. To form the matrix of the system, $O(n^2L)$ operations are needed. The formation of the kth column of free terms requires $O(nL)$ operations for k in the range from 1 to c. In total, $O(n(n + c)(n + L)) = O(n^3 + n^2L)$ operations are needed to assign the continuous controls if c is a constant. Ultimately, $O(n^3 + n^2L)$ operations are needed to assign all the controls.

By the definition of the big O notation, there exist constants α and β such that the assignment of all the controls at the framework transitions requires not more than $\alpha(n^3 + n^2L) + \beta$ simple arithmetic operations. This estimate is linear in the total length of the tests (which is proportional to the plant control time). To make this estimate acceptable for practical application, a relatively small n should be used. It is worth noting that the execution time of the evolutionary programming algorithm considerably increases with increasing n because the search space grows exponentially with n. In this paper, n did not exceed 130. The specific values of the constants α and β depend on the implementation of the fitness function evaluation algorithm (if this algorithm is thoroughly implemented, these coefficients are not large). In the approach proposed in [14], with which we compare the approach proposed in this paper, the execution time of the fitness function evaluation algorithm also linearly depends on the plant control time; however, the constants there are typically much larger because simulation is used.

2.6. Operators of the Evolutionary Programming Algorithm

In this section, we describe the selection, mutation, and crossover operators used in the proposed evolutionary programming algorithm.

2.6.1. Selection operator. This operator is used to find in the current generation the fittest individuals and add them to the intermediate generation. To compare individuals, a fitness function is used. This function assigns to each individual a number that characterizes how well this individual suits for solving the problem. The greater this number, the better is the individual. In this paper, we use tournament selection [9]. In this method, k individuals are randomly selected from the current generation. Then, the fittest individual among them is chosen and added to the intermediate generation. The tournament is held as many times as there are individuals in the generation.

2.6.2. Mutation operator. This operator executes the following operations with a prescribed probability: change in the initial state and addition, deletion, or change of a random transition in the FSM framework. By the change of transition, we mean the change of the state into which the transition is performed to a randomly selected state.

2.6.3. Crossover operator. Two individuals take part in the crossover operation, and this operation produces two new individuals. Denote the parent individuals by *P1* and *P2*, and the descendant individuals by *C1* and *C2*. For the initial states *C1.is* and *C2.is*, one of the following relations holds: *C1.is* = *P1.is* and *C2.is* = *P2.is* or *C1.is* = *P2.is* and *C2.is* = *P1.is*. Next, for each cell *t*[*i*][*j*] of the transition table, one of the

following values is selected with identical probability: $C1.t[i][j] = P1.t[i][j]$ and $C2.t[i][j] = P2.t[i][j]$ or $C1.t[i][j] = P2.t[i][j]$ and $C2.t[i][j] = P1.t[i][j]$.

2.7. The Proposed Method

Summarizing the foregoing, we formulate the proposed method for generating control FSMs using evolutionary programming based on training examples.

1. A plant is specified (for example, in the form of an emulator) with the known control interface and the ability of reading and writing the input and control parameters.

2. The control system has a hierarchical structure. Operational modes of this system are distinguished, and for each of them an FSM is constructed using the proposed method. The upper-level FSM, which is responsible for switching modes, can be constructed manually or generated using the method described in [14].

3. For each mode, the following actions are repeatedly performed:

3.1. Choice of parameters. Predicates of the input parameters are used as input controls for the FSM, and changes in the control parameters are used as its output controls. A tuple of numbers of the length equal to the number of control parameters is associated with each transition of the FSM.

3.2. Test creation.

3.2.1. Start the plant and register the parameters as often as possible. For different modes, different numbers of parameter sets are used as tests. The collection of these sets for each run is a test example. The sequences thus registered are called reference sequences.

3.2.2. Heuristic choice of the number of tests. As the number of tests increases, the quality of the result improves; however, the time needed to collect the data and the execution time of the evolutionary programming algorithm used to construct the FSM also increases.

3.3. The use of evolutionary programming.

3.3.1. An FSM framework—an FSM with a fixed number of states containing transitions associated with transition execution conditions and output controls (but the values of controls are undefined)—is used as an individual.

3.3.2. Choice of the fitness function. This function must reflect the degree of similarity of the reference (generated by a human) and generated by the FSM output controls. The form of this function affects the complexity of the algorithm because it solves systems of linear or nonlinear equations of which each corresponds to one continuous control parameter.

3.3.3. The set of predicates that form conditions for the transition execution is created heuristically.

3.3.4. For each predicate, a code that implements this predicate based on the input parameters is written in a programming language.

3.3.5. Creation of an operator that transforms the framework into an FSM that maximizes the fitness function over all possible FMSs corresponding to the framework.

3.3.5.1. Run the FSM framework on test examples and register the sequence of executed transitions.

3.3.5.2. For each step of the plant operation, determine the values of the control parameters in terms of the initial values and unknown changes at the transitions. For the continuous controls, this value is the sum of the value at the preceding step and the change at the current step; for the discrete controls, this is the value at the last executed transition.

3.3.5.3. Substitution of the expressions for the control parameters into the fitness function. Thus, the fitness function is parameterized by the values of the controls at the transitions of the framework.

3.3.5.4. Maximization of the fitness function. If the fitness function is appropriate, the expression for the controls depending on the discrete parameters can be obtained explicitly. The values of the continuous controls are determined by solving a system of equations. This system is obtained by equating the partial derivatives of the fitness function with respect to the unknown controls to zero. If the fitness function is appropriate, then this system is linear. Thus, the fitness function should be "appropriate" both for the discrete and continuous parameters. Function (2.1) satisfies this condition.

3.3.6. Choice of the model for the evolutionary programming algorithm and the choice of its characteristics (the number of individuals in a generation, the size of elite group, selection and crossover functions, the step size, and so on).

3.3.7. Run of the algorithm.

3.4. Run of the resultant FSMs and estimation of their behavior.

Fig. 4. Snapshot of the FlightGear simulator display at the time when the acceleration starts.

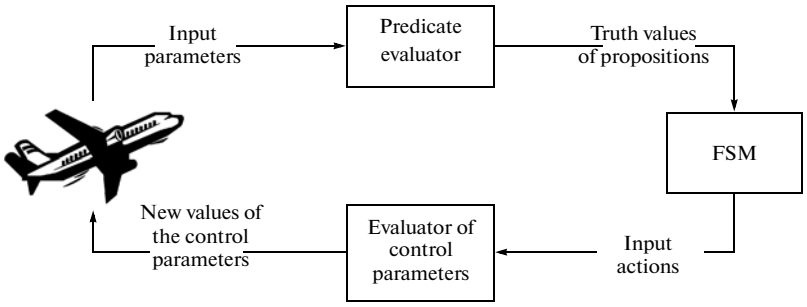

Fig. 5. Aircraft control using the FSM.

3. TESTING THE METHOD EFFICIENCY

To check the efficiency of the proposed method, we generated an FSM for controlling an aircraft executing a loop-the-loop maneuver.

3.1. Interaction of the Generated FSM with the model of an UAV

There are simulators that can model aircraft flight. We chose the open-source cross-platform simulator *FlightGear* (http://www.flightgear.org), which allows one to steer a model aircraft manually or in automatic mode. Figure 4 shows a snapshot of the FlightGear display at the time when the acceleration starts.

The simulator makes it possible to register the flight parameters (velocity, flight direction, and so on) and the aircraft parameters (the position of the rudder, ailerons, state of the starter, and so on). The flight parameters are the input parameters for the control system, and the aircraft parameters are the controlling parameters because their changes help steer the aircraft. The values of the controlling parameters are formed by the FSM (see Fig. 5).

3.2. Generation of an FSM for the Loop-the-Loop Execution

This problem can be formulated as follows: design an FSM under whose control a model aircraft executes a loop-the-loop maneuver and then flies evenly for several seconds.

3.2.1. Use of the proposed method. We briefly consider the main steps of the application of the proposed method for this example: we formed the necessary and sufficient set of propositions concerning the aircraft state; designed three collections of tests that were considered independently of each other (each collection consisted of ten tests of which each included several thousands of sets of input and output control parameters, and the parameters were registered ten times per second); the evolutionary programming algorithm was run for each of the three collections of tests and each set of the algorithm parameters.

Here are the parameters of the algorithm and their values: the generation size was 100 individuals, the number of FSM states was in the range from two to five, the probability of mutation was 0.5, the tournament selection was used, the elite group consisted of two individuals, and the step time length was 0.1 s.

Computations were performed on one core of a computer with an Intel Core 2 Duo T7250 2 GHz processor under Microsoft Windows XP. Programs were written in Java.

On the average, the algorithm execution time was ten hours for one set of parameters and one collection of tests. About 2000 generations were produced. Therefore, the creation and processing of one generation took 20 s on the average or 0.2 s per one FSM, which is much less than in [14], where the creation of one generation took about 5 min.

The propositions were as follows: x_0 stands for x_1 engine is on;x_1 stands for the acceleration of the change in the aircraft heading is greater than zero; x_2 stands for the change in the aircraft heading is greater than zero; x_3 stands for the deviation from the initial heading is less than one degree; x_4 stands for the deviation from the initial heading is greater than zero (deviation to the left); x_5 stands for the acceleration of the roll change is greater than zero; x_6 stands for x_7 velocity of the roll change is greater than zero; x_7 stands for the roll is small (less than 1°); x_8 stands for the roll is greater than zero (the aircraft tilts to the right); x_9 stands for the acceleration of change in the vertical aircraft velocity is greater than zero; x_{10} stands for the velocity of change in the vertical aircraft velocity is greater than zero; x_{11} stands for the vertical velocity is low (less than 0.1 m/s); and x_{12} stands for the vertical aircraft velocity is greater than zero (the aircraft is ascending).

The controlling devices are magneto, starter, throttle, ailerons, elevation rudder, and rudder. The two first devices are discrete, and the others are continuous. For example, the control *switch on the starter* is discrete and turn the rudder to the right by 0.5° is continuous. Experiments with the evolutionary programming algorithm showed that FSMs with three or four states have fairly good behavior, while the structure of FSMs with a greater number of states is more complex and their behavior deteriorates.

3.2.2. Registered tests. Figures 6a–6f show 12 frames from the video of a training example for a loop-the-loop maneuver.

Notice that this is the classical loop-the-loop maneuver, but, due to the fact that the camera that registers the maneuver changes its position, it seems that another (more complicated) maneuver is executed.

Here are two references to video of maneuvers performed by a human operator. The first of them is a successful loop-the-loop maneuver (http://www.youtube.com/watch?v=G5Kcx0ohpNo); this flight was included in the training set. The second video shows the unsuccessful maneuver (http://www.youtube.com/watch?v=OGVTch-a97A); this flight was not included in the training set.

3.2.3. Results. The evolutionary programming algorithm was run about 50 times, and in each of them the FSM with the best value of the fitness function was selected. The runs differed in the parameters and collections of tests. The flights of models controlled by the selected FSMs were visually examined, and the FSM that gave the flight most close to the "perfect loop-the-loop maneuver" was regarded to be the best one. This FSM has four states and 68 transitions (see table). It is worth noting that the perfect loop-the-loop maneuver can be different from the reference maneuver executed manually.

Examination of the execution of the loop-the-loop maneuver controlled by the best FSM showed that three versions of the execution are possible depending on the parameters of the environment and the aircraft.

1. In most cases, the aircraft model executes one loop-the-loop maneuver, as under manual control, and flies further.

2. Sometimes, the aircraft model executes several loop-the-loop maneuvers with a certain interval. This happens when the values of the aircraft model at the end of the maneuver are similar to those at its beginning. In this case, the FSM is at the same state after the maneuver as it was at its beginning, and the model behavior goes in cycles. In our experiments, we observed two loops in succession, but no longer sequences of loop-the-loops were observed.

3. The model can fail to execute the loop-the-loop maneuver because the FSM is unable to meet the challenge; however, this is a rare case.

The determination of conditions under which each of these cases is realized is a subject to further investigation.

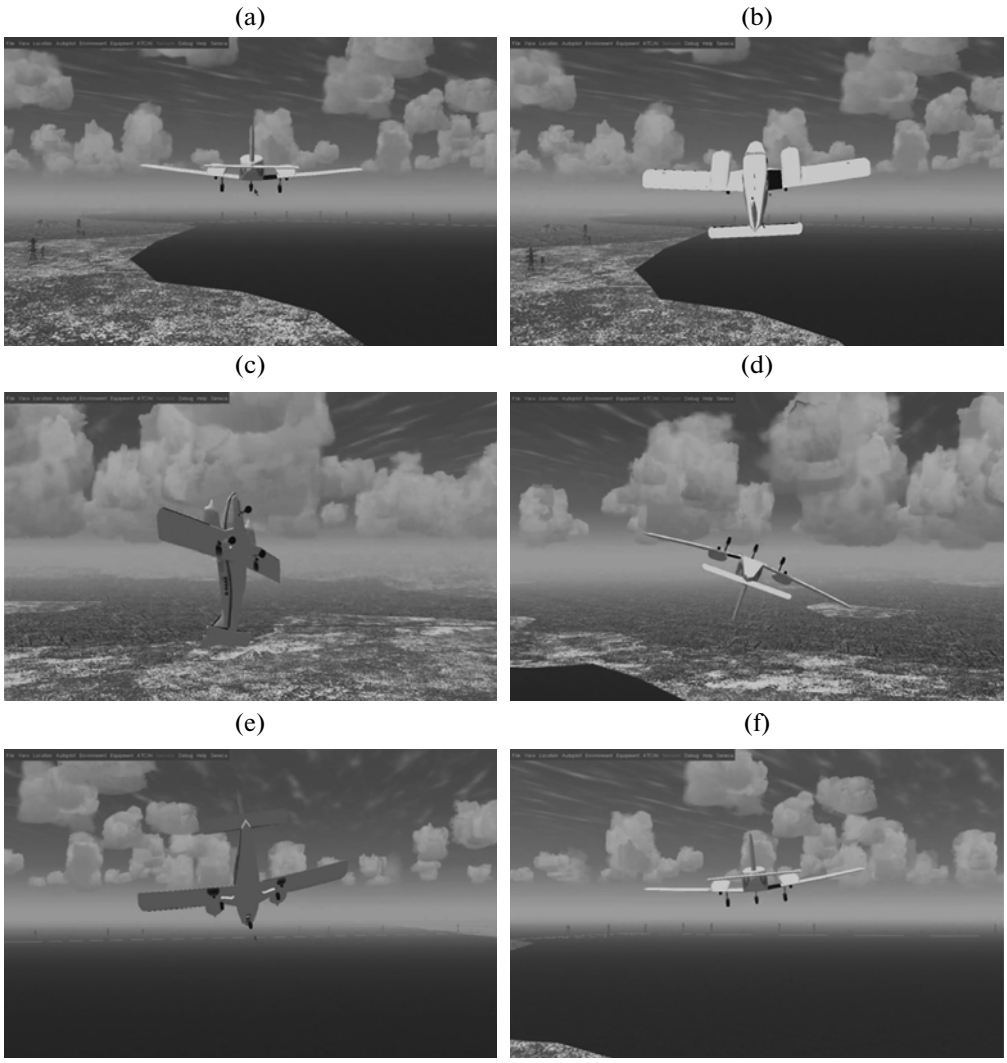

Fig. 6. Frames from the video of a training example of executing the loop-the-loop maneuver.

3.3. Example of Flight under the Control of the Best FSM

Here are references to three realizations of the loop-the-loop maneuver under the control of the best FSM: execution that is close to the perfect one (http://www.youtube.com/watch?v=TzrLoJjjVTA); execution with tilting to the left and subsequent leveling (http://www.youtube.com/watch?v=C6WV7x2bqE8); and sequential execution of two loop-the-loops (http://www.youtube.com/watch?v=yFiG4yz67Ks).

Figures 7a–7f show eleven frames of the first of these videos.

The Table describes the transitions of the best FSM for executing the loop-the-loop maneuver (the number of states is four, the initial state is zero, and the number of arcs is 68).

We emphasize that the complete transition table is used in the proposed method. For each state, $2m$ transitions can be defined, where m is the number of predicates. In the example under consideration, $m = 13$. However, none of the fours states is characterized by all of the predicates. If there is no certain transition in the table, then the FSM retains its state and parameters of the control devices at this transition.

Before the loop-the-loop maneuver is started, the engine is on. For that reason, the starter is not used in the course of its execution—the column *starter* contains only zeros. This device was included in the FSM generation because additional tests can be used in which the starter must be switched (for example, if the engine fails). The last column also contains only zeros, but the corresponding device can be used in other tests.

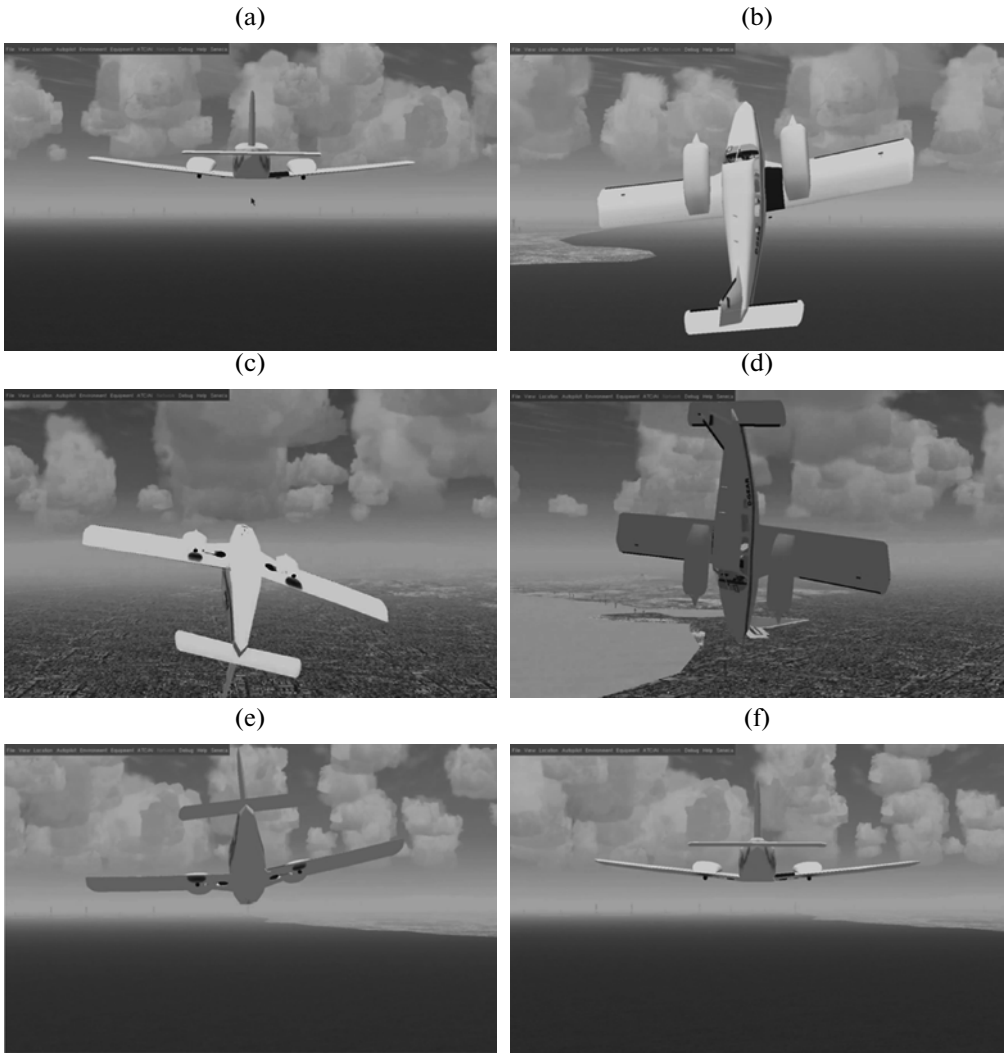

Fig. 7. Frames from the video of an aircraft executing a loop-the-loop maneuver controlled by the best FSM.

CONCLUSIONS

An evolutionary programming method based on training examples for the generation of FSMs controlling an object with complex behavior is developed. The method is experimentally investigated, and it is shown that the automatically generated FSM ensures better control than the manual control. In most cases, the automatically generated FSM correctly accomplishes the task. It is shown that the proposed method makes it possible to estimate the produced FSMs much faster than its prototype described in [14]. The proposed method was used to control a model of an UAV executing a loop-the-loop maneuver. In addition to high performance of the method, it has one more advantage compared with the case when the fitness function is evaluated by simulation—the method proposed in this paper does not require this function to be adapted to the control mode or to another object.

ACKNOWLEDGMENTS

This work was supported by the federal targeted program *Research and Academic Teaching Staff in Innovative Russia* for 2009–2013.

REFERENCES

1. N. I. Polikarpova and A. A. Shalyto, *Automata-Based Programming* (Piter, St. Petersburg, 1991) [in Russian], http://is.ifmo.ru/books/_book.pdf

2. A. A. Shalyto, *SWITCH-Technology. Algorithmization and Programming Logic Control Systems* (Nauka, St. Petersburg, 1998) [in Russian].
3. P. Angeline and J. Pollack, "Evolutionary module acquisition," in *Proc. of the Second Annual Conf. on Evolutionary Programming* (MIT Press, Cambridge, 1993), pp. 154–163. http://www.demo.cs.brandeis.edu/papers/ep93.pdf
4. D. Jefferson, R. Collins, C. Cooper, et al., "The genesys system: Evolution as a theme in artificial life," in *Proc. of the Second Conf. on Artificial Life* (Addison–Wesley, 1992), pp. 549–578. www.cs.ucla.edu/~dyer/Papers/AlifeTracker/Alife91Jefferson.html
5. F. N. Tsarev and A. A. Shalyto, "The use of genetic programming for generating a finite state machine in the smart ant problem," in *Proc. of the 4th Int. Conf. on Integrated Models and Soft Computations in Artificial Intelligence* (Fizmatlit, Moscow, 2007), pp. 50–597. http://is.ifmo.ru/genalg/_ant_ga.pdf
6. "Genetic programming technology for the generation of finite state machines controlling systems with complex behavior," *Intermediate Report on Experimental Studies, St. Petersburg State University of Information Technologies, Mechanics, and Optics, St Petersburg, 2008.* http://is.ifmo.ru/genalg/_2007_03_report-genetic.pdf.
7. D. A. Parashchenko, F. N. Tsarev, and A. A. Shalyto, "A technology for simulating a class of multiagent systems based on automata-based programming using the game of competing flying saucers as an example," *Project Documents, St. Petersburg State University of Information Technologies, Mechanics, and Optics, St Petersburg, 2006.* http://is.ifmo.ru/unimod-projects/plates
8. A. Coates, P. Abbeel, and A. Y. Ng, "Learning for control from multiple demonstrations," in *Proc. 25th Int. Conf. on Machine Learning, Helsinki, 2008*, pp. 144–151.
9. L. A. Gladkov, V. V. Kureichik, and V. M. Kureichik, *Genetic Algorithms* (Fizmatlit, Moscow, 2006) [in Russian].
10. S. Russell and P. Norvig, *Artificial Intelligence: A Modern Approach* (Prentice Hall, Upper Saddle River, N.J., 2003; Vil'yams, Moscow, 2003).
11. J. R. Koza, *Genetic Programming: On the Programming of Computers by Means of Natural Selection* (MIT Press, Cambridge, (1992).
12. V. M. Kureichik, "Genetic algorithms: State of the art, problems, and perspectives," J. Comput. Syst. Sci. Int. **38**, 137–152 (1999).
13. V. M. Kureichik and S. I. Rodzin, "Evolutionary algorithms: Genetic programming," J. Comput. Syst. Sci. Int. **41**, 123–132 (2002).
14. 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**, 265–282 (2010).
15. F. N. Tsarev, "A method for constructing control finite state machines based on test examples using genetic programming," Inform.-Upravl. Sist., No. 5, 31–36 (2010).

Translated by A. Klimontovich