Automatic system identification based on coevolution of models and tests

Sylvain Koos, Jean-Baptiste Mouret and Stéphane Doncieux

Abstract— **In evolutionary robotics, controllers are often designed in simulation, then transferred onto the real system. Nevertheless, when no accurate model is available, controller transfer from simulation to reality means potential performance loss. It is the reality gap problem. Unmanned aerial vehicles are typical systems where it may arise. Their locomotion dynamics may be hard to model because of a limited knowledge about the underlying physics. Moreover, a batch identification approach is difficult to use due to costly and time consuming experiments.**

An automatic identification method is then needed that builds a relevant local model of the system concerning a target issue. This paper deals with such an approach that is based on coevolution of models and tests. It aims at improving both modeling and control of a given system with a limited number of manipulations carried out on it. Experiments conducted with a simulated quadrotor helicopter show promising initial results about test learning and control improvement.

I. Introduction

In evolutionary robotics ([1], [2]), evolutionary algorithms are used to design controllers or robots' morphologies: a selective pressure based on the description of the expected behavior leads the search to a solution or a set of solutions. Such an approach relies on the evaluation of the different solutions on a set of performance objectives. But an experimental setup means in particular that the robot can easily be set to a given state. The system can also show risky behaviors when exploring the solution space. To avoid these problems and, moreover, to speed up the evaluation of the solutions, this evaluation step is often performed within a simulator instead of the real system (see [3] for discussion).

Once the algorithm has converged, it may be transferred onto the real system. But, because of the discrepancies between the simulator and the real system, this step often means performance loss or even total loss of the previously selected behavior. An accurate simulation model of the system is therefore needed for a good transfer. This transfer problem, also called reality gap [4], becomes a critical issue for systems that can't be easily accurately modeled like bird-sized unmanned aerial vehicles (UAVs) and small helicopters. Because of their sizes and their speeds, these flying robots bring into play little-known dynamics with low/medium Reynolds numbers ([5]).

In order to get a good simulation model of the system, one can resort to learning its dynamics with a system identification method (see e.g. [6]). Such an approach consists in performing some experiments on the system to generate data from which a model is learned. Once an accurate model is found, it can be used as a simulator of the system, to design an optimal controller for instance. Nevertheless, a batch identification approach assumes a convenient mean to generate data on the system. For physical systems like UAVs, this assumption requires costly and time consuming experimental setup (pilot, wind tunnel,...). We want then to reduce the number of experiments performed on the system.

To this end, the *active learning* approach (see [7], [8], [9]) provides an alternative to batch identification: the data generation becomes part of the identification process in order to perform only meaningful experiments concerning the target issue (trajectory tracking, forward locomotion, ...). One promising way in active learning is *coevolution*, a method that relies on evolutionary concepts.

Bongard et al. introduces in [10] an estimation-exploration algorithm that implements an identification method based on coevolution of models and tests. The models are evaluated from previously generated test data and the test (a controller here) that discriminates at most between these simulators is transferred on the system to generate new meaningful identification data. This process is iterated until a good simulator is found. Such a method allows to reduce the number of tests required to explore efficiently the state space.

In the same vein, de Jong et al. addresses in [11] theorical issues about a coevolution process with multiobjective evaluation called Pareto-Coevolution. It brings into play learners and tests. A learner is evaluated by interaction terms, reflecting each its quality on a given test. In such a context, the existence of a minimal set of tests that leads to optimal learners is proved.

In Bongard's approach, the models are only evaluated from a global test dataset by comparing real data generated from start with predicted ones. But to obtain simulators that are relevant for the control issue, it seems more intuitive to evaluate them directly by some interactions with test controllers, then in a Pareto-Coevolution context.

Our approach is an implementation of the estimation-exploration algorithm [10] based on Pareto-Coevolution concepts [11] to improve in a joint process both modeling and control of a robotic system with few experiments carried out on it. The problem includes then two main goals:

- to obtain a good simulator of the system by local identification of its dynamics;
- to find a relevant controller for a given task/problem.

Two populations are brought into play: models and tests. Tests are made of a controller, an initial state and a target

Sylvain Koos, Jean-Baptiste Mouret and Stéphane Doncieux are with the Université Pierre et Marie Curis (UPMC) - Paris 06, Institut des Systèmes Intelligents et de Robotique (ISIR), CNRS UMR 7222, F-75005, Paris, France; e-mail: {koos,mouret,doncieux}@isir.fr. This work is part of the ROBUR project.

state. If a test is selected the corresponding trajectory of the real system is generated. Models are learned from trajectories' data. As in Pareto-Coevolution, each model is evaluated by some interactions, also called performance objectives, one by test trajectory. The control task is then directly taken into account for model evaluation. Tests are selected according to some objectives computed with the set of the best models like discrimination power as in [10] and, optionally, control quality concerning the defined issue.

An application to the control of a simulated quadrotor helicopter based on [12] has then been performed in order to evaluate our approach. A physical simulator of quadrotor dynamics plays the role of the "real" system. We want then to find good models of this simulator according to the selected tests, which have to be relevant on the specified control task.

After a review of some related works in the next section, section III deals with the coevolution process and the different elements which are brought into play: the models and the tests. The implementation of the algorithm is detailed in section IV, in particular the evaluation objectives for each kind of elements. Next, section V details the application to the quadrotor's control. Section VI shows the main results obtained with this application that are discussed in section VII. The paper ends with some conclusions in section VIII.

II. Previous work

System identification is a highly multidisciplinary issue as it's resorted to when modeling any partially-known or unknown system. Whatever the system is, the approach is similar: data are generated and a model is learned. Nevertheless, the data generation can raise some problems with systems like UAVs, for which each experiment means costly experimental setup.

In [6] and [13], a Markov decision process model of an helicopter is learned with data collected from a trajectory initially performed by a pilot. The data are then generated before the identification task. A controller is next designed from the obtained model used as a simulator before it is transferred onto the UAV. The approach shows good results, but such a batch approach is not suitable for more open behaviors because it requires to collect all experimental data about the given problem beforehand.

In order to limit the number of experiments, the modeling process has to be led to meaningful parts of the state space of the system concerning the addressed issue. To this end, the data generation can be included in the identification process with an *active learning* approach [7], [8], [9]: from a given target (trajectory, behavior,...) the system explores effectively its environment to generate pertinent identification data. The data generation becomes then an integral part of the identification process. Among active learning methods, *coevolution* is a promising approach based on evolutionary concepts. Coevolution designates the joint evolution of two (or more) populations affected by their interactions.

Bongard et al. introduce in [10] an estimation-exploration algorithm that implements a system identification method based on coevolution of models and tests: the models try to model at most the system from previously generated data while the tests try to create new pedagogical data by discriminating at most between the models. For their robotic applications, a test means a controller designed from the learned model for a specified task. The method has been successfully applied to forward locomotion for a simulated quadrupedal robot. Models are sets of parameters and control is provided by neural networks.

The estimation-exploration algorithm has been later implemented on a real four-legged machine [14]: the robot infers its own structure by performing actions that discriminate at most between the built models. The models are still sets of parameters. From these models the machine can generate forward locomotion. Besides, if a leg is removed, the coevolution process makes models adapt to this new structure and new locomotion's behaviors can then be generated again. In contrary to the previous approach, the tests are only actions and the controller search is performed after the identification task.

Such a method leads to two main advantages. First, as an experiment on the real system is only conducted when a pertinent test is selected, the number of experiments is drastically reduced in comparison with a batch identification approach. The modeling task can secondly be led to meaningful parts of the state space by the evaluation process of the tests, which affects the data generation. We can then obtain a local model good enough to design relevant controllers for a given control task. We can also explicitly improve the models towards this specific task. The evaluation process can bring into play interactions with test controllers to quantify the model's proximity with the physical system concerning the control task.

This is the context of Pareto-Coevolution, introduced by de Jong et al. in [11]. It consists in finding a relevant minimal set of tests to improve the quality of a learner population concerning some underlying objectives of a given problem. As these objectives are often unknown, the learners are evaluated by a set of interactions with tests. Each interaction evaluates the performance of one learner on a given test: each test gives rise to an objective in a multiobjective meaning. In this Pareto-Coevolution context, it is then proved that a minimal set of tests that distinguishes at most between the models' interaction values is an ideal evaluation set and leads the learners to a maximization of the underlying objectives.

There is nevertheless a difference concerning the implementation of the coevolution process between Pareto-Coevolution and estimation-exploration's approach. Coevolution is a priori simultaneous in Pareto-Coevolution context, that is, both populations are evaluated in the same time. In [10], this simultaneity aspect is replaced by a more sequential coevolution in two main phases. Our approach, detailed in the next section, is based on this sequential process that we call *double evolutionary loop*.

III. Approach

The coevolution process relies on two populations: model and test populations. We first describe what is meant by test

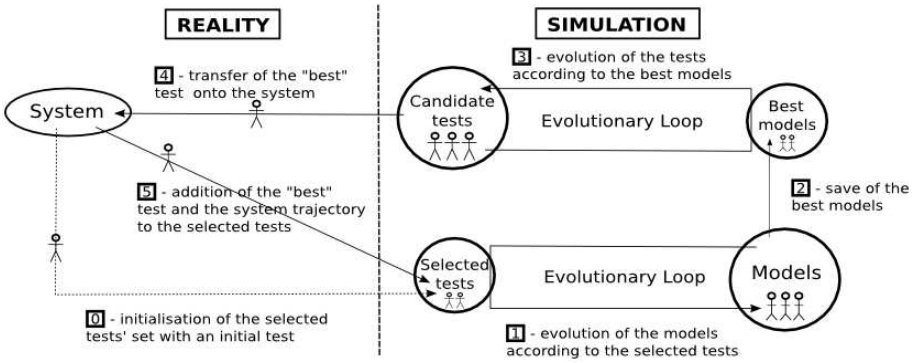

Fig. 1. Outline of the whole algorithm - A test is made of a controller, an initial state and a target state - Besides, a selected test also contains the corresponding trajectory of the real system.

before detailing the modeling issue. After the description of these elements, our approach is introduced. An outline of the whole method is shown on Figure 1.

Notations: at time t, the state of the system is called η_t and ω_t designates a vector of motor commands.

A. Test description

A candidate test Υ is made up of a controller, an initial state, a target state. A controller $\mathfrak{C}$ is a function which returns motor commands ω_t from the current state η_t and the target state of the system η_d:

- $\omega_t = \mathfrak{C}(\eta_t,\ \eta_d)$

A controller $\mathfrak{C}$ can be evaluated on the real system as follows: with a given initial state η_1 and a given target state η_d, the command returned by $\mathfrak{C}$ is applied on the system at each time step and the successive states of the system are recorded. The corresponding trajectory T is then summed up by a dataset:

- $T = \{(\omega_t,\ \eta_t,\ \eta_{t+1}), t \in [0,\ n]\}$, where n is the length (i.e. the number of steps) of the control sequence

If a candidate test is selected, the controller is transferred onto the real system, the corresponding trajectory is generated and the data are linked to the selected test structure. A whole test is thus generated by each experiment performed on the real system.

For simplicity, we assume in this paper that a candidate test can be summed up by N parameters: proportional-integral-derivative (PID) controller's gains, neural network's weights, initial state, target state, etc. Three evolutionary operators can then be defined on this vector representation:

- a *mutation operator* modifies the value of one parameter;
- a *crossover operator* builds a new candidate test by mixing two parameter vectors of the current population;
- a *reproduction operator* copies a candidate test of the current population to the next one without change.

B. Model description

A model $\mathfrak{M}$ is a function that predicts $\hat{\eta}_{t+1}$, the next state of the system, from its current state η_t and the applied commands ω_t:

- $\hat{\eta}_{t+1} = \mathfrak{M}(\omega_t,\ \eta_t)$

The models are evaluated on the selected test trajectories. The model structure doesn't take any prior information on the system into account. Moreover we assume that the model structure allows to define three operators: mutation, crossover and reproduction. Nevertheless, as the definition of these operators is highly dependent on the model structure, we'll only detail them in the application description (see V-A).

C. A double evolutionary loop

In order to initialize the algorithm, we assume that an initial test Υ_0 is available, that is a controller $\mathfrak{C}_0$, an initial state η_1^0, a target state η_d^0 and the corresponding trajectory T_0 of the real system. We define then a set of selected tests Ω. At the initialization, $\Omega = \{\Upsilon_0\}$.

The models are learned from test data and evaluated by comparing their prediction to each selected test trajectory. The test management is a little bit more complicated. The candidate tests are ranked thanks to a set of objectives concerning the models. As in [10], candidate tests that discriminate at most between the models are selected. But test evaluation can include other objectives too like control performance in order to lead the search to more efficient controllers and then the identification to more meaningful state space areas concerning the control issue. The influence of such a quality objective will be discussed later.

The coevolution process, or *double evolutionary loop*, is implemented by the sequence below and iterated as long as no good pair (model, controller) is found:

1) *model loop*: the best models are looked for from data of the selected tests' set Ω (step 1 in Figure 1), next saved (step 2);
2) *test loop*: candidate tests are evolved from the previously found best models (step 3);
3) *data generation*: one among the more relevant candidate tests is transferred onto the real system (step 4) and added with the generated data to Ω (step 5).

Both loops are evolutionary algorithms and bring into play the three operators presented above. There is a question left: considering one model and a set of selected tests, how can this model pertinently be evaluated? Likewise, how can we

select the best candidate test to transfer onto the system? The next section details the evaluation and selection processes for both populations.

IV. Evaluation process

As models and tests are evaluated on a set of objectives, each evolutionary loop boils down to a multiobjective search of optima in a given objective space. In this section, some concepts and tools concerning multiobjective problems are introduced first in order to describe next the evaluation objectives of each population.

A. Multiobjective concepts & NSGA-II ranking method

Let i be an individual evaluated by the objectives obj_j^i to minimize, $j \in [1,\ N]$. The dominance relation between the individuals is a partial order relation defined as follows:

- i_1 dominates $i_2 \iff \begin{cases} \forall j \in [1,\ N],\ obj_j^{i_1} \leq obj_j^{i_2} \\ \exists j \in [1,\ N],\ obj_j^{i_1} < obj_j^{i_2} \end{cases}$

For a given problem, the set of the non-dominated individuals, i.e. the best solutions, is called *Pareto front*, noted $\mathcal{P}$. On a given population, a non-dominated set $\mathcal{S}$ can be defined that contains its non-dominated individuals.

An *ideal point* z^* [15] can be defined on a given set of objectives. Its coordinates z_j^* are the minimal values for each objective on the Pareto front $\mathcal{P}$:

- $z_j^* = min_i\ (obj_j^i),\ j \in [1,\ N],\ i \in \mathcal{P}$

As the non-dominated set $\mathcal{S}$ of a population is an approximation of $\mathcal{P}$, an *approximate ideal point* $\hat{z}^*$ can be defined too:

- $\hat{z}_j^* = min_i\ (obj_j^i),\ j \in [1,\ N],\ i \in \mathcal{S}$

From these definitions, a multiobjective evolutionary algorithm addresses two issues:

- to make $\mathcal{S}$ converge to the Pareto front $\mathcal{P}$;
- to maintain a good spread of solutions in $\mathcal{S}$.

Among the recent multiobjective evolutionary algorithms, NSGA-II shows good performance on both issues with an reasonable computational cost (see [16] for discussion). The algorithm we used in model and test loops is based on the NSGA-II ranking method[1].

The NSGA-II ranking method [17] is based on the dominance relation between individuals. For a given population, the individuals of the non-dominated set $\mathcal{S}$ get 1 as rank value. If these individuals are temporarily removed from the population, we can define a new non-dominated set on it. The corresponding individuals get 2 as rank value and so forth.

Such a ranking method doesn't explicitly maintain diversity among the non-dominated sets. Deb et al. suggest the use of a crowding-distance term which favours individuals in lesser crowded regions of the objective space [17]. Moreover, individuals located on the edges of the non-dominated set are given an infinite crowding distance and thus systematically selected.

[1]The selection method is not the same as in [17], but as our approach is a priori not dependent on this implementation choice, it shouldn't have a strong influence on the results

Both evolutionary loops are designed as follows: three steps are iterated for a certain number of generations.

1) construction of the offspring of the current population based on elitism and ranking selection with the three operators: mutation, crossover, reproduction (for details, see parts III-A and III-B);
2) evaluation of the new individuals by the corresponding set of objectives;
3) sorting of the new population according to the NSGA-II ranking method and addition of the non-dominated individuals to the non-dominated set $\mathcal{S}$.[2]

At the end of each loop, the approximate ideal point $\hat{z}^*$ is constructed from the found non-dominated set. For each non-dominated set, we define then a best trade-off for our specific issue by choosing the nearest individual to $\hat{z}^*$ in $\mathcal{S}$ according to an Euclidean distance.

From this shared loop structure, the two populations differ by their evaluation objectives, that are described in the next parts.

B. Models' evaluation objectives

As described in part III-B, the models are learned from the selected test trajectories. An objective can then be defined that reflects the prediction performance of a given model against the data of a trajectory (see Figure 2). We call it *open-loop objective* and its value is the Mean Squared Error (MSE) of the model prediction on the whole trajectory.

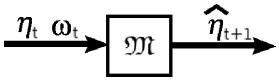

Fig. 2. Outline of the open-loop process.

Nevertheless, the simulation model and the system have to show similarities that concern the control issue too. We introduce therefore a *closed-loop objective*, computed in a closed-loop process as schematized in Figure 3. With several elements of a given test - a controller $\mathfrak{C}$, an initial state η_1, a target state η_d - and a given model $\mathfrak{M}$, the two calculations below are iterated for $t \in [1,\ n]$:

1) $\omega_t = \mathfrak{C}(\eta_t,\ \eta_d)$;
2) $\hat{\eta}_{t+1} = \mathfrak{M}(\omega_t,\ \eta_t)$.

The trajectory T' obtained in this way can be compared with the true trajectory T of the test: the MSE between the predicted states $\hat{\eta}_{t+1}$ of T and η_{t+1} of T' is the value of the closed-loop objective.

These two objectives seem relatively close, but some preliminary results have shown that they are not redundant. As the closed-loop objective is computed from a temporal sequence of system states, it accumulates the model's errors while the open-loop one doesn't. Let N be the number of tests at the beginning of the loop, a model is then evaluated by $2N$ objectives.

[2]An individual that becomes dominated by another one is, of course, removed from $\mathcal{S}$.

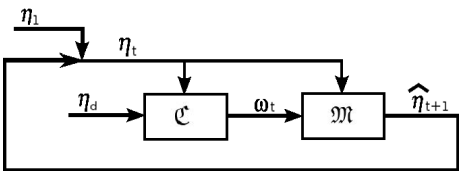

Fig. 3. Outline of the closed-loop process.

As the model Pareto front $\mathcal{P}^i_{\mathfrak{M}}$ at the i^{th} iteration is contained in $\mathcal{P}^{i+1}_{\mathfrak{M}}$ [3], the model population is randomly generated only once at the beginning of the algorithm. It allows to extend the non-dominated set found at i in the additional dimensions.

Notations: the model non-dominated set is called $\mathcal{S}_{\mathfrak{M}}$ and the corresponding best trade-off, $\mathfrak{M}^*$.

C. Tests' evaluation objectives

The test loop takes place like the model one. It boils down to the choice of a controller, an initial state and a target state to generate a new trajectory of the real system, i.e. new pertinent data. To that end, we want to select a new candidate test that discriminates at most between the individuals of the model non-dominated set $\mathcal{S}_{\mathfrak{M}}$ [10].

Let $\mathfrak{M}$ be a model and Υ a candidate test. The corresponding simulated trajectory can be built according to the previously described closed-loop process. Given the control task, each trajectory can be summed up by the distance $\delta^{\Upsilon}_{\mathfrak{M}}$ between the final state reached by the system η_n and the target state η_d of the test:

- $\delta^{\Upsilon}_{\mathfrak{M}} = \parallel \eta_n - \eta_d \parallel$

The first objective we introduce is the discrimination power of a test [10]. A test Υ is even more discriminating as the variance of the $\delta^{\Upsilon}_{\mathfrak{M}}$ values is large on the set $\mathcal{P}_{\mathfrak{M}}$. The objective value to minimize is then the inverse of this variance.

But, with this objective only, candidate tests whose trajectories are near instabilities of the system are preferred [18]. The neighborhoods of the instabilities are indeed parts of the state space where two models can easily be discriminated, even if they are very close. Moreover, it is difficult to model the system's behavior near an instability. To avoid such modeling problems, we assume that the relevant behaviors we look for stay away from instabilities. A second objective is then needed to take into account this assumption.

Consider a model of the system, the best trade-off $\mathfrak{M}^*$ for instance. Consider the candidate test generated from the triplet $\Upsilon = (\mathfrak{C},\ \eta_1,\ \eta_d)$. A corresponding trajectory T can be built with $\mathfrak{M}^*$. Consider yet the candidate test generated from $\Upsilon' = (\mathfrak{C}',\ \eta_1,\ \eta_d)$, where $\mathfrak{C}'$ is a slightly mutated version of $\mathfrak{C}$. The corresponding trajectory T' built with $\mathfrak{M}^*$ is expected to be similar to T. If it's not, there is an instability nearby.

The second objective to minimize is then the variance of the distances $\delta^{\Upsilon'}_{\mathfrak{M}^*}$ values computed each with a mutant $\mathfrak{C}'$ of the evaluated controller $\mathfrak{C}$. This value is even smaller as the corresponding trajectory stays away from the instabilites of the system.

A third objective can be added: the control quality of the test. As we want to obtain a pertinent controller for the system, the use of such an objective is indeed attractive. The term $\delta^{\Upsilon}_{\mathfrak{M}}$ defined above can be used to quantify this objective to minimize. Its usefulness will be discussed later.

By choice, the controller population is randomly generated at each beginning of the test loop. The addition of a selected test between the iterations i and i+1 can indeed modify a lot the computation of the discrimination power once this new test is learned. Nevertheless, as the multiobjective algorithm used here doesn't resort to archives, we could also use the population of the previous test loop as initial population.

Notations: the test non-dominated set is called $\mathcal{S}_{\Upsilon}$ and the corresponding best trade-off $\Upsilon^* = (\mathfrak{C}^*, \eta^*_1, \eta^*_d)$.

V. Application to the quadrotor's control

UAVs' study is currently an important robotic field because of their various applications. Moreover, UAVs are systems that can hardly be accurately modeled and experiments on it means costly facilities like a wind tunnel. Batch identification methods are then difficult to use.

We have envisaged an application to quadrotor control from Adigbli et al. [12], which designed controllers tested with a physical model of the UAV. To validate our approach in a relatively simple context, the "real" system we want to model and control is this physical model. We have then two main points to check:

1) the best models we obtained are good simulators of the "real" system in a neighborhood of the test trajectories;
2) the controllers of the selected tests are relevant concerning the given control task.

We use Linear Genetic Programming as learning method and detail it in the first part. The section deals then with the physical model and the used controller.

A. A learning method: LGP

A system identification approach goes always with a learning method. In evolutionary context, it is often resorted to Genetic Algorithms for parametric identification ([19], [20]) and (often tree-based) Genetic Programming (GP) for symbolic/non-parametric identification ([21], [22]).

Here, we are only interested in non-parametric identification. We have selected a page-based Linear Genetic Programming (LGP) [23], that is a linear version of GP. Previous comparisons between tree-based GP and LGP on various problems (see [24]) show indeed very promising results with LGP and LGP's implementation is simple.

```
  01. X2=X1*X1
x 02. X1=X1*3.5
  03. X2=1.1+X1
x 04. X1=2.1*18.2
  05. X3=X2+X2
```

Fig. 4. Example of a LGP model - (X1, X2) in input, X3 as output - marked instructions are introns.

[3] A non-dominated individual remains non-dominated when a new objective is added.

A model is then a sequence of instructions as pictured in Figure 4, i.e. elementary calculations (addition, subtraction, multiplication, division) involving the inputs and constants. In such a program, the same symbol can represent the corresponding input or a temporary vector. Each model contains 5 pages and each page is made up of 10 instructions. LGP brings into play three operators:

- a *mutation operator* modifies single instructions of a model;
- a *page-based crossover* swaps single pages between two models [25];
- a *reproduction operator* copies a model of the current population to the next one without change.

As in tree-based GP methods, an individual contains introns, instructions that don't affect its prediction whatever the inputs. In LGP, the intron sequences are yet easier to detect and to remove temporarily. It allows then to accelerate significantly the evaluation step with a low additional computational cost.

B. Physical model of the quadrotor - Backstepping controller

A quadrotor helicopter is an UAV that is propelled by four rotors as schematized in Figure 5. In this part, we only address the issue of modeling and controlling the quadrotor's attitude[4], that is the angles Φ, Θ and Ψ. The state vector η_t is thus :

- $\eta_t = (\Phi_t,\ \Theta_t,\ \Psi_t,\ \dot{\Phi}_t,\ \dot{\Theta}_t,\ \dot{\Psi}_t)$

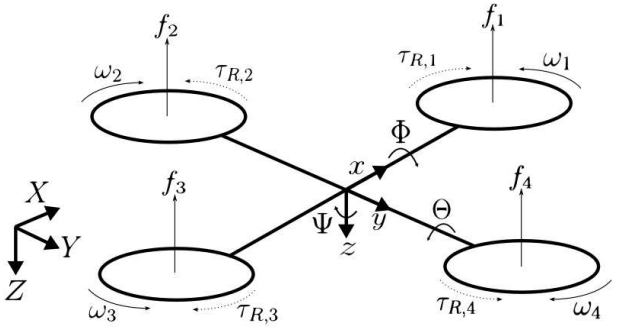

Fig. 5. Outline of the quadrotor helicopter.

As shown in [12], let I_x (resp. I_y or I_z) and τ_Φ (resp. τ_Θ or τ_Ψ) be respectively the moment of inertia and the torque applied, both about the axis x (resp. y or z), let J_R be the torque of inertia of each rotor, the attitude's dynamics can be modeled by the equations below. In the application, this model is the "real" system we want to simulate and control.

$$\begin{aligned}
\ddot{\Phi} &= \tfrac{\tau_\Phi}{I_x} - \tfrac{J_R\ \pi}{I_x}\ \dot{\Theta} + \dot{\Theta}\ \dot{\Psi}\ (\tfrac{I_y - I_z}{I_x}) \\
\ddot{\Theta} &= \tfrac{\tau_\Theta}{I_y} - \tfrac{J_R\ \pi}{I_y}\ \dot{\Phi} + \dot{\Phi}\ \dot{\Psi}\ (\tfrac{I_z - I_x}{I_y}) \\
\ddot{\Psi} &= \tfrac{\tau_\Psi}{I_z} + \dot{\Phi}\ \dot{\Theta}\ (\tfrac{I_x - I_y}{I_z})
\end{aligned}$$

From the applied torques $\tau_\Phi,\ \tau_\Theta,\ \tau_\Psi$ and the current state of the system η_t, we can compute its next state by applying the fourth-order Runge-Kutta method to the model and calculating first the new rotational speeds $\dot{\Phi}_{t+1},\ \dot{\Theta}_{t+1},\ \dot{\Psi}_{t+1}$, then the new angles $\Phi_{t+1},\ \Theta_{t+1},\ \Psi_{t+1}$.

The used controller is derived from the backstepping controller described in [12]. It consists in one proportional-integral-derivative controller (PID) controller per angle. Each controller can therefore be summed up by 9 parameters: 3 gains per angle. The torque vector τ^{ctrl} obtained with the PID controllers is next translated in a command vector ω with a conversion chart. The control sequence lasts 2 seconds and commands are sent to the motors each 20 ms.

As described in III-A, a test contains an initial state and a target state too. In order not to increase unnecessarily the number of parameters for this application, we address the issue of the attitude stabilization from the initial state $\eta = \mathbf{0}$ to the target state $\eta_d = (-30°,\ 30°,\ -70°,\ 0,\ 0,\ 0)$. Thus, only the 9 controller parameters are needed to define one test.

VI. Results

In this section, we want first to show some important points on test learning. In the first part, the tests are evaluated with two objectives only. Some results on the addition of a control quality objective are next presented in a second part. The parameter values used for the performed experiments are shown in Table I (values in brackets are only used in the second part).

TABLE I

Main parameter values.

global process	model loop		test loop		
number of iterations	number of generations	population size	number of generations	population size	number of objectives
7	15000	250	500 (750)	100	2 (3)

A. Global results on the learning of test trajectories

The first points to verify concerns the learning method LGP: does it allow to learn a good model of the physical simulation on a test trajectory? To answer this question, we record the closed-loop objective values obtained with the test Υ_0 and the best trade-off models selected at each iteration of the double loop process in 5 runs[5]. The corresponding means and standard deviations are shown in Table II.

TABLE II

Closed-loop objective values obtained on Υ_0 with the best trade-off model $\mathfrak{M}^*$ selected at each iteration.

it.	0	1	2	3	4	5	6	7
mean	0.204	0.024	0.029	0.129	0.341	0.357	0.394	0.389
st.dev.	0.175	0.030	0.030	0.167	0.250	0.327	0.265	0.403

The test Υ_0 is significantly learned after the first and the second iterations. Moreover, the models contain in average a higher percentage of intron sequences to 50%. These results

[4]For any details concerning the physical model and the backstepping controller, see [12].

[5]The best trade-off model at the iteration 0 is the best model of the initial random population.

validate the choice of LGP as learning method and the size of the models.

Nevertheless the later iterations are accompanied with a performance loss. In fact, only one run performs badly from the 3^{rd} iteration, but 4 runs do from the 4^{th} and 5^{th} ones. The last one doesn't show any significant loss. This perfomance loss is mainly due to the growing objectives number. After some iterations, the model evaluation space becomes indeed too high. The non-dominated set can't converge efficiently to the Pareto front in the model loop with the specified number of iterations. Thus, we often find models that are on the edges of the model Pareto front, then good on some test trajectories and bad on the others.

It is difficult to calculate relevant stastitics on several runs for the other selected tests as they are often different between two runs. The Figure 6 shows information on the selected tests for the more successful run among the 5 ones. Each line corresponds to the closed-loop objective values on a given selected test computed with the successive best trade-off models.

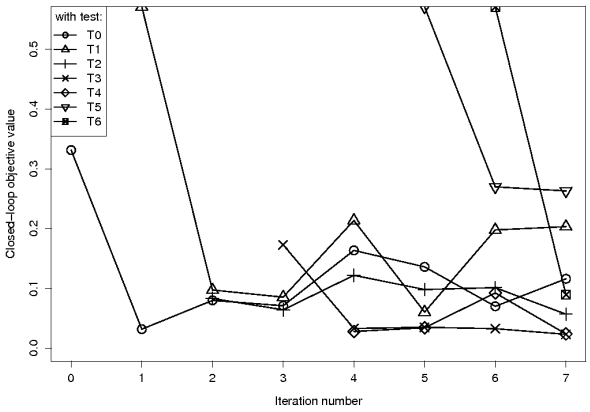

Fig. 6. Graph of the closed-loop objective values on the selected tests computed with the best trade-off model selected at each iteration of the evolutionary double loop.

According to the Figure 6, in this run, not only the first test but also the next ones are learned on a permanent basis. A little performance loss appears at the 6^{th} iteration with the addition of the test Υ_5. Nevertheless, it shows that the algorithm can find relatively good trade-off models in spite of the growing objective number.

B. Influence of a control quality objective

The second main point that we want to achieve consists in finding relevant controllers for the "real" system. To that end, we compare two versions of the algorithm concerning the test evaluation:

1) with two objectives;
2) with a control quality objective more.

To compare these two versions, we have conducted 25 runs of the test loop for each version on a same model non-dominated set, that is after the same run of the model loop. The control quality value have then been recorded on the "real" trajectory of the test selected at the end of the test loop. Some statistics are shown in Table III.

TABLE III
STATISTICS ON CONTROL QUALITY WITH EACH VERSION.

	number of experiments	control quality mean	control quality standard deviation
version 1	25	0.72	3.11
version 2	25	0.19	0.29

The tests selected in the version 2 show control quality values significantly lower than in the version 1. This is mainly due to the fact that in version 1, selected tests can be relevant about the discrimination power and the instability proximity issues, but totally irrelevant concerning the control task. The addition of a control quality objective is then necessary to lead the search to meaningful control trajectories.

Nevertheless, in both versions, a controller significantly better than $\mathfrak{C}_0$ is rarely found. There are two main explanations. On the one hand, due to the evaluation space that grows with the number of iterations in the model loop, the algorithm has been run with a low number of iterations (7). On the other hand, the initial tests have been designed from not so bad controllers that perform relatively well in simulation, then tests with control quality of the same order are often found, but better ones aren't.

VII. DISCUSSION

As tests and models are evaluated by several factors, an intuitive and meaningful way to address model and test selections was to translate our problem into a multiobjective one: each learning/control goal becomes a full-fledged objective. But, an iterative addition of tests raises some questions in such a context.

The first point concerns multiobjective algorithms. Roughly, the actual algorithms are indeed relevant with a number of objectives less than 5, a limit already reached at the 3^{th} iteration of the evolutionary double loop, that is after the addition of 2 tests only. An important issue is then to reduce the number of objectives. To that end, one can try to merge the open-loop objective on all trajectories because this objective can be defined on any dataset and not only on a trajectory. Nevertheless, it would not solve the whole problem. Another solution consists in limiting the number of tests. We'll come back to this issues later on.

The second point involves the number of generations used for the model loop. The higher the number of tests is, the higher the model evaluation space is and the more it takes time to explore efficiently the model Pareto front from the previous one. On the one hand, too few generations don't allow the convergence of the non-dominated set to the Pareto front. Thus, no good trade-off is found. On the other hand, a too high number of generations leads to overfitting on the first trajectories and the model non-dominated set can't be efficiently extended in the added dimensions. These

issues are all the more crucial that the NSGA-II ranking method favours individuals that are on the edges of the non-dominated set thanks to an infinite crowding distance term.

To address both problems, one can resort to a number of generations that grows with iterations, but there is no easy way to design such a function. A more pertinent approach introduces a diversity term that encourages models to spread over the Pareto front by adding a new evaluation objective [26]. Although it adds one more objective, in contrary to the crowding-distance this method allows to favour individuals that would be dominated else, but which are yet pertinent as good trade-offs between the different objectives.

Another approach consists in using a vector of tests with a constant size. At each double loop iteration, some new tests are added to this vector by replacing the oldest ones. It will then fix the evaluation space dimension of the models. Moreover, as old data are progressively "forgotten", such a method would allow the models to adapt to changes of the system dynamics like system damages, collisions, addition/removal of system parts, etc. The process could become less dependent on the initial test Υ_0 too.

The test evaluation can also be improved. On the one hand, staying away from instabilities is convenient, but it penalizes perhaps some relevant behaviors depending on the system. On the other hand, objectives that evaluate if a trajectory is risky or not can be meaningful when transferring onto the real system.

A last unsolved question deals with the first test Υ_0. In the application, we design indeed not so bad first controllers $\mathfrak{C}_0$ from [12]. It would be interesting to use worse controllers, or even incomplete initial trajectories in order to study the robustness of the coevolution process in more realistic situations. This issue will be examined in future work.

VIII. Conclusions and future prospects

This paper addressed the reality gap problem in the case of controller transfer, an important issue in robotics. An identification method of nonlinear systems based on coevolution of models and tests has been implemented. The goal consists finally in automatically modeling and controlling systems (or parts of a system) without the need of an accurate prior model and/or a lot of experiments.

The application to the control of the simulated quadrotor helicopter shows promising results on the modeling and the control issue. Nevertheless, in spite of its intuitive advantages, the iterative addition of tests asks some problems in a multiobjective context. An alternative with a fixed number of tests could be considered. Moreover, previous works on neural networks controllers [27] might be used to address the control issue in a more generic way.

References

[1] J.-A. Meyer, P. Husbands, and I. Harvey, "Evolutionary Robotics: A Survey of Applications and Problems," *Lecture Notes in Computer Science*, pp. 1–21, 1998.

[2] S. Nolfi and D. Floreano, *Evolutionary robotics*. MIT Press, 2000.

[3] S. Nolfi, D. Floreano, O. Miglino, and F. Mondada, "How to evolve autonomous robots: Different approaches in evolutionary robotics," in *Artificial Life IV*, pp. 190–197, 1994.

[4] N. Jakobi, P. Husbands, and I. Harvey, "Noise and the Reality Gap: The Use of Simulation in Evolutionary Robotics," *Lecture Notes in Computer Science*, pp. 704–704, 1995.

[5] E. de Margerie, J. Mouret, S. Doncieux, and J. Meyer, "Artificial evolution of the morphology and kinematics in a flapping-wing mini-UAV," *Bioinspiration and Biomimetics*, vol. 2, no. 4, p. 65, 2007.

[6] P. Abbeel, V. Ganapathi, and A. Ng, "Learning vehicular dynamics, with application to modeling helicopters," *Advances in Neural Information Processing Systems*, vol. 18, p. 1, 2006.

[7] D. Mitrovic, S. Klanke, and S. Vijayakumar, "Adaptive optimal control for redundantly actuated arms," in *From Animals to Animats: Proceedings of the 10th International Conference on the Simulation of Adaptive Behavior (SAB)*, pp. 93–102, 2008.

[8] P. Robbel, "Active Learning in Motor Control," Master's thesis, University of Edinburgh, UK, 2005.

[9] S. Thrun, "Exploration in active learning," in *Handbook of Brain and Cognitive Science* (M. Arbib, ed.), MIT Press, 1995.

[10] J. Bongard and H. Lipson, "Nonlinear System Identification Using Coevolution of Models and Tests," *Evolutionary Computation, IEEE Transactions on*, vol. 9, no. 4, pp. 361–384, 2005.

[11] E. Jong and J. Pollack, "Ideal Evaluation from Coevolution," *Evolutionary Computation*, vol. 12, no. 2, pp. 159–192, 2004.

[12] P. Adigbli, C. Grand, J.-B. Mouret, and S. Doncieux, "Nonlinear attitude and position control of a micro quadrotor using sliding mode and backstepping techniques," in *7th European Micro Air Vehicle Conference (MAV07)*, (Toulouse, France), 2007.

[13] P. Abbeel, A. Coates, M. Quigley, and A. Ng, "An Application of Reinforcement Learning to Aerobatic Helicopter Flight," in *Advances in Neural Information Processing Systems: Proceedings of the 2006 Conference*, MIT Press, 2007.

[14] J. Bongard, V. Zykov, and H. Lipson, "Resilient Machines Through Continuous Self-Modeling," *Science*, vol. 314, no. 5802, pp. 1118–1121, 2006.

[15] K. Deb, *Multi-Objective Optimization Using Evolutionary Algorithms*. Wiley, 2001.

[16] V. Khare, X. Yao, and K. Deb, "Performance Scaling of Multi-objective Evolutionary Algorithms," *Lecture Notes in Computer Science*, pp. 376–390, 2003.

[17] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan, "A fast and elitist multiobjective genetic algorithm: NSGA-II," *Evolutionary Computation, IEEE Transactions on*, vol. 6, no. 2, pp. 182–197, 2002.

[18] J. Bongard, "Action-selection and crossover strategies for self-modeling machines," in *Proceedings of the 9th annual conference on Genetic and evolutionary computation*, pp. 198–205, ACM Press New York, NY, USA, 2007.

[19] K. Kristinsson and G. Dumont, "System identification and control using genetic algorithms," *Systems, Man and Cybernetics, IEEE Transactions on*, vol. 22, no. 5, pp. 1033–1046, 1992.

[20] K. Tan, Y. Li, D. Murray-Smith, and K. Sharman, "System identification and linearisation using genetic algorithms with simulated annealing," in *Proceedings of the 1st IEE/IEEE International Conference on Genetic Algorithms in Engineering Systems: Innovations and Applications*, pp. 164–169, 1995.

[21] J. Bongard and H. Lipson, "Automated reverse engineering of nonlinear dynamical systems," *Proceedings of the National Academy of Sciences*, vol. 104, no. 24, p. 9943, 2007.

[22] G. Gray, D. Murray-Smith, Y. Li, K. Sharman, and T. Weinbrenner, "Nonlinear model structure identification using genetic programming," *Control Engineering Practice*, vol. 6, no. 11, pp. 1341–1352, 1998.

[23] M. Heywood and A. Zincir-Heywood, "Page-based linear genetic programming," in *Systems, Man, and Cybernetics, 2000 IEEE International Conference on*, vol. 5, 2000.

[24] M. Brameier and W. Banzhaf, *Linear Genetic Programming*. Springer, 2007.

[25] M. Heywood and A. Zincir-Heywood, "Dynamic page based crossover in linear genetic programming," *Systems, Man and Cybernetics, Part B, IEEE Transactions on*, vol. 32, no. 3, pp. 380–388, 2002.

[26] J.-B. Mouret and S. Doncieux, "Overcoming the boostrap problem in evolutionary robotics using behavioral diversity," in *Proceedings of the Congress on Evolutionary Computation 2009 (to appear)*, 2009.

[27] J.-B. Mouret, S. Doncieux, and J.-A. Meyer, "Incremental evolution of target-following neuro-controllers for flapping-wing animats," in *From Animals to Animats: Proceedings of the 9th International Conference on the Simulation of Adaptive Behavior (SAB)* (S. Nolfi et al., ed.), pp. 606–618, 2006.