Coevolutionary modelling of a miniature rotorcraft

Renzo DE NARDI and Owen E. HOLLAND
University of Essex,
Department of Computing and Electronic Systems,
Colchester CO4 3SQ, United Kingdom;
rdenar@essex.ac.uk,owen@essex.ac.uk

Abstract. The paper considers the problem of synthesising accurate dynamic models of a miniature rotorcraft based on minimal physical assumptions, and using the models to develop a controller. The approach is based on the idea of building models that predict accelerations, and is implemented using evolutionary programming in a particularly efficient co-evolutionary framework. Both the structure and the parameters of the nonlinear models are jointly identified from real data. The modelling method is demonstrated and validated on a miniature quadrotor rotorcraft, and a controller based on the model is then developed and tested.

1. Introduction

For decades, the wide range of potential applications of unmanned aerial vehicles (UAVs) has made them the subject of academic and commercial research. Quadrotor rotorcraft like the one used in this study (Figure 1) are typical examples of very small and capable flying

Figure 1. Quadrotor rotorcraft. Note the reflective markers used for tracking.

machines. Mechanically very simple to build and maintain, robust to crashes, lightweight, powerful and manoeuvrable, and readily available in the marketplace, these flying machines are becoming the platforms of choice for many research teams ([7],[9],[12],[13]).

The challenge now is in fully exploiting the potential of these machines; in this paper we are seeking a methodology able to identify a system using only data collected during normal piloted flight, and without requiring any prior formal knowledge of the structure of the system model. The advantages of this approach are clear: as well as eliminating

the need for in-depth knowledge of a specialist engineering domain, such as aeronautics, it offers the possibility of automatically identifying unknown and novel systems. Such systems are still being developed by talented craftsmen who build novel flying machines (or variations of existing concepts) relying almost entirely on their insight [14], [18],[15].

A second motivation is the possibility of using automatic modelling for automatic damage recovery (e.g. [6]), where after automatically relearning or adapting the system model, new control strategies that cope with the damage can be obtained.

In any exercise in modelling, it is first necessary to specify the domain of use for the model in question, as this allows us to define the types and extents of discrepancies from the real system that can be tolerated. In our case we plan to use the model to produce a dynamic simulator that, within the normal envelope of use of the machine, is sufficiently accurate for the purpose of developing and testing a controller. In this simulator-based approach, the form of the model is not important as long as the model replicates the input-output behaviour of the actual process. However, for comparability with other methods, we will develop a model in the form of a set of nonlinear differential equations (i.e. in state space form), a common way of representing dynamic systems:

$$^{1}\dot{\mathbf{x}}(t) = f\left(\mathbf{x}(t), \mathbf{u}(t-\boldsymbol{\tau})\right). \tag{1}$$

The next two sub-sections will give overviews of evolutionary modelling, and of modelling quadrotors. Our own approach is explained in detail in Section 2, and Section 3 reports the results of the experiments conducted to validate the approach.

1.1. Quadrotor Modelling and Control

Many publications deal with quadrotors, and most of them are dedicated to modelling and control. The control techniques analysed range from simple PID methods [8] to more complex techniques like LQR, sliding mode or integral backstepping ([7],[9]). A notable exception is [19] where the vertical dynamic of the quadrotor is learned using Locally Weighted Linear Reagression and a controller is trained using reinforcement learning. Most authors begin by proposing models based on first principles. A model for the thrust of each rotor is generally used, and the balance of forces and moments at the quadrotor's centre of mass allows the computation of the dynamic behaviour of the machine. More sophisticated models ([13],[16]) can include the aerodynamic effects of blade flapping, or simple models for the ground effect ([7]). Model parameters are mainly derived from static tests; if a specific component is known to be critical for control, specific tests are devised to model it correctly. Ultimately the engineer uses his insight and the results of control experiments to decide which effects need to be modelled, and which estimated, so that the final controller will yield the desired performance.

1.2. Canonical and Evolutionary System Identification

System identification in aeronautics is of course a well developed field of research. Most established methods tackle the problem of estimating the parameters of a model with a known structure, one that is generally based on first principles. Methodologies that can

[1]The vector $\mathbf{x} = [\phi, \theta, \psi, u, v, w, p, q, r]$ represents the state (ϕ, θ, ψ are the quadrotor's attitude angles, u, v, w and p, q, r are its linear and angular velocity in the body frame of reference), $\mathbf{u} = [u_P, u_R, u_Y, u_T]$ is the input vector made up by the pitch, roll, yaw, and throttle commands and $\boldsymbol{\tau}$ is the vector of input time delays.

also determine the model structure, for example by using such techniques as artificial neural networks, have also been developed (e.g. [11]), but the general need for a model to be transparent and understandable makes those techniques of limited practical use.

The use of evolutionary computation for system identification is not new, but it is only recently that it has been applied to the identification of models of the type and size useful in a domain like ours. For example, Bongard and Lipson have successfully demonstrated the use of co-evolution to regress the dynamics of complex nonlinear systems [4]. The idea behind the techniques is simple: a set of fitness maximizers is evolved, but the tests against which the performance of the maximizers is measured are also under evolutionary control. Each period of evolution of the models is interleaved with a period of evolution of the tests; the fitness of the models is obtained by testing them with the evolved tests, while the fitness of the tests is the variance that each produces in the fitness of the models. The tests therefore evolve to be as discriminative as possible when measuring the performance of the models. In [5] Bongard and Lipson also suggest partitioning a complex problem by regressing each modelled dimension independently of the others; this allows the methodology to scale up and approach problems out of the reach of standard techniques.

Common to any system identification approach there is the fact that any effect not present in the dataset cannot be learned. In our case, the experimenter has to make sure that the flight envelope of interest is adequately covered by the collected data.

2. Our approach

The only domain knowledge we use is the assumption that the system can be sufficiently well approximated as a 6DoF rigid body. This might seem to be rather limiting, but in practice a wide range of vehicles, from wheeled robots to aircraft, helicopters and even ships can be treated in this way. Physics tells us that any motion that our system exhibits is due to the effects of forces and moments (i.e. linear and angular accelerations) applied to the rigid body. If we can relate these instantaneous accelerations to the state and inputs of the system, we will be able to predict the motion of the object under study. This concept paves the way for the two main ideas of our approach:

- using a general and computationally efficient co-evolutionary method to infer the structure and the parameters of the nonlinear relationships between the inputs, the state, and the accelerations;
- directly modelling the accelerations in the body coordinates using the laws of physics to propagate the state forward in time so that the effects of the translation and rotation of the body's frame of reference can be explicitly taken into account.

The sizes of our state and input vectors (12 and 4 respectively) mean that the number of possible nonlinear functions that could relate them to the accelerations is too large for any method of extensive search, and so an evolutionary approach is indicated. We have chosen genetic programming (GP) for its ability to handle both the structure and the parameters of the model, and to deal with both linear and nonlinear functions. The efficiency of the coevolutionary setup is also important, since every function evaluation involves integrating the full 6 dimensional state of the model, which is computationally expensive.

Given that all the state variables and inputs are available from the data, and that we can precompute the relevant accelerations offline, an obvious approach is to search for

the model that minimizes the error between the predicted and the computed accelerations. However, when the acceleration prediction is integrated forward in time, any errors will be accumulated, creating an obvious risk of divergence. A better approach is to integrate the predicted acceleration for a specified number of time steps, and select the model that minimizes the error between the predicted and the computed value of the state variable itself. Any divergence will produce a higher error, ensuring a search for models with good long term prediction ability. As the predicted acceleration is successively integrated through time, it will affect not just its own, but also the other state variables. Prediction errors will propagate through the system just as they would when the model is used as a simulation tool. In effect, our evolutionary algorithm will produce a model that can cope with the effects of its own errors.

A vehicle travelling in a given direction, and at the same time rotating, will experience a sideslip force as a result of inertia. These effects are nonlinear [3], which makes the model learning problem even harder. The discrete time update of the state variables can be written in body coordinates as:

$$\begin{bmatrix} u \\ v \\ w \end{bmatrix}_{t+1} = R_{b_t}^{b_{t+1}} \left(\begin{bmatrix} u \\ v \\ w \end{bmatrix}_t + \begin{bmatrix} a_x \\ a_y \\ a_z \end{bmatrix}_t \Delta t \right) \qquad \begin{bmatrix} p \\ q \\ r \end{bmatrix}_{t+1} = \left(\begin{bmatrix} p \\ q \\ r \end{bmatrix}_t + \begin{bmatrix} \dot{p} \\ \dot{q} \\ \dot{r} \end{bmatrix}_t \Delta t \right) \quad (2)$$

where u, v, w are the velocities in the quadrotor body frame of reference, p, q, r are the rotational velocities about the axes, $R_{b_t}^{b_{t+1}}$ is the matrix transformation that rotates the body frame from its orientation at time t to that at $t+1$; $\mathbf{a} = [a_x, a_y, a_z]$ and $\boldsymbol{\alpha} = [\dot{p}, \dot{q}, \dot{r}]$ are respectively the linear and angular accelerations in body coordinates. Equation 2 shows that the linear accelerations are not simply the derivatives of the linear velocities, and so an additional nonlinear transformation is needed to compute them. By performing the transformation $R_{b_t}^{b_{t+1}}$ in our integration routine, we simplify the learning task.

2.1. Data filtering, computation of derivatives and integration

The first step in our method is the data collection. The 6DoF pose of the vehicle, and the input from the pilot, are recorded at $100Hz$. However, we require not only the absolute pose of our vehicle, but also its first and second derivatives (i.e. the angular and linear velocities, and the angular and linear accelerations). We first apply a low pass filter to reduce the effects of noise. A transfer function frequency plot from our data showed the dynamics of our system to be quite slow, with most of the frequency content below $4Hz$; this allowed us to set the cutoff frequency to $5Hz$. To avoid phase distortion, and to allow for delay compensation, we use a finite impulse response (FIR) filter with 200 taps.

The need to integrate the state equations forward in time is a significant computational drawback of any time-domain method. However, given the limited bandwidth of our system, we can mitigate this problem by downsampling our data. We then compute the first and second order derivatives as first order differences. During the computation of the derivatives the necessary changes of coordinates are performed in order to obtain the mathematical counterpart of the integration procedure described next.

As a final check, we integrate the computed accelerations forward in time to verify that the resultant time series matches the original time history. With a requirement for a relative squared error (RSE) lower than 5% within a time span of 750 steps[2] we can

[2]The length normally used in the regression algorithm, equivalent to $30s$ of clock time.

safely achieve a downsampling factor of 4, which brings the sampling frequency to $25Hz$. The trim values (i.e. the control values corresponding to the hover condition) for each of the control data series are then subtracted; this is standard practice in time domain methodologies.

To produce the development over time of the state variables, the accelerations predicted by the model need to be integrated from some initial state. As well as executing the discrete time update equation for linear and angular velocities (i.e. equation (2)), the integration routine also needs to compute the position and attitude. In order to avoid the gimbal lock problem arising from the use of Euler angles, and also to improve accuracy, the attitude is represented in quaternion form [3].

2.2. Co-evolutionary Setup

The coevolutionary setup of our algorithm consists of two main steps that are interleaved in time: in the first, the models are evolved, and in the second, the tests are evolved.

In our case a model is simply a GP expression tree with a maximum depth of 5. The inner nodes of the trees are the usual functionals $\{+, -, *, /\}$ while the leaves can be any of the inputs or a constant value. To reduce the computational complexity we have opted for using 14 parallel hill climbers in place of a population based strategy; at each generation the parent is replaced if the offspring generated by mutation performs better. Mutation is the only evolutionary operator used, and either a macro or a micro mutation is used with equal probability. A macro mutation randomly selects a node in the tree and replaces it with a newly generated one, removing or generating child nodes as needed. A micro mutation selects a node and replaces it with a newly generated subtree. To generate a new subtree, nodes are drawn at random with a probability of 0.7 of picking a leaf. Following a macro or micro mutation, one of the constants in the tree is perturbed with a probability of 0.5 by adding a random value from a Gaussian distribution with mean 0 and variance 0.1. The initial value of a constant node is chosen from a uniform distribution ranging from -5 to 5. Finally, mutation is applied to the delay of each input: with probability 0.1, a delay can be replaced with a new random value between 0 and 5 (the maximum allowed delay). All delays are originally initialised to 0.

In our algorithm, tests are simply short subsets of consecutive data chosen from the training dataset. The first sample of the set represents the initial conditions, and the subsequent samples are used by the integration routine. The average squared error of the model computed from those samples is summed over all the tests in the suite to yield the model fitness. The position at which a subset of points is taken from the dataset is controlled by evolution; mutation is applied by adding a randomly generated value to the current starting index of the data in the test. In the case of the tests, the population of 5 individuals is evolved using five parallel hill climbers. The fitness function used for a test is the variance that each test produces in the current population of models, thereby choosing for identification the parts of the dataset not well described by the population of models. This stops the optimization from focusing too much on dynamic states that are overrepresented in the data (e.g. hovering). To avoid cyclic oscillations of fitness, we do not use the evolved tests directly for the evaluation of the models, but instead maintain a suite of 6 tests to which we add the last best test generated. If the test overlaps one of the tests already in the suite, it replaces it, otherwise it replaces the oldest test in the suite.

[3] Although the core integration routine uses quaternions, the attitude is expressed in Euler angles when passed to the modelling equations.

Bloat tends to increase the size of the models significantly, so after each cycle of the algorithm, we try to simplify each model by replacing a randomly selected subtree with a constant. We then try to reconverge the tree by evolving it using only the Gaussian mutation operator. If the resulting tree performs no better than the original, it is discarded.

An iteration of the algorithm typically consists of 600 generations of model evolution, 80 generations of test evolution, and one instance of tree simplification. A typical run of the full algorithm contains 30 iterations.

3. Experiments

3.1. The quadrotor

The quadrotor used in the study (see figure 1) is a commercially available model [1] powered by four brushless motors fitted with 8" propellers. The only modification made for these experiments was the addition of the 5 infrared tracking markers. For data collection the quadrotor was manually flown in our flying arena which is equipped with a Vicon MX [2] infrared motion capture system; this system tracks and resolves the quadrotor's 3D position and orientation in real time with high accuracy and precision (of the order of millimetres) with a sampling frequency of $100Hz$. A standard RC transmitter was used to fly the quadrotor manually; the commands were recorded at $100Hz$ and synchronised to the flight data. The transmission delay of the RC transmitter is $18ms$, and the data capture delay is typically under $10ms$; both delays were ignored since at $25Hz$ (the post filtering data sampling rate) they are smaller than one time step.

A quadrotor is by design a mechanically unstable system since even if the four rotors are driven at the same speed, the rotational moments produced by each rotor will not cancel out exactly due to minor differences in drag and lift. MEMS gyros are commonly used on this type of rotorcraft to provide active stabilisation based on rotational speed feedback. In [12] Gurdan et al. describe the low level stabilisation algorithm used in our quadrotor; it is basically a set of three independent PD loops, one for each rotational axis. No dynamic models were used to design the machine or the controller, and the authors relied solely on their insight and experience to tune the controller empirically.

During the flight we did not have direct access to the low level motor inputs; only the control inputs to the stabilisation loops were recorded. In contrast to the 'first principles' models discussed in 1.1 our system identification technique will provide a lumped model of the quadrotor plus the stabilisation controller. We do not consider this as a limitation, but instead we see it as illustrating the flexibility of our approach. The user can decide at what level to model the system, without the need to understand either the requirements or the implications for the structure of the model.

3.2. Identification

After filtering and downsampling, a series of 13800 data points was extracted; 11800 of them were used for training with the remainder reserved for validation. Each sample point consists of 3 arrays, the state $\mathbf{x}$, the computed accelerations $\boldsymbol{a}$,$\boldsymbol{\alpha}$ and the control input $\mathbf{u}$. During evolution, the individual tests were conducted using short chunks of 750 consecutive sample points[4]. The state information in the dataset is used only to define the initial

[4]Longer time windows would be able to to enforce a search for even better long term prediction abilities but at the expenses of increased computational needs; the length of 750 was empirically found to be a good trade off.

condition of the system; from that moment onward, the state is computed using the integration routine. At each timestep, the model under test is fed the control inputs, and the current state. Since the delay of each input is controlled by the evolutionary algorithm, it is important to remember that some of the control data might not be the "current" ones. The predicted accelerations and the remaining "true" accelerations from the dataset form the inputs to the integration routine.

The fitness of a given model is the average absolute error between the true state variable and the one predicted by the integration routine; for example, for a model that predicts the variable $\dot{e}$, the fitness ($f_{\dot{e}}$) would be:

$$f_{\dot{e}} = -\frac{\sum_{n=0}^{N}\left(\sum_{t=0}^{T}|e_t - \hat{e}_t|\right)_n}{N * T} \qquad e \in \{u, v, w, p, q, r\}$$

where T is the number of samples in each of the tests, N is the number of tests in the test suite, e_t is the true and $\hat{e}_t$ is the predicted state variable. The algorithm also delivered a very similar performance using the classic squared error fitness metric.

We can now move towards the production of a full model. Each of the six equations was first identified independently, and then, for each one, the best model found after 30 repetitions of the coevolutionary algorithm was selected. Then we simply provided the predictions of the set of six models as the input to the integration routine. After setting the state to some initial value from the dataset we propagated the system forward in time with the recorded control input as the only input. To have a better understanding of the algorithm's precision and reliability we repeated the procedure of producing a full model 30 times. A qualitative example of the evolution in time of the state variables for a window of $7s$ (randomly selected from the validation data) is shown in Figure 3.2. The pre-

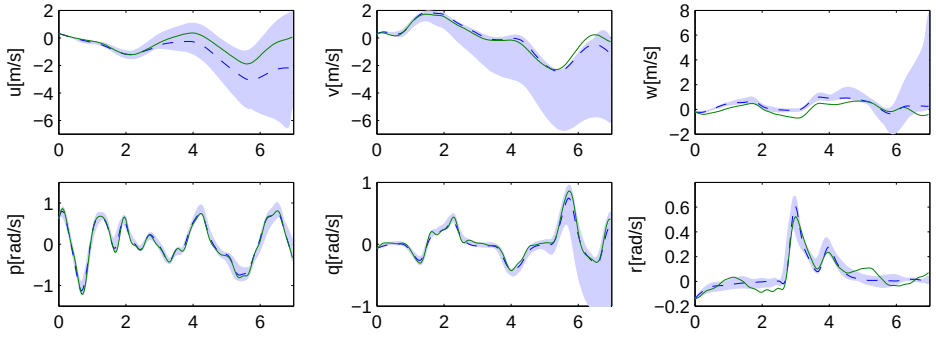

Figure 2. Prediction over a $7s$ time window; true data (continuous line), best model (dashed line), envelope of the min and max values predicted within the whole population of 30 models (shaded area).

dictions appear to be in good agreement with the recorded data; this is especially so for the angular velocities since they depend directly on the control input. The linear velocities suffer a larger error as consequence of the accumulation of the angular velocity error. As expected the error increases as a function of time, but even the worst models still appear well behaved.

In table 1 we can see the equations[5] of the model with the median error performance produced by the 30 runs of the GP process (we will refer to the median model as *modelM*

[5]We have simplified the expressions and rounded the constant terms to one decimal place for better readability.

from now on); we chose this as a good representative of the full set of models, but a similar structure can also be seen in the best model.

Median model
$a_x = 10.0\boldsymbol{\theta} - 0.6\boldsymbol{u} + 0.18 - 0.5q - 0.56pr + 0.1v - 0.6u_Y$
$a_y = \mathbf{0.3} - 9.2\boldsymbol{\phi} - 0.6\boldsymbol{v} + v/(4.3 + \phi - r)$
$a_z = 28.2\boldsymbol{u_{T-5}} - \boldsymbol{w} + 2.1u_P^2 - 0.02$
$\dot{p} = 10.3\boldsymbol{u_{R-1}} - 4.15\boldsymbol{p}$
$\dot{q} = -9.8\boldsymbol{u_P} - 3.3\boldsymbol{q} - 1.2\theta + 0.2u + 9.8qu_Ru_Y + (\psi - qu_P)(-u_Y + qu_Ru_Y)$
$\dot{r} = -7.8\boldsymbol{u_{Y-2}} - 1.9\boldsymbol{r} + 2qu_Y - r^2u_{Y-2}$

Table 1. Model with median error performance from 30 repetitions of the algorithm. Terms in bold appear (directly or after an appropriate series expansion) in every single one of the 30 models. The symbol u_{X-k} indicates the input X delayed by k timesteps.

Although understanding the model produced by our evolutionary algorithm is far from simple, it is interesting and instructive to try to analyse the structures present in all 30 models. (They are highlighted in bold in table 1). First, in the expression of the linear accelerations a_x and a_y, we can recognize something like the small angle approximation to the projection of gravity (i.e. 10.0θ and 9.2ψ). At hover the thrust is exactly equal to the weight, so any pitch or roll movements will project the accelerations $\sin(\theta)\mathbf{g}$ and $\sin(\phi)\cos(\theta)\mathbf{g}$ in the forward and lateral direction; again, our terms are approximations of these. The reader familiar with aerodynamics will also spot the Stokes' drag (the expression of the viscous drag appropriate for relatively slow speeds) appearing in the terms $0.6u$ for a_x and $0.6v$ for a_y; this also appears in the expression of a_z as $-w$. The vertical acceleration is seen to be proportional to the throttle input u_T; this makes sense because in a quadrotor the throttle directly controls the mean speed of the rotors, which is proportional to the thrust (and our brushless motor controller can exactly maintain the required speed thanks to rotational speed feedback). Of course, the inertia of the motors and rotors reduces the bandwidth of the propulsion group; our GP expression models this as a control delay u_{T-5}. The angular accelerations show the presence of the PID controller loop. We know from [12] that the controller produces a change in rotor speeds proportional to the control input; this explains the terms $10.3u_{R-1}$,$9.8u_P$ and $7.8u_{Y-2}$ (in $\dot{p}$,$\dot{q}$ and $\dot{r}$ respectively). The controller output is also inversely proportional to the rotational speed in the controlled axis, giving rise to the terms $-4.15p$, $-3.3q$ and $-1.9r$

Although they were derived from real and noisy data, the models produced here are completely deterministic; a way of simulating the disturbances naturally produced in rotors suggested in [9] is to add some form of noise to the control inputs. In all our simulations we have used Gaussian noise with a standard deviation equal to 10% of the maximum value of the control input in the data used for identification.

3.3. Control

We then moved on to test the model in conjunction with a controller. As a first step we manually tuned a PID controller on the test quadrotor to perform waypoint navigation. The position and attitude estimation provided by the Vicon system were used as the feedback signals for a series of nested PID loops, with the inner layer consisting of three PD loops controlling pitch, roll, and yaw angle using the attitude and angular speed infomation from the motion capture system. The outer layer consisted of three PD loops, two of which,

given the lateral and longitudinal distance from the next waypoint (i.e. the current error in body coordinates), would output a pitch and a roll angle to be tracked by the loops of the inner layer. The third PID loop controlled the throttle input as a function of the altitude error.

The next step was to investigate the possibility of producing a controller directly based on the model.We were not attempting at this stage to produce an optimal controller (indeed, better tracking results than those presented here can be found in the literature e.g. [17]); a simple but effective controller was all that was required. Our previous experience in evolving controllers had been successful ([10]) so we decided to evolve from scratch the parameters of the PID controller we had already developed. We used a simple approach based on evolutionary strategies: each candidate controller would attempt to fly *modelM* on a randomly generated course. The controller's fitness was defined as proportional to the distance covered in the allotted time, and inversely proportional to its deviation from the course (for a detailed explanation we refer to [10]).

To compare the performance of the controller in the simulated and in the real system, we used one of the trajectories generated during the evolution of the PID controllers. We first recorded the path followed in simulation, and then recorded the path of the real quadrotor flying the same trajectory while controlled by the evolved PID. The recorded trajectories are plotted in Figure 3. The high similarity between the paths followed by the

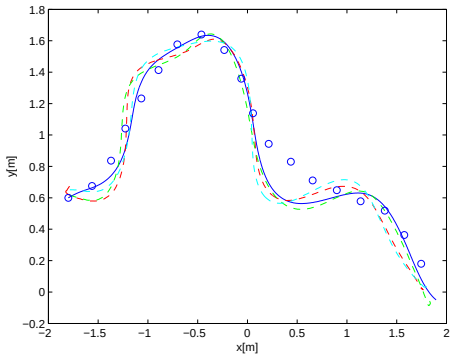

Figure 3. Evolved PID controlling the quadrotor and its model; simulated using the model (blue continuous line) and three repetition of the task on the real quadrotor (red,green and cyan dashed lines).

real quadrotor and the simulation experimentally validates our modelling technique, and shows that a PID controller evolved in simulation can be transferred to the real system. Both the model and the quadrotor behave poorly (but more important similarly) right after the sharpest bend; we believe this to be a result of our suboptimal choice of controller structure.

4. Conclusion

The methodology presented here has been shown to be capable of identifying the dynamics of a non-trivial platform without the need for any specialised domain knowledge. Stages traditionally left to the abilities of the skilled engineer, such as the choice of the model structure or of the relevant inputs, have been automated. The use of genetic programming permits the nonlinearities in the model to be handled naturally, along with the identification

of all its parameters, enabling the results achieved in simulation to be transferred to the real platform.

Although we have successfully demonstrated the principle, this work will need to be further validated and extended to be of any practical use. At present the technique does not provide any adequate characterisation of the parameters in the model, nor does it include a sensitivity analysis of the model output to parameter changes.

Future research will address the complementary problem of automatically building a controller; we are hopeful that this will reach the same levels of performance as more traditional approaches, with the advantages of being a fully automatic methodology.

5. Acknowledgment

Our thanks go to Richard Newcombe and Julian Togelius for many insightful discussions, and to Swarm Systems Ltd. for financial support.

References

[1] Ascending technologies GmbH. http://www.asctec.de/main/index.php?id=4&pid=2&lang=en&cat=pro.

[2] Vicon MX homepage. http://www.vicon.com/products/viconmx.html.

[3] P. Abbeel, V. Ganapathi, and A. Y. Ng. Modeling vehicular dynamics, with application to modeling helicopters. In *Neural Information Processing Systems*, December 2006.

[4] J. Bongard and H. Lipson. Nonlinear system identification using coevolution of models and tests. *IEEE Transaction on evolutionary computation*, 9(4):361–384, August 2005.

[5] J. Bongard and H. Lipson. Automated reverse engineering of nonlinear dynamical systems. *PNAS*, 104(24):9943–9948, June 2007. p.

[6] J. Bongard, V. Zykov, and H. Lipson. Resilient machines through continuous self-modeling. *Science*, 314:1118–1121, 2006.

[7] S. Bouabdallah. *Design and control of quadrotors with application to autonomous flying*. PhD thesis, EPFL, 2007.

[8] S. Bouabdallah, A. Noth, and R. Siegwart. PID vs LQ control techniques applied to an indoor micro quadrotor. In *Proceeding of IROS 2004*, 2004.

[9] I. D. Cowling, O. A. Yakimenko, J. F. Whidborne, and A. K. Cooke. A prototype of an autonomous controller for a quadrotor UAV. In *Proceedings of ECC 07*, 2007.

[10] R. De Nardi, J. Togelius, O. Holland, and S. Lucas. Neural networks for helicopter control: Why modularity matters. In *IEEE Congress on Evolutionary Computation*, July 2006.

[11] W. E. Faller and S. J. Schreck. Neural network: Applications and opportunities in aueronautics. *Progress in Aerospace Sciences*, 32:433–456, 1996.

[12] D. Gurdan, J. Stumpf, M. Achtelik, K. Doth, G. Hirzinger, and D. Rus. Energy-efficient autonomous four-rotor flying robot controlled at 1khz. In *The 2006 International Conference on Robotics and Automation.*, September 2006.

[13] G. M. Hoffmann, H. Huang, S. L. Waslander, and C. J. Tomlin. Quadrotor helicopter flight dynamics and control: Theory and experiment. In *Proceedings of the AIAA Guidance, Navigation, and Control Conference*, 2007.

[14] P. Muren. The proxflyer. http://www.proxflyer.com.

[15] K. Nakamura. Mr. Kimio NAKAMURA's Coaxis Micro Helicopter. http://liaison.ms.u-tokyo.ac.jp/agusta /coaxis/nakamura.html.

[16] P. Pounds, R. Mahony, and P. Corke. Modelling and control of a quad-rotor robot. In *ACRA 2006*.

[17] M. Valenti, B. Bethke, G. Fiore, J. How, and E. Feron. Indoor multi-vehicle flight testbed for fault detection, isolation, and recovery. In *AIAA Conference on Guidance, Navigation and Control*, 2006.

[18] A. Van de Rostyne. The pixelito. http://pixelito.reference.be.

[19] S. L. Waslander, G. M. Hoffmann, J. S. Jang, and C. J. Tomlin. Multi-agent quadrotor testbed control design: Integral sliding mode vs. reinforcement learning. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2005.