Evolution of Neural Networks for Helicopter Control: Why Modularity Matters

Renzo De Nardi, Julian Togelius, Owen E. Holland and Simon M. Lucas
Department of Computer Science
University of Essex, UK
{rdenar, jtogel, owen, sml}@essex.ac.uk

***Abstract*— The problem of the automatic development of controllers for vehicles for which the exact characteristics are not known is considered in the context of miniature helicopter flocking. A methodology is proposed in which neural network based controllers are evolved in a simulation using a dynamic model qualitatively similar to the physical helicopter. Several network architectures and evolutionary sequences are investigated, and two approaches are found that can evolve very competitive controllers. The division of the neural network into modules and of the task into incremental steps seems to be a precondition for success, and we analyse why this might be so.**

I. Introduction

A. The helicopter platform

The work described in this paper is part of the UltraSwarm project, a research effort that aims to produce a flock of miniature helicopters able to perform cluster computation across a wireless network [10][7]. The two main areas of research are coordinated flight using nearest neighbor rules (i.e. flocking) and mobile cluster computing. At this stage we are interested in developing an indoor "proof of concept" system; the development of a suitable autonomous aerial vehicle is the first step towards the goal of coordinated flight.

1) Characteristics of the helicopter: The aerodynamic design of the vehicle is not among our primary objectives, so to speed up the development a commercially available electric model helicopter is being used (see [2]). The advantages of using model helicopters for our project are many: they are generally cheap, easily available, and robust. Cost competition ensures simplicity of design, but also limits the overall quality standard, and differences in flying qualities are therefore common even between helicopters of the same type. In addition, since these helicopters are not designed to carry any payload, we expect to see variations in their dynamic characteristics when loaded with the additional hardware needed in our experiments.

As is often the case in the model toy market, the design of these helicopters is largely the result of the experimental work of a committed designer, rather than a conventional engineering effort; this means that no adequate dynamic model of these novel helicopters is available. Under these circumstances, it is clear that the methodologies of control system design traditionally used in the aircraft control community become very difficult to apply. This paper examines in some detail a controller design methodology more suited to these circumstances, in that it does not require a detailed knowledge of all the characteristics of the helicopter.

2) Onboard sensing and computation: Given the limited payload (approximately 40g), only a small computer and a limited set of sensors can be carried. The chosen computer, a Gumstix Basix400, weighs 8g; a 15g digital IMU (inertial measurement unit), consisting of 3 accelerometers and 3 rate gyroscopes installed in orthogonal x,y,z directions, will be used as the main sensor for stabilization. For navigation purposes, the absolute position of the helicopter will be estimated by an infrared tracking system installed in our test arena. The inertial and positional data, appropriately fused onboard using a Kalman filter, will permit the full estimation of the helicopter state $[x, y, z, u, v, w, \phi, \theta, \psi, p, q, r]$[1].

3) Simulation environment: We are currently awaiting the full installation of the infra-red tracking system which will be used to obtain the flight data necessary for deriving the dynamic model of our helicopter. As an interim measure to enable a preliminary validation of our approach, we are currently using the freely available helicopter simulator included in the Autopilot software suite [1]. This simulator accurately reproduces the dynamics of the XCell 60 model helicopter. Blade element theory is used as the basis for the computation of rotor thrust and drag forces, and the main rotor dynamics and stabilizing bar are modelled as proposed in Mettler *et al.* It updates its state and receives a new command from the controller every 20 ms (50 Hz), while the dynamic equations are integrated with a timestep of 10 ms. [13]. The simulator outputs the same state variables as will be available from the Hirobo helicopter, and accepts the same flight control inputs. Although qualitatively similar to the Hirobo in all essential respects, the simulated helicopter is much less stable (it is almost unflyable manually), and so it definitely constitutes a challenging test bench for our design approach.

B. System identification

In order to develop a good controller, we need to be able to evaluate its performance, but testing a controller directly on a real helicopter is prohibitively time-consuming, and may

[1]The state vector follows the conventional notation used in the aircraft control community; x, y, z and u, v, w are respectively the position in the inertial reference frame and the velocity in the helicopter's frame of reference, while ϕ, θ, ψ and p, q, r are respectively the rotations and rotational velocity about the axis of the helicopter

be unsafe in the early stages. Whether the controller is to be designed manually or automatically, it is necessary to be able to predict its performance using a good model. Unfortunately, although nowadays there exists a well-established body of knowledge concerning the modeling of fixed-wing aircraft, this is not true of rotary-wing aircraft, especially at small or micro-scales.

1) Modelling from first principles: Because of its complexity and severe nonlinearity, the understanding of helicopter aerodynamics is still relatively poor, and the direct estimation of model parameters from experimental flight data still remains the only reliable method for producing accurate models. However, a suitable mathematical representation of the helicopter model can be used to embed specific domain knowledge about the helicopter dynamics to some extent. In [13] and [11] a nonlinear helicopter model is derived on the basis of aerodynamic principles, and from its linearization a state model is derived. A global optimization of the model parameters to fit the flight data in the frequency domain is then conducted. By making provision for the selection of meaningful data, and by enabling the possibility of including several equilibrium points in the flight envelope, a sufficiently accurate helicopter model can be produced.

2) Principles-agnostic modeling: In an alternative approach, Abbeel *et al.* [3] present a more general time domain identification method. The originality of the approach lies in the idea of predicting physical accelerations instead of the direct state coordinates. Linear regression is then applied in the time domain, allowing the system to learn the free parameters of the model that best fit the acceleration data collected during flight. Subsequent integration and appropriate changes of coordinates then lead to the full helicopter state description. Special attention is devoted to producing good long term estimates by minimizing the prediction error several steps ahead. Learning to predict the model‘s accelerations proved to be particularly effective in modeling the inertial effects, since they are of course the direct consequences of acceleration.

These two methods, although similar in many respects, have different strengths. A general model is of much broader potential use since it does not assume any knowledge of the peculiarities of a specific helicopter. Such a model is particularly suitable for controller design; the automatic learning of a control policy is a typical example where the use of domain knowledge is inherently limited. In contrast, the specific contributions of the design characteristics to the dynamic model are more clearly revealed by modelling based on first principles, where every estimated parameter has a clear physical meaning. Since we do not presently have access to suitable design information about our chosen vehicle, in this project we will use the approach of Abbeel *et al.* [3]

C. The orthodox approach to helicopter autopilots

Probably the simplest way to approach helicopter control is by controlling the four helicopter inputs by means of modular control logic in the form of multiple SISO (single-input single-output) control loops (see for example [20] or [4]); the PID controller that comes with the helicopter simulator is of this type. Unfortunately this methodology has limited capability for handling uncertainty, disturbance, and coupling among channels. There is a clear trade-off between design simplicity (in the form of independently designed control loops) and the performance of the controller.

To improve the tracking performance when executing complex trajectories, controllers must be designed to exploit the full state vector, and to compute the control outputs taking account both of the nonlinear characteristics, and of the coupling between modes. The control system literature presents a variety of suitable techniques (e.g. $\mathcal{H}_\infty$ [12], nonlinear model predictive control [20]), with well known strengths and weaknesses. These analytical methods are known to produce effective controllers, but the insight of the control system engineer and his knowledge of the vehicle characteristics often play a major role in achieving successful results.

D. Neuroevolution as a method of developing controllers

The term neuroevolution refers to the practice of using evolutionary algorithms to define the weights, and sometimes also the topology, of a neural network [24]. Neuroevolution has proven to be an effective methodology for designing controllers for simple robotics problems [18] involving legged or wheeled robots, and also for agents in computer games [21][23]; both problem domains are similar to the problem of helicopter control in that they involve the time-critical control of objects in real or simulated physical space.

E. Previous machine learning approaches to helicopter control

Ng *et al.* [15][16], used reinforcement learning to learn to fly a large model helicopter using a simulation built around the acceleration based model described in section I-B. Reinforcement learning was used to learn the weights of very small custom-topology neural networks, which were used as the controllers. The networks were separated in input and output space, so that for example only those inputs judged relevant for pitch were provided to the network dealing with pitch control; only a few connections between the networks were present. With this approach, Ng *et al.* managed to produce a controller able to fly the real helicopter along complicated trajectories used in model flying competitions.

In [19] a GA was used to determine the optimal gains of a predesigned pitch controller for a simulated helicopter. A floating point representation of the genes appeared to facilitate the evolution process. However, limiting the study to the longitudinal channel alone is a drastic simplification of the general problem.

Evolutionary computation has also been employed to refine the parameters of a predesigned fuzzy rule controller [9]. Again, separate functional modules (longitudinal, lateral, altitude) were used in the controller. The simultaneous learning of the weights of all the rules turned out to be possible only if a suitable initial set of rules was provided.

Giving evolution the freedom to increase the ruleset from a simple reduced subset demonstrated the ability to produce an adequate ruleset from scratch, although much of the controller was still hard-wired.

F. On evolutionary algorithms and reinforcement learning

With a little change of perspective, evolutionary computation can be seen as an alternative to, or a form of, reinforcement learning. Among the differences are that evolutionary computation is population-based, and that changes to the controllers are carried out only after the completion of a trial, rather than continuously as in most classical reinforcement learning approaches. The relative merits and applicability of evolutionary computation and reinforcement learning are a topic of current debate. In this case, however, there is a clear advantage in using evolutionary methods: Ng *et al.* needed to artificially structure the training task into many "task-segments" in order to balance exploration and exploitation, whereas by using artificial evolution we can evaluate performance on the whole task, thus reducing the human involvement in behavior shaping.

II. Methods

A. PID controller

The simulator comes with a handcrafted PID controller, tuned to perform waypoint following. The controller is designed as four separate parts: two single loop PID modules for controlling yaw and altitude respectively, and two modules controlling longitudinal and lateral motion. The lateral and longitudinal modules each contain two nested PID loops, the inner controlling the roll angle ϕ (pitch angle θ in the case of the longitudinal module), and the outer controlling the distance from the waypoint in terms of body coordinates y (x). The controller is completely specified by the values of the feedback constants; there are eighteen in our case. This controller, although clearly not optimal, provides a good baseline against which to compare our evolved solutions. Its modular structure inspired the topology of the network presented in section V, and was also useful during the training phase (see section IV).

Comparing the expressive power of PID controllers with that of neural networks is far from simple. The PID controller is linear and therefore its performance degrades significantly when the system state deviates from the equilibrium point at which the PID is tuned; the nonlinearities of the system are clearly difficult to handle. On the other hand, the PID integrates information over time, correcting for any steady-state error, something a normal multi-layer perceptron cannot do but which a recurrent neural network can.

B. Neural networks

The controllers we developed were based on feed-forward neural networks, using the $tanh$ activation function at all nodes. Some of the networks were organized as multi-layer perceptrons (MLPs) with two layers of weights (see fig. 1), but others used less connected topologies (see fig. 3). The networks were fed some or all of the helicopter's 12 state variables, and were also given information about the position of the current goal (the next waypoint) relative to the position of the helicopter. After appropriate scaling, the outputs of the neural network were treated as some or all of the actuator commands to be sent back to the simulator.

C. Evolutionary algorithm

1) Selection: The connection weights of the neural networks were set by an evolutionary algorithm. The evolutionary algorithm was essentially a (10+23) Evolution Strategy, with occasional self-adaptation. The algorithm worked as follows: At initialization, the first population was filled with neural networks with small random weight values. All networks were evaluated (given a fitness) on the task, and the population was sorted according to fitness. The worst 23 individuals were then replaced with new individuals formed from the best-performing 10 individuals (the elite). The algorithm did not employ recombination; each new individual was just a mutated copy of a randomly selected member of the elite.

2) Mutation: Mutation consisted of adding a random value (drawn from a Gaussian distribution with mean 0 and standard deviation D) to each connection weight in the network. D was in general set equal to 0.01, but in some experiments self-adaptive mutation was used, so D was itself subject to mutation. This was arranged as follows: Before the weight mutation was applied, D was mutated by multiplying it with e to the power of a value drawn from a Gaussian distribution with mean 0 and standard deviation 1.

D. Tasks

The fitness function used for the evolutionary algorithms was based on progress along a path defined by an ordered chain of waypoints; it was similar to the method used in [23], but was extended to three dimensions. The positions of the waypoints were randomly generated at the beginning of each trial, and the waypoint component of fitness (chain progress P_{chain}) was proportional to the number of waypoints visited in the correct order. A waypoint was deemed to be visited if the centre of the helicopter approached within 1 foot of it; the mean distance between successive waypoints was 17.5 feet. The full fitness function (f) depended not only on progress along the track, but also on the orthogonal distance from the shortest path between the waypoints, and on the ability of the controller to maintain correct altitude and heading:

$$f = \frac{\sum_{i=0}^{N} \left(w_h P_{chain} \mid z - z_{next} \mid - \mid \psi - \psi_{ref} \mid \right)}{N}. \quad (1)$$

Where N is the number of timesteps allowed to execute the task (for the evolution $N = 1000$ was used) and z_{next}, ψ_{ref} are respectively the altitude of the next waypoint and the fixed reference heading. The factor w_h is equal to one if the helicopter is on the shortest path between waypoints and decays as the cube of the orthogonal distance from it, thus penalizing controllers that do not follow the shortest path. The factor w_h forces a null instantaneous fitness whenever a controller is not able to keep a suitable altitude; however,

in the early stages of development, crashes into the ground were frequent, and if this happened the fitness was set to a negative value equal to the difference between N and the number of elapsed timesteps. The controllers that spent the longest time aloft were selected as members of the elite, thereby helping the early stages of the evolutionary process.

III. A partial overview of preliminary non-working approaches

In this section we survey some of the more noteworthy unsuccessful approaches that eventually led to the successful methods presented in sections IV and V. In section VIII we analyse these failed attempts using the conceptual framework developed by Calabretta *et al.* [5].

A. (Not) Evolving monolithic networks from scratch

In the first approach, we tried to evolve a full MLP with two layers of weights, using the standard version of the evolutionary algorithm presented in section II-C, and a fitness function based only on progress along the waypoint chain. The MLP was given all 12 state variables (position, velocity, angles and rotational velocity), plus the location of the next waypoint in 3 dimensions, and the network's four outputs were used to drive all the control surfaces of the helicopter.

Results were not encouraging. In some cases, no goal-directed behaviour at all was observed, save that of maintaining enough altitude not to crash into the ground. In some other cases, the helicopter drifted slowly towards the first waypoint, but rarely reached it. In all cases, the helicopter was continuously spinning around the z axis, though the speed with which it did this varied.

Guessing that the failures were due to the continuous z-rotation of the helicopter, we decided to remove yaw control from the neural network and let part of the original PID controller handle the yaw, so making sure that the helicopter was always pointing in the same direction (due north). The neural network retained control over the three other actuator dimensions. Controllers evolved with this setup were only marginally better than those with neurally controlled yaw, leading us to think that further modularization was needed.

B. (Not) Evolving yaw control together with the rest of the task

Finding that the monolithic approach above did not work, we decided to divide up the controller into several small modules. Throughout a series of experiments, we tried to evolve the yaw control at the same time as evolving the control of the other three actuator dimensions.

We tried a variety of network modularizations and hybrid network-PID solutions which ultimately led us to the network used in section V. We were indeed able to evolve stable yaw, but only if the trials lasted for 50 time steps or less. Evolutionary runs using trials longer than 50 timesteps did not even achieve yaw stabilization, and consequently waypoint-following did not occur. We had the clear feeling that the fitness reward for moving forward in the waypoint chain was working against the yaw stability as the length of the trials increased. A variety of alternative fitness functions were tested, always including progress along the track, but also trying out such factors as rotational speed, angular difference between current orientation and due north, and so on. None of these proved to be really successful.

These results suggested the need to split the evolution into two different phases: a first short evolution of the yaw stabilization, which served as a stepping stone for the second phase in which we evolved waypoint-following. To avoid rewarding solutions that had poor yaw stability, a fitness gain for correctness was still included (equation 1) when evolving waypoint-following.

IV. First working approach: Incrementally substituting a PID with neural networks

Given the poor results of the first attempt to evolve a controller based on a full MLP (see section III), we decided to use the PID controller delivered with the simulator to enforce functional separation in the network.

A. Methods

The first phase was the evolution of the yaw controller. For this purpose a very simple neural network, with four connections in all, was evolved to stabilize the yaw. This network was evolved using a fitness function that linearly penalised deviation from the target heading. Each trial lasted for only twenty timesteps, and evolution produced a fairly good solution within several tens of generations. During the second phase, the yaw network was free to evolve along with the rest of the controller.

The second phase was divided into three steps; in all steps the same fitness function was used. In step 1, a three layer MLP was substituted for the PID controller's guidance layer; it took as its input the distances from the waypoint, and the velocity of the helicopter, (both in body coordinates) and had as outputs the desired pitch and roll attitude to the longitudinal and lateral PID's.

In step 2, the two inner PID loops (longitudinal and lateral) were removed and substituted by two separate MLPs. These were evolved to act on the information given by the neural outer layer, and they output helicopter motor commands (longitudinal and lateral cyclic control, and collective pitch). After this step, we had a working helicopter controller consisting solely of neural networks.

In step 3, the outer network layer, which was frozen during step 2, was allowed to evolve further, along with the inner network layer. In this way, the networks could coadapt to each other, potentially allowing them to exploit modes of cooperation not possible for the purely linear PID controllers.

B. Results

The controller based on the inner network reached a good fitness level within 200 generations in all ten replications of the experiment. Evolution of the outer network showed more variability, but within 500 generations an outer network had been evolved in all replications which gave the controller a reasonable, if not good, performance. When both networks

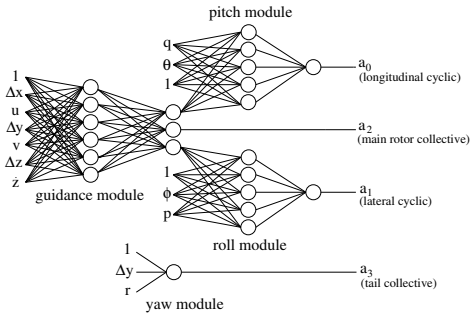

Fig. 1. Topology of the substitution network.

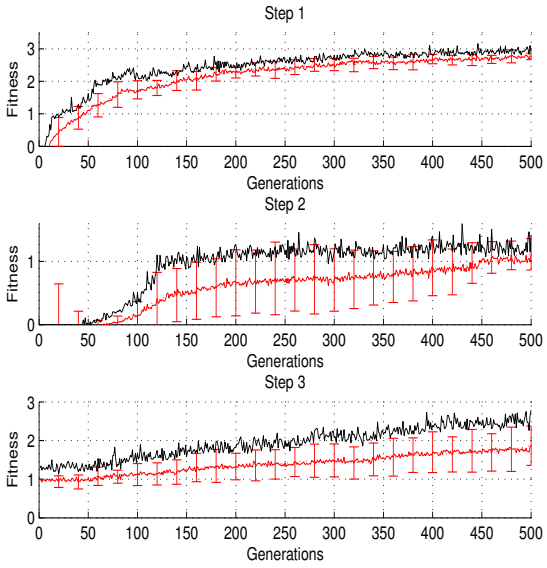

Fig. 2. Evolving the substitution network in three steps. Best fitness (black line) and average of the best fitnesses (gray line) over 10 repetitions of the evolution are shown. Error bars show the standard deviation calculated every 20 generations.

were further evolved together, however, very good performance was reached in every replication (see fig. 2).

V. Second working approach: Incremental/Simultaneous evolution of modular networks

A. Methods

The first phase, evolving a simple yaw stabilizer, was repeated exactly as in the previous working approach.

For the rest of the controller, three relatively simple custom-topology neural networks (fig. 1) were evolved simultaneously using the standard fitness function. The three networks respectively output longitudinal cyclic control, lateral cyclic control, and the collective pitch; the topologies of the networks are depicted in figure 3. The longitudinal network had the following inputs: longitudinal distance to waypoint (Δx), u, q and θ. Similarly, the lateral network made use of the lateral distance to the waypoint (Δy), v, p and ϕ. The collective pitch network used the difference in altitude of the waypoint (Δz) and $\dot{z}$.

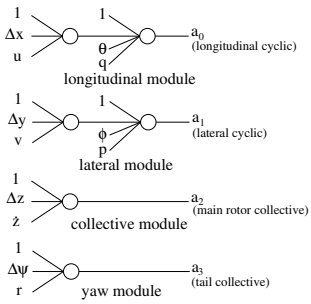

Fig. 3. Topology of the modular network.

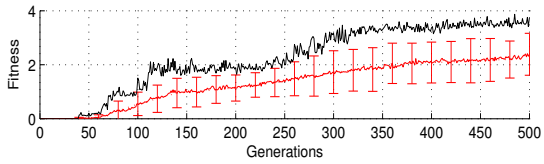

Fig. 4. Evolving the modular network. Best fitness (black line) and average of the best fitnesses (gray line) over 10 repetitions of the evolution are shown. Error bars show the standard deviation calculated every 20 generations.

B. Results

The experiment was replicated ten times. The variability in fitness within the first few hundred generations was quite high, but within 500 generations very good performance was reached in all ten replications, as shown in figure 4.

VI. Evolving PID gains

In order to obtain a controller to serve as a meaningful standard of comparison in our performance tests, we evolved the gains of PID controllers structurally identical to the handcrafted controller shipped with the simulator. The evolutionary runs proceeded by setting all the gains of the controllers to 0, and then first evolving the yaw control component of the PID. The fitness function and trial length were as described in section IV for the yaw stabiliser. Evolution produced fairly good solutions within several tens of generations. Without this preliminary step, the continuous z-rotation of the helicopter prevented evolution from obtaining successful controllers as similarly reported in section III. The rest of the PID was then unfrozen, and all the gains were evolved together to perform the full task. Reasonable performance was usually reached within the first two hundred generations (see fig. 5).

VII. Performance comparison

The fitnesses reached by the evolved controllers during the evolution process can serve as a useful first comparison among the solutions obtained. However, to better understand the peculiarities of the different controllers, additional tests were performed on four tasks differing from the one used in the evolutionary process:

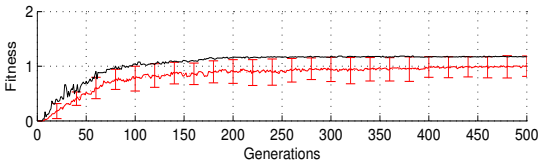

Fig. 5. Evolving PIDs. Best fitness (black line) and average of the best fitnesses (gray line) over 10 repetitions of the evolution are shown. Error bars show the standard deviation calculated every 20 generations.

1) Task 1, Sparse waypoints: The first test was simply a repetition of the task used for the controller evolution, but over a much longer timespan. The extra time allowed for values of P_{chain} bigger than 1.0 since the helicopter was able to fly the whole waypoint chain more than once.

2) Task 2, Close waypoints: Closer waypoints were chosen for the second task, with the average distance between the waypoints now set to 6 feet.

3) Task 3, Sparse waypoints in presence of wind: The waypoints were chosen with the same criteria used for task 1, but an external disturbance in the form of wind gusts was added to the simulation in order to test the robustness of the controller. The wind was simulated as a time varying vector ($[0, 10]$ ft/s) added to the helicopter velocity.

4) Task 4, Sparse waypoints with varying gross weight: This task provided an understanding of the controllers' ability to handle variations in the helicopter's weight (and mass). The maximum random weight variation was set to fifty percent since the payload of a small helicopter can often reach this limit.

In addition, since the compound fitness used for evolution (equation 1) gives only limited insight into the specific abilities of the controllers, three more specific performance indices were developed:

- *progress along the waypoint chain* P_{chain} *(higher is better),*
- *mean deviation from the shortest path* $e_p = \sum_{i=0}^{N} \mid w_h \mid$ *(lower is better),*
- *mean heading error* $e_h = \sum_{i=0}^{N} \mid \psi - \psi_{next} \mid$ *(lower is better).*

Table I shows the results of the tests; the best values are printed in bold. The waypoint layout was randomly generated at the start of every evaluation, and each task was repeated 20 times; the average value (and standard deviation) of the performances obtained is shown (in parentheses). The penalty for crashing into the ground was a fitness of 0, rather than a negative fitness as during evolution. On the wind task, the number of crashes in 20 trials is given; this is a measure of the ability of the controller to maintain stable flight. In all the test tasks the performance was evaluated during a predefined period of 3000 timesteps (corresponding to 60s of flight time).

Figures 6 and 7 show the trajectories of an evolved PID controller and an evolved modular neural network controller respectively, performing the same task (task 1) for 1100 timesteps. The PID exhibited a very conservative strategy, slowing down when still far away from the waypoint and almost stopping when close to it. The network-based controller instead retained a better control over the helicopter speed that produced a more linear trajectory and better progress along the waypoint chain.

TABLE I

PERFORMANCE OF THE VARIOUS CONTROLLERS ON DIFFERENT TASKS

	Handcrafted PID	Evolved PID	Substitution Network	Modular Network
Close waypoints				
P_{chain}	0.96 (0.077)	1.85 (0.091)	2.27 (0.182)	**2.38** (0.106)
e_p	0.95 (0.216)	0.40 (0.048)	1.04 (0.21)	**0.34** (0.044)
e_h	**0.005** (0.003)	0.010 (0.002)	0.457 (0.002)	0.013 (0.001)
Sparse waypoints $[10, 25ft]$				
P_{chain}	0.46 (0.121)	0.74 (0.273)	1.35 (0.127)	**1.64** (0.081)
e_p	1.61 (0.265)	0.84 (0.192)	2.5 (0.748)	**0.63** (0.242)
e_h	**0.007** (0.001)	0.015 (0.004)	0.459 (0.003)	0.018 (0.002)
Sparse waypoints with wind $[0, 10ft/s]$				
P_{chain}	0.27 (0.231)	0.49 (0.330)	0.95 (0.434)	**1.07** (0.565)
e_p	1.99 (3.705)	2.58 (2.880)	3.54 (3.097)	**1.33** (1.651)
e_h	0.069 (0.083)	0.160 (0.216)	0.430 (0.146)	**0.062** (0.049)
Crash	7	2	2	4
Sparse waypoints with variable weight $[\pm 50\%]$				
P_{chain}	0.54 (0.128)	0.83 (0.235)	1.3 (0.170)	**1.53** (0.137)
e_p	1.68 (0.271)	0.80 (0.153)	2.28 (0.503)	**0.618** (0.112)
e_h	**0.012** (0.002)	0.017 (0.005)	0.48 (0.02)	0.031 (0.017)

Generally, the modular network evolved in section V performed best in all four variations of the task, as it always progressed the furthest of the four controllers along the waypoint chain, and always moved in a more or less straight line to the waypoint. The substitution network was a close second as far as progress along the chain was concerned, but had a more mixed performance when it came to deviation from the shortest path, and was the only controller that has any significant problems keeping the desired heading.

The performance of the PID controllers was much worse than the neural network controllers when it came to progress along the waypoint chain, except in the situation with the waypoints closer together, where its performance was comparable to (if slightly lower than) the neural networks. This suggests that the reason for the good performance of the neural network controllers was not only the superiority of evolutionary tuning over manual tuning, but also the lack of nonlinearity in the PID controllers. It simply does not seem possible to evolve a purely linear controller than is robust enough to perform well over the various task variations and performance measures we used here.

VIII. ANALYSIS

Comparing the results of the various approaches we have tried, it is obvious that the task and controller representations need to be divided up in some way in order for evolution to be able to produce adequate controllers. Both when evolving monolithic controllers building on a PID, and when evolving modular controllers sequentially, the task was divided up on a temporal basis, yielding a form of incremental evolution. In both cases, the controllers could also be said to be divided spatially, in a mixed serial/parallel way in the first approach, and in a parallel separated fashion in the second case.

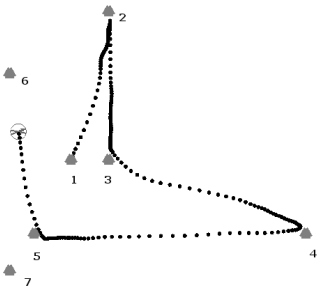

Fig. 6. Trajectory of an evolved PID after completing 1100 timesteps of a typical sparse waypoint task. In this particular run the progress along the path (after 3000 timesteps) was 1.14, the mean trajectory error 0.39 and the mean heading error 0.019. The dots mark the position of the helicopter every 10 timesteps.

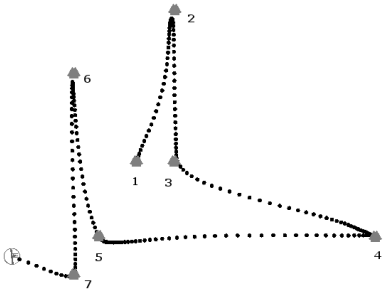

Fig. 7. Trajectory of a modular network controller after completing 1100 timesteps of the sparse waypoint task used in Fig.6. At the and of the run (3000 timesteps) the progress along the path was 1.70, mean trajectory error 0.38 and mean heading error 0.016.

A. The case for unified tasks and controllers

Helicopter control is a problem where the various input dimensions are strongly coupled in a nonlinear fashion. Changing one actuator level - such as the collective - most often requires a compensatory adjustment of the other actuator levels. This implies that the best control would be achieved by an integrated controller in which the parts of the controller responsible for the different output representations could communicate with each other; this is in direct contrast to the state of affairs when using parallel modular networks, as in our second working approach.

In the context of simpler control problems than helicopter control, it has been argued that as many decisions as possible about the structure of the controller should be left to the evolutionary process [17]. This is because the description of behaviour from the perspective of the agent's sensorimotor system, the *proximal description of behaviour*, is inherently more suited to such decisions than the human observer's *distal description of behaviour*, which is couched in high-level terms which may not be appropriate to the specific problem at hand.

B. The effects of modularity and task division

Given these indications that a minimum of human tampering in the structuring of the controller and fitness function might be desirable, it remains to be explained why we found the exact opposite in this case.

1) Incremental evolution: Incremental evolution is an established technique for evolving solutions to hard problems in control [6][8][22]. This technique developed as a response to the fact that, for hard problems, all individuals of the first generation are likely to have the same very low fitness, and to be far from any region of the fitness space where fitness would increase; they are effectively in a flat region of fitness space. Incremental evolution solves this problem by dividing up the task into simpler subtasks, creating a "pedagogic" sequence of tasks leading to the full task.

2) Neural interference: The paper by Calabretta *et al.* [5] addresses the issue of evolving behaviourally complex organisms. Their arguments are based on experiments attempting to evolve neural networks for an abstract neuroscientifically-inspired task, in which the network has to perform two related but different tasks using the same input. Two reasons for the poor performance of monolithic neural networks on such tasks are advanced: neural interference, and genetic interference. Neural interference is the consequence of the fact that, if the same neural connections can potentially be used by more than one mechanism, they probably will be so used. Whichever mechanism is learned first might exploit those connections in such a way that a second mechanism cannot be learned without disrupting the first mechanism. This effect can be eliminated, as Calabretta *et al.* and others have shown, by dividing up the network into modules that are interconnected only when they really need to be.

3) Genetic interference: Genetic interference is a consequence of genetic linkage. With any reasonably high mutation rate, several mutations are made in every generation at different positions in every genome. This means that a beneficial mutation in one part of the genome is likely to be accompanied by a disadvantageous mutation in another part of the genome, meaning that the individual with this genome gets a low fitness, and that the beneficial mutation is likely to be lost. Calabretta *et al.* managed to alleviate this problem to some extent by coevolving two separate neural modules, each responsible for implementing one of the two mechanisms required to solve their task; a similar approach is used in the SANE neuroevolution method, amongst others [14].

4) Search space dimensionality: Another factor which should not be forgotten is the sheer reduction in search space dimensionality due to the absence of intermodular connections. In a fully connected neural network, we have $O(n^2)$ connections; in a network consisting of several modules with only limited interconnections this number will in general be much smaller.

C. An expressivity/learnability tradeoff?

There are thus ample theoretical indications that the division of tasks and networks into subtasks and subnetworks that proved successful in our experiments above should actually improve the chances of evolving good solutions. However, given the nature of the control problem, this strategy may

at the same time decrease final fitness, as the human designers will have imposed constraints on the evolutionary mechanism. This points to a trade-off between the ease of development and the theoretical maximum performance of a controller when developed by an evolutionary algorithm. This trade-off echoes a similar trade-off when designing controllers manually, as described in section I-C, a similarity which might be more than coincidental.

IX. Conclusion

A. Competitiveness of the controllers

Both of the two approaches we tried for evolving neural networks generated very capable controllers that significantly outperformed the human-designed PID controller. They also outperformed the attempts by the authors of the article to control the vehicle manually. It is interesting to note that the evolved neural networks were quite robust when the parameters of the task, vehicle and environment were varied, something that could not be said about the PID controllers. This may be due to two factors: evolution is better at tuning weights and gains than are humans; and the nonlinearity of neural networks makes them better suited for handling such variations than linear controllers. Such robustness is obviously important when transferring controllers to real vehicles.

B. Effects of modularity and incrementality on evolvability

Without incorporating some domain knowledge in the evolutionary process, we have not been able to evolve a controller for doing anything more advanced than not crashing. Domain knowledge has been introduced either by using parts of a hand-designed PID controller, or by using knowledge of which inputs should be relevant to which outputs. Another form in which human judgement (though not really domain knowledge) was used was that the yaw control network always had to be evolved before the rest of the controller. Similar phenomena have been reported by several researchers in evolutionary neural networks, who have found both modularity of the controller and incrementality of the task to be necessary for evolving solutions to complex problems [22][5]; they can be at least partially explained with the models they have provided. But this approach also resonates well with common wisdom in manual controller design: the simpler and less interconnected a controller is, the easier it is to tune. Whether and how artificial evolution can be made to overcome this phenomenon is still an open research question.

C. Future work

The work presented in this paper aimed at developing a methodology for the UltraSwarms project, which will now go on to develop the physical helicopter platform further, to refine the system identification method, and to try the method proposed in this paper on the real vehicles. Ultimately, the project aims to use the neural networks evolved in this way as the lowest software layer in a complex system supporting flocking and cluster computation. As a related research topic, it could be interesting to investigate the performance of algorithms that allow neural network topologies, including modular separation, to be evolved together with the connection weights; the helicopter control problem is a challenging problem with obvious real-world applications, and a successful solution could have wide applicability.

References

[1] http://autopilot.sourceforge.net/.

[2] http://model.hirobo.co.jp/products/0301-905/index.html.

[3] P. Abbeel, V. Ganapathi, and A. Y. Ng. Modeling vehicular dynamics, with application to modeling helicopters. In *Neural Information Processing Systems*, December 2005.

[4] G. Buskey, J. Roberts, P. Corke, and G. Wyeth. Helicopter automation a using a low-cost sensing system. *Computing & Control Engineering Journal*, 15(2):8 – 9, April-May 2004.

[5] R. Calabretta, A. Di Fernando, G. P. Wagner, and D. Parisi. What does it take to evolve behaviorally complex organisms? *BioSystems*, 2002.

[6] D. Cliff. Incremental evolution of neural network architectures for adaptive behaviour. Technical Report CSRP 256, University of Sussex, Brighton, UK, 1993.

[7] R. De Nardi, O. Holland, J. Woods, and A. Clark. SwarMAV: A swarm of miniature aerial vehicles. In *Proceedings of the 21st Bristol International UAV Systems Conference*, April 2006.

[8] F. Gomez and R. Miikkulainen. Incremental evolution of complex general behavior. *Adaptive Behavior*, 5:317–342, 1997.

[9] F. Hoffmann, T. J. Koo, and O. Shakernia. Evolutionary design of a helicopter autopilot. In *3rd On-Line World Conf. on Soft Computing (WSC3)*, 1998.

[10] O. E. Holland, J. Woods, R. De Nardi, and A. Clark. Beyond swarm intelligence: The UltraSwarm. In *Proceedings of the IEEE Swarm Intelligence Symposium (SIS2005)*. IEEE, 2005.

[11] M. La Civita, W. C. Messner, and T. Kanade. Modeling of small-scale helicopters with integrated first-principles and system-identification techniques. In *American helicopter society 58th annual forum*, 2002.

[12] M. La Civita, G. Papageorgiou, W. C. Messner, and T. Kanade. Design and flight testing of a high-bandwidth $\mathcal{H}_\infty$ loop shaping controller for a robotic helicopter. In *Proceedings of the AIAA Guidance, Navigation, and Control Conference*, number AIAA-2002-4836, 2002.

[13] B. Mettler, M. Tischler, and T. Kanade. System identification of a model-scale helicopter. Technical Report CMU-RI-TR-00-03, Robotics Institute, Carnegie Mellon University, Pittsburgh, PA, 2000.

[14] D. E. Moriarty and R. Miikkulainen. Efficient reinforcement learning through symbiotic evolution. *Machine learning*, 22:11–32, 1996.

[15] A.Y. Ng, A. Coates, M. Diel, V. Ganapathi, J. Schulte, B. Tse, E. Berger, and E. Liang. Autonomous inverted helicopter flight via reinforcement learning. In *Proceedings of the International Symposium on Experimental Robotics*.

[16] A.Y. Ng, H.J. Kim, M.I. Jordan, S. Sastry, and S Ballianda. Autonomous helicopter flight via reinforcement learning. *Advances in Neural Information Processing Systems*, 2004.

[17] S. Nolfi. Evolutionary robotics: Exploiting the full power of selforganization. *Connection Science*, 10(3-4), 1998.

[18] S. Nolfi and D. Floreano. *Evolutionary robotics*. MIT Press, Cambridge, MA, 2000.

[19] M. G. Perhinschi. A modified genetic algorithm for the design of autonomous helicopter control system. In *Proceedings of the AIAA Guidance, Navigation and Control Conference*, pages 1111–1120, 1997.

[20] D. H. Shim, H. J. Kim, and S. Sastry. A flight control system for aerial robots: Algorithms and experiments. In *IFAC Control Engineering Practice*, 2003.

[21] P. Spronck. *Adaptive Game AI*. PhD thesis, University of Maastricht, 2005.

[22] J. Togelius. Evolution of a subsumption architecture neurocontroller. *Journal of Intelligent and Fuzzy Systems*, 15:15–20, 2004.

[23] J. Togelius and S. M. Lucas. Evolving controllers for simulated car racing. In *Proceedings of the Congress on Evolutionary Computation*, 2005.

[24] Xin Yao. Evolving artificial neural networks. *Proceedings of the IEEE*, 87:1423–1447, 1999.