Robustness Analysis of Evolutionary Controller Tuning using Real Systems

Mario A. Gongora, Benjamin N. Passow, and Adrian A. Hopgood

***Abstract*— A genetic algorithm (GA) presents an excellent method for controller parameter tuning. In our work, we evolved the heading as well as the altitude controller for a small lightweight helicopter. We use the real flying robot to evaluate the GA's individuals rather than an artificially consistent simulator. By doing so we avoid the "reality gap", taking the controller from the simulator to the real world. In this paper we analyze the evolutionary aspects of this technique and discuss the issues that need to be considered for it to perform well and result in robust controllers.**

I. Introduction

The demand for autonomous helicopters from industry, military and in the civil sector has been growing rapidly [1]. One reason for this is the fact that helicopters are versatile in their maneuverability. Unfortunately, this also makes the platform very difficult to control. Therefore, much research is done in this area, often with rather large helicopters with a rotor span of more than a meter [2], up to rotor spans of over 3 meters [3]. These large helicopters present an obvious safety risk and also emit fumes, are loud, and test set-ups and experiments are complex. In our work, we use a very small and lightweight helicopter. This platform can be used indoors, is low cost, less of a safety risk, and very flexible in its application. We will refer to this platform as *Flyper* – flying performing robot – as this is its name in various projects.

In our work we present ongoing research in achieving stable and robust control for this indoor helicopter. We applied a genetic algorithm (GA) to find PID control parameters of the heading as well as the altitude controllers running on the embedded system. Instead of using an artificially consistent simulator we used the actual robot to evaluate the control parameters' fitness. In this paper we present and discuss our results from the analysis of executing GA using real systems to evaluate the fitness, and their relevance in creating robust controllers.

We will use results presented elsewhere in combination with some new results presented in this paper, and perform an analysis on the critical issues we have identified, mainly: the measurement of fitness, performance of the evolution process, and the robustness vs. accuracy of the final controller.

The remainder of this paper is organized as follows. In section II we provide an overview of related work in terms of GA tuned controllers and execution of GA in real systems. In section III we will present and discuss the structure of the GA we used for our experiments and in section IV the setup to execute the fitness function on the real platform (*Flyper*). In section V we analyze the results and section VI concludes the paper and suggests further work.

II. Background

First, let us provide a succinct introduction to the platform we have used to experiment so that it can be seen in the context of this paper. The autonomous helicopter is based on a Twister Bell 47 small indoor model helicopter. It is a coaxial rotor helicopter with twin counter-rotating rotors (it does not have a tail rotor) with 340 mm span, driven by two motors, and uses two servos to control the rotor blades' plane angles. The weight of the helicopter in its original state is approximately 210 grams without battery and it can lift up to 120 grams. This helicopter has six degrees of freedom (DOF) controlled by four inputs. It can fly for approximately 10 minutes with its standard battery. Figure 1 shows the helicopter diagram and its 6 degrees of freedom.

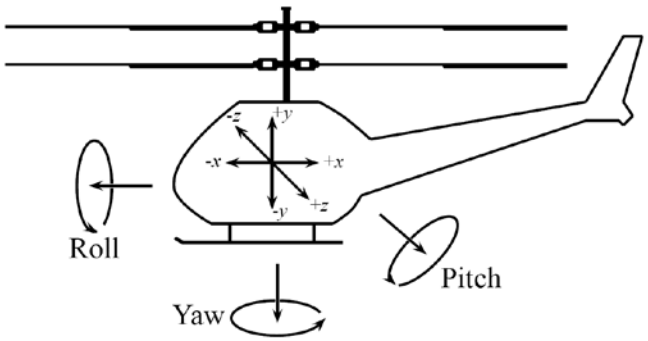

Fig. 1. Diagram showing the six degrees of freedom of the helicopter used in this paper.

The controller deals with the four inputs to this system: two position values for the actuators that control the plane of the rotors, and two power values for the motors that drive each rotor. The actuators that control the rotors' plane,

Mario A. Gongora is with the Centre for Computational Intelligence (CCI), De Montfort University, Leicester, UK, (phone: +44 116 207 8226; email: mgongora@dmu.ac.uk).

Benjamin N. Passow is with the Institute of Creative Technologies (IOCT), De Montfort University, Leicester, UK, (email: benpassow@dmu.ac.uk).

Adrian A. Hopgood is a Professor and Dean of the Faculty of Technology at De Montfort University, Leicester, UK, (email: aah@dmu.ac.uk).

directly affect the pitch and roll degrees of freedom, which in turn affect the *x, y* and *z* positions. The rotor driving motors affect the *y* position in terms of their total combined power, and the yaw in terms of their difference. This shows the complexity of the control process for this particular platform in which some DOFs are controlled directly (*y*) and others indirectly (yaw) relative to the control actions, in addition to the fact that most DOFs are inter-dependable (roll and pitch with *x, y* and *z*).

The original radio controlled system has been replaced by an onboard single chip embedded computer. All the control programs run in this onboard computer, which can communicate with a desktop computer via Bluetooth for telemetry purposes.

A. Control Architecture

Computational intelligence methods have been applied and well studied in controlling unmanned aerial vehicles (UAV) capable of vertical take-off and landing (VTOL) [4, 5, 6, 7]. Others have shown that traditional control techniques, such as proportional-integral-derivative (PID) controllers, work well in VTOL control [8, 9, 10].

Kadmiry and Driankov describe a TSK-type fuzzy system to control the altitude and attitude of a small size, unmanned helicopter [4]. They use a mathematical model obtained from an existing platform to model the TSK fuzzy system and consequent parameters. It is stated that the large class of non-linear plants can well be represented by the TSK models with only minor changes. On the other hand, the formal system identification can be very difficult for a complex system such as a helicopter. Using a PID control technique, only a few certain gain and border parameters need to be identified.

Puntunan and Parnichkun introduce a heading direction and floating height controller for a single rotor helicopter [10]. The control system uses a proportional plus derivative controller (PD) to maintain the helicopter's heading and height, while a human pilot controls the horizontal movements remotely. Puntunan and Parnichkun present test results that confirm stable controlling capability with a relative small margin of error.

Sanchez, Becerra, Velez present in [8] an unmanned helicopter control system combining a Mamdani type fuzzy logic controller [11] with PID controllers. The Fuzzy Inference System (FIS) controls the translational movement while the PID controllers handle the altitude and attitude of the helicopter. The system was tested via simulation for hovering and slow velocities which showed good performance.

Saripalli, Montgomery, and Sukhatme introduce an autonomous helicopter which uses differential GPS, an inertial measurement unit (IMU), and a sonar sensor to determine the helicopter's position and attitude [9]. Their system is based on PI controllers. Seven test flights confirm the successful control and landing of the helicopter. This work shows that PI controllers work well and the integral control part is very useful in helicopter control.

B. Evolving Controllers

Identifying and tuning control parameters for such vehicles can be a challenging task. Fleming and Purshouse present in [12] a survey of EC in control systems engineering. A wide spectrum of control related applications are presented including a section on parameter optimisation and on-line applications. It is discussed that few real-time applications use EC methods for control. Additionally, it is mentioned that little work shows actual results rather than simulated results. A simulator of the corresponding system is very often used in order to evaluate the individuals' fitness within a GA.

Jacobi, Husbands, and Harvey discuss the "reality gap" and research the effect of noise in a simulator to evolve control systems for a Khepera robot [13]. A simulator's limitations need to be identified so that it does not describe properties that do not exist in the real world or does not ignore properties that are essential to the real world [14]. The authors show that the control system can be successfully evolved for the real robot using only a simulator but that great care must be taken when implementing the simulator while adding the right amount of noise. Further, they state that building a simulator for such a simple robot system is much easier than for most other robots. A formal model for the helicopter used in this work is clearly much more complicated than for the Khepera robot.

C. Evaluation in Simulation

Sekaj and Sramek present methods based on GAs for the design of robust controllers [15]. The methods are applied to a nonlinear differential equation and compared to other methods, all in simulation. The results are promising, but no application other than in simulation has been presented.

In [16], Shim, Koo, Hoffmann, and Sastry present a comprehensive study of control design for an autonomous helicopter. Three different control methodologies are compared and discussed: linear robust multi-variable control, nonlinear tracking control, and fuzzy logic control with evolutionary tuning. A GA is used to identify and tune the consequent parameters of four controllers using fitness evaluated in a simulation. The controllers are designed and evaluated on an artificial model created from aerodynamics models.

Perhinschi [17] used a GA to identify the gain parameters of linear differential equations which are used to stabilize and control a helicopter's longitudinal channel. The results of four different GA strategies are compared by three criteria employing the fitness of the best individuals. The GA used a linearised model of a helicopter and the controller performance was not tested in simulation or on a real system.

Mao shows in [18] a robust flight controller for a helicopter evolved using a GA. The H-infinity mixed sensitivity design approach is used for the development of

the controller. The GA evolves the design parameters based on a mathematical model and the final results are tested only in simulation.

In [19], Zufferey et al. discuss the "reality gap" and propose a methodology for creating a simulator from a formal model. The system is tested on a blimp, a real indoor airship, and the results are discussed. Although the results for the simple navigational task are promising, this method is useful only to this specific type of vehicle which is much simpler and more stable than a helicopter system.

D. Evaluation on the actual System

Ahmad, Zhang, and Readle present an on-line GA-tuned PI controller system [20]. In this paper they present a system for tuning a heating system's controller, which is optimised in between control cycles. This is possible due to the slow response time required by such a system due to its high thermal inertia.

Nolle et al. present a simulated annealing (SA) approach to parameter identification where solutions are evaluated on the actual system [21]. The approach shows promising results outperforming trained experts in terms of time needed and fitness of the results.

Phillips, Karr, and Walker introduce a fuzzy logic based flight controller for a UH-H1 "Huey" helicopter [6]. A GA is used to find the parameters of the fuzzy controllers, evaluating the individuals on a formal numerical model of the helicopter. The resulting controller is tested in simulation and on the actual helicopter. The tests on the real system showed oscillations and a small problem in the design with the fuzzy logic controller where the simulation showed no problems.

III. Genetic Algorithm Implementation

Any evolutionary search bases its performance on the evaluation of the fitness of the individuals (chromosomes) it is manipulating; in most cases, this evaluation is achieved with the help of a simulator. In fact, this part of the GA execution has traditionally been done in the computer where the GA is running, using either the calculations of the real problem (if it is computer based), a data set (of historical data collected about the problem), or a simulator (if it is an external or hardware system). For this last case the main problem is to get a suitable simulator which can only be as good as the model used to create it. Getting a model then involves solving the issues of system identification and accuracy of the model vs. computational resources required to simulate it.

In the work presented here, rather than using a simulator, we use the actual system for the evaluation of the individuals for the GA. The GA itself runs on a host computer, for every generation, individual chromosomes are transferred to the system to be controlled using a communication link. The chromosomes are tested in the real target system where the controllers' response is recorded; these results are transmitted back to the host computer to perform the fitness evaluation. This section gives details on how the GA has been configured.

For the purpose of this study we started with two PID controllers: one for controlling the heading of the helicopter (i.e. controlling the yaw DOF) and one for controlling the altitude (i.e. controlling the *y* DOF). These two DOFs have the characteristic that they are controlled indirectly by two control actions (power to each motor) that have to interact to achieve two single, yet inter-dependent effects.

A. Solution Encoding

To tune a controller for *Flyper* we encoded five integer values in the range specified in table I in the GA's chromosome. The values correspond to the PID parameters for a particular controller. Limiting the search space to these values has been determined by taking a multiple of the parameters that have been identified creating an initial ad hoc hand-tuned controller.

TABLE I
GA'S PARAMETER VALUE RANGES

Parameter	Chromosome	Heading PID	Altitude PID
Proportional gain	0 – 200	0 – 2.00	0 – 2.00
Integral gain	0 – 100	0 – 1.00	0 – 1.00
Integral state max	0 – 16	0 – 400	0 – 400
Integral state min	0 – 16	0 – 400	0 – 400
Derivative gain	0 – 400	0 – 4.00	0 – 40.0

B. Initial Population

The initial population for the GA could be primed using initial "semi-optimal" solutions such as based on the hand tuned parameters to kick-start the optimization process. We decide against this for two main reasons. First, we do not know if the hand tuned control parameters are near-optimal or if they could lie in a local minimum of the search space. Secondly, we want to evaluate the performance of the GA rather than the controller, so we initialized the system with a fresh and widespread random population. All initial individuals are created with random values within the range specified in table I. The population size of 20 was chosen, as it is small enough to have a reasonable evaluation of each generation while providing enough individuals to maintain variety.

C. Fitness Function

Each individual's fitness is evaluated on the real system rather than in a simulated environment. The evaluation function is shown in equation 1.

$$e = \sum_{t=0}^{n} (r_t - s)^2 \qquad (1)$$

where e is the measure of error, t is the control cycle, n is the total number of control cycles for the complete evaluation, r_t is the reading at control cycle t and s is the set-point. Squaring the current error increases the selective pressure on the individuals causing the GA to find better solutions quicker. For example, if an individual has an error of 5 and another has an error of 10, the first would have a fitness 4 times higher than the second. In this way, squaring the error within the fitness function provides an exponentially higher penalty to larger errors. The sum of all the squared errors is the measure used to determine the final fitness of an individual. The equivalent fitness is inverse proportional to this measurement of error.

D. Selection

The selection of individuals to "survive" to the next generation is an important part of the GA. Our selection method is based on the roulette wheel strategy but without the possibility that an individual is chosen more than once. This method enables even the weakest individual to be chosen, although fitter individuals are more likely to be selected.

E. Genetic Operators

In this work, elitism is applied, which means that the best individual of every generation is automatically copied to the next generation without the need to be selected first. In combination with this, 20% of the old population's individuals are copied to the next generation using a roulette wheel like system.

For the crossover operator, two individuals are selected to generate one offspring. This new individual is created by taking the mean of every chromosome's loci of the parents. This method is applied in order to get 40% of the new population.

Mutation is the source of new variety. In this work, a probabilistic random mutation is used on every loci of the selected individuals to form 40% of the new population. This method of mutation uses a bell shaped probability where the chance of a small mutation to take place is higher than the chance for a big mutation to take place.

F. Termination Criteria

Often, there is a termination criterion in place where the GA is stopped when a certain fitness, by one or more individuals, is reached. Additionally, another termination criterion often used is where the GA is stopped when no increase in fitness is found within a defined number of generations. Because in this work we are studying the behaviour of the GA applied to a real world system at this stage we will only use a time-out as the termination criteria, stopping the GA only after a specific number of generations is reached.

In a typical run, we let the evolution execute 30 generations which resulted in 600 individuals tested per run.

IV. System Setup

The embedded on-board computer consists of a single-chip microcontroller; it receives data from three sonar sensors arranged to determine the attitude and altitude from a smooth surface, and an electronic compass to determine the heading. A bluetooth module provides a communication link between the microcontroller and a host computer. This link can be used to stop the helicopter in case of an emergency but more importantly, it provides the means for a host computer to gather flight telemetry for performance analysis and to send control parameters to be tested by the helicopter.

The program running on the on-board computer is capable of reading all sensor data and calculating the four actuator outputs using four separate PID controllers, in average, 13 times a second.

Since finding good PID control parameters is a challenging task [22], we used the following evolutionary technique to tune the heading and altitude control parameters.

A. Tuning the Heading Controller

To tune the heading control parameters, the helicopter was attached to a ball bearing supported turntable. This stand restricted its movement to turn to between 90 and -90 degrees from its middle position (at 0). The evaluation of one individual took about 20 seconds and the system got another 20 seconds to cool down, before the next individual was evaluated. Each individual was tested by perturbing the helicopter to each side and analyzing the controller's reaction attempting to reach the set-point 0.

In an initial step, an individual's chromosome (a set of parameters for the controller) is sent from the host computer to the embedded controller using the communications link. Thereafter, the helicopter starts the motors and the controller reacts on the heading error based on the parameters received. Every control cycle, the value of the current heading is sent back to the host computer running the GA to evaluate the current individual using (1).

The helicopter is initially perturbed by 90 degrees from the set point by driving the two rotors with different power levels. The helicopter turns but cannot go beyond 90 degrees as the experimental setup physically blocks it there. At this point the controller takes over and starts responding while being evaluated by the host computer. After 92 control cycles the evaluation and controller are paused and the helicopter is perturbed -90 degrees, i.e. into the other direction. The controller and its evaluation start again.

This setup, in combination with the GA running on a host computer, enables the automatic implementation and evaluation of individuals and thus the execution of the GA on the real system without any human intervention.

B. Tuning the Altitude Controller

Tuning the altitude controller of the helicopter requires the robot to fly relatively freely while keeping the other

controllers influence as small as possible. For this we built a stand that allows the small helicopter to take off and fly in a height of up to 1.4 meters while being fixed in yaw, roll, and pitch angles. The stand has been made from small aluminum rods to keep the mass at a minimum. The weight of the stand is effectively neutralized by using a spring system. The pulling force of the spring system is equal to the spring constant *D* multiplied by the extended length of the springs. To keep the weight neutralizing pulling force as constant as possible, the extended length of the spring system changes very little over any altitude of the stand. The helicopter's weight, while being attached to the stand, is not increased. Still, the mass of the helicopter has increased by the small mass of the stand's arm.

In order to keep the helicopter from over-heating, we stopped the GA from time to time for a few minutes. In the case of the altitude control tuning we were not able to use a fan to cool down the helicopter in between the evaluations as it would have been detected by the altitude sonar sensor. This resulted in much longer execution times for the whole process. In all, we were able to execute less runs of the altitude controller tuning than for the heading controller.

The same GA strategy that was used for tuning the heading controller was used to tune the altitude controller. Every GA individual is evaluated by its performance reaching and keeping at predefined altitude set-points. First, the helicopter needs to reach an altitude of 90 cm for 10 seconds, then the set altitude is reduced to 50 cm for another 10 seconds, and finally the helicopter is supposed to land by continuously and slowly reducing the height. The measure of error is again the sum of all squared errors meaning that bigger errors result in a far worse fitness compared to smaller errors.

For both experimental setups we connected an external power supply to power the helicopter over the duration of the GA.

V. Results

The two controllers (heading and altitude) where tuned separately using the setup described in the previous section. Then they were evaluated by letting *Flyper* fly freely, using both controllers at the same time. In this section, first we will analyze the performance of the GAs while running, and then we will analyze the performance of the final controllers in actual flights.

A. Performance of the GAs

Figure 2 and 3 show the typical performance of the GAs in terms of fitness of the best individual (inverse to the measure of error) of each generation when evolving the control parameters for both controllers. Figure 2 shows the evolution of the best individual for the heading controller. Figure 3 shows the evolution of the best individual for a run of the altitude controller.

The main aspect to observe here is that, although we used elitism, the best fitness evolution didn't occur in a monotonic way as it should be in a computer based GA. When using elitism, each time a generation finishes, the best individual is saved for the next generation; in an artificially consistent system the main effect should be that the best fitness in a given generation is never worse than in the previous generation. Between generations we would then expect either to find the same *best* individual or to get a new better one.

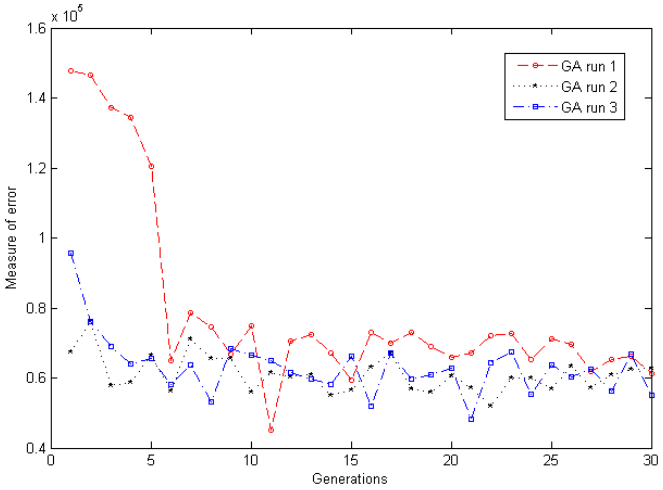

Fig. 2. Evolution of the best individual for 3 runs of the heading controller tuning.

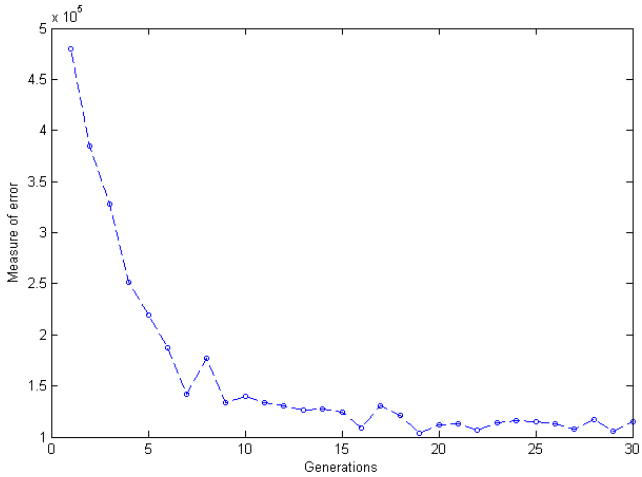

Fig. 3. Evolution of the best individual for a run of the altitude controller tuning.

Because the fitness of these chromosomes is being evaluated using *Flyper*, we have seen now that due to the high instability, noise, and sensitivity of the system, we get a variability in the results which causes the fitness of a particular chromosome to be different in different tests. Re-evaluating a known individual in an artificially consistent system would be redundant as the same exact fitness would be found. In contrast, in the case of our setup, we need to re-evaluate every individual as it will most probably result in a slightly different fitness, possibly worse. The main issue being that we need to make sure the individual not only has a

high fitness in a particular case, but is consistent across generations; in other words, that is not only a good solution in one test, but a robust solution across many tests.

As the individuals evolve, we can clearly see an improvement in the controller but with variable performance of every individual. An important aspect is that in any case, we are getting better individuals, and more importantly, we are getting slightly less variability. We can see that it is unlikely for the error to become much lower, and of course, impossible to be eliminated. At the end of the GA execution we will have to choose a *best* individual as the final outcome to the GA.

Normally, choosing the best solution at the end of the GA is a trivial task: juts get the individual with the highest fitness. In our case now it becomes an issue to analyze carefully. Now we don't just have a best individual, as we have to consider the possibility that if we run the current individuals again, we might find a different best individual within the same population. We decided then to test a few of the best individuals of a completed GA run, for a number of times each, and rather than measuring the exact fitness in each run, we looked at their repeatability. This gave us the individual that would perform the best controlling *Flyper* in general situations, and was found among the best individuals of the last population as reported in [23]. In doing this, we are finding that a GA that evaluates the fitness in real systems does not only evolve towards better fitness, but it evolves individuals that are more robust during repeated tests. From the results analyzed and presented in [23] we can see that the better the average fitness of a population, the better its repeatability.

We have to conclude that although the fitness function is not considering repeatability explicitly, the population is still evolving toward it. This can be explained by the fact that testing the individual using the real system, across generations, results in the effect of testing the individuals for robustness. This is demonstrated by the fact, that even using elitism, the *best* individual of one generation cannot be guaranteed to survive into the next, even if no new *best* individual is created, as another individual from the same population could have a better fitness if all were tested for a second time.

This effect proves as well, that we do not need to include robustness tests explicitly in the fitness function to achieve this. It is a matter of discussion of which method would be more efficient: measuring robustness as part of the fitness, or making up for this indirectly by instead using more generations. This would be the next step to analyze in this respect.

B. Performance of the final Controllers

After choosing a final *most robust* individual from each controller, *Flyper* was freed from the experimental arrangement to restrain its movement. The two controllers were executed simultaneously in the on-board computer and the helicopter left to fly autonomously for a short time.

The initial tests provided some mixed results. When flying unrestrained and with combined controllers, two main problems appeared: the heading controlled developed an offset error, and the altitude controller became very unstable. Figure 4 shows the altitude and the altitude control command. Figure 5 shows the heading and the heading control command.

The altitude oscillations were expected to severely affect the heading controller, as the heading depends on the power setting of the rotors.

Analyzing the behavior of the altitude controller, we determined that the GA had evolved a too high integral value for the PID controller. During evolution, the helicopter was attached to the altitude experimental set up restrain. Although the spring system successfully eliminated the gravitational pull of the stand weight, its mass was still there affecting the inertia of the helicopter when trying to gain altitude, the integral term was then exaggerated to help the controller respond to the time lag produced by the extra mass. Once that mass was eliminated from the system, the integral gain caused the system to become unstable.

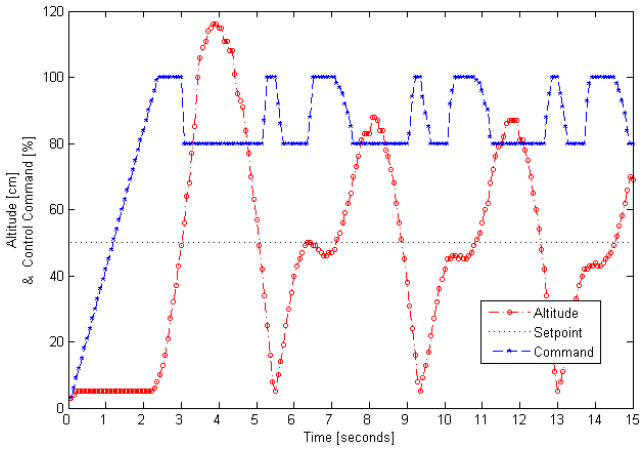

Fig. 4. Performance of the altitude controller with *Flyper* flying freely running both controllers.

To test this we eliminated the integral gain from the controller and in effect, it achieved the results we were expecting. Figure 6 shows the altitude when controlled with only PD gains.

The system behaved much better at this stage. The reality gap became apparent again, as although we were using the real system, the need to restrain it to avoid accidents produced changes to the system that in turn resulted in similar effects to that of using a simulator. But not all was as expected. Surprisingly, the heading controller didn't change that much.

We were expecting the heading to have changed as much as the altitude controller, not only for the lack of restrain, but because of the effect produced by the altitude severe oscillation. The performance of the heading controller and

the offset error seen in figure 5 remained comparable. As expected, the heading performance was better with the new altitude control parameters, but not the overall robustness of it.

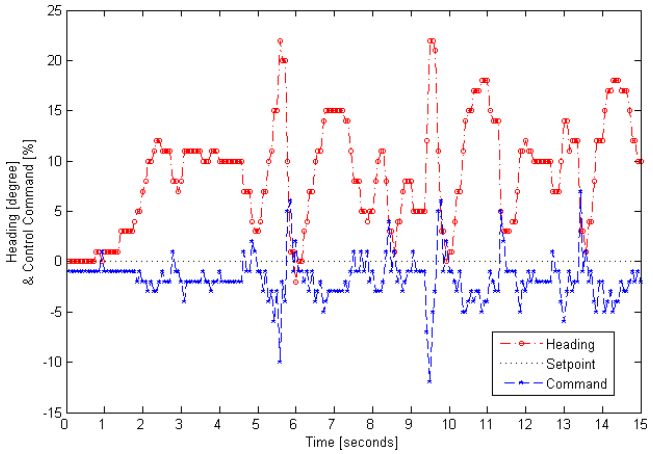

Fig. 5. Performance of the heading controller with *Flyper* flying freely running both controllers.

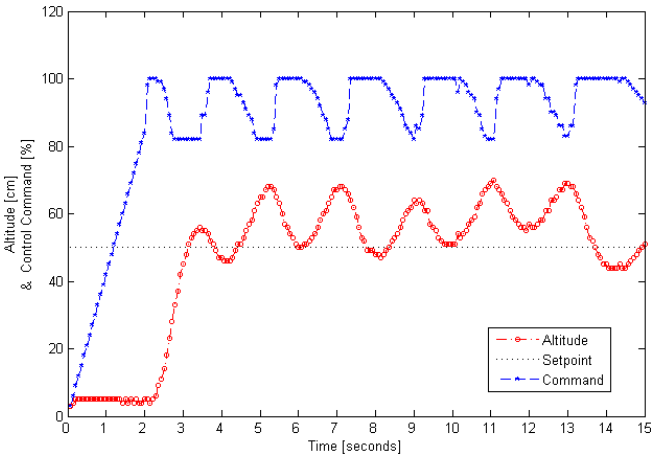

Fig. 6. Performance of the altitude controller without the integral part, i.e. PD only controller.

To analyze the real robustness of the heading controller, we subtracted the offset error (since it is constant) and found it to be very reliable. Figure 7 shows the heading and the control command when combined with the PD only altitude controller.

The peak to peak error is not very different from that to figure 5, which is surprising given that in figure 5 it was coping with a severe interference from the altitude oscillations. The overall robustness of the heading controller was very high in comparison with the altitude controller.

C. Overall Analysis

The overall performance of the controllers can be considered very satisfactory given that it was evolved starting from a completely random population and with a relatively low number of generations.

The difference in performance found between the heading and altitude controller seems to be due to the different effects produced by the experimental setups. The mechanical restrains used for the altitude controller affected more the performance of the system, with respect to a free system, as compared to the effect of the restrains used for the heading controller. Therefore the reality gap was larger in the case of the altitude controller's tuning.

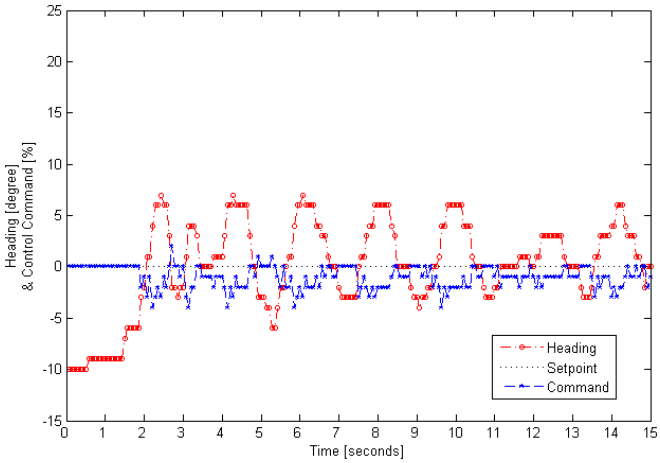

Fig. 7. Performance of the heading controller in combination with the PD only controller, and error offset eliminated.

The reality gap was not breached entirely anyway due to the need to restrain the helicopter during evolution, which affected the behavior of the system. Further tests will be required to determine if this gap can be breached at all using the real system in these conditions. It is safe to assume that such gap does not exist if the *pure* real system could be used. But in cases such as the presented in this paper, where the pure system cannot be used in practice, the reality gap remains an issue.

Still, we confirm with our analysis that evolving a controller on real systems, even with constrained characteristics, can produce robust solutions, which is very difficult to achieve with a simulator.

VI. Conclusion

We have analyzed the behavior of an evolutionary optimization process using real systems and have concluded that robustness is a trait that is always gained with such a setup. This can be achieved even if robustness is not evaluated explicitly on the fitness function. Further work can be done to determine which would be a more efficient strategy, including explicit robustness test in the fitness function or making up for it with longer time-out for the evolutionary process.

The reality gap will still exist if the pure system cannot be used for practical reasons, such as restrains for safety purposes. It will be a matter of further analysis to determine if the gap can be reduced more by better simulation or by more efficient restrains on the real system.

REFERENCES

[1] P. Spanoudakis, L. Doitsidis, N. C. Tsourveloudis, and K. P. Valavanis, “A market overview of the vertical take-off and landing uavs,” in Workshop on Unmanned Aerial Vehicles, 11th Mediterranean Conference on Control and Automation, Rhodes, Greece, 2003.

[2] A. Coates, P. Abbeel, and A. Ng, “Learning for control from multiple demonstrations,” in the 25th International Conference on Machine Learning, ICML, Helsinki, Finland, 2008.

[3] B. Ludington, E. Johnson, and G. Vachtsevanos, “Augmenting uav autonomy,” Robotics & Automation Magazine, IEEE, vol. 13, no. 3, pp. 63–71, 2006.

[4] B. Kadmiry and D. Driankov, “A fuzzy gain-scheduler for the attitude control of an unmanned helicopter,” IEEE Transactions on Fuzzy Systems, vol. 12, pp. 502–515, 2004.

[5] M. Sugeno, Fuzzy Modeling and Control: Selected Works of M. Sugeno. CRC Press, 1999, ch. Development of an Intelligente Unmanned Helicopter, pp. 13–43.

[6] C. Phillips, C. L. Karr, and G. Walker, “Helicopter flight control with fuzzy logic and genetic algorithms,” Engineering Applications of Artificial Intelligence, vol. 9, no. 2, pp. 175–184, 1996.

[7] H. Yamada, J. Takeuchi, G. Matsumoto, and M. Ichikawa, “A flying object using hardware implemented, vision processing and motor control system with adaptive neural network,” in Neural Information Processing, 2002. ICONIP ’02. Proceedings of the 9th International Conference on, vol. 2, 18-22 Nov. 2002, pp. 685–690vol.2.

[8] E. Sanchez, H. Becerra, and C. Velez, “Combining fuzzy and pid control for an unmanned helicopter,” in Annual Meeting of the North American Fuzzy Information Processing Society, Unidad Guadalajara, Mexico, 2005, pp. 235–240.

[9] S. Saripalli, J. Montgomery, and G. Sukhatme, “Vision-based autonomous landing of an unmanned aerial vehicle,” in Robotics and Automation, 2002. Proceedings. ICRA’02. IEEE International Conference on, vol. 3, Washington, DC, May 2002, pp. 2799–2804.

[10] S. Puntunan and M. Parnichkun, “Control of heading direction and floating height of a flying robot,” in IEEE International Conference on Industrial Technology, vol. 2, Bangkok, Thailand, 2002, pp. 690–693.

[11] E. H. Mamdani, “Application of fuzzy algorithms for simple dynamic plant,” IEE Proceedings on Control Theory and Applications, vol. 121, pp. 1585 – 1588, 1974.

[12] P. Fleming and R. Purshouse, “Evolutionary algorithms in control systems engineering: a survey,” Control Engineering Practice, vol. 10, no. 11, pp. 1223–1241, 2002.

[13] N. Jacobi, P. Husbands, and I. Harvey, Lecture Notes in Computer Science - Advances in Artificial Life.1em plus 0.5em minus 0.4emSpringer, UK, 1995, ch. Noise and the Reality Gap: The Use of Simulation in Evolutionary Robotics, pp. 704–720.

[14] R. A. Brooks, “Intelligence without reason,” in Proceedings of the 12th International Joint Conference on Artificial Intelligence, J. Myopoulos and R. Reiter, Eds.1em plus 0.5em minus 0.4emSydney, Australia: Morgan Kaufmann publishers Inc.: San Mateo, CA, USA, 1991, pp. 569–595.

[15] I. Sekaj and M. Sramek, “Robust controller design based on genetic algorithms and system simulation,” in Decision and Control, 2005 and 2005 European Control Conference. CDC-ECC ’05. 44th IEEE Conference on, 12-15 Dec. 2005, pp. 6881–6886.

[16] H. Shim, T. Koo, F. Hoffmann, and S. Sastry, “A comprehensive study of control design for an autonomous helicopter,” in Decision and Control, 1998. Proceedings of the 37th IEEE Conference on, vol. 4, Tampa, Florida, USA, December 1998, pp. 3653–3658.

[17] M. Perhinschi, “A modified genetic algorithm for the design of autonomous helicopter control system,” in Proceedings of the AIAA Guidance, Navigation and Control Conference, 1997, pp. 1111–1120.

[18] J. Mao, “Robust flight controller design for helicopters based on genetic algorithm,” in Proceedings of Fifth IFAC Congress, Barcelona, 2002.

[19] J. Zufferey, A. Guanella, A. Beyeler, and D. Floreano, “Flying over the reality gap: From simulated to real indoor airships,” Autonomous Robots, vol. 21, no. 3, pp. 243–254, 2006.

[20] M. Ahmad, L. Zhang, and J. Readle, “Online genetic algorithm tuning of a pi controller for a heating system,” in Genetic Algorithms In Engineering Systems:Innovations And Applications, 1997. GALESIA 97. Second International Conference On (Conf. Publ. No. 446), 2-4 Sept. 1997, pp. 510–515.

[21] L. Nolle, A. Goodyear, A. Hopgood, P. Picton, and N. Braithwaite, “Improved simulated annealing with step width adaptation for langmuir probe tuning,” Engineering Optimization, vol. 37, no. 5, pp. 463–477, 2005.

[22] P. De Moura Oliveira, “Modern heuristics review for pid control systems optimization: A teaching experiment,” in Proceedings of the 5th International Conference on Control and Automation, ICCA’05, 2005, pp. 828–833.

[23] Passow, B. Gongora, M. Coupland, S. Hopgood, A.A., Real-time Evolution of an Embedded Controller for an Autonomous Helicopter, In Proc. 2008 IEEE Congress on Evolutionary Computation (CEC 2008), Hong Kong, 2008.