This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TEVC.2017.2703142, IEEE Transactions on Evolutionary Computation

A Platform that Directly Evolves Multirotor Controllers

David Howard, *Member, IEEE.*

***Abstract*—We describe an experimental platform that uses differential evolution to automatically discover high-performance multirotor controllers. All control parameters are tuned simultaneously, no modeling is required, and, as the evolution occurs on a real multirotor, the controllers are guaranteed to work in reality. The platform is able to run back-to-back experiments for over a week without human intervention. Self-adaptive rates are shown improve solution fitness whilst (at least) maintaining convergence times. This platform is the first to allow for evolutionary robotics experimentation to occur safely and repeatedly on real multirotors. High-performance controllers are evolved despite noisy fitness evaluations, real-world sensory noise, low population sizes and limited numbers of evolutionary generations.**

I. INTRODUCTION

CONSIDER a fleet of identical multirotor Unmanned Aerial Vehicles, such as one might find in a research lab. The individual multirotors may be in different states of hardware degredation, or may have been recently upgraded with e.g., new propellers. The multirotors may be fitted with different payloads, which may be mounted in a number of configurations and potentially actuated. This will shift the multirotor's mass distribution, or introduce momentum from the actuation. All of these factors may affect flight performance. Additionally, the multirotor may be required to perform different types of missions with different desired flight characteristics, e.g., smooth transitions would aid in video acquisition, whereas more abrupt maneuvers may be beneficial for tracking an unpredictable moving target. Despite these factors, multirotor controllers are usually identically parameterised, and therefore not specific to a given hardware state, payload, or desired in-mission behaviour. One reason for this is that tuning pre-mission usually requires a model (which should be updated each time), and/or a pilot (which is costly and time-consuming).

Motivated by a desire to automatically create controllers for an arbitrary multirotor and payload as a pre-mission closed-loop optmizer, we present a platform that follows an Evolutionary Robotics (ER) approach, and evolves multirotor controllers from scratch within 3-6 hours. ER is a promising approach it is ambivalent to the physical shape of the multirotor. ER provides the flexibility to define fitness in multiple ways, and the optimisation landscape is likely to be multimodal, discontinuous, and noisy — evolutionary algorithms are known to handle these features well [1].

David Howard is with the Cyber-Physical Systems Program, CSIRO, QCAT, 1 Technology Court Pullenvale Brisbane 4069 Australia david.howard@csiro.au

Previous attempts to perform ER on flying robots typically require a simulator (e.g., [2], [3]), as testing stochastically-optimised controllers on real flying robots can be destructive. We follow a different approach, presenting the first platform that can safely, repeatedly and non-destructively evolve controllers for multirotors directly on the robot, which provides the benefits that the controllers are guaranteed to work on the multirotor in reality, and that the optimisation quality isn't tied to quality of the simulator/model.

The platform consists of a protective cage that contains the multirotor, and a fan which provides significant wind disturbances. A host PC monitors the state of the multirotor, including health checking, dangerous state monitoring and recovery, and performance estimation, and runs the optimisation process. A pair of physical tethers prevent the multirotor from reaching irrecoverable states, and a power tether allows for 24-hour operation. The platform is reliable enough to run for over a week without human intervention. Unlike many evolutionary robotics approaches, especially those involving flying robots, we can execute back-to-back experimental repeats to confirm statistically testable hypotheses.

Differential Evolution (DE) [4] is selected as it has shown promising results in controller optimisation specifically [5], [6]. Self-adaptive DE is used (e.g., [7]) as different multirotors and payloads are likely to prefer different learning rates. As we are limited to real-time evaluations, we use self-adaptation to reduce the number of generations required.

We report on two experiments. On a hovering task with wind disturbances, we test the hypothesis that self-adaptive DE rates generate higher fitness controllers than both default "best-guess" rates, and static rates set using the best rates discovered in the self-adaptive experiment. Secondly, we demonstrate the ability of the platform to generate payload-specific controllers by altering the payload on the multirotor and showing the adaptation in both control parameters and learning rates.

A. Contributions

This work demonstrates the first platform to permit continual (24/7) closed-loop (no human required) ER on dynamic flying robots, including experimental repeats. Other significant contributions include (i) a demonstration of the benefits of self-adaptation in this hardware evolutionary flying robotics scenario, and (ii) an in-depth analysis of the resulting control parameters and self-adaptive rates.

B. Structure

The next section discusses related work and motivates our design decisions. We describe the hardware and software

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TEVC.2017.2703142, IEEE Transactions on Evolutionary Computation

composition of the platform in Section III. Sections IV, V, and VI describe the multirotor, evolutionary algorithm, and experimental setup respectively. Section VII presents the results and analyses the performance, evolved control parameters, and self-adaptive rates. Section VIII analyses the evolved controllers. Section IX describes the experiment in which the payload is varied and analyses the results of that experiment, and Section X concludes with a discussion and outlook.

II. Background

A. Hardware Evolutionary Robotics

This work falls under the broad remit of Evolutionary Robotics (e.g., [8]), which involves the use of an evolutionary algorithm to optimise the controllers of a physical robot. Evolutionary robotics can also involve a morphological aspect, where the body of the robot is also optimised — either with human [9] or automated [10] construction. Due to the focus of our study, we use the acronym ER to refer to controller optimisation only, with a static physical robot morphology.

A typical ER experiment involves a physical robot and a population of controllers. Each controller is sequentially assessed for its ability to perform some task, giving each population member a fitness value which can then be iteratively improved upon though stochastic generation of new controllers using, e.g., mutation and recombination. Controller representations vary, although neural networks [11] and parameterisations of known control structures are popular choices.

Modelling or simulation are frequently used to expedite ER experiments on real robots. As a simulator is a necessarily imperfect capture of reality, controllers evolved in simulation may perform suboptimally on a real robot — the 'reality gap'. Sufficient levels of sensory noise [12] and explicitly selecting for transferability [13] have shown promise in circumventing this problem, however real hardware evolution remains the most likely method to concurrently exploit the physical state of the robot and guarantee performance in reality, e.g., [14]. Our platform allows this type of evolution on multirotors for the first time.

Coevolution of a series of models with a set of tests have shown promise in matching models to reality, and are capable of adapting to hardware changes online, i.e., the removal of a leg from a hexapod [15]. This technique has been used to model a multirotor [16], where tests were a series of sections of a time series of multirotor flights, however the model was generated from human-piloted flight tests. We note that such research typically focuses on model generation, rather than the discovery of high-performance controllers.

Recent research involves pre-computing a map of possible behaviour-performance relations, which is used to quickly search for damage-compensating behaviours for a legged robot [17]. With a reported precalculation time of 2 weeks per robot, such approaches are infeasible for us. Our aim is to create specialised controllers in a short time frame for subsequent field deployment.

Examples of hardware evolution of flying robots is scarce, due to the potentially destructive nature of the stochastic optimisation process. Control has been evolved for a blimp [18], which has slow dynamics and is therefore easily recoverable from failure states. The aforementioned coeveolutionary approach [16] requires a pilot to generate a model, and additionally focuses on the modelling rather than control. Height and yaw control of a miniature single-propeller rotorcraft [19] has been evolved on a real robot, however the rotorcraft is very small so applications are limited, and some gains were incorrectly set as the physical restraints on the rotorcraft did not accurately capture the conditions of free flight. Generally, the aforementioned multirotor ER works do not provide experimental repeats, so statistical tests between different approaches are infeasible. Our platform is the first to safely and reliably permit such testing.

Recent work evolves behaviour trees to allow a (simulated, then real) micro multirotor to fly through a window when trapped in a room [20]. Using human-legible behaviour trees allowed for manual tweaking in the evolved rulebase — allowing the real multirotor to raise its escape success rate to 54% from 46%, although the reality gap is evident as the simulation success rate was 88%.

The most similar approach to ours evolves controllers on real multirotors using a Bee Colony Algorithm [21]. As with our approach, controllers are optimised from scratch. The authors also demonstrate an online derivative, with slight performance loss between online and offline modes. We note that the authors do not include environmental disturbances (the required behaviours are generated indoors with no wind, and as such are easier to attain), and batteries are still used and must be replaced as needed, which is time-inefficient and requires sporadic human intervention, thus not providing the capability to easily perform experimental repeats.

Overall, our decision to focus on evolving on real multirotors is based on the necessity for the controllers to work in reality and account for hardware and payload differences. Our research therefore focuses on creating a platform that accounts for these factors, and on the effects of self-adaptation to reduce the number of hardware trials required.

B. Multirotor Controllers & Tuning

1) Controllers: The most popular multirotor controller is the linear Proportional/Integral/Derivative (PID) [22], which assumes that a reasonably accurate linear approximation of the multirotor's state space can be made. Owing to a lack of extreme speeds, attitudes, and transitions, this assumption holds in many real-world missions (that do not involve aerobatics). PIDs attempt to minimise control error by specifying three control parameters (*gains*) per error signal. Typically for flying robots, there is one error signal per degree of freedom (the roll ρ, pitch θ, and yaw ψ angles, plus x, y and z position). For each degree of freedom, three gains weight the relative impact of the raw error signal (proportional), plus its integral and derivative. As the position control quality directly relates to the corresponding attitude control, PID controllers are typically nested, with the attitude control loop sitting inside the position control loop, as highlighted in e.g., [23].

Another linear control technique — gain scheduling — involves splitting the state space into a number of regions, each

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TEVC.2017.2703142, IEEE Transactions on Evolutionary Computation

with different gain values. This results in a piecewise-linear approximation, however this subdivision typically requires the input of an expert to decide on the transition regions, and tuning multiple gain regions can be time-consuming.

Nonlinear controllers include Linear Quadratic Regulators, H∞, and neural networks (see [24] for an overview). They are more complex in nature, and provide accurate control in cases where the linear statespace assumption breaks down, e.g., when flying extreme acrobatic maneouvers. However, they are more difficult to implement, have more tunable variables, and typically require an individual model per multirotor.

We elect to evolve the control parameters of PIDs, as we can safely assume linearity in the statespace for the mission types we consider. Additionally, PIDs rely only on the measured process variable, not on knowledge of the underlying process, and as such are broadly applicable. PID controllers are readily interpretable, can reveal underlying features of the platform/mission/payload, and require no modelling (as we don't require a model, any multirotor with an airframe length $<$800mm can be used with the platform). Most importantly, PID is commonly implemented on the majority of off-the-shelf flight control boards — our platform can therefore optimise most multirotors.

2) PID Tuning: Tuning is the process of setting control parameters so as to successfully control a given plant — in our case a multirotor. In practice, PID tuning requires striking a balance between different gain settings. An overview of tuning methods is provided [23]. As an optimisation task, real multirotor PID tuning can be difficult despite the relatively small number of dimensions. Performance assessment is based on noisy sensors, control is through noisy actuators, and the control parameters strongly interact with each other.

Typical tuning methods, such as the heuristic Ziegler-Nichols method [22], optimisers such as the Lopez method [25], and analytical approaches e.g., root-locus [22], have all been successfully applied to tune PID multirotor controllers. A pilot is typically required to fly suitable trajectories and assess performance [26], which can be difficult and time consuming. Model-based tuning circumvents the need for a pilot, but model generation itself usually requires a pilot, as well as making control quality dependent on model quality. The requirement for remodelling every time a payload is changed is also time consuming. Reinforcement learning has shown promising results [27], [28], but again requires a model or pilot.

We note that traditional tuning methods (e.g., [22]) tend to separate position from attitude, assuming that optimal attitude control leads to optimal position control. By evolving all parameters simultaneously we disregard this assumption, and potentially harness coupling (the interactions between control parameters) to generate improved control quality.

We require a model-free, pilot-free tuning method, that allows us to define performance on multiple levels of abstraction; from angular errors and position errors, through to complex higher-level behaviours. Due to these prerequisites, an evolutionary algorithm presents itself as an obvious candidiate.

C. Differential Evolution

Differential Evolution (DE) [4] is an efficient evolutionary algorithm based on vector manipulation. We focus on DE as it has been shown to converge expediently compared to e.g., Genetic Algorithms [29], CMA-ES [30], and Particle Swarm Optimisation (PSO) [30] (important as evaluations are costly in ER). Algorithmically, it is insensitive to the initial population distribution (important as we are forced to use low population sizes), and as we evolve controllers from scratch without knowing optimal value ranges, we can harness the property that the DE parameter space is unbounded during search. DE also permits incremental fine-tuning of parameters, which is beneficial for our task of controller tuning. It is also easy to understand, which is important for other users of the platform.

The performance of DE is known to be sensitive to the setting of the rates CR and F [31]. Due to the wide variety of multirotors and payloads we expect our platform to optimise, and given this rate sensitivity, we expect that a single homoegenous rate setting will be suboptimal across all combinations, and as such consider self-adaptation as a method of tailoring the optimisation process to each specific case to reduce the number of generations required.

D. Self-Adaptation

Self-adaptation is a parameter control technique that allows the mutation rates of an evolutionary algorithm to vary, typically based on their performance [32]. The literature indicates that self-adaptive mutation can be beneficial to the evolutionary process, by tailoring the learning rates to the problem explicitly, and allowing rates to change through time in response to the state of the population [33].

Self-adaptation has been shown to be beneficial in the context of simulated [3] and hardware ER [33] in terms of setting rates for evolutionary algorithms (although we note that such experimentation has not occurred on flying robots). We note the use of self-adaptation in reducing the number of generations required per experimental repeat [34], which is important as our evaluations cannot occur faster than real-time. We also note the use of individual rate-restarts [35] using a 1+1 Evolution Strategy [36] as an effective technique to (i) dissuade premature convergence into unfavourable areas of the rate space, and (ii) 'rescue' the optimisation process from unfavourable rate settings.

Self-adaptive DE allows *CR* and *F* to vary during an experiment, with either a single global setting of each rate, or one setting per population member — both versions have shown performance benefits compared to static rates [7], [37].

E. Literature Summary

Overall, we have shown a niche for our platform in automatically generating specialised controllers that depend on the type of multirotor and its payload, and in generating statistically testable experimental repeats. To open our platform to as many multirotors as possible, we have decided to focus on platform that optimises PID controllers, as a closed loop system on real multirotors to guarantee real-world performance. We have also:

1) Justified the use of PIDs as human-legible, and generally-applicable controller representations.
2) Highlighted the suitability of DE as a high-performance evolutionary algorithm that has shown promise in evolving PID controllers.
3) Motivated our investigation into the use of self-adaptation in this scenario, to provide a more flexible learning process given the different optimisation conditions that our platform is likely to experience, and to reduce the required number of evaluations.

III. Platform Components

Before commencing with the platform description, we introduce some nomenclature which will be adopted throughout the remainder of the article. The platform evolves a *population* of *individuals*, where each individual is a set of control parameters, fitness, and *CR* and *F*. An *experiment* is a set of ten optimisation runs, which is used to generate statistics. A single run is called an *experimental repeat*. Each repeat lasts for a number of *generations* — until convergence — with each generation involving the creation of a number of new individuals which are evaluated on the test problem, and potentially replace current population members. An *evaluation* involves the current control parameters being tested on the multirotor, and culminates with a fitness value being assigned to the controller. For ease of reference, a full listing of symbol definitions is provided in Appendix A.

A. Physical Platform

The hardware setup includes a safety-net covered frame onto which an LED strip light — allowing operation at night — and down-pointing camera are mounted (dimensions shown in Fig.1). Two mechanical tethers (nylon wire) prohibit (i) flipping by preventing tilt angles $> 60^o$, and (ii) rotation in excess of $\pm 160^o$. A 3mm foam mat absorbs landing shock and damps contact noise. An oscillating fan provides wind disturbances of ≈2m/s with an oscillation period of 8 seconds and total traversal angle of 105^o. Two cables attach to the multirotor: constant power is provided at 24V, and communication is handled through a serial cable that connects to the host PC. Power for the entire testbed can be remotely toggled through an SSH connection.

B. Software Framework

The host PC runs the Extended State Machine (ESM) framework [38] to manage and monitor experiments. ESM is a real-time state machine, which is especially important for evolving multirotor controllers as the multirotor is highly dynamic and requires high state/control update rates. Internally, ESM describes all system components (sensors, motors, software components) and defines the data connections between them.

An experiment is defined as a number of states (e.g., *multirotor in air*, *evolve a controller*), together with suitable transitions between those states. Altering the state machine structure allows us to quickly alter what is tested, and how we test it. ESM is responsible for (remotely) starting experiments, receiving state information, commanding angles via the serial connection, assigning controller fitness, tracking population statistics, and calling DE to generate new individuals for testing.

ESM monitors for (i) mechanical tether breakage, (ii) camera communication loss, and (iii) tether interference with the height sensor, and automatically restarts evaluations that are subject to such failures. ESM continually checks for dangerous multirotor states, and can intervene to terminate an evaluation early if the following states are detected:

1) *high* (maximum height of 18cm exceeded),
2) *highSpeed* (maximum velocity of 50cm/s horizontal / 25cm/s vertical exceeded),
3) *tilted* (maximum roll or pitch angle of 15^o exceeded),
4) *highCurrent* (maximum current of 15A exceeded),
5) *ctrlLimit* (maximum rate for upper PWM limit of 75s^{-1} exceeded),
6) *onGround* (aircraft touching the ground for 1s),
7) *touching* (part of the landing gear touching the ground),
8) *rotated* (maximum yaw error of 45^o exceeded).

IV. Multirotor

A. State Estimation

The multirotor under evaluation estimates its state through a MEMS-based IMU and external camera. The IMU provides attitude Euler angles (roll ϕ, pitch θ, yaw ψ) at 400Hz. Angular rates (ω_p, ω_q, ω_r) are derived from two consecutive Euler angles. Height h is calculated from Euler angles and readings from a 20Hz IR rangefinder that is mounted on the airframe. Attitude angles are processed through a Kalman Filter, and height through a complimentary filter.

Offboard position estimates for North p_n and East p_e are provided at 60Hz by a machine vision camera that is mounted 200cm from ground level, using the pixel position of a multirotor-mounted LED. Velocities (v_n, v_e, v_h) are computed through linear regression of five consecutive position estimates. This complete state estimate includes position, attitude, angular rates, and velocities, and provides a 3D position error <5mm and heading error $<2^o$.

B. Controllers

PIDs use the estimated state to control the multirotor's position and attitude at 400Hz. We use a static nested PID structure as shown in Fig.2 to account for each the multirotor's six degrees of freedom — horizontal position (p_n and p_e) is controlled by the outer loop, and the attitude (ϕ, θ, ψ) and height h by the inner loop — note that ϕ and θ depend on the outputs of the horizontal position PIDs.

The outer-loop PIDs generate setpoints θ_{sp} and ϕ_{sp} (Fig.2). The inner-loop PIDs outputs δ_ϕ, δ_θ, δ_ψ, and δ_t represent commanded changes in attitude and thrust, which are scaled in the range of attainable motor PWMs l_{ul}=1000 and l_{um}=2000, and passed to a linear mixer which produces one speed controller command per motor m, e.g., u_1 to u_m.

Each variable is limited to a maximum error l_{er} (10cm for h, 15^o for attitude, 15cm for p_n/p_e) before being input

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TEVC.2017.2703142, IEEE Transactions on Evolutionary Computation

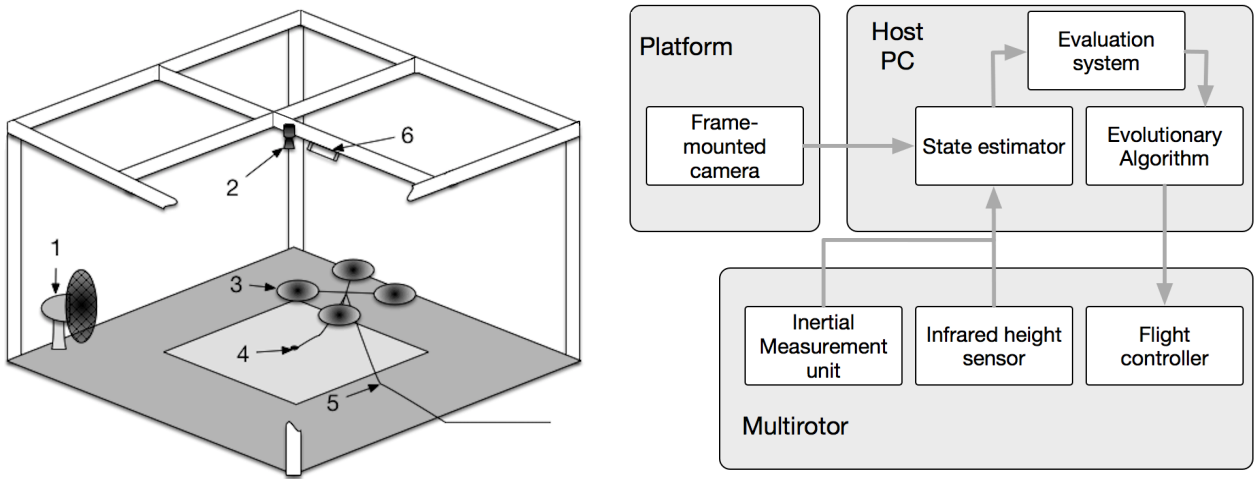


Fig. 1. (left) Diagram of the platform, showing (1) the fan, (2) camera, (3) multirotor, (4) physical tether, (5) data/power tether, and (6) light. The camera height is 200cm and padded floor area is 271cm^2. The light grey floor area depicts a standard flight area of $\approx 60cm$ in x and y, and 20cm in z. (right) showing the major components. The evolutionary algorithm generates and modifies controllers. Control parameters detemine control commands, which are sent to the flight controller and executed. The frame-mounted camera and IMU/IR height sensor readings are sent to the host PC and combined in the state estimator. The state estimator is used to assess fitness and monitor system errors (e.g., multirotor out of bounds). The state estimator and evaluation system are run in real-time using ESM.

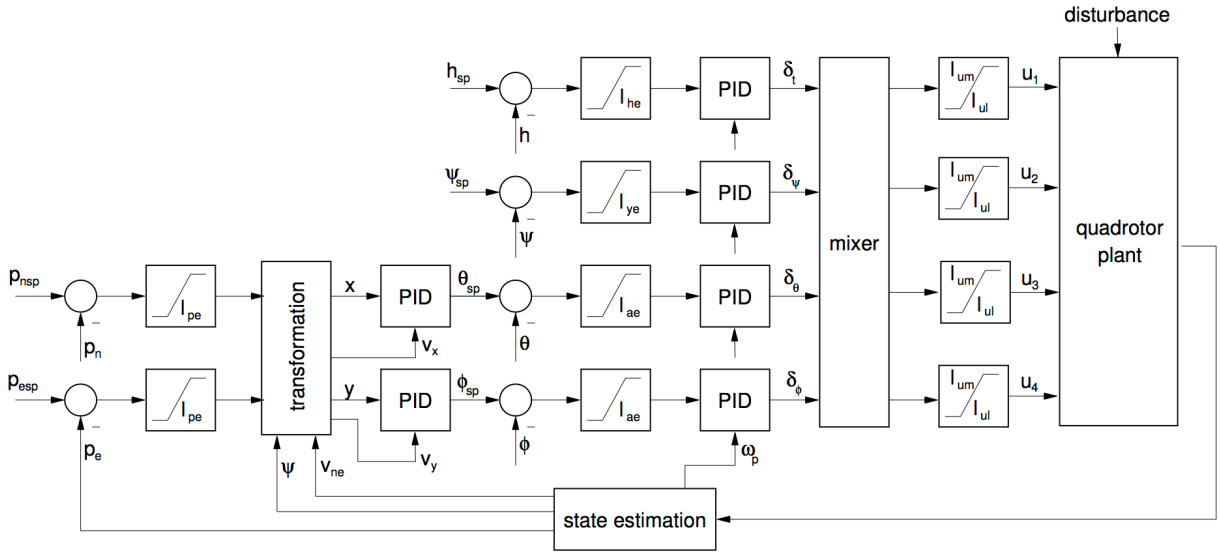


Fig. 2. PID control structure, showing attitude and position loops. Parameters l_{he} l_{ye} l_{ae} denote error limits for height yaw and attitude respectively. l_{ul} and l_{um} are minimum and maximum motor commands, and δ_ϕ, δ_θ, δ_ψ, and δ_t are command inputs to a mixer which produces speed controller commands u_1, u_2, u_3, and u_4. The disturbance is input from the fan.

to the PID, which attempts to minimise the error e between the desired setpoint (variable with subscript $_{sp}$), and the state estimate value for that variable following (1). Here, o is the PID output, t is the instantaneous time, τ is the integration timestep from 0 to t, and K_p, K_i, and K_d are tunable gains that define the response of the controller to raw error, integral error, and the derivative of the error respectively. With three tunable gains per degree of freedom and six degrees of freedom, a total of 18 floating point numbers are used to represent each individual in the population.

$$o(t) = K_p e(t) + K_i \int_0^t e(\tau)d\tau + Kd\frac{d}{dt}e(t) \qquad (1)$$

V. Evolutionary Algorithm

We use DE/rand/1/bin, which we use throughout as it has shown promising results in evolving PID gains [39]. At the start of an experiment, we initialize N=20 controllers of size $S = 18$ randomly within bounds shown in (2), where l_{cmd} is a generalised maximum possible command (PWM) for each of the control parameters V: l_{cmd} for $\phi/\theta/\psi/h$=500, and p_n/p_e=15cm.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TEVC.2017.2703142, IEEE Transactions on Evolutionary Computation

$$K_{\mathrm{pV}} = K_{\mathrm{iV}} = K_{\mathrm{dV}} = (0, \frac{l_{\mathrm{cmd}}}{l_{\mathrm{er}}}] \quad (2)$$

Each individual is tested on the platform, and assigned a fitness f. Per generation, a donor vector v is created for each 'parent' individual p as in (3), where F, $(0 < F \leq 2)$ is the differential weight, and $r1$, $r2$, and $r3$ are unique individuals that are selected uniform-randomly.

$$v = r3 + F(r1 - r2) \quad (3)$$

A 'child' vector c is created for each parent by probabilistically merging p and v. For each vector index i, c_i = v_i if $i == R$ or $rand < CR$, otherwise c_i = p_i. $rand$ is a uniform-random number between 0 and 1, *CR*, $(0 < CR \leq 1)$, is the crossover rate, and R is a random vector index, ensuring $c \neq p$. Each child is then evaluated and assigned a fitness f, with c replacing its parent p if f_c is superior to f_p. When every child has been evaluated, the next generation begins.

Section II-C notes two broad techniques for self-adaptation of DE rates, with either one rate set per experiment, or one rate set per individual. We adopt the latter approach, as during an experiment, the population of controllers may be at different stages of the evolutionary process, requiring different simultaneous individualised rates.

For self-adaptive experiments, each initial population member rates are *CR*=0.5 and *F*=1.0 — the midpoints of the possible ranges for each rate [4]. When new population members are created, the parent's *CR* and *F* are altered following an Evolutionary Strategy [36] (respecting rate limits) from its parent as in (4). The parent's rates are unaltered. The child adopts the altered rates in subsequent generations, which affects the impact of *CR* and *F* on the DE process. Static "baseline" mutation rates follow the original DE [4]; *CR*=0.5, *F*=0.8.

$$\mu \leftarrow \mu * e^{N(0,1)} \quad (4)$$

Note that comparing self-adaptive strategies is not a current focus of this work; our approach was chosen as it is conceptually simple (so the effect of changing the parameters is clear), and has previously shown promising results in ER [34]. For general comparisons between self-adaptive algorithms, we refer the interested reader to [32].

Self-adaptation reduces the number of hardware evaluations required before convergence, by allowing useful rates to be automatically selected throughout the duration of an experiment. Note that self-adapatation also allows for generation of an archive of rates for certain multirotor/payload combinations, to bootstrap subsequent experiments.

Preliminary experimentation revealed that rates could occasionally cause non-convergence, when *CR* becomes too high to create any useful parent-to-child mappings, meaning the parent is never replaced. To counter this, a rate reset strategy (following, e.g., [35]) is used — an individual's rates are uniform-randomly reinitialised if, for five consecutive generations (heuristically selected to balance search stability and convergence times), no improvement in fitness is generated from the children generated for that population slot. When a controller can successfully fly the entire waypoint set, it no longer triggers rate resets.

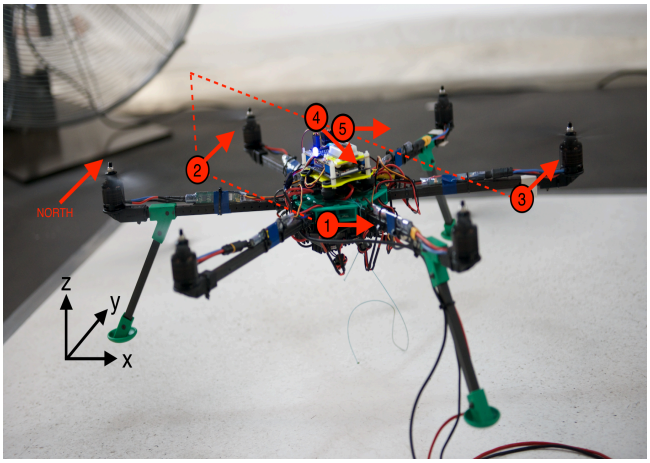


Fig. 3. Trajectory. The multirotor is commanded to (1) hover at a height of 10cm with a yaw of 40^o (2) move 8cm North and 8cm West with a yaw of 0^o (3) increase height to 14cm and move 16cm South and 16cm East (4) return to the centre of the cage with a yaw of 80^o (5) alter yaw to 40^o. A transition occurs every 8 seconds, total flight time not including take-off is 40s. The fan can be seen in the top-left of the image.

VI. Experimental Setup

We present a series of experiments to compare three versions of the platform: for clarity we refer to these versions as algorithmic (or ALG, baseline/default rates, $CR = 0.5, F = 0.8$), self-adaptive (SA), and static (STAT, static rates set from the mean best self-adaptive rates, $CR = 0.48, F = 0.29$).

We wish to ascertain differences in terms of fitness, convergence times, and DE rate settings. We aim to prove the hypothesis that self-adaptation is beneficial to the evolution of multirotor controllers in ER scenarios, and demonstrate the ability of our platform to perform statistical tests to prove this hypothesis.

We assess the platform variants in a hovering task, where a multirotor attempts to fly a 5-waypoint trajectory as shown in Fig.3. With an eight-second gap between transitions, the total evaluation time is 40s. The trajectory is designed to sufficiently excite all of the PID gains to generate a viable hover controller. Hovering is initially chosen as is it a basic requirement of multirotors, and the resultant control parameters can be extended into various more specialised behaviours. Note that other behaviors can be attained by altering the fitness function and selecting a suitable trajectory. Note that this is a challenging optimisation problem; the platform must deal with real-world levels of sensory and actuator noise, varying fitness evaluations for the same control parameters, and must optimise in few generations as evaluations in ER are expensive.

The initial population is bootstrapped — random controllers are generated until each of the initial controllers prevents the landing gear from touching the ground for 0.2s. Once the population is full of such controllers, an evaluation follows Algorithm (1), where t_1=5s, t_2=4s, t_3=40s, and t_4=5s are empirically-determined timeouts, designed to prevent persis-

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TEVC.2017.2703142, IEEE Transactions on Evolutionary Computation

tant unwanted behaviors (t_1, t_2, t_4) or terminate a successful evaluation (t_3).

Note that *airborne* is a state where the multirotor is not touching the ground for >1s. During an evaluation, fitness accumulates every 2ms cycle by adding f_{cycle}, calculated according to Appendix B, to a running total f, with a theorectical maximum $f = 160000$. If the multirotor enters a dangerous state, its flight is terminated and it retains its fitness.

To account for noisy fitness evaluation, any controller that completes the 40s evaluation is immediately reevaluated and assigned the mean fitness. If the controller completes the evaluation on both occasions, it is said to be a success.

ALGORITHM 1: an evaluation, as processed by ESM. ESM continually monitors error states, together with associated timeouts (line 2,5,7,10,12). Fitness culminates every 2ms when airborne (line 13).

```
1   f = 0, n = 0, f_tmp = 0, n_tmp = 0
2   while (t < t_1) ∧ ¬airborne ∧ ¬high ∧ ¬highSpeed
        ∧¬tilted ∧ ¬highCurrent ∧ ¬ctrlLimit
3     f_tmp += f_cycle, n_tmp += 1
4     if n_tmp > n then f = f_tmp, n = n_tmp
5     if touching ∧ (t ≤ t_4) then f_tmp = 0, n_tmp = 0
6   if airborne
7     while (t < t_2) ∧ ¬high ∧ ¬highSpeed ∧ ¬tilted
          ∧¬highCurrent ∧ ¬ctrlLimit ∧ ¬onGround
8       f_tmp += f_cycle, n_tmp += 1
9       if n_tmp > n then f = f_tmp, n = n_tmp
10      if touching ∧ (t ≤ t_4) then f_tmp = 0, n_tmp = 0
11    if t ≥ t_2
12      while (t < t_3) ∧ ¬high ∧ ¬highSpeed ∧ ¬tilted
            ∧¬highCurrent ∧ ¬ctrlLimit ∧ ¬onGround∧
            ∧¬rotated
13        f += f_cycle
```

Successful controllers are reset to their start positions (centre of the floor area with ψ=40^o) to ensure a fair test between controllers; before this point we dont require an accurate fitness assignment, being more interested in discovering controllers that can exhibit basic flight competency. When the entire population consists of successful controllers (convergence), the experiment ends. Previous experimentation has shown this termination criterion to be suitable, as 300 extra generations resulted in fitness gains of <2% of the converged total [6].

VII. Results

Each of the three experiments is repeated ten times. All statistical significance assessments are performed with the Mann-Whitney U-test as we cannot guarantee normally-distributed samples.

A. Performance

The most notable result is that SA generates populations with greater average best fitness ($f = 127185.2$) than the other variants (both p< 0.01, ALG $f = 124020.7$, STAT $f = 123531.1$; Fig. 4(a), Table I). This trend is echoed for average mean fitness (SA $f = 122562.6$, ALG $f = 117483$, STAT $f = 118450.3$, Fig. 4(b)). As the static rate setting is based on the best rates found in the SA experiment, we suggest that it is in-experiment adaptivity, rather than the final rate setting, that is responsible for the observed increase in fitness in the SA version by permitting fine-grained improvements towards the end of an experiment.

Following this trend for average low fitness (Fig. 4(c)), SA $f = 117480.6$ is statistically superior to ALG ($f = 110795$, p< 0.01). SA is statistically similar to STAT ($f = 112323.8$, p> 0.01). This is thought to be related to the convergence criterion used — as the experiment terminates immediately when the entire population can survive for 40s, the final individual to achieve this will have had no time to finesse the controller further, leading to a lack of separation in mean low fitness between the three variants. SA displays the most standard deviation for low fitness (4565, compared to 3104 for ALG and 2337 for STAT). This highlights an issue in poorly-performing SA controllers, brought about by a combination of poor rates and poor control parameters, settling certain controllers in local minima that can be difficult to escape from. Practically this does not affect the ability of SA to generate useful controllers, as the lowest-fitness controllers are not used by the multirotor. No rate resets were required in the SA experiment, showing that self-adaptation allows for consistent fitness improvements for the entire population, despite the additional variance inherent in the process.

Fitness graphs (Fig. 4(a)-(c)) shows the ALG profile for high, average, and low fitness consistently trails behind the others, and presents a more gradual, linear profile compared to the sinusoidal SA and STAT profiles. STAT consistently outperforms SA in initial generations (possibly due to the additional search required to find feasible SA in early generations), however SA regains the advantage within ≈18 generations due to the ability to tailor the search process to the state of the population explicitly.

The additional complexity of SA rates could reasonably be expected to lengthen convergence times. However, SA (mean convergence generation = 24.6) is statistically similar to STAT (value=27.4, p> 0.01), and significantly expedites convergence compared to ALG (value = 70.5, p< 0.01) (Fig. 4(d)). As STAT is also significantly faster than ALG, this observation seems tied to good rate setting rather than in-experiment rate variation. Note the SA outlier (diamond) in Fig. 4(d), illustrating the variance involved in self-adaptive rate setting. This is echoed in the standard deviation for SA (10.6) compared to STAT (6.3). Practically, SA cuts the time required per experiment by ≈66% compared to ALG. Given that SA is statistically similar to STAT, it seems to be the best option for use in our platform, as SA's ability to perform context-sensitive parameter setting should allow the platform to handle more variance in terms of the multirotors and payloads it optimises for.

SA as a whole can be seen to flexibly search the solution space, as evidenced in Fig. 4(a), which show more fitness variance in the most fit controllers at the start (generations 18 - 28, higher standard error / more emphasis on solution space

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TEVC.2017.2703142, IEEE Transactions on Evolutionary Computation

TABLE I
PERFORMANCE AND CONVERGENCE METRICS FOR THE THREE EXPERIMENT TYPES.

	ALG		SA		STAT	
Parameter	Mean	Stdev	Mean	Stdev	Mean	Stdev
Best f	124020.7	1355	127185.2	2576	123531.1	2864
Mean f	117483	2060	122562.6	2820	118450.3	2243
Low f	110795	3104	117480.6	4565	112323.8	2337
Converged	70.5	15.9	24.6	10.6	27.4	6.3

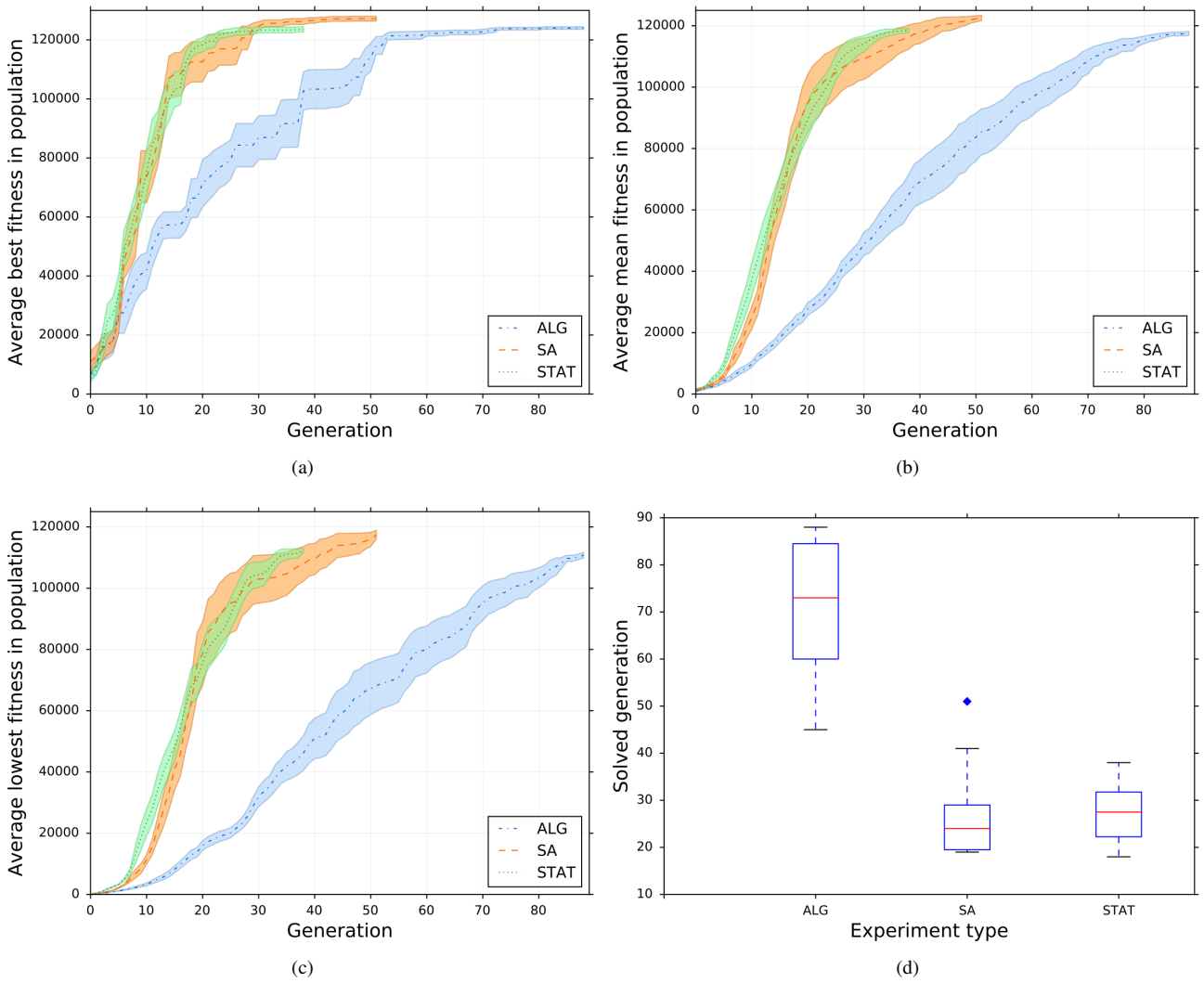


Fig. 4. (a) Best, (b) average, and (c) worst fitness for the experiment types. Shaded areas denote standard error. (d) Showing a boxplot for the convergence generations of all experiment types — diamonds denote outliers. Plots continue until the final repeat of a given experiment type is completed.

exploration), and less variance near the end of an experiment (generation 28 onwards, lower standard error / more emphasis on exploitation). As a comparison, STAT populations are seen to have approximately equal fitness errors (as their rates cannot change, population variance is similar throughout an experiment). We propose that this is the main reason for both the low convergence generation, and the higher fitness values in the SA experiment.

Hover stability is assessed by hovering the multirotor at waypoint 1 for 30 minutes with the 2m/s fan using the best controller. Horizontal position and height did not exceed 5cm while the multirotor during hover using the highest-fitness SA gainset. Yaw error did not exceed 10^o and the multirotor did not stress either of the mechanical tethers. To assess the generalisability of the evolved controllers, we repeat the assessment using a 5m/s fan, offset to 90^o of the original fan's position. The same controller is shown to generalize to the more difficult conditions, achieving error for horizontal position=10cm, height=2.5cm and yaw=3.1^o without further evolution in these more challenging conditions. All measured

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TEVC.2017.2703142, IEEE Transactions on Evolutionary Computation

errors are taken within the 95th percentile of absolute error.

B. Self-adaptive Rates

Practically, *CR* is the per-allelle probability of including a given element of the donor vector v in the new child c, so higher *CR* reduces the liklihood of c strongly resembling its parent p. Differential weight *F* affects the magnitude of changes in the donor vector.

The average final rates *CR* and *F* in the SA experiment are lower than the average initial rates, which indicates that self-adaptation is used to stabilise the search process towards the end of the experiment, and is responsible for the ability for SA to find higher fitness controllers (see Fig. 4(a)). Higher rates are seen early on, as SA induces more population variance between subsequent generations to expedite the search of the solution space. The higher fitness values achieved by SA compared to STAT shows that the ability to alter rates throughout an experiment, rather than just the setting of beneficial rates, is beneficial to the controller evolution.

Mean crossover rate *CR*, seen in Fig. 5(a), shows an approximately stable rate progression through the first 50 generations, raising slightly to a maximum of 0.539 after 16 generations, before declining to 0.482 after 22 generations, and maintaining that value until generation 50. After an initial growth period (generations 0-10), standard error is also stable throughout, at $\approx$ 0.07.

F, depicted in Fig. 5(b), shows a more directed profile, whose mean value descends through the first 20 generations from an initial setting of 1.04 to 0.381 at generation 20, followed by a more gradual decrease to 0.29 at generation 50. The stable selection of this rate through generations 20-50 indicate that SA favours this value, and the stability of the rate setting justifies our decision to base the STAT rate settings on the mean of the SA experiment settings.

As with *CR*, standard error for *F* Fig. 5(b) is seen to initially grow as more variance is induced by the self-adaptive mechanism. Gradual error growth is observed between generations 0 ($\approx$0.044) and 10 ($\approx$0.129), followed by a steady decrease in error from $\approx$0.09 at generation 20 to $\approx$0.05 at generation 50. As *F* finishes with lower error than *CR*, and has a more clearly selected-for rate setting, we conclude that the setting of *F* is more critical to the evolutionary process. This is in line with previous experimentation, e.g., [4].

The most expediently-converged experiment (16 generations) displays a *CR* profile that raises to 0.58 after 2 generations, before declining to 0.51 at generation 5 and peaking to a maximum of 0.705 at generation 7. Following this, *CR* declines rapidly to a final value to 0.461 at generation 16. *F* for this experiment starts at $\approx$1.0, drops to 0.36 at generation 5, and, following a brief plateau to 0.34 at generation 8, descends to a low of 0.158 at generation 12 before slightly increasing to 0.194 at generation 16. This final value is below the mean population value for *F* (0.29). For *CR* and *F*, the combination of higher rates (more exploration) at the start of an experiment and lower rates (more stable search) towards the end accounts for the low convergence generation of this controller.

VIII. Control Parameters

To aid in interpreting the control parameter results, *high* gains — far from 0 — increase the strength of control responses to the error signal, whereas *low gains*, which are closer to 0, have a weaker response to control commands. Practically a balance is needed; low gains tend to produce sluggish responses, and high gains may overshoot or cause controller oscillations [22]. The performance of a controller is a result of all 18 gains, although certain gains may be more or less critical to performance, depending on, e.g., multirotor type, payload, etc. We adopt the shorthand of P for proportional gains, I for integral gains, and D for derivative gains.

A. Best Single Parameter Set

For the best single controllers (the first column per experiment in Table II), we note that D gains are the lowest for all position variables (the controllers are less strongly influenced by the derivative of the error signal over time than the other gain types). This indicates that the controllers are more sensitive to D errors than P or I error signals.

In terms of the magnitude, position gains follow the similar descending trend of I/P/D, P/I/D, P/I/D for h, x, and y respectively — Kiy for ALG (9.71, compared to 3.847 for SA and 2.491 for STAT) is the notable exception, which generates small amounts of control oscillation and accounts for the reduced fitness of this controller. As the only difference between the experiments is the use of self-adaptation, the inability to concurrently parameterise the entire gain set with suitable values can be directly attributed to the inability to locate suitable instantaneous settings of *CR* and *F*.

A similar trend is seen for attitude variables for roll (ρ) and pitch (θ) (I/P/D, P/I/D respectively), but yaw (ψ) for each experiment is different (D/P/I for ALG, I/P/D for SA, P/D/I for STAT). This indicates that more combinations of gains for yaw are acceptable, as they do not follow a strcit order. Except for yaw, SA has universally lower D gains than STAT, which may account for the comparatively lower fitness in the latter as the higher D gains cause small, persistant oscillations, especially in horizontal position control.

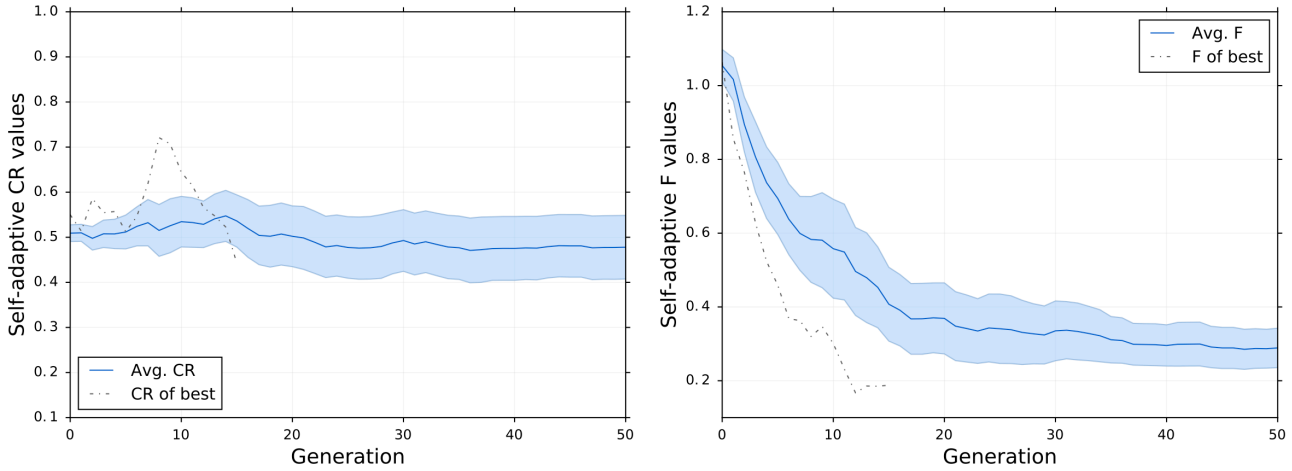
Between the three experiments, we note that SA does not always produce the lowest or highest gain, but can rather concurrently find gains that compliment each other. This highlights the existance of coupling between the control variables, which is reflected in the performance of the gains and makes the search space more complex to successfully traverse. Exploiting coupling is a stated reason for wishing to simultaneously optimise position and attitude gains (see Section II-B2).

B. Population Controller Parameters

The final two columns per experiment type in Table II show values of the mean gain, and its standard deviation.

With the exception of $K_{\mathrm{d}\psi}$=3.15, D gains are lower than P and I across all experiments and degrees of freedom. As with the best controllers, D gains also have the smallest standard

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TEVC.2017.2703142, IEEE Transactions on Evolutionary Computation



(a) (b)

Fig. 5. Showing population average and single best (fastest converged experiment) self-adaptive values for DE parameters (a) *CR* and (b) *F* in the first 50 generations of the SA experiment. The shaded areas denote standard error.

deviations, meaning the values of D gains are most critical to the performance of the controller as all flyable controllers have D gains in a small region of the available parameter space. I gains have higher average standard deviation than P or D gains, indicating that they have a wider range of acceptable values and are less critical to the performance of the controller, see Table II.

All experiments follow the descending trend of h=I/P/D, x=P/I/D, x=P/I/D for position — we reason that as x and y are practically identical in terms of the control they require and the state space they can explore, it follows that the gains for x and y are similar. ALG gains K_{ih}, K_{px}, K_{py}, and K_{iy} are notably higher than those of the other experiment types, which may account for the reduced performance of ALG controllers.

IX. Varying the Payload

To demonstrate the ability of the platform to create context-sensitive controllers and corresponding SA rates, we attach a payload — an off-centre 280g lead weight — to the front of the multirotor. Note that the weight and off-centre positioning make this optimisation much more difficult than the first hovering experiment.

We run the best SA gainset (original best f=129131) 30 times without evolution with the new payload, giving a best $f = 74687.7$, and an average $f = 21352.3$ with a standard deviation of 17258.7. More importantly, 28 of the 30 repeats did not achieve a *success*, and a standard SA experiment with random initial controllers was terminated after 5 hours having achieved only 11 of the required 20 viable precursor controllers in the bootstrapping process. Combined, these factors highlight need for retuning the controller when equipping multirotors with different payloads.

To compensate for the increased mission difficulty, a form of incremental evolution is used to create a population as before to $N = 20$, where each population member is mutated from the best SA controller, $\pm 25\%$ of the initial range for each gain (ranges are determined following (2)). The experiment then follows the procedure for an SA experiment, until convergence.

The new population converges in 55 generations, achieving a best fitness of f=120540, with a mean population f=116420 and low population f=113382. Best fitness is almost double the best fitness of the original best controller with the new payload and achieves over 93% of the fitness of the original controller (129181) in the first experiment (without payload).

Fig. 6(a) shows that the highest-fitness controller is found after 30 generations. The fitness profile for best fitness shows an initial best f =≈72,000 — a result of the bootstrapping process — and slowly improves until generation 21, where it jumps to f =≈112,000. Mean fitness and low fitness follow more gradual profiles.

The difficulty of this optimisation is highlighted by two periods of rate resets — around generations 34 and 44. At generation 34, the resets are used to induce more variance to underperforming controllers, jumping the mean *CR* from 0.19 to 0.54 and *F* from 0.63 to 0.91 (Fig. 6(b)). A corresponding increase in low fitness from ≈83,000 to ≈93,000 (generations 35 to 38) can be seen in Fig. 6(a). The second reset stabilises the search process, mainly be decreasing *CR* from 0.61 to 0.41. Together with a high *F*, this suggests that certain controllers have a number of control parameters correctly set (lower *CR*), but are far from the optimal value (higher *F*).

Mean SA rates are higher than in the original experiment — new *CR*=0.537 compared to ≈0.48, and *F*=1.18 compared to ≈0.29. This indicates that the more converged initial population requires higher rates of evolutionary search to find the correct control parameters. As the population is more converged in terms of control parameters — and therefore performance — these high rates are not as disruptive as in the original experiment. Fig. 6(b) shows that generational variance in *CR* and *F* is similar; a decrease until generation ≈30, followed by a sharp uneven and generally upwards trend until the end of the experiment. The first successful controller

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TEVC.2017.2703142, IEEE Transactions on Evolutionary Computation

TABLE II
THE SINGLE BEST CONTROLLER, AND THE POPULATION MEAN AND STANDARD DEVIATION FOR EACH FINAL EVOLVED CONTROL PARAMETER PER EXPERIMENT. MEAN AND STANDARD DEVIATION ARE TAKEN OVER 10 REPEATS.

	Algorithmic			Self-adaptive			Static SA		
Parameter	Single Best	Mean	Stdev	Single Best	Mean	Stdev	Single Best	Mean	Stdev
K_{ph}	-4.61	-7.51	2.8	-9.35	-9.35	2.4	-8.34	-9.34	2.3
K_{ih}	-10.1	-12.94	6.2	-10.91	-11.52	3.7	-15.44	-10.4	3.1
K_{dh}	-2.38	-2.7	0.7	-3.13	-2.80	0.7	-3.22	-3.29	0.8
K_{px}	-14.176	-10.585	4.4	-8.873	-6.914	2.65	-7.965	-6.422	1.84
K_{ix}	-6.343	-5.378	4.35	-2.929	-4.958	3.26	-2.585	-3.886	2.3
K_{dx}	-1.937	-1.825	0.81	-1.888	-2.121	0.64	-2.313	-1.884	0.76
K_{py}	8.21	9.323	4.73	6.208	5.993	2.29	8.051	5.666	3.07
K_{iy}	9.71	5.896	3.95	3.847	3.471	2.38	2.491	3.264	2.48
K_{dy}	1.809	1.632	0.81	1.321	1.837	1.29	2.21	1.936	0.99
$K_{\mathrm{p}\phi}$	-11.253	-9.766	3.41	-8.805	-9.259	2.75	-6.921	-9.119	3.39
$K_{\mathrm{i}\phi}$	-15.457	-19.371	12.73	-12.772	-10.445	5.35	-12.133	-7.787	4.74
$K_{\mathrm{d}\phi}$	-2.192	-2.265	0.67	-1.531	-2.07	0.72	-2.044	-2.112	0.88
$K_{\mathrm{p}\theta}$	-8.701	-7.055	3.59	-11.587	-8.083	3.34	-9.151	-5.925	2.56
$K_{\mathrm{i}\theta}$	-7.179	-11.046	8.88	-11.368	-8.828	4.88	-5.951	-7.233	5.77
$K_{\mathrm{d}\theta}$	-1.537	-2.126	0.55	-2.384	-1.988	0.54	-3.447	-2.105	0.76
$K_{\mathrm{p}\psi}$	-7.428	-6.770	2.98	-8.554	-6.553	2.63	-12.212	-8.092	3.44
$K_{\mathrm{i}\psi}$	-5.075	-8.712	6.14	-13.951	-6.532	4.12	-5.516	-7.488	3.92
$K_{\mathrm{d}\psi}$	-10.117	-5.430	2.61	-8.211	-5.214	3.15	-6.758	-5.988	2.97

emerges when SA rates are low (*CR*=0.2, *F*=0.55) — an example of more fine-grained search being used to create a solving controller from a near-solving controller.

Controller variance shows an average absolute difference from the initial setting of 13.65% for P gains, 12.57% for I gains and 9.96% for D gains, with mean actual changes of 5.74, 2.81 and 0.75 respectively). The lower variance of D gains reinforces the notion that they are the most sensitive.

Adaptation to the payload mainly occurs through K_{iy}, which decreases by 24% from 1.098 to 0.831. Noting that y is most strongly affected by ρ, we also observe an sizeable increases in $K_{\mathrm{p}\rho}$ (from -11.0564 to -9.33999) and $K_{\mathrm{i}\rho}$ (from -7.54815 to -6.45127). As these are the three most strongly varying gains, and backed up by the flight performance (indicated by fitness), we present this result as evidence of a successful context-sensitive adaptation to the payload, which causes off-centre moments of inertia and drag coefficients on this axis. A more detailed interpretation of these results is that y position suffers from overshoot due to momentum of the increased weight, and as such reduces its integral term to reduce observed oscillations in the initial controllers that are induced by the payload. Another effect of the payload is an increased latency on roll commands, hence higher gains are selected for roll. Together, these gains reach a solution that allows the payload to be successfully accounted for.

X. SUMMARY

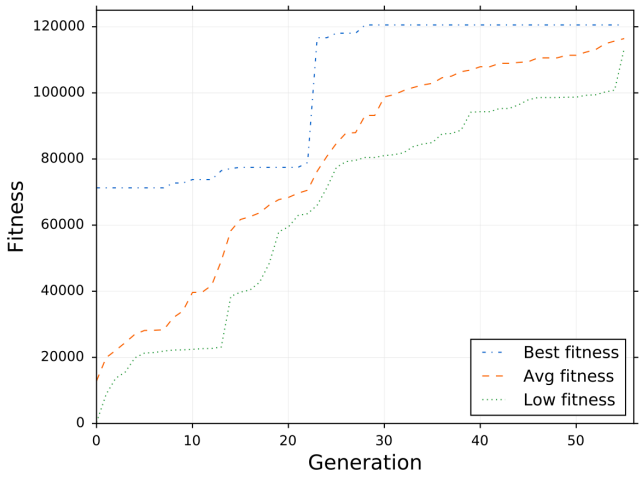
In this article we have presented the first platform that allows evolutionary robotics experiments to be performed nondestructively and repeatedly, directly on multirotor flying robots. The platform automatically generates context-sensitive multirotor controllers from scratch in a closed loop with no requirement for a model or pilot. Contributions are both experimental (allowing for the statistical testing of ER hypotheses), and functional (potentially allowing for more successful mission outcomes and longer flight times in the real world).

Self-adaptation is shown to discover mutation rates and control parameters that are specific to the hardware state of the multirotor, and the payload it carries. In the first experiment, self-adaptation generates controllers with the statistically highest fitness, and the fastest convergence times. We note the utility of self-adaptation for our platform specifically, as it will be required to adapt to a wide range of multirotors, payloads, etc. In the second experiment, when varying the payload, we justify our approach in terms of showing context sensitivity, as different self-adaptive rates and control parameters are discovered.

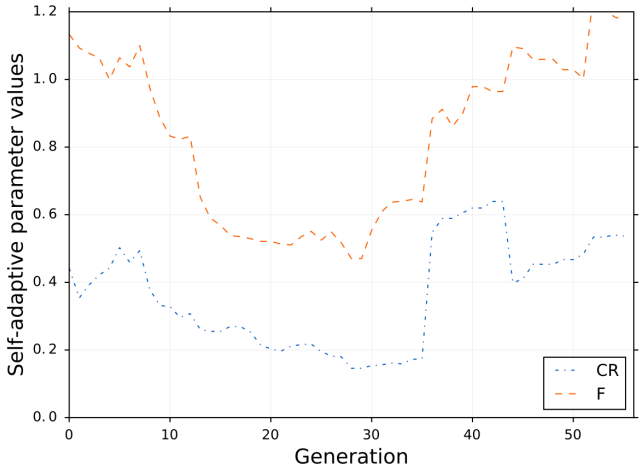
As PID is the most common multirotor controller, the platform will be able to optimise most multirotors. As the controllers are directly evolved on the robot, they are guaranteed to work in reality. By adding a battery as a payload pre-evolution, removing the tethers, replacing the camera position estimate with one derived from a differential GPS, and transferring the controller to the pre-existing onboard computer, the multirotor can be flown freely. Initial experimentation has shown this approach is viable.

Future research includes, in the short-term, testing a number of multirotors and payloads, which will allow us to analyze their effects on the optimal rates and control parameters. Comparing different EAs or controller representations may be a fruitful research direction. Taking a longer view, we note a recent focus by companies (e.g., BAE) and research institutions in creating specialised aircraft in small numbers, which are designed for a specific task and part of a wider trend towards application-specific bespoke hardware. This follows a move from "Evolutionary algorithms to the evolution of things" [10]. Platforms such as ours will have a place in the design of controllers and repeatable testing of robot designs, where robot morphologies will be heterogeneous and discovered through evolutionary processes.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TEVC.2017.2703142, IEEE Transactions on Evolutionary Computation



(a)



(b)

Fig. 6. Showing (a) best, mean, and low fitness, and (b) self-adaptive DE parameters, when the payload is changed.

Appendix A
Symbol Definitions

l_ω : max. pitch and roll rate in closed-loop system/ (115^o/s)
$l_{\omega\mathrm{n}}$: pitch and roll rate noise threshold (30^o/s)
l_{vhn} : horizontal velocity noise threshold (5cm/s)
l_{vh} : max. horizontal velocity in closed-loop system (15cm/s)
l_{vvn} : vertical velocity noise threshold (2cm/s)
l_{vv} : max. vertical velocity in closed-loop system (20cm/s)
l_{a} : attitude range limit (15^o)
l_{h} : height range limit (10cm)
l_{hc} : core height limit (5cm)
l_{yc} : core yaw limit (15^o)
l_{y} : yaw range limit (160^o)
l_{pc} : core position limit (8cm)
l_{p} : position range limit (20cm)

v_{n} : north velocity
v_{e} : east velocity
v_{v} : vertical velocity
p_{n} : north position
p_{e} : east position
h : height
p_{nsp} : north position setpoint
p_{esp} : east position setpoint
h_{sp} : height setpoint
ψ : yaw
θ : pitch
ϕ : roll
ψ_{sp} : yaw setpoint
θ_{sp} : pitch setpoint
ϕ_{sp} : roll setpoint
ω_{p} : roll rate
ω_{q} : pitch rate
ω_{r} : yaw rate

Appendix B
Fitness Function

$$f_{\mathrm{cycle}} = f_{\mathrm{a}_i} + f_{\mathrm{vh}_i} + f_{\mathrm{vv}_i} + f_{\mathrm{h}_i} + f_{\mathrm{y}_i} + f_{\mathrm{p}_i} + f_{\mathrm{l}_i} + f_{\omega_i}$$

$$f_{\mathrm{l}} = \begin{cases} 0, & \text{if PWM limit reached} \\ 1, & \text{otherwise} \end{cases}$$

$$f_{\mathrm{a}} = \max\{1 - \frac{|\phi_{\mathrm{sp}} - \phi|}{l_{\mathrm{a}}}, 0\} + \max\{1 - \frac{|\theta_{\mathrm{sp}} - \theta|}{l_{\mathrm{a}}}, 0\}$$

$$f_{\mathrm{vh}} = \max\{1 - \frac{\mathrm{db}\{\sqrt{v_{\mathrm{n}}^2 + v_{\mathrm{e}}^2}, l_{\mathrm{vhn}}\}}{l_{\mathrm{vh}}}, 0\}$$

$$f_{\mathrm{vv}} = \max\{1 - \frac{\mathrm{db}\{|v_{\mathrm{v}}|, l_{\mathrm{vvn}}\}}{l_{\mathrm{vv}}}, 0\}$$

$$f_{\omega} = \max\{1 - \frac{\mathrm{db}\{|\omega_{\mathrm{p}}|, l_{\omega\mathrm{n}}\}}{l_{\omega}}, 0\} + \max\{1 - \frac{\mathrm{db}\{|\omega_{\mathrm{q}}|, l_{\omega\mathrm{n}}\}}{l_{\omega}}, 0\}$$

$$f_{\mathrm{h}} = \begin{cases} \max\{\frac{|h_{\mathrm{sp}} - h|}{4(l_{\mathrm{h}} - l_{\mathrm{hc}})}, 0\}, & \text{if } |h_{\mathrm{sp}} - h| > l_{\mathrm{hc}} \\ \frac{3|h_{\mathrm{sp}} - h|}{4l_{\mathrm{hc}}} + \frac{1}{4}, & \text{otherwise} \end{cases}$$

$$f_{\mathrm{y}} = \begin{cases} \max\{\frac{|\psi_{\mathrm{err}}|}{4(l_{\mathrm{y}} - l_{\mathrm{yc}})}, 0\}, & \text{if } |\psi_{\mathrm{err}}| > l_{\mathrm{yc}} \\ \frac{3|\psi_{\mathrm{err}}|}{4l_{\mathrm{yc}}} + \frac{1}{4}, & \text{otherwise} \end{cases}$$

$$f_{\mathrm{p}} = \begin{cases} \max\{\frac{p_{\mathrm{err}}}{4(l_{\mathrm{p}} - l_{\mathrm{pc}})}, 0\}, & \text{if } p_{\mathrm{err}} > l_{\mathrm{pc}} \\ \frac{3p_{\mathrm{err}}}{4l_{\mathrm{pc}}} + \frac{1}{4}, & \text{otherwise} \end{cases}$$

$$p_{\mathrm{err}} = \sqrt{(p_{\mathrm{nsp}} - p_{\mathrm{n}})^2 + (p_{\mathrm{esp}} - p_{\mathrm{e}})^2}$$

$$\psi_{\mathrm{err}} = \mathrm{wrap}\{\psi_{\mathrm{sp}} - \psi\}$$

$$\mathrm{wrap}\{\alpha\} = \mathrm{atan2}(\sin(\alpha), \cos(\alpha))$$

$$\mathrm{db}\{x, l\} = \begin{cases} x, & \text{if } x > l \\ 0, & \text{otherwise} \end{cases}$$

f_{hover} : fitness for hover controller
f_{cycle} : fitness for one control step
f_{a} : fitness for pitch and roll
f_{vh} : fitness for horizontal velocity
f_{vv} : fitness for vertical velocity
f_{h} : fitness for height
f_{y} : fitness for yaw
f_{p} : fitness for horizontal position
f_{ω} : fitness for pitch and roll rates

Acknowledgment

D. Howard would like to thank Torsten Merz and Alberto Elfes for their helpful discussions. This work was supported

by the OCE Postdoctoral fellowship in Evolutionary Aerial Robotics, and CSIRO's Biosecurity and Data61 Business Units.

REFERENCES

[1] S. Doncieux, J.-B. Mouret, N. Bredeche, and V. Padois, "Evolutionary robotics: Exploring new horizons," in *New horizons in evolutionary robotics*. Springer, 2011, pp. 3–25.

[2] 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.

[3] D. Howard and A. Elfes, "Evolving spiking networks for turbulence-tolerant quadrotor control," in *International Conference on Artificial Life (ALIFE14)*, 2014, pp. 431–438.

[4] R. Storn and K. Price, "Differential evolution – a simple and efficient heuristic for global optimization over continuous spaces," *J. of Global Optimization*, vol. 11, no. 4, pp. 341–359, Dec. 1997.

[5] A. Biswas, S. Das, A. Abraham, and S. Dasgupta, "Design of fractional-order pi λ d μ controllers with an improved differential evolution," *Engineering applications of artificial intelligence*, vol. 22, no. 2, pp. 343–350, 2009.

[6] D. Howard and T. Merz, "A platform for the direct hardware evolution of quadcopter controllers," in *Intelligent Robots and Systems (IROS), 2015 IEEE/RSJ International Conference on*. IEEE, 2015, pp. 4614–4619.

[7] A. Nobakhti and H. Wang, "A simple self-adaptive differential evolution algorithm with application on the alstom gasifier," *Applied soft computing*, vol. 8, no. 1, pp. 350–370, 2008.

[8] S. Nolfi and D. Floreano, "Evolutionary robotics. the biology, intelligence, and technology of self-organizing machines," 2001.

[9] J. B. Pollack and H. Lipson, "The golem project: Evolving hardware bodies and brains," in *Evolvable Hardware, 2000. Proceedings. The Second NASA/DoD Workshop on*. IEEE, 2000, pp. 37–42.

[10] A. E. Eiben and J. Smith, "From evolutionary computation to the evolution of things," *Nature*, vol. 521, no. 7553, pp. 476–482, 2015.

[11] D. Rumelhart and J. McClelland, *Parallel Distributed Processing*. Cambridge, MA: MIT Press, 1986, vol. 1 & 2.

[12] N. Jakobi, P. Husbands, and I. Harvey, "Noise and the reality gap: The use of simulation in evolutionary robotics," in *Advances in artificial life*. Springer, 1995, pp. 704–720.

[13] S. Koos, J.-B. Mouret, and S. Doncieux, "Crossing the reality gap in evolutionary robotics by promoting transferable controllers," in *Proceedings of the 12th Annual Conference on Genetic and Evolutionary Computation*, ser. GECCO '10. New York, NY, USA: ACM, 2010, pp. 119–126.

[14] H. Heijnen, D. Howard, and N. Kottege, "A testbad that evolved hexapod controllers in hardware," in *Robotics and Automation (ICRA), 2017 IEEE/RSJ International Conference on*. IEEE, 2017, p. In press.

[15] J. C. Bongard and H. Lipson, "Nonlinear system identification using coevolution of models and tests," *Evolutionary Computation, IEEE Transactions on*, vol. 9, no. 4, pp. 361–384, 2005.

[16] O. E. Holland and R. D. Nardi, "Coevolutionary modelling of a miniature rotorcraft." in *Intelligent Autonomous Systems 10 (IAS10)*. IOS Press, January 2008.

[17] A. Cully, J. Clune, D. Tarapore, and J.-B. Mouret, "Robots that can adapt like animals," *Nature*, vol. 521, no. 7553, pp. 503–507, 2015.

[18] D. Floreano, J.-C. Zufferey, and J.-D. Nicoud, "From wheels to wings with evolutionary spiking circuits," *Artificial Life*, vol. 11, no. 1-2, pp. 121–138, 2005.

[19] M. Gongora, B. Passow, and A. Hopgood, "Robustness analysis of evolutionary controller tuning using real systems," in *Evolutionary Computation, 2009. CEC '09. IEEE Congress on*, 2009, pp. 606–613.

[20] K. Y. W. Scheper, S. Tijmons, C. C. de Visser, and G. C. H. E. de Croon, "Behavior trees for evolutionary robotics," *Artificial Life*, vol. 22, no. 1, pp. 23–48, Feb 2016.

[21] P. Ghiglino, J. L. Forshaw, and V. J. Lappas, "Online evolutionary swarm algorithm for self-tuning unmanned flight control laws," *Journal of Guidance, Control, and Dynamics*, vol. 38, no. 4, pp. 772–782, 2015.

[22] K. J. Åström and T. Hägglund, *Advanced PID control*. ISA-The Instrumentation, Systems, and Automation Society; Research Triangle Park, NC 27709, 2006.

[23] K. H. Ang, G. Chong, and Y. Li, "Pid control system analysis, design, and technology," *IEEE Transactions on Control Systems Technology*, vol. 13, no. 4, pp. 559–576, July 2005.

[24] A. Isidori, *Nonlinear control systems*. Springer Science & Business Media, 2013.

[25] A. Marlin, *Process Control*. McGraw-Hill Companies, 1995.

[26] H. Shim, T. J. 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, 1998, pp. 3653–3658.

[27] 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 *Experimental Robotics IX*. Springer, 2006, pp. 363–372.

[28] P. Bouffard, A. Aswani, and C. Tomlin, "Learning-based model predictive control on a quadrotor: Onboard implementation and experimental results," in *Robotics and Automation (ICRA), 2012 IEEE International Conference on*. IEEE, 2012, pp. 279–284.

[29] N. Karaboga and B. Cetinkaya, "Performance comparison of genetic and differential evolution algorithms for digital fir filter design," in *Advances in Information Systems*, ser. Lecture Notes in Computer Science, T. Yakhno, Ed. Springer Berlin Heidelberg, 2005, vol. 3261, pp. 482–488.

[30] R. Dong, "Differential evolution versus particle swarm optimization for pid controller design," in *Natural Computation, 2009. ICNC '09. Fifth International Conference on*, vol. 3, Aug 2009, pp. 236–240.

[31] J. Brest, S. Greiner, B. Boskovic, M. Mernik, and V. Zumer, "Self-adapting control parameters in differential evolution: a comparative study on numerical benchmark problems," *IEEE transactions on evolutionary computation*, vol. 10, no. 6, pp. 646–657, 2006.

[32] A. E. Eiben, R. Hinterding, and Z. Michalewicz, "Parameter control in evolutionary algorithms," *IEEE Transactions on Evolutionary Computation*, vol. 3, no. 2, pp. 124–141, Jul 1999.

[33] A. E. Eiben, G. Karafotias, and E. Haasdijk, "Self-adaptive mutation in on-line, on-board evolutionary robotics," in *Self-Adaptive and Self-Organizing Systems Workshop (SASOW), 2010 Fourth IEEE International Conference on*, Sept 2010, pp. 147–152.

[34] G. Howard, E. Gale, L. Bull, B. de Lacy Costello, and A. Adamatzky, "Evolution of plastic learning in spiking networks via memristive connections," *IEEE Transactions on Evolutionary Computation*, vol. 16, no. 5, pp. 711–729, 2012.

[35] J.-M. Montanier and N. Bredeche, "Embedded Evolutionary Robotics: The (1+1)-Restart-Online Adaptation Algorithm," in *New Horizons in Evolutionary Robotics*, S. S. S. in Computational Intelligence, Ed. Springer, 2011, pp. 155–169.

[36] M. Eigen, *Ingo Rechenberg Evolutionsstrategie Optimierung technischer Systeme nach Prinzipien der biologishen Evolution*. mit einem Nachwort von Manfred Eigen, Friedrich Frommann Verlag, Struttgart-Bad Cannstatt, 1973.

[37] M. G. Omran, A. Salman, and A. P. Engelbrecht, "Self-adaptive differential evolution," in *Computational intelligence and security*. Springer, 2005, pp. 192–199.

[38] T. Merz, P. Rudol, and M. Wzorek, "Control system framework for autonomous robots based on extended state machines," in *Autonomic and Autonomous Systems, 2006. ICAS'06. 2006 International Conference on*. IEEE, 2006, pp. 14–14.

[39] C. Anil and R. Padma Sree, "Tuning of proportional integral derivative controllers for integrating systems using differential evolution technique," *Indian Chemical Engineer*, vol. 53, no. 4, pp. 239–260, 2011.

David Howard David is a Research Scientist in the Autonomous Systems Program in Data61, a part of CSIRO. His research interests include evolvable hardware, evolutionary robotics, neuro-evolution, learning classifier systems, and the application of evolutionary algorithms to challenging real-world robotic optimisation tasks.