USING CMA-ES FOR TUNING COUPLED PID CONTROLLERS WITHIN MODELS OF COMBUSTION ENGINES

K. Henclová[*]

Abstract: Proportional integral derivative (PID) controllers are important and widely used tools of system control. Tuning their gains is a laborious task, especially for complex systems such as combustion engines. To minimize the time an engineer spends tuning the gains in a simulation software, we propose to formulate a part of the problem as a black-box optimization task. In this paper, we summarize the properties and practical limitations of gain tuning in this particular application. We investigate the latest methods of black-box optimization and conclude that the Covariance Matrix Adaptation Evolution Strategy (CMA-ES) with bi-population restart strategy, elitist parent selection and active covariance matrix adaptation is best suited for this task. Details of the algorithm's experiment-based calibration are explained as well as derivation of a suitable objective function. The method's performance is compared with that of PSO and SHADE. Finally, its usability is verified on six models of real engines.

1. Introduction

In a running combustion engine, one or more PID controllers ensure that certain quantities (e.g. intake pressure or exhaust gas temperature) remain constant or within given range. Since these quantities may be naturally related and affect each other, controllers are often coupled and their gains cannot be tuned independently of each other.

When engines are modeled, as in simulation software WAVE[1], controllers may and need to be tuned using simulations with little or no knowledge about the transfer function that describes the system. The complicated finite-element model

---

[*]Kateřina Henclová; Czech Technical University in Prague, Faculty of Nuclear Sciences and Physical Engineering, Břehová 78/7, CZ-115 19 Praha 1, Czech Republic, E-mail: katerina.henclova@fjfi.cvut.cz
[1]WAVE is 1D engine and gas dynamics simulation software package developed by Ricardo Software [33] and it was used by the author of this paper. This software has been utilized by world's leading automobile companies for many years and has proven its modelling quality.

cannot be described by a simple formula and, moreover, it comes with its own modeling and discretization errors.

Presently, manual work makes up a major part of the controller tuning process. This lengthy procedure is based on trial and error and requires a knowledgeable and experienced control engineer. For systems with a single controller (or multiple but decoupled controllers), simple rules of thumb can be employed (e.g. Ziegler-Nichols [38]). Similar, already-solved problems can also provide a guideline. However, when having a complicated or unique system of coupled controllers, the complexity of the task makes it very difficult to solve even for an experienced control engineer. Moreover, in our application of PID controllers within combustion engine models, other professionals need to tune the controllers as well, creating the need for a simple-to-use, robust tool.

Our goal is to formulate at least a part of the problem and deliver a method that would eliminate or significantly lower the need for manual tuning. It should find a solution within acceptable time and with as little user interaction as possible, making the controller tuning accessible for users with little experience. When combined with simple tuning rules or educated guess, our method is to use the provided solution approximation as a starting point and quickly find a more refined solution. Further testing of solutions found by our method (such as verifying their robustness) is, however, still left for manual post-processing. In principle, it is simple to replace the proposed objective function by its expected value over parametric uncertainties. However, the uncertainties on the parametric space are not available for the considered application and thus it is not a part of this work. Moreover, such approach would significantly increase the computational load.

The proposed method is used to design the controller gains for a predefined trajectory of the controller. For more complex system that requires the use of gain scheduling [1] or fuzzy PID [31], the proposed method will be used to design the gain in the operational point designed by the supervising engineer. The engineer will also be responsible for selection of the gain scheduling or fuzzy rules.

The PID tuning problem with either one controller or multiple but decoupled or symmetric controllers can be and has been reformulated as a black-box optimization problem and solved with an appropriate method. Evolutionary algorithms too have been used to tune PID controllers, e.g. genetic algorithm [27], differential evolution (DE) [5,22], particle swarm optimization (PSO) [7,8,29] and many hybrids [22,26].

The tuning problem with multiple coupled controllers can too be formulated as an optimization problem, albeit more complex. (To the author's knowledge, there have not been any papers dealing with such problems.) A simple trick can be used to carefully transform the multiple objectives (one for each controlled quantity) into a single objective and thus enable the problem to be solvable by usual means.

The time budget sets the greatest limitation. With simulations taking up to several minutes each, we aim for an overnight or a one-day computation on a regular PC, i.e. a few thousand simulation runs at most. This imposes high expectations upon efficiency of the method used.

Considering properties of the problem, we choose to use a variant of the Covariance Matrix Adaptation Evolution Strategy (CMA-ES) [15,19,20,24], an evolutionary algorithm founded deep in probability theory. It has proven to be very effective and robust method in the extensive testing of Black-Box Optimization

Benchmarking (e.g. [4, 18]), surpassing the above mentioned algorithms and many others (on the relevant sort of problems). Despite its fame in the optimization community and large number of practical applications, it has so far been little used for tuning PID controllers [22, 23, 37] or similar problems [16, 17].

In this paper, we experiment with the algorithm's settings and come up with a method that meets practical requirements, such as tolerable runtime, and is thus fit for common use by engineers working with complicated simulations (e.g. users of WAVE). Moreover, we demonstrate its applicability on six models of real-world engines with one, two and three controllers.

For comparison with CMA-ES, we use PSO (as implemented in the software package DEAP [6]) and a restarted version of SHADE (Success-History based Adaptive DE [34]) with an implementation based on that by Tanabe [35]. We show that they do not perform by far so well as CMA-ES.

2. Formulation of the problem

In this section, the multiple-controller tuning problem is formulated as a problem of numerical optimization. Compared to other research on controller tuning [5, 7, 8, 22, 26, 27], dealing with coupled controllers requires an extra level of complexity.

2.1 PID controllers in the context of engine simulations

PID controllers are well known and powerful tools in system control [9, 30]. Their input is the error function

$$e(t) = |\text{actual}(t) - \text{target}(t)|, \tag{1}$$

i.e. the time-dependent absolute difference between the desired target value and the actual value of a quantity (as measured by a sensor or computed by a model). The output control signal that defines the system's subsequent reaction is given as

$$C(t) = Pe(t) + I \int_0^t e(\tau)d\tau + D \ \frac{d}{dt}e(t), \tag{2}$$

where P, I and D are the proportional, integral and derivative gains, respectively.

In our application, the controllers' implementation is provided within the simulation software WAVE. All the computations as well as the whole engine model are given to us as a black box (as by an end user) and we cannot analyze them. The software as well as the models are calibrated to mirror the real world. The simulation is fully deterministic and we do not introduce any random perturbations of environment variables during the tuning process.

We input the controllers' gains into the simulation software and obtain computed trajectory of the controlled quantities over time. Our goal is to find such *constant* gains P, I and D for each controller, that the corresponding controlled quantities converge to the target value and do so as quickly as possible. The entire tuning process is offline but, due to nonlinearities in the system, the curve is not as smooth as it is common for simple systems.

In this work, the initial values and the (nonzero) constant target values of the controlled quantities are provided and comply the context of combustion engines.

2.2 Definition of the objective function

With k controllers within a system, each determined by three constant gains P, I and D, there are $3k$ gains to be tuned: $x = (P_1, I_1, D_1, \ldots, P_k, I_k, D_k)$. When the controllers' gains are set and the whole simulation is run, it outputs the error functions $e_i(t) = e_i(x, t)$, $i = 1, \ldots, n$, described by Eq. (1). It remains to process e_i so that the final function value contains all information about the input's (candidate solution's) quality. Without loss of generality, we always assume that that higher quality inputs have lower function values, thus having the problem of numerical minimization. For practical purposes, the minimizer found need not be unique.

In this section, vector x is arbitrary but fixed. Hence, for simplicity of notation, x shall be left out from the functions' arguments.

2.2.1 Controlling multiple objectives

Let us assume for a moment that once controllers' gains are set, quality of the setting is characterized by functions $F_i(t) : \mathbb{R}^{3k} \to \mathbb{R}$, $i = 1, \ldots, n$, that describe the n controlled quantities' development (simulation time t being given), i.e. n objectives to be optimized. In our case, the number of controllers is the same as the number of controlled quantities (i.e. $k = n$) but, in general, this is not required. We expect k to be small, typically $k \leq 5$.

These objectives can be combined into a single one using a weighted sum (see e.g. [28]). Then the objective function describing the response of the whole model is $F(t) = \sum_{i=1}^{n} w_i F_i(t)$, where w_i is the weight constant corresponding to the i-th controlled quantity. Since the range of F_is may differ significantly depending on the corresponding units, we make them comparable by setting $w_i = p_i/\text{scale}_i$, where p_i describes priority of the i-th objective (in this work, we always have $p_i = 1$, $i = 1, \ldots, n$) and scale_i contains knowledge of its typical value or range. We further set $\text{scale}_i = \text{target}_i > 0$ to be the (constant and nonzero) target value of the i-th controlled quantity. target_i is the remainder of the integral under its (constant) curve, considering that the simulation time is equal for all the objectives. Thus we have
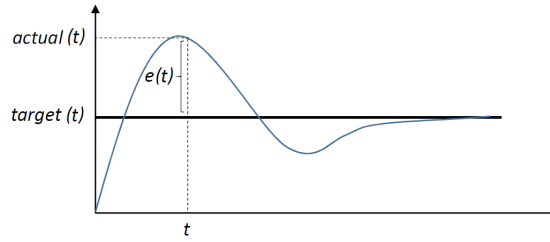
$$F(t) = \sum_{i=1}^{n} \frac{1}{|\text{target}_i|} F_i(t) \tag{3}$$

and it remains to define $F_i(t)$.

2.2.2 Step-response based objective function

Defining $F_i(t)$ is based on the step response curve, see Fig. 1. The step response describes how the controlled quantity behaves after an engine is started. The abrupt step change of the target value is the most difficult part for the controllers to deal with. Once the engine is running, the changes in the environment and the target values are not so sudden. Thus, when we tune the controllers for the "worst case scenario", there is a good reason to believe (supported by empirical evidence) that they will be able to control other cases as well. When the environment changes too much, e.g. hitting the extremes of RPM (rotations per minute), the controller gains may be changed using gain scheduling [1].

Fig. 1 *For a given setting of controllers, the error at time t is given as $e(t) = |actual(t) - target(t)|$.*

Functions F_i are naturally given by formula

$$F_i(t) = \int_{t_0}^{t} T(\tau)\, E(e_i(\tau))\, d\tau, \quad t_0 \geq 0, \tag{4}$$

where t_0 is the initial time (typically $t_0 = 0$), $T(\tau)$ is a function of time, $E(e_i(\tau))$ is a function of error, $e_i(\tau)$ is the error as defined in Eq. (1). In our case, t is given by the end of the simulation. Function $E(e_i(\tau))$ describes the concern over the actual error size, while $T(\tau)$ the concern over when the error occurs (i.e. $T(\tau)$ can also be perceived as weight). The commonly used criteria of step response quality (IAE, ITAE, ITSE or ISE) all fit this framework. Considering the application, the ITAE criterion [10] was chosen:

$$\text{ITAE}(t) = \int_{0}^{t} \tau |e(\tau)|\, d\tau. \tag{5}$$

The information about when error occurs must not be lost even after combining multiple objectives in Eq. (3), leading to further modifications. First, the time is shifted, so that the value of one second loses its importance as a factor. Then, nonconvergent solutions could be penalized. Or the importance of time can be stressed by increasing the degree of its polynomial. Or the beginning of the time interval can be left out, as the corresponding large error is – from the optimization viewpoint – only noise. Various possibilities have been experimented with and the best option found was to set

$$F_i(t) = \int_{t_0}^{t} (\tau + 1)|e_i(\tau)|\, d\tau \quad t_0 \geq 0. \tag{6}$$

Selection of shift $t_0 \geq 0$, its justification and influence upon performance is discussed later.

2.2.3   The objective function

In previous steps, we have constructed the objective function

$$F(x, t) = \sum_{i=1}^{n} \frac{1}{|\text{target}_i|} \int_{t_0}^{t} (\tau + 1)|e_i(x, \tau)|\, d\tau, \tag{7}$$

where n, $target_i$ and $0 \leq t_0 < t$ are given and the error functions e_i are computed by the simulation software.

This objective function can be easily modified to fit the needs of other applications. For example, one can incorporate regularization to enforce smooth solutions.

2.3 Character of the problem and practical limitations

Character of the problem plays a crucial role when choosing a fitting method to solve it. For any set of controllers' gains, the FEM simulation provides us with all data necessary to compute the objective function value Eq. (7) but we are forced to take it as a black box. We can carefully assume that, in general, the objective function is non-convex, non-linear, non-quadratic and highly multimodal. It is probably continuous and noiseless, but we know nothing of its conditioning or geometry. There are no derivatives available and neigher can we presume smoothness (that would allow for derivative approximation). Since gains cannot be tuned independently of the others, the objective function is non-separable.

Practical limitations must be considered as well. Engine simulations, as in WAVE, take up to several minutes (per case – when parameters like rotations per minute etc. are fixed), but the time budget is limited. Usually aiming for an overnight computation, we are allowed only a few thousand evaluations of the objective function. That is, when parallelization is used, and therefore we require the optimization algorithm to be parallelizable. Last but not least, the algorithm must be robust – it must reliably produce decent results.

3. The optimization method

Metaheuristic and evolutionary methods have been extremely successful when tackling hard black-box optimization problems. In order to find the best-fitting method for our application, results of the extensive Black-Box Optimization Benchmarking (BBOB) were consulted [3, 4, 18, 21]. The Covariance Matrix Adaptation Evolution Strategy with bi-population restart scheme (BIPOP-CMA-ES) has proven its efficiency in the tests as well as in a growing number of practical applications. However, its use for tuning of PID controllers remains scarce [22, 23, 37].

3.1 CMA-ES: idea of the basic algorithm

The Covariance Matrix Adaptation Evolution Strategy (CMA-ES) [14, 15] is an evolutionary algorithm that uses stochastic and algebraic tools to define optimally diverse population of candidate solutions in an area that seems to be most promising. The size of the area and its location are determined based on the algorithm's previous experience with the objective function. New candidate solutions are sampled from a multivariate normal distribution:

$$x_k \sim m + \sigma \mathcal{N}(0, C), \quad k = 1, \ldots, \lambda, \tag{8}$$

whose mean m and covariance matrix C are adapted in each generation along with the general step size σ. The number of sampled candidate solutions λ is called the population size.

The mean in the new generation is defined as weighted average of several best-ranking individuals (the parent set) of the last generation. The weights are constant and depend only on the individuals' relative ranking. The covariance matrix is supposed to capture information about the objective function's features (curvature) around the mean. The matrix of the previous generation is weighted and further improved by exploiting the maximum likelihood principle (when estimating the distribution's parameters from the data) and by adding information of the overall progress across generations (the evolution path). The latter technique is also used for controlling the overall step length σ.

Authors of CMA-ES have designed the method so that minimal user interaction or manual setting of parameters is necessary. The starting point is needed in order to initiate the first mean $m^{(0)}$ but otherwise all the parameters are assigned usable default values. A user might want to change only the initial step length $\sigma^{(0)}$ and the population size λ. The parent set size is given as $\mu = \frac{1}{2}\lambda$. The population size is by default very small, given by the formula $\lambda_{\text{def}} = 4 + \lfloor 3 \log(d) \rfloor$, where d is the problem dimension [15]. For a given budget, smaller population size means faster adaptation. More often than not, this is a desirable property. Choice of $m^{(0)}$ and $\sigma^{(0)}$ is described further.

3.2 Extensions of the basic algorithm

The basic algorithm can be upgraded. In this application the elitist BIPOP-aCMA-ES version is used, i.e. Covariance Matrix Adaptation Evolution Strategy [15] with active covariance matrix updates [24], elitist scheme of parent selection [19] and bi-population restart strategy [20]. The choice of this particular variant, as well as setting the algorithm's parameters, is supported by numerical experiments described in Section 4.

3.2.1 Restart strategies

Restart strategies enable the basic algorithm to become more robust by outweighing premature convergence. They are based on changing the population size, balancing exploration of the search space and exploitation of the most promising areas.

The default population size λ_{def} is small and the optimal population size may be much larger. The IPOP (increasing population) strategy doubles the population size (the factor of 2 is empirical) every time a new restart is launched, while other parameters remain unchanged [2].

The more advanced BIPOP (bi-population) restart strategy [20] makes use of two interlacing regimes. The first one uses the IPOP restart strategy, while the second uses varying small populations. After each restart, it is decided which of the two regimes is to be applied next depending on whose count of conducted function evaluations is lower. The maximal population size is limited by the number of restarts under the first regime.

Two restart criteria need to be set carefully and adjusted for each model, because they depend on its numerical values. `TolFunHist` defines the tolerance for range of the best function values in recent history. `TolFun` defines the tolerance for function values of the current generation. For both criteria, it was observed that it is generally beneficial to try larger values first and diminish them if they invoke

restarts too often (occurrence of premature restarts is usually obvious). Alternatively, `TolFun` can be set to a small value, preferably a fraction of `TolFunHist`. That way, most restarts are called by `TolFunHist` and only one restart criterion needs tuning.

3.2.2 Elitist selection

There are multiple options of choosing parents of a new generation [19]. The basic algorithm selects all parents from the current generation (non-elitist selection). However, this scheme does not exploit very good solutions found early in the search. They are used as parents within the one generation that produced them and then are lost. Selecting the parent set among the individuals of the current generation and their parents as well (elitist selection) helps to preserve the exceptionally good individuals until they are superseded and thus amplify their influence. This approach speeds up the convergence and is advantageous in our problem, where good solutions are scarce. The downside is that it may, and sometimes it does, lead to premature convergence to a local optimum.

3.2.3 Active covariance matrix adaptation

In the basic algorithm, the successful individuals are used for the covariance matrix adaptation. Variance in directions that have proven to be beneficial is increased and thus they are preferred when sampling next generation. With the so-called active updates, information hidden in the unsuccessful individuals is exploited as well [24]. As opposed to passive decay over time, variance in detrimental directions is actively decreased. Besides telling the method where to go, we also tell it where *not* to go.

3.3 Important properties of CMA-ES and their use

CMA-ES does not use gradients and it does not even presume their existence. Moreover, it does not even use the actual values of the objective function once relative ranking has been assigned to the candidate solutions (except for some stopping/restart criteria). As a result, transformations of the objective function that have no effect upon the relative ranking of individuals do not effect the method's performance, making it more robust.

Further, the method exhibits invariance to invertible linear transformations of the search space. In particular, CMA-ES is invariant to scaling of variables (coordinate axes), which is the key property that makes it well-suited for tuning multiple controllers: parameters of one controller are usually of roughly the same scale, but with multiple controllers, the scaling may differ by many orders. Scaling of variables is used in the following way. The algorithm is given an initial approximation of the solution (hereafter called the reference point): $s = (s_1, \ldots, s_N)$. Then, when the algorithm wants to evaluate vector $v = (v_1, \ldots, v_N)$, it scales it by the reference point's elements' magnitudes and inputs vector w in the engine model: $w = (|s_1| \cdot v_1, |s_2| \cdot v_2, \ldots, |s_N| \cdot v_N)$. Because the scale coefficients are all positive and signs of the variables are essential, the algorithm must always have a zero vector as its starting point. Also, the initial step length is set to $\sigma^{(0)} = 1.0$.

Algorithm 1 Elitist BIPOP-aCMA-ES

---
set λ, μ
initialize $m, \sigma, C = I, p_\sigma = 0, p_c = 0$
initialize $restart\_regime = 1, count_1 = 0, count_2 = 0$
while termination criteria not met **do**
    while restart criteria not met **do**
        if not first generation in a restart **then**
            for $i = 1, \ldots, \mu$ **do**
                $x_{i+\mu} = x_i$ {parents of previous generation}
                $f_{i+\mu} = f_i$ {parents' objective function values}
            end for
        end if
        for $i = 1, \ldots, \lambda$ **do**
            $x_i \sim \mathcal{N}(m, \sigma^2 C)$ {sample new population from normal distribution}
            $f_i = evaluate(x_i)$ {evaluate x_i with objective function}
        end for
        sort $x_i, i = 1, \ldots, \lambda + \mu$ acc. to f_i {assign relative (descending) ranking}
        $m^* = m$
        $m = update\_m(x_i, \ldots, x_\mu)$ {move the mean utilizing the parents}
        {the evolution paths contain information about past progress}
        $p_\sigma = update\_p_\sigma(p_\sigma, \sigma^{-1} C^{-1/2}(m - m^*))$ {isotropic evolution path update}
        $p_c = update\_p_c(p_c, \sigma^{-1}(m - m^*), \|p_\sigma\|)$ {anisotropic evolution path update}
        $C = update\_C(C, p_c, (x_1 - m^*)/\sigma, \ldots, (x_{\lambda+\mu} - m^*)/\sigma)$ {covariance matrix update}
        $\sigma = update\_\sigma(\sigma, \|p_\sigma\|)$ {step size update}
        if $restart\_regime = 1$ **then**
            $count_1 = count_1 + \lambda$
        else
            $count_2 = count_2 + \lambda$
        end if
    end while
    if $count_1 < count_2$ **then**
        restart_regime = 1
    else
        restart_regime = 2
    end if
    reinitialize parameters and variables acc. to selected restart regime
end while

---

4. Experiment-based calibration of the optimization method

For the experiments, the 1D engine simulation software package WAVE by Ricardo Software was used [33], as the primary incentive was to develop a working automated tuner of PID controllers within WAVE. CMA-ES Python code provided by the method's author [11] was used.

Since models of engines take several minutes to run, while the CMA-ES run-time is negligible, time will be measured in number of evaluations of the objective function (= number of simulation runs).

4.1 The basic testing model

For the algorithms' calibration and basic testing, a quick-to-run model with three strongly coupled controllers was used. It is known that these controllers can be tuned well using only P and I gains (setting $D = 0$) as well as using all available P, I and D gains.

The model is depicted in Fig. 2. It represents a single cylinder (orange circle) engine. The blue "clouds" contain information about the surroundings (e.g. ambient pressure and temperature or initial fluid composition). The thick black lines depict the ducts. The green element is an orifice – an opening of variable diameter. The yellow PID elements are the controllers to be tuned. From left to right, the first PID controller controls the orifice diameter, the second one controls the fuel-air ratio by manipulating the fuel injector, and the third controls the compression ratio (the ratio of the maximum to minimum volume in the cylinder). The arrow-like elements are actuators that perform the actual mechanical control based on the control signal outputted by the corresponding controllers.

The controlled quantities measured by sensors (depicted as gray circles) are: indicated mean effective pressure (IMEP; the average pressure acting upon the piston during its cycle; controlled by adjusting the fuel-air ratio), exhaust gas temperature (controlled by the compression ratio, i.e. the ratio of largest and smallest possible capacity of the combustion chamber), volumetric efficiency (the
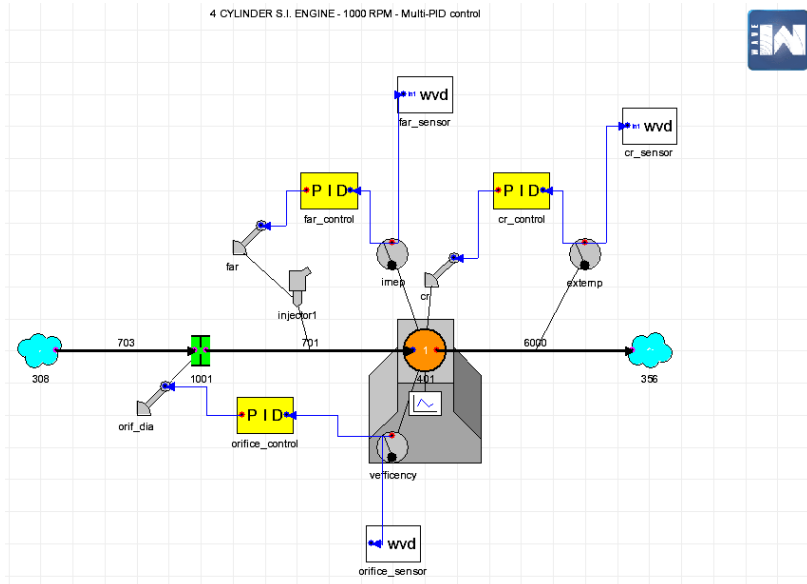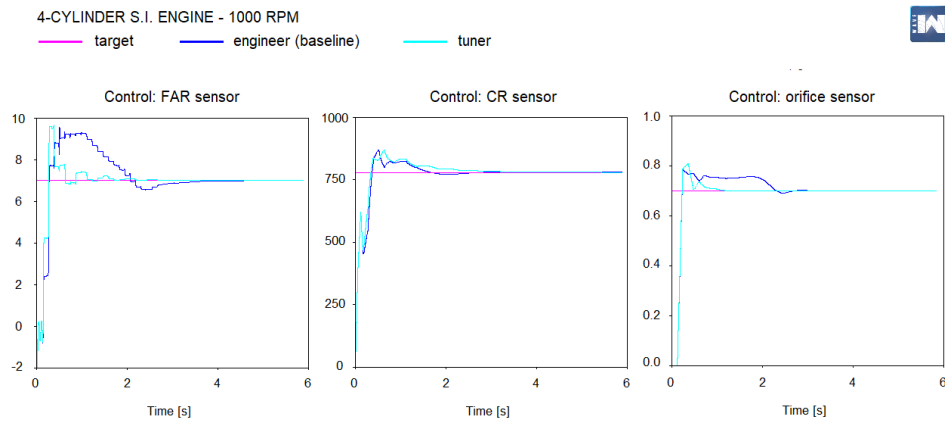

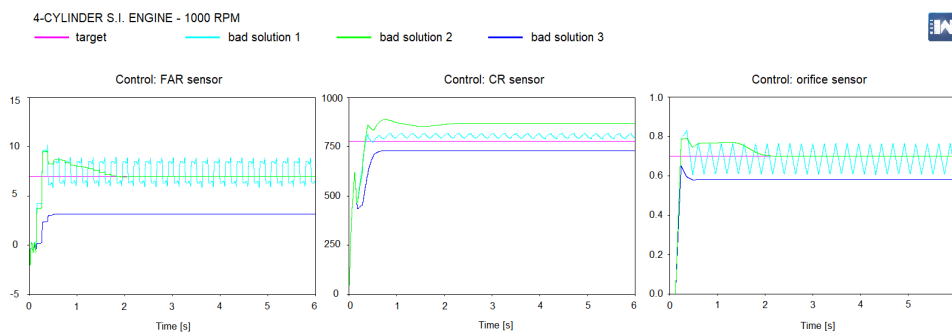**Fig. 2** *The basic testing model in WAVE.*

ratio of the volume of fluid actually displaced by a piston; controlled by opening of the orifice). Clearly, they influence each other heavily, so the controllers are coupled and cannot be tuned independently of each other.

When start of the engine is simulated, the controllers face a simultaneous step change of all three target values. Appropriate target values of the controlled quantities are provided. In Fig. 3, each plot contains two step responses. Controllers tuned by an engineer (hereafter called the baseline solution) yield the dark blue curves. This solution is sufficient but, apparently, a solution found by our method (light blue) is even better. More importantly, it can be reached with little user interaction, saving a lot of expert manual work.

Fig. 3 *The basic testing WAVE model: comparison of good solutions. Pink: the target value, dark blue: tuned by engineer, light blue: tuned by the algorithm.*

However, most candidate solutions are unacceptable: oscillatory, convergent to a different value in one or more criteria, or even divergent (see Fig. 4). Nevertheless, it is important to observe the development of the quantities' values in order to determine the shift t_0 used in the objective function. Larger t_0 means greater risk

Fig. 4 *The basic testing WAVE model: unsatisfactory solutions still provide enough information to estimate a fitting shift.*

335

of leaving out important information but possibly faster search. As a rule of thumb, a time point is chosen just before the initially erratic curve starts to follow a trend. It seems fitting to set the total simulation time such that the shift makes up the first 20%.

Setting $t_0 > 0$ is specific to our problem, where it is guaranteed empirically that leaving out most of the transient part (i.e. where $0 < \tau < t_0$) does not lead to unacceptable solutions. This approach reduces the "noise" in the objective function caused by the information contained in the transient part, making the problem easier to solve by an optimization method. However, this approach certainly cannot be used in all applications of PID controllers, especially when large overshoot is an issue, e.g. due to circuit breakers.

4.2 Tuning CMA-ES

CMA-ES was tuned using the basic testing model with PI controllers only (i.e. all derivative gains were set to zero as is common in this application). The experimental evidence summarized in Tab. II shows that the average run times of the tuned method are very similar for PI and PID modes. The D parameters were, however, included in testing – both with the basic test model (Tab. II) and other, real-world models (Tab. III).

Because the method uses randomness and each run is different, each test was run ten times and minimum, maximum and average run times were computed. The same reference point (i.e. scaling) of rather 'poor quality" is used for all runs of all tests. Compared to the vector of a good solution, its elements are off by two orders of magnitude on average.

There are two simple termination criteria. Either the target objective function value is hit or the budget of 12000 function evaluations is depleted. A different solution might be found each time, but setting the target value to 0.5 ensures high quality of any of them (all the corresponding step responses are almost identical). In practice, however, the algorithm should be stopped manually by the user because the desired objective function value varies greatly from model to model and the target value is difficult to estimate. We recommend the current-best solution to be plotted as its corresponding set of step responses. Then the human intervention is simple enough to be done even by an inexperienced user.

The results are summarized in Tab. I. The best setting found is always compared to the setting, where one or two specified attributes were changed.

When choosing a variant of CMA-ES, the results show that the best combination is the elitist parent selection scheme with active updates of the covariance matrix and cumulative step-size adaptation (CSA) step size adaptation. CSA [15] uses information of the algorithm's overall progress across generations, while the alternative two-point step-size adaptation (TPA) [12]) implements a line search along the direction of the latest mean shift.

Then we experiment with the initial population size λ_0 (i.e. the population size before the first restart). For a given cost, smaller populations enable more generations than large populations, causing faster adaptation [15]. Larger populations perform more exhaustive search. Using smaller than default populations is discouraged by Hansen but bigger populations can sometimes be more efficient,

setting		min	max	average
CMA-ES variants	active, not elitist	644	10877	4145
	elitist, not active	327	5361	2184
	not active, not elitist	812	$> 12000^*$	> 5334
adaptation method	TPA	926	$> 12000^*$	> 4724
λ_0, μ_0	$\lambda_0 = 2\lambda_{\mathrm{def}}, \ \mu_0 = \lambda_{\mathrm{def}}$	1533	8812	3273
	$\lambda_0 = 2\lambda_{\mathrm{def}}, \ \mu_0 = 1/2\lambda_{\mathrm{def}}$	697	10973	4181
`TolFunHist`	0.8×1.5	987	8980	3147
	1.2×1.5	674	4990	2058
	2.0×1.5	871	4960	2837
shift t_0	0.0 s	919	9567	3687
	0.6 s (10%)	354	11169	2963
BEST setting		268	2267	1098

Tab. I *Experiments' results comparing various settings of CMA-ES and the objective function, where always only the specified attribute or two were changed from the BEST setting (shift $t_0 = 1.2s$ (20%), CMA-ES variants = active and elitist, adaptation method = CSA, population size $\lambda_0 = \lambda_{def}$, parent set size $\mu_0 = 1/2\lambda_{def}$, restart parameter `TolFunHist` = 1.5). Entries marked *: number of evaluations once exceeded the maximum number of iterations allowed. When computing the corresponding average value, the value of 12000 was used in such cases, resulting in lower-bound for the average run time.*

so double initial population size is tested: $\lambda_0 = 2\lambda_{\mathrm{def}}$. The parent set size μ_0, by default equal to half of the population size, is then either doubled as well (i.e. $\mu_0 = \frac{1}{2}\lambda_0$) or it stays the same (i.e. $\mu_0 = \frac{1}{2}\lambda_{\mathrm{def}}$).

The best working value of the influential restart parameter `TolFunHist` (tolerance in function value history) for the basic testing model was found to be $n/2$, where $n = 3$ is the number of controlled quantities. It was further observed that setting $n/2$ seems to be almost universally usable in our context, even though the optimal value differs for each model. The tolerance in function value was set to `TolFun` $= 0.1$ for the basic testing model and later roughly adjusted for other models, scaled in proportion to the "usual" numerical values of the objective function and expected threshold.

In the objective function, shift $t_0 = 1.2$ second proved to be most effective.

For detailed parameter description see the CMA-ES tutorial [13].

4.3 Testing the method's robustness

Now the baseline solution is taken and other reference points are derived from it by multiplying each of its elements by factors $10^{-3}, 10^{-2}, 10^{-1}, 10^1$, and 10^2 (scaling by 10^3 proved to be too challenging for all tested methods). They simulate various quality of user's estimate and are used to test the consequent change in the tuned method's performance.

In these tests, both PI and PID controllers are considered. Since there are no PID baseline solutions with nonzero D parameters available, they are estimated based on the corresponding magnitudes of P and I parameters.

Results are summarized in Tab. II together with other methods' robustness tests. The run times are very good: of 120 runs, 8 reached 2000 to 3000 evaluations and only two exceeded 3000 evaluations. Surprisingly, there is no greater difference between the 6-dimensional PI setting and 9-dimensional PID setting, dimensionality being balanced by the system's greater flexibility. Closer look at algorithms' behavior suggests that too large search area (i.e. scaling) leads to "needle in haystack" type of situation. The run times are also influenced by the fact that there exist good solutions at the "10^{-1} level".

To conclude, when the reference point (scaling) is within a reasonable range of two orders of magnitude in each coordinate, the average run times are very usable. A user should be able to provide such a reference point – an estimate of the solution. If in doubt, as a rule of thumb, magnitudes of the reference point's coordinates should be chosen rather smaller than larger.

5. Comparison with PSO and SHADE

Performance of the above-described variant of CMA-ES on the problem of tuning coupled controllers was further compared with performance of two other prominent evolutionary algorithms: the particle swarm optimization (PSO) and the differential evolution (DE) in its success-history based adaptive variant SHADE.

5.1 PSO

PSO [25] is inspired by bird flocks or fish schools, where every individual moves by itself, yet the whole self-organized system acts as a single organism. Each particle within a swarm moves in the search space as it is assigned a different "velocity" vector in each generation. This vector is defined as a combination of the previous "velocity", the individual's best known position so far and the swarm's (or sub-swarm's) best known position. This way, the whole swarm moves towards historically best areas.

For PSO implementation, the default setting of PSO in the DEAP (Distributed Evolutionary Algorithms in Python) optimization framework [6] is used, adding only the essential scaling of variables as described above.

5.2 SHADE

The basic DE algorithm [32] maintains a population of candidate solution, which are further combined and tested for better objective function values. Each candidate solution X is tested against a new point Z, which was obtained as a binary crossover of X and $Y = A + f(B - C)$, where A, B and C are three distinct candidate solutions in the population and $f \in [0, 2]$ is a parameter. The crossover probability of each vector element is given by parameter $cr \in [0, 1]$. Both parameters and the population size greatly influence efficiency of the search.

Considering the context of our problem and the performance-comparing tests in [36], the SHADE variant of DE was chosen to compete with CMA-ES. This upgrade [34] introduces adaptation of parameters cr and f based on success history that is stored in an external archive.

Implementation of the SHADE algorithm by Tanabe was used [35] and tuned in accordance with its author's recommendations [36] and additional experiments, setting, most notably, initial cr = 0.5, initial $f = 0.5$ and population size = $2n$, where n is the problem dimension. One restart criterion was added to the standard SHADE algorithm, based on recommendations in [36]: if the number of objective function evaluations exceeds 1000, restart can be launched if the best-so-far solution is not updated for $50 \times n$ evaluations. The particular numbers were set based on experiments and the first condition proved very important.

Unlike in CMA-ES and PSO, the reference point is not used for scaling but to define boundaries of the area, from which the initial population is sampled. Each coordinate of the reference vector is multiplied by -10 and 10 to define the boundaries. Candidate solutions in further generations are not restricted.

5.3   PSO and SHADE experiment results

The experiments with PSO and SHADE were performed with same set of 13 reference points as in the previous tests of CMA-ES, with 5 runs of each test. The stopping criteria were either hitting the target value of the objective function or depleting the budget of 10000 objective function evaluations. See Tab. II for results.

PSO was able to outperform CMA-ES but was not very reliable. Its good performance was limited to very good reference points and the algorithm very often (36 out of total 65 runs) did not converge within the given budget.

The performance of SHADE was more consistent but very slow compared to CMA-ES. Same as PSO, SHADE never converged within the given budget for the reference points of baselines multiplied by 10^1 and 10^2 or the one used for CMA-ES calibration.

6.   Verification of the method on models of real engines

It was shown that, on the testing model, the described version of CMA-ES performs very well and clearly outperforms PSO and DE. Its usability is now verified on models of real engines provided by Ricardo. The testing set consists of two models with 1 controller (labeled M1.1, M1.2), three models with 2 controllers (labeled M2.1, M2.2, M2.3) and one model with 3 controllers (labeled M3.1). A baseline setting of controllers is given for each model: a PI controller setting tuned manually by an engineer.

The single-controller models are tested with 5 reference points: the baseline solution and its element-wise multiples by factors $10^{-2}, 10^{-1}, 10^1$ and 10^2. The multi-controller models are tested with three reference points: the baseline solution and its multiples of 10^{-1} and 10^1. The primary reason for this restriction is the enormous time consumption. Each test is run five times. A run is terminated upon

reference p.	PSO runs					SHADE runs					CMA-ES runs		average run time		
	#1	#2	#3	#4	#5	#1	#2	#3	#4	#5	min	max	PSO	SHADE	CMA-ES
PI baseline	113	61	–	–	–	130	396	341	2100	155	12	168	–	624	76
10^1 PI b.	–	–	–	–	–	–	–	–	–	–	540	2064	–	–	1049
10^2 PI b.	–	–	–	–	–	–	–	–	–	–	821	5700	–	–	2061
10^{-1} PI b.	56	76	48	39	32	342	156	1560	260	538	61	816	50	571	317
10^{-2} PI b.	893	-	483	226	403	5155	1514	2783	1437	2123	222	1334	–	2602	592
10^{-3} PI b.	-	-	-	-	1600	4118	1090	2102	1243	1422	259	1617	–	1995	812
PID baseline	562	28	105	71	28	428	307	497	441	658	32	256	159	466	67
10^1 PID b.	–	–	–	–	–	–	–	–	–	–	62	1462	–	–	1102
10^2 PID b.	–	–	–	–	–	–	–	–	–	–	953	4130	–	–	2022
10^{-1} PID b.	33	164	104	280	43	239	418	209	437	390	141	782	125	339	343
10^{-2} PID b.	387	624	301	1186	–	6524	1018	1150	3297	1556	202	941	–	2709	580
10^{-3} PID b.	–	–	1362	1734	1680	1673	2125	1945	1477	2163	416	2324	–	1877	1138
calib. ref. p.	–	–	–	–	–	–	–	–	–	–	268	2267	–	–	1098

Tab. II *Results of algorithm testing on the basic model. 5 PSO runs, 5 SHADE runs and 10 CMA-ES runs were performed for each of 13 reference points. The value of "–" means that a satisfactory solution (i.e. solution with function value less than 0.5) was not found within the provided budget of 10000 function evaluations. Average run time was not computed if one or more runs did not finish within the given budget.*

reaching the (empirically set) target value specific to the given model or budget depletion. Solutions are not necessarily unique.

All results are summarized in Tab. III.

model	reference p.	min	max	aver.
M1.1	PI baseline	2	68	28
	10^1 PI b.	35	153	79
	10^2 PI b.	95	519	225
	10^{-1} PI b.	20	120	66
	10^{-2} PI b.	49	296	123
M1.2	PI baseline	1	22	9
	10^1 PI b.	4	28	11
	10^2 PI b.	80	225	187
	10^{-1} PI b.	34	100	51
	10^{-2} PI b.	57	181	94
M2.1	PI baseline	11	66	35
	10^1 PI b.	244	280	255
	10^{-1} PI b.	4	32	21
M2.2	PI baseline	8	98	29
	10^1 PI b.	60	770	364
	10^{-1} PI b.	44	107	64
M2.3	PI baseline	9	78	32
	10^1 PI b.	250	757	629
	10^{-1} PI b.	49	1188	347
M2.3	PID baseline	10	91	57
	10^1 PID b.	274	857	522
	10^{-1} PID b.	82	1576	749
M3.1	PID baseline	41	331	152
	10^1 PID b.	827	1763	1268
	10^{-1} PID b.	179	3867	2476

Tab. III *Real-world models*

6.1 Single-controller models

The single-controller models are included because of their importance in practical use and they represent the typical use of a PI controller in a combustion engine. Model M1.1 contains a turbocharger and M1.2 contains a twin turbocharger.

With dimension being only 2, the run times for both models are very short. Good reference points for single controllers can be obtained by commonly used rules of thumb (e.g. Ziegler-Nichols) or provided by previous (personal or programmed) experience with similar models.

6.2   Multi-controller models

The two-controller models M2.1 and M2.2 do not pose a greater challenge than the single-controller models. For M2.3, PI and PID control is compared. The extreme differences in minimal and maximal run time values of "10^{-1} PI baseline" and "10^{-1} PID baseline" tests are caused by non-optimal setting of the restart parameter tolerance in function value history (`tolhistfun`) described above. After adjustment, the "10^{-1} PI baseline" and "10^{-1} PID baseline" run times drop and the results of PI control resemble those of M2.2.

Unlike the previous cases, the three-controller model M3.1 requires a full PID control. The reference point was obtained by estimating the D gains and adding them to the given (not very good) PI engineer-tuned baseline "solution". It can be seen that this model is considerably harder to tune than the previous models. While the algorithm has no trouble finding the same near-optimal "solutions" similar to the one given by an engineer, it was hard to get to an actual global optimum, when all three controlled quantities converge.

7.   Conclusion

This paper has shown how the Covariance Matrix Adaptation Evolution Strategy can be applied to the problem of tuning the gains of multiple coupled PID controllers within combustion engine simulations. It was shown that its version with bi-population restart scheme, elitist parent selection and active covariance matrix updates is capable of finding good parameters of up to three PID controllers through minimization of a fitting objective function. The method has been calibrated on a testing model and verified on models of real-world engines, showing its practical usability and tolerable computation times even for poor-quality reference points. On the testing model, CMA-ES clearly outperformed PSO and SHADE methods.

Acknowledgement

References

[1] ÅSTRÖM K. J., WITTENMARK B. Adaptive Control: Second Edition. Courier Corporation, 2013. ISBN 9780486319148.

[2] AUGER A., HANSEN N. A restart CMA evolution strategy with increasing population size. In: B. McKay et al., editors, *The 2005 IEEE International Congress on Evolutionary Computation (CEC'05)*, 2005, 2, pp. 1769–1776, doi: 10.1109/CEC.2005.1554902.

[3] AUGER A., FINCK S., HANSEN N., ROS R. BBOB 2009: Comparison Tables of All Algorithms on All Noiseless Functions. Technical Report RT-0383, INRIA, April 2010.

[4] AUGER A., FINCK S., HANSEN N., ROS R. BBOB 2010: Comparison Tables of All Algorithms on All Noiseless Functions. Technical Report RT-388, INRIA, September 2010.

[5] BINGUL Z. A new pid tuning technique using differential evolution for unstable and integrating processes with time delay. In: *Neural Information Processing: 11th International Conference, ICONIP 2004, Calcutta, India, November 22-25*, pp. 254–260, Berlin, Heidelberg, 2004. Springer Berlin Heidelberg. ISBN 978-3-540-30499-9.

[6] FORTIN F.A., RAINVILLE F.M., GARDNER M.A., PARIZEAU M., GAGNÉ. C. DEAP: Evolutionary Algorithms Made Easy. *Journal of Machine Learning Research*, 13, pp. 2171–2175, Jul 2012.

[7] GAING Z.L. A particle swarm optimization approach for optimum design of PID controller in AVR system. *IEEE Transactions on Energy Conversion*, 19(2), pp. 384–391, June 2004. ISSN 0885-8969.

[8] GHOSHAL S.P. Optimizations of PID gains by particle swarm optimizations in fuzzy based automatic generation control. *Electric Power Systems Research*, 72(3), pp. 203–212, 2004, doi: 10.1016/j.epsr.2004.04.004.

[9] GOODWIN G.C., GRAEBE S.F., SALGADO M. E. *Control System Design*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 1st edition, 2000. ISBN 0139586539.

[10] GRAHAM D., LATHROP R.C. The Synthesis of Optimum Transient Response: Criteria and Standard Forms. *Transactions of the American Institute of Electrical Engineers, Part II: Applications and Industry*, 72(5), pp. 273–288, Nov 1953. ISSN 0097-2185.

[11] HANSEN N. cma 1.1.06: Python Package. https://pypi.python.org/pypi/cma.

[12] HANSEN N. CMA-ES with Two-Point Step-Size Adaptation. Research Report RR-6527, INRIA, 2008.

[13] HANSEN N. The CMA Evolution Strategy: A Tutorial. http://arxiv.org/abs/1604.00772, 2016.

[14] HANSEN N., KERN S. Evaluating the CMA evolution strategy on multimodal test functions. In: *International Conference on Parallel Problem Solving from Nature*, pp. 282–291. Springer, 2004.

[15] HANSEN N., OSTERMEIER A. Completely derandomized self-adaptation in evolution strategies. *Evolutionary Computation*, 9(2), pp. 159–195, 2001, doi: 10.1162/106365601750190398.

[16] HANSEN N., NIEDERBERGER A.S.P., GUZZELLA L., KOUMOUTSAKOS P. Evolutionary optimization of feedback controllers for thermoacoustic instabilities. In: *IUTAM Symposium on Flow Control and MEMS*, pp. 311–317. Springer, 2008, doi: 10.1007/978-1-4020-6858-4_36.

[17] HANSEN N., NIEDERBERGER A.S.P., GUZZELLA L., KOUMOUTSAKOS P. A method for handling uncertainty in evolutionary optimization with an application to feedback control of combustion. *IEEE Transactions on Evolutionary Computation*, 13(1), pp. 180–197, 2009.

[18] HANSEN N., AUGER A., ROS R., FINCK S., POSIK P. Comparing Results of 31 Algorithms from the Black-Box Optimization Benchmarking BBOB-2009. *Workshop Proceedings of the GECCO Genetic and Evolutionary Computation Conference 2010*, pp. 1689–1696, 2010.

[19] HANSEN N., ARNOLD D.V., AUGER A. Evolution Strategies. In: J. Kacprzyk and W. Pedrycz, editors, *Springer Handbook of Computational Intelligence*, chapter 44, pp. 871–898. Springer Berlin Heidelberg, 2015. ISBN 9783662435052.

[20] HANSEN N. Benchmarking a BI-population CMA-ES on the BBOB-2009 Function Testbed. In: *Proceedings of the 11th Annual Conference Companion on Genetic and Evolutionary Computation Conference: Late Breaking Papers*, GECCO '09, pp. 2389–2396, 2009. ISBN 978-1-60558-505-5.

[21] HANSEN N., FINCK S., ROS R. COCO – COmparing Continuous Optimizers: The Documentation. Research Report RT-0409, INRIA, May 2011.

[22] IRUTHAYARAJAN M.W., BASKAR S. Evolutionary Algorithms Based Design of Multivariable PID Controller. *Expert Syst. Appl.*, 36(5), pp. 9159–9167, July 2009. ISSN 0957-4174.

[23] IRUTHAYARAJAN M.W., BASKAR S. Covariance Matrix Adaptation Evolution Strategy Based Design of Centralized PID Controller. *Expert Syst. Appl.*, 37(8), pp. 5775–5781, August 2010. ISSN 0957-4174.

[24] JASTREBSKI G.A., ARNOLD D.V. Improving evolution strategies through active covariance matrix adaptation. In: *IEEE Congress on Evolutionary Computation – CEC 2006*, pp. 2814–2821, 2006.

[25] KENNEDY J., EBERHART R.C., SHI Y. *Swarm Intelligence*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2001. ISBN 1-55860-595-9.

[26] KORANI W.M., DORRAH H.T., EMARA H.M. Bacterial foraging oriented by Particle Swarm Optimization strategy for PID tuning. In: *Computational Intelligence in Robotics and Automation (CIRA), 2009 IEEE International Symposium on*, pp. 445–450, Dec 2009.

[27] KWOK D.P., SHENG F. Genetic algorithm and simulated annealing for optimal robot arm PID control. In: *Evolutionary Computation, 1994. IEEE World Congress on Computational Intelligence., Proceedings of the First IEEE Conference on*, pp. 707–713. IEEE, 1994.

[28] MARLER R.T., ARORA J.S. Survey of multi-objective optimization methods for engineering. *Structural and multidisciplinary optimization*, 26(6), pp. 369–395, 2004.

[29] MONTAZERI-GH M., JAFARI S., ILKHANI M.R. Application of particle swarm optimization in gas turbine engine fuel controller gain tuning. *Engineering Optimization*, vol. 44, pp. 225–240, 2012.

[30] OGATA K. *Modern Control Engineering*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 2nd edition, 1990. ISBN 0135891280.

[31] PETROV M., GANCHEV I., TANEVA A. Fuzzy PID control of nonlinear plants. *Intelligent Systems, 2002. Proceedings. 2002 First International IEEE Symposium*, 1, pp. 30–35. IEEE, 2002.

[32] PRICE K., STORN R.M., LAMPINEN J.A. *Differential Evolution: A Practical Approach to Global Optimization*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2005. ISBN 3540209506.

[33] Ricardo Software. WAVE Manual, 2017.

[34] TANABE R., FUKUNAGA A. Success-History Based Parameter Adaptation for Differential Evolution. *IEEE CEC*, 2013, pp. 71–78.

[35] TANABE R., FUKUNAGA A. Evaluating the performance of SHADE on CEC 2013 benchmark problems. *IEEE CEC*, 2013, pp. 1952–1959.

[36] TANABE R., FUKUNAGA A. Tuning Differential Evolution for Cheap, Medium, and Expensive Computational Budgets. *IEEE CEC*, 2015, pp. 2018–2025.

[37] WAKASA Y., KANAGAWA S., TANAKA K., NISHIMURA Y. Pid controller tuning based on the covariance matrix adaptation evolution strategy. *IEEJ Transactions on Electronics, Information and Systems*, 130(5), pp. 737–742, 2010.

[38] ZIEGLER J.G., NICHOLS N.B. Optimum Setting for Automatic Controller. *ASME transaction*, 1942, pp. 759–768.