A Noise-Resistant Mixed-Discrete Particle Swarm Optimization Algorithm for the Automatic Design of Robotic Controllers

Cyrill Baumann and Alcherio Martinoli

Abstract— **The automatic design of well-performing robotic controllers is still an unsolved problem due to the inherently large parameter space and noisy, often hard-to-define performance metrics, especially when sequential tasks need to be accomplished. Distal control architectures, which combine precoded basic behaviors into a (probabilistic) finite state machine offer a promising solution to this problem.**

In this paper, we enhance a Mixed-Discrete Particle Swarm Optimization (MDPSO) algorithm with an Optimal Computing Budget Allocation (OCBA) scheme to automatically synthesize distal control architectures. We benchmark MDPSO-OCBA's performance against the original MDPSO as well as the Iterated F-Race (IRACE) and the Mesh Adaptive Direct Search (MADS) algorithms on both a benchmark function with different noise levels and design problems of distal control architectures. More specifically, we evaluate the algorithms using high-fidelity simulations in three increasingly challenging scenarios involving parallel and sequential tasks. Additionally, the best performing controller generated in simulation by each optimization algorithm is compared with a manually designed solution and validated with physical experiments.

The analysis on the benchmark function with different noise levels demonstrates MDPSO-OCBA's high robustness to noise. The comparison on the robotic control design problems shows that, without any meta-parameter tuning, MDPSO-OCBA is able to generate the best performing control architectures overall, closely followed by IRACE. They significantly outperform MADS for the more complex and noisier scenarios, resulting in competitive controllers in comparison to the manually designed one.

I. INTRODUCTION

Effectively designing and optimizing control algorithms for robotic systems plays a crucial role in enabling them to accomplish increasingly complex tasks. While many robotic systems are still designed manually, automatic control design is gaining popularity. Tasks solved with automatically designed controller range from obstacle avoidance [1] to more complicated assignments, such as an excavation task in [2].

In order to achieve controllers capable of solving these tasks, different control architectures have been proposed. A currently highly popular option is represented by lightweight and proximal architectures (such as Artificial Neural Networks, ANNs) in combination with meta-heuristic optimizers (e.g., PSO, c.f. [1] for example) or Reinforcement Learning (RL, cf. [3] for example). Such architectures present numerous advantages, such as, RL only needing either a positive or negative feedback or, for other metaheuristic methods, the definition of fitness functions of arbitrary complexity. However, they also imply several known (and still unsolved) challenges, including, for example, the interpretability (for humans), traceability and verifyability.

Another, promising alternative are distal control architectures in combination with a possibly robust optimization method. A distal controller includes a set of basic behaviors (such as 'turn clockwise' or more high-level such as 'phototaxis'), which an arbitrator, potentially a probabilistic one, combines to solve a given problem. In [4], [5], [6], [7], Iterated F-Race (IRACE) is used to create probabilistic Finite State Machines (FSMs) combining basic behaviors. In [8], a framework leveraging grammatical evolution is proposed to carry out the same operation. The main advantages of such distal control architectures is the ability to combine automatic control generation with interpretability and verifiability of the resulting controller due to the natural ease of humans in understanding FSMs. Furthermore, reuse of basic behaviors, while also possible in proximal architectures, is highly facilitated through the functional abstractions introduced in the distal control architecture.

Regardless of the chosen control architecture, an important challenge in automatic design of robotic controllers is the definition of the performance metric of the targeted scenario. This is often done ad hoc, though there are several dedicated works focusing on metric definitions for a specific problem, such as [9], [10] or [11]. However, this work uses sequential tasks (i.e. subtask A has to be solved before starting subtask B), for which the design of performance metrics, to the best of our knowledge, has not been well-studied.

The main contributions of this work are thus three-fold. First, we enhance the Mixed-Discrete Particle Swarm Optimization (MDPSO) algorithm, firstly introduced in [12], with an Optimal Computing Budget Allocation (OCBA) scheme [13] in order to handle noisy performance evaluations of candidate solutions inherent to challenges in robotic controller design. The enhanced MDPSO-OCBA is compared to the original MDPSO first, before being benchmarked against two competitive mixed-integer optimization algorithms: IRACE [14] which has been previously used to generate FSMs and Mesh Adaptive Direct Search (MADS) for constrained optimization [15], a state-of-the-art extension of the generalized pattern search algorithm class.

Second, while MDPSO has been thoroughly tested and benchmarked on noise-free benchmark functions [12], we apply it here to a noisy benchmark function as well as to three concrete, highly stochastic scenarios of automatic

The authors are with the Distributed Intelligent Systems and Algorithms Laboratory, School of Architecture, Civil and Environmental Engineering, École Polytechnique Fédérale de Lausanne (EPFL), Switzerland.
firstname.lastname@epfl.ch.
This work is financially supported by Mitsubishi Electric Corporation, Japan.

control generation for wheeled robots. This represents a challenging benchmark suite that, to our knowledge, has not been considered before for this algorithm.

Third, we propose to encode the performance metric of sequential tasks using discrete steps in the fitness landscape. Depending on the achievement of sub-tasks, the robot's performance metric is increased permanently (e.g., doubles). Since sequential tasks occur in many complex real life scenarios, by defining the performance metric using such discrete steps, the efficiency and transparency of its design can be simplified significantly for numerous problems. This comes at the price of introducing additional difficulties in the optimization problem. By using three scenarios with gradual introduction of sequential tasks eventually constituting a simplified version of a search and rescue mission, we therefore demonstrate that, despite discrete steps in the performance metrics, the overall optimization problem remains viable.

In order to prove the validity of the FSMs generated in simulation, the best performing controller produced by each optimization algorithm is then compared with a manually designed solution and validated with physical experiments.

The paper is organized as follows: first, the problem tackled, the approach, and the experimental set-up are described. We then present and discuss the experimental results and end with some conclusive remarks.

II. APPROACH

In order to automatically generate controllers, the approach used in this work is based on the representation of a behavioral arbitrator as a FSM, as introduced in [4]. For each state of the FSM, a Basic Behavior (BB) needs to be chosen out of a set $\mathscr{B}$ of predefined BBs. Likewise, for every transition in the FSM, the target state needs to be defined, and one or multiple conditions for the transition need to be selected out of a set $\mathscr{C}$ of predefined conditions. To allow for some flexibility, both conditions and BB have attributes which need to be specified. Each FSM candidate is then evaluated using a high-fidelity robotics simulator, according to a predefined, scenario-dependent cost function. Additionally, a given FSM can be expressed as a vector of mixed-integer numbers using an FSM-encoding grammar. The associated optimization problem is therefore concerned with finding the vector which minimizes the cost function obtained through the high-fidelity simulator.

This optimization problem can be categorized more precisely as a black-box Mixed-Integer Nonlinear Programming (MINLP), characterized by both continuous and discrete (or categorical) variables, and nonlinear objective functions. While for this work variable bounds are the only constraints needed, for more complex tasks one could also imagine to add the most critical objectives, such as collision avoidance.

As a result, two main components are needed for the approach presented: a FSM encoding grammar and an optimization algorithm.

A. FSM encoding grammar

A generic FSM $\mathscr{F}$, with N_s states and $N_D = N_S(N_S-1)$ possible state transitions, can be encoded as a vector of both discrete and continuous numbers as follows:

$$\mathscr{F} = [[B]\ [D]\ [C]\ [A_B]\ [A_C]] \tag{1}$$

where:

- B consists of $B_1...B_{N_S} \in \mathscr{B}$, corresponding to the selected behaviors for the N_S behavior slots, where $\mathscr{B}$ is the set of available behaviors.
- D consists of $D_1...D_{N_D} \in \{1,..,N_S\}$, corresponding to the destinations or target behavioral slots of every transition.
- C consists of $C_1...C_{N_C \cdot N_D} \in \mathscr{C}$ corresponding to the conditions for each transition to happen, where $\mathscr{C}$ is the set of available transition conditions and N_C to the number of conditions allowed per transition.
- A_B consists of $A_1...A_{N_{BP} \cdot N_S} \in [0,1]$, corresponding to the parameters of the behaviors, where N_{BP} is the number of parameters allowed per behavior.
- A_C consists of $A_1...A_{N_{CP} \cdot N_C \cdot N_D} \in [0,1]$, corresponding to the parameters of the conditions, where N_{CP} is the number of parameters allowed per condition.

As the length of vector $\mathscr{F}$ corresponds to the length of the cardinality of the sets above, the total number of variables for a FSM can thus be expressed as $N_V = N_S(N_S + N_{BP} + N_C(N_S-1)(1+N_{CP}))$. The numerical values used in this work are $N_C = 2$, $N_{BP} = 2$, $N_{CP} = 1$ and $N_S \in \{1,2\}$ depending on the scenario.

The behaviors and conditions used in this work are summarized in Tables I and II. Figure 1 gives an example of a FSM and its corresponding 16 dimensional vector. The translation from a FSM to a corresponding vector or vice-versa can be achieved using Equation 1 as well as Tables I and II. It is worth noting that the sensor threshold parameter of the conditions allows for highly flexible use of conditions. For example, the condition "above grey floor at T" can even be used in a limited way to detect black floor, given the right threshold. Furthermore, as shown in Figure 1, transitions onto itself, as well as multiple same states or conditions with the same or different arguments, are possible.

Fig. 1: Example of a generated FSM for the *Exploration with Stopping* scenario ($N_S = 2$). The initial state is displayed by a double circle, other states by a single circle. State transitions are indicated by arrows. The FSM displayed here is encoded through the mixed-integer vector given by: 1 3 1 1 4 5 4 4 0.91 0.21 0.21 0.12 0.96 0.39 0.24 0.81

Hence, the optimization problem can be defined as

$$\min_{\mathscr{F}} \sigma(\mathscr{S}(\mathscr{F})) \tag{2}$$

where $\mathscr{S}$ corresponds to the resulting robot behavior in the simulator for a given $\mathscr{F}$, and σ is the cost function, depending on the targeted scenario. The σ's for the scenarios considered here are given in Table III. Note that due to the stochastic nature of robotics, a precise mathematical formulation of $\mathscr{S}$ is usually impossible, explaining the use of a high fidelity simulator.

TABLE I: Set $\mathscr{B}$ of predefined basic behaviors A more detailed definition of each behavior can be found in Appendix A.

ID	Behavior	Parameter 1	Parameter 2
1	Obstacle avoidance	agressivity	distance threshold
2	Light following	agressivity	unused
3	Stop	unused	unused

TABLE II: Set $\mathscr{C}$ of predefined transition condition

ID	returns true at time T if	Parameter
0	always true	unused
1	above grey floor at T	sensor treshold
2	above white floor at T	sensor treshold
3	obstacles in range at T	sensor treshold
4	no obstacles in range at T	sensor treshold
5	above black floor at T	sensor treshold
6	above black floor at any $t \in [0...T]$	sensor treshold

B. Optimization algorithms

In order to optimize a MINLP problem, numerous state-of-the-art algorithms are available. However, for our application, we are particularly interested in algorithms which are able to handle stochastic MINLP. In this work, we introduce MDPSO enhanced with OCBA. We consider both variants introduced in [12], with and without explicit diversity preservation. MDPSO's performance is compared to MADS [15] through the implementation NOMAD [16], [17] and IRACE [14] with its available implementation in R. Both MADS and IRACE have been employed on noisy MINLP before, making them ideal candidates for benchmarking MDPSO-OCBA.

1) MADS: MADS is a state-of-the-art direct search algorithm with rigorous convergence analysis. It extends the Generalized Pattern Search algorithms with local exploration in an asymptotically small mesh spanning the optimization variable space. The algorithm has been designed to solve difficult blackbox optimization problems involving nonsmooth, nonlinear cost functions, as well as mixed-discrete variables. This corresponds well to our problem's characteristics. Moreover, the NOMAD implementation used in this work includes many recent improvements on the original algorithm including noise resistance [18].

2) IRACE: IRACE is a state of the art racing algorithm which has already shown its capabilities in similar noisy MINLP optimization problems, such as in [4]. IRACE is an iterated version of the F-Race algorithm, which is based on evaluating a finite set of candidate solutions through systematic allocation of computational resources. Initially similar to a brute-force approach, the algorithm then drops underperforming candidates in order to increase the computational budget available to decide the best performing candidates.

3) MDPSO: MDPSO is a mixed-discrete variant of PSO, where the MD part can be roughly described as an additional rounding introduced to obtain discrete variables. PSO is based on having a finite set of particles deployed in the optimization variable space. Particles are represented by position (the actual candidate solution) and velocity vectors. Each particle's velocity in this space is then influenced by the personal and neighborhood best candidate solutions found so far, as well as an inertia term.

Given the previous superior performances of noise-resistant PSO variants in robotic control shaping under highly stochastic conditions [1], highly competitive when compared to Reinforcement Learning [19] and evolutionary approaches [20], noise-resistant MDPSO seems a promising option for solving mixed-integer problems in the very same context.

We therefore have enhanced the original, MDPSO algorithm introduced in [12] with an OCBA scheme, in order to better handle the noise inherent to robotic systems as shown in Algorithm 1. Akin to *OCBA_C* of [21], for every PSO iteration, after initial n_0 evaluations of each particle's solution, OCBA determines, based on means and standard deviations, how much of the total available iteration budget B_i each candidate solution should be allocated in order to enable an optimal performance comparison between them. Note that the difference between *OCBA_C* of [21] and this algorithm is in the way particle positions are updated, as shown in Algorithm 2.

In [12], an explicit diversity preservation mechanism was introduced which allows to avoid premature convergence in some cases by adding a motion vector away from the neighborhood's best position with varying weight, depending on the convergence of the particles. However, as this mechanism does not always result in better optimization performance [12], we also consider a version of MDPSO without explicit diversity preservation in this work. This "MDPSO without diversity preservation" corresponds to a classical, OCBA-enhanced noise-resistant PSO as in [21], with the additional rounding to obtain discrete variables as in [12].

The used PSO neighborhood in this work has been set to global in an effort to limit the number of meta-parameters. The evaluation budget available per iteration has been set experimentally to $B_i = 3N_p$, where N_p is the number of particles used by the PSO. Further OCBA parameters used are the number of initial particle evaluations $n_0 = 2$ and $\Delta = 2$ the number of re-evaluations allocated together.

For the sake of conciseness, further in-depth descriptions of the different optimization algorithms are omitted here. We refer the reader to the respective works [12] for MDPSO, [13]

Algorithm 1: MDPSO OCBA

Initialize particles
$n \leftarrow 0$
while $n < N_E$ **do**
  Initialize iteration budget: $B_i \leftarrow 3N_p$
  $n \leftarrow n + B_i$
  for N_p *particles* **do**
    Evaluate new particle position n_0 times
    $B_i \leftarrow B_i - n_0$
  end
  while $B_i > 0$ **do**
    Allocate Δ samples among current positions and personal bests using OCBA
    Evaluate allocated samples
    Recalculate mean and variance for new evaluations
    $B_i \leftarrow B_i - \Delta$
  end
  for N_p *particles* **do**
    Update personal best
    Update neighborhood best
    Update particle position (Alg. 2)
  end
end

for OCBA, [15] for MADS and [14] for IRACE.

C. Scenarios

Initially, the optimization algorithms are challenged on a mixed-discrete benchmark function composed of both a 2 dimensional Rastrigin and a 2 dimensional Rosenbrock function as shown in Equation 3. It is worth noting at this point that there have been recent efforts in creating mixed integer benchmark suites such as [22]. The use of such suites for in-depth comparisons of MDPSO-OCBA against state of the art algorithms on benchmark functions is undisputed. However, as this work focuses on the application and performance of MDPSO-OCBA for automatic design of behavioral arbitrators in robotics, such an in-depth comparison would exceed the scope of this paper. Therefore, the benchmark function in Equation 3 has been designed to match both discrete and continuous dimensions of the first control design scenario, while enabling comparison across different noise levels. f_{rastr} corresponds to a standard 2D Rastrigin function, $f_{rosenbr}$ to a standard 2D Rosenbrock function. To enforce a unique global best (of value -5 at $\{0,0,0\}$) the Rastrigin values are offset by -5. Additive, zero-mean gaussian noise with a standard deviation of n_{std} is added in both cases.

$$f(d,c_1,c_2,n_{std}) = \mathcal{N}(0,n_{std}) + \begin{cases} f_{rastr}(c_1,c_2) - 5 & \text{if } d = 0 \\ f_{rosenbr}(c_1,c_2) & \text{if } d = 1 \end{cases}$$
$$\text{s.t. } d \in \{1,2\} \text{ and } c_1,c_2 \in [-5.12, 5.12] \quad (3)$$

In a second step, all the selected optimization algorithms are challenged with the generation of robotic controllers for the three scenarios summarized in Table III. The goal of the *Exploration* scenario is for the robot to explore the arena (see Figure 2), while avoiding obstacles. This is a basic, well-investigated but also highly relevant problem for real-world applications. The respective, fully continuous, cost function $\sigma_E(\mathscr{S}(\mathscr{F}))$ has been adapted from [23]. $\sigma_E(\mathscr{S}(\mathscr{F}))$ results in lower cost for controllers which maximize the traveled distance as well as the distance from the closest obstacle.

In the *Exploration with stopping* scenario, the robot has to explore the arena until it encounters black floor where it is supposed to stop. This scenario's cost function σ_{ES} differs from σ_E in the additional term γ_S introducing a discrete step by pulling the cost to zero while the robot is on a black floor.

In the final scenario, *Exploration with sequential targets*, corresponding to a simplified search and rescue scenario where a robot has to find the victim before following, for instance, a radio-ping to the rally point. Its cost function σ_{EST} is similar to σ_{ES}, however, the coefficient γ_S has been replaced by γ_{ST} which includes two discrete steps: it halves the cost once the black patch has been found and drops it to zero only once the robot successfully arrives in the white zone, after having visited a black spot.

It is worth noting, that in addition to the explicit sequential tasks specified before, there are implicit parallel tasks involved in all the scenarios. For example, the exploration scenario consists of moving while avoiding obstacles.

From an optimization perspective, the *Exploration* scenario consists of a three dimensional problem including one discrete (categorical) and two continuous variables. Both the *Exploration with stopping* and *Exploration with sequential targets* scenario correspond to 16 dimensional problems including 8 discrete (categorical) and 8 continuous variables each.

Algorithm 2: Update particle i's position $\mathscr{P}_i$

Input: $\mathscr{P}_i^t$, Current timestep t, previous displacement vector V_i^{t-1}, personal and neighborhood's best positions B_i^t, B_n^t, weights α, β_p, β_n
Output: $\mathscr{P}_i^{t+1}$
Initialize variables
$\gamma_c \leftarrow 0$
$\hat{V}^t \leftarrow \overrightarrow{0}$
$r_1, r_2, r_3 \leftarrow$ random numbers
if *diversity preservation* **then**
  Calculate γ_c (c.f., [12])
  $\hat{V}^t \leftarrow \mathscr{P}_i^t - B_n^t$
end
Calculate new displacement vector
$V_i^t \leftarrow \alpha V_i^{t-1} + \beta_p r_1 (B_i^t - \mathscr{P}_i^t) + \beta_n r_2 (B_n^t - \mathscr{P}_i^t) + \gamma_c r_3 \hat{V}^t$
Update particle position
$\mathscr{P}_i^{(t+1)} \leftarrow \mathscr{P}_i^t + V_i^t$
for *every discrete dimension* $d \in \mathscr{P}_i^{t+1}$ **do**
  Discretize d using Nearest Vertex Approach
  $d \leftarrow round(d)$
end

TABLE III: Configurations for the considered scenarios.

Scenario	N_S[a]	N_E[b]	σ[c]
Exploration (σ_E)	1	500	$\frac{1}{T_E}\sum_{k=1}^{T_E}\left(1-\sigma_{mv,k}(1-\sigma_{oa,k})\right)$
Exploration & Stop (σ_{ES})	2	2000	$\sigma_E\,\gamma_{S,k}$
Expl. with seq. targets (σ_{EST})	2	2000	$\sigma_E\,\gamma_{ST,k}$

[a] N_S: number of states of the resulting FSM controller
[b] N_E: available evaluation budget per optimization run
[c] σ: respective cost functions to be minimized using

$$\gamma_{S,k} = \begin{cases} 0 & if\ on\ black\ floor \\ 1 & otherwise \end{cases}$$

$$\gamma_{ST,k} = \begin{cases} 0 & if\ on\ white\ floor\ and\ on\ black\ floor\ at\ t<k \\ 0.5 & if\ on\ black\ floor\ at\ t\leq k \\ 1 & otherwise \end{cases}$$

and $\sigma_{mv,k} = \frac{||pos_k - pos_{k-1}||}{V_{max} * dT}$ the travelled distance at t=k and $\sigma_{oa,k} = \frac{min(detected\_obstacle\_distances)}{max\_sensor\_range}$ the distance to the closest obstacle at t=k both normalized between 0 and 1.

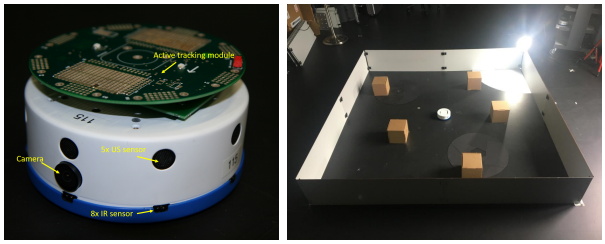


Fig. 2: A Khepera IV robot with the active marker module (left) in its initial position in the real 2 x 2 m arena including obstacles (right). It is worth noting that while the overall floor color seems black to human eyes, it is seen as grey by the robots' infrared sensors.

III. Experiments

All robotic experiments were conducted using a single Khepera IV robot [24], either simulated or real, equipped with an active marker module featuring two LEDs for enabling accurate tracking with the SwisTrack software [25] (see Figure 2). Khepera IV are differentially driven robots with a diameter of 14 cm and a maximal speed of $\sim$ 81 cm/s. Simulations have been carried out in Webots [26], an open-source, high-fidelity robotics simulation platform. For the simulated robots, sensors and actuators were calibrated to match those of the real robots. The experiments in this work are based exclusively on the infrared sensors of the Khepera IV robot, leveraging them as proximity sensors and as ambient light sensors. Finally, four infrared sensors underneath the robot allow for the distinction of different floor colors. The camera and ultrasound sensors are not used.

A. Experimental setup

Independently of the optimization algorithm used and the targeted scenario, the experimental setup remains the same. The optimization algorithm is initialized with its configuration and N_E depending on the scenario (cf. Table III). During the optimization, each candidate solution $\mathscr{F}$ is evaluated in simulation using Webots, where the returned cost-value $\sigma(\mathscr{S}(\mathscr{F}))$ depends on the scenario. The Webots world remains the same for all scenarios and is an exact replication of the real arena depicted in Figure 2, including obstacles, both black and white floor patches, as well as a light source. Using a light-following behavior, the light source enables the robot to navigate efficiently towards the white floor.

Each optimization algorithm-scenario combination is run ten times. In order to investigate the truthfulness of the recorded performance, each final candidate FSM is then re-evaluated twenty times in simulation. Finally, the best resulting FSM for each algorithm-scenario combination is evaluated five times in reality using the setup depicted in Figure 2. Furthermore, a manually designed FSM evaluated in the same way serves as baseline. A solution is considered competitive if its cost is not higher than the upper quartile of the manual solution, rounded up to 0.05.

All optimization algorithms have knowledge about the lower and upper bounds of every variable as well as their type (discrete/continuous). Note that while both the NOMAD and the IRACE implementation also support categorical variables, MDPSO does not. As a result, in order to achieve a fair comparison, we decided to enforce the use of only discrete and continuous values for all algorithms. Consequently, no variable dependencies were provided to any algorithm either.

For MDPSO and MADS the parameters used can be found in Table IV. It is worth noting that in order to ensure a fair comparison, we used default parameters for all algorithms, without resorting to any meta-parameter tuning. Naturally, better learning performances could be achieved using fine-tuned meta-parameters for each optimization algorithm and scenario. However, there is no guarantee that one set of tuned meta-parameters will perform well on a different scenario, resulting thus in potentially highly specific results.

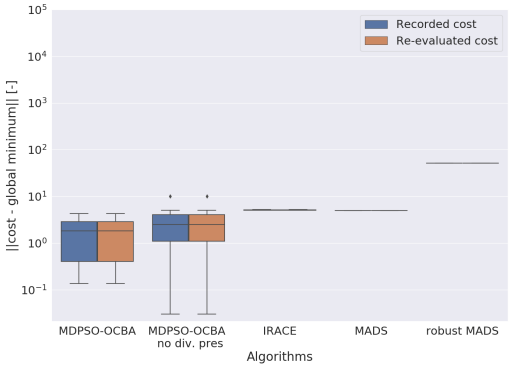
TABLE IV: Parameters used for MDPSO and MADS using the notations [12] and [17] respectively

MDPSO				MADS	
N_p	$3N_V$	γ_{c0}	2	LH SEARCH	true
α	0.8	γ_{d0}	0.7	ROBUST MADS	false*
β_l	1.4	γ_{min}	1E-10	ANISOTROPIC MESH	false
β_g	1.4			RANDOM EVAL SORT	true
λ	0.1			MODEL EVAL SORT	false
				OPPORTUNISTIC EVAL	false

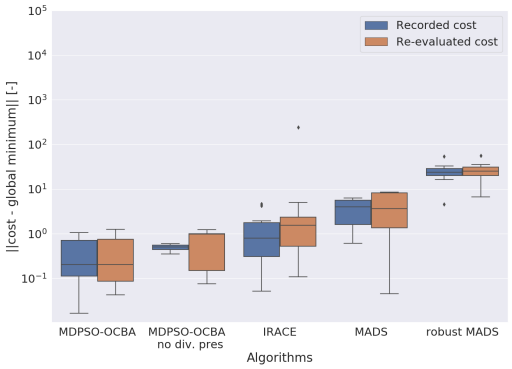
* both robust and non robust versions have been explored in the scope of this work. However, the non-robust version resulted in better performance as reported in the results.

IV. Results

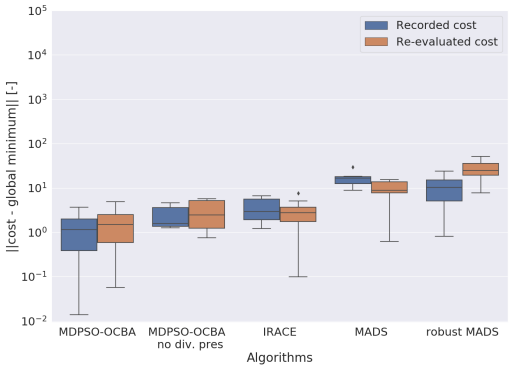
Figure 3 show the performance for all algorithms on the benchmark function mentioned before using different levels of noise. For this evaluation, each algorithm-noise configuration was run 20 times and re-evaluated using the benchmark function without noise to obtain the true performance. We can observe that, while both MDPSO-OCBA variants perform consistently well, the moderate noise results in slightly better mean performance for the MDPSO-OCBA



(a) no noise ($n_{std} = 0$)



(b) moderate noise ($n_{std} = 1$)



(c) high noise ($n_{std} = 10$)

Fig. 3: Comparison of the optimization algorithms for the benchmark function using different levels of noise.

than the no-noise situation. This is supposedly due to the additional exploration induced by the noise. Both IRACE and MADS deliver consistently too, where MADS does not perform as well as soon as noise is involved. However, it is especially unexpected to see the robust MADS performing *worse* than the non-robust variant, even in presence of noise. In fact this observation (which was also repeated for the control generation scenario - not shown here for brevity), suggested us to exclusively consider the non-robust version of MADS in the robotic comparative study. While a detailed analysis of the cause of ROBUST-MADS' worse performance would be out of scope of this contribution, we suspect that the mixed-discrete aspect of our optimization problem is the cause. In fact, the smoothing-based noise-resistance technique introduced in [18] and used in ROBUST-MADS has been developed and demonstrated to increase MADS' performance for continuous, unconstrained problems.

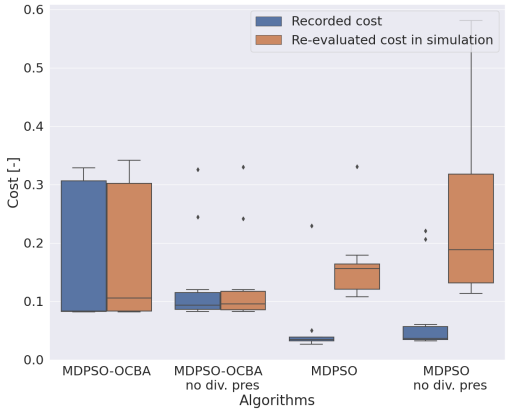


Fig. 4: Comparison between MDPSO and MDPSO-OCBA for the *Exploration with sequential targets* scenario.

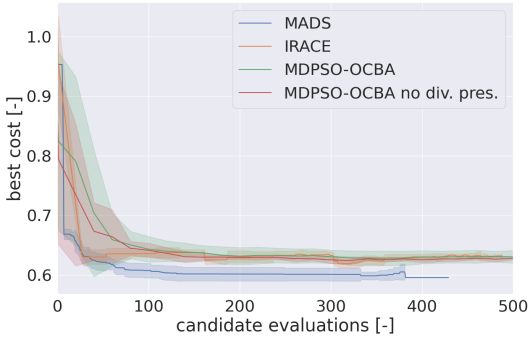


Fig. 5: Comparison between the three optimization algorithms for the *Exploration* scenario. The shaded area corresponds to the standard deviation among optimization runs.

Figure 4 shows the direct comparison of MDPSO vs MDPSO-OCBA for the *Exploration with sequential targets* scenario. MDPSO-OCBA results not only in a better performing solution, but also in a significantly decreased the gap between recorded and re-evaluated cost, corresponding to a better assessment of the learning progress within PSO.Very similar results were obtained using the other scenarios and are thus not reproduced here. It is also worth noting that the results are coherent with the ones reported in [21] for continuous PSO.

Figure 5 shows the representative resulting learning process for the three scenarios, taking as example the *Exploration* scenario. The number of controller evaluations (the most computationally expensive operation in each iteration) for each algorithm is used as optimization progress index. We note that MADS shows the fastest convergence, but all algorithms reach a competitive score in less than 100 (respectively 250 for the other two scenarios) evaluations.

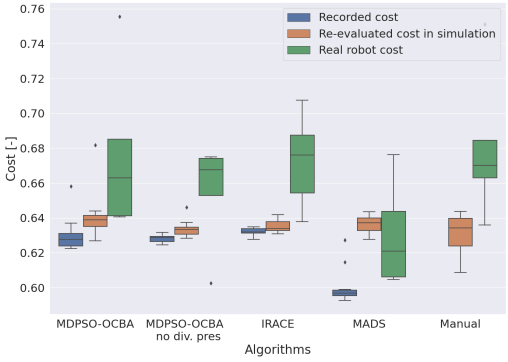


Fig. 6: Recorded versus re-evaluated cost both in simulation and reality of the best performing solutions for the *Exploration* scenario for each of the three optimization algorithms.

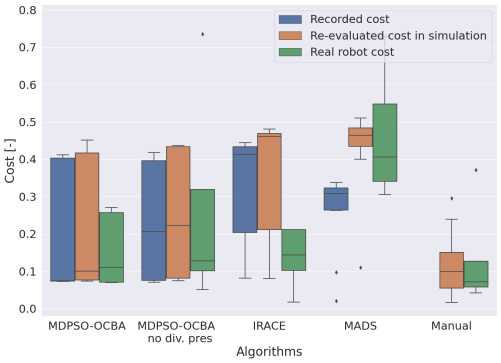


Fig. 7: Recorded versus re-evaluated cost both in simulation and reality of the best performing solutions for the *Exploration with stopping* scenario for each of the three optimization algorithms.

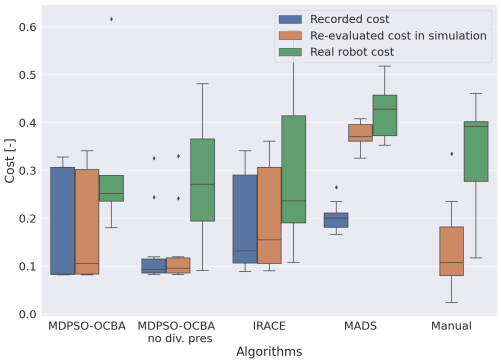


Fig. 8: Recorded versus re-evaluated cost both in simulation and reality of the best performing solutions for the *Exploration with sequential targets* scenario for each of the three optimization algorithms.

MADS is also the only algorithm to stop before reaching the maximal evaluation budget due to convergence of its mesh grid. In fact, the individual MADS runs start converging at around 370 evaluations, which explains the lack of standard deviation after such value. Note that this only happens for the *Exploration* scenario, for the other two scenarios MADS does not converge before reaching the evaluation budget. Both MDPSO-OCBA and IRACE do not converge prematurely for any of the scenarios. Of course, it would be possible to tune both MDPSO-OCBA and IRACE to converge earlier as well, though this would involve a meta-parameter tuning which was intentionally avoided here. Finally, it is worth noting that MDPSO-OCBA without diversity preservation converges faster than MDPSO-OCBA with diversity preservation, as the latter forces MDPSO-OCBA to explore the variable space more thoroughly.

According to the self-reported performances of each algorithm in Figure 5, it seems that MADS performs best. However, by comparing the recorded performances during the optimization with a twenty-fold re-evaluation of the final FSM candidates of every algorithm, shown in orange in Figures 6, 7 and 8, we notice that MADS, and to a lesser degree all algorithms, overestimate their performance during the optimization process (i.e. report lower costs than those obtained subsequently through re-evaluations). Nonetheless, using their respective techniques for noise handling, the reported costs for both MDPSO-OCBA and IRACE at the end of the optimization runs, reported in blue in Figure 6, are reasonably truthful. As we are using the non-robust version of MADS, a strong underestimation of the recorded cost and a corresponding large difference of cost during re-evaluation was expected.

Overall, for the *Exploration* scenario, the performance in simulation of all three algorithms is quite comparable. All algorithms found a competitive solution (with a re-evaluated cost < 0.65) in ten out of ten trials, with the exception of one run of MDPSO-OCBA which is considered an outlier.

The five-fold evaluation of the best performing candidate solution of each algorithm on real Khepera IV robot is shown in green in Figure 6. While the MADS candidate seems to perform best, followed by both MDPSO-OCBA candidates, these differences are most probably simply due to the inherent noisy nature of real robot experiments, since their candidate solutions are almost identical.

Figure 7 shows that for the *Exploration with stopping*, both MDPSO-OCBA algorithms perform best in simulation (orange), with the one with diversity preservation slightly better than without, tightly followed by IRACE. It is further worth noting that out of ten trials, IRACE found a competitive solution (with a re-evaluated cost < 0.2) in three, MADS in one, MDPSO-OCBA in six and MDPSO-OCBA without diversity preservation in five, respectively. This implies that among the considered algorithms, both MDPSO-OCBA versions generate competitive solutions to the manually designed one most consistently.

The evaluation in reality of the best performing candidate, reported in green in Figure 7, shows, however, that the best performing candidate of IRACE is fully competitive with the solutions found by the MDPSO-OCBA algorithms. MADS's solution, on the other hand, performs significantly worse.

Figure 8 shows the results of the *Exploration with sequential targets* scenario. A comparison of the re-evaluated costs

(in orange) shows that MDPSO-OCBA without diversity preservation performs best in simulation. It is worth noting that out of 10 trials, IRACE generated five competitive results (cost < 0.2), MADS zero, MDPSO-OCBA six and MDPSO-OCBA without diversity preservation eight, respectively. However, the performance in reality of the best candidates of IRACE, as well as both MDPSO-OCBA solutions are comparable, especially given the large noises present. MADS, on the other hand, underperforms again compared to the other algorithms both in simulation and reality.

V. Discussion

In order to compare quantitatively the overall effect of the algorithm choice on the resulting performance in simulation, a Friedman test was performed on the results for the three robotics problems. There was a marginally significant, large effect in performance between at least two groups ($\chi^2(4) = 9.33$, $p = 0.053$, Kendall's W = 0.78). The statistical marginal significance was expected given the relatively small sample size, resulting in a low sensitivity.

Dunn's post-hoc test for multiple comparisons with Benjamini-Hochberg p-value correction to reduce the false discovery rate revealed significant differences between MADS and Manual ($p < 0.01$), as well as marginally significant differences between both MDPSO-OCBA variants and MADS ($p < 0.1$) as well as IRACE and Manual ($p < 0.1$). Given these results, we can conclude that MADS is outperformed by MDPSO-OCBA and that the manual solution outperforms IRACE. However, further experiments are needed to draw overall quantitative conclusions between both MDPSO-OCBA variants and IRACE.

The comparison on the benchmark function highlights both the performance and noise resistance of MDPSO-OCBA. It also shows an increase in performance when using the diversity preservation variant introduced in [12]. However, no similar improvement can be observed for the considered robotic problems. This is supposedly due to different characteristics of both fitness landscape and noise of the optimization problems, rendering the diversity preservation mechanism less effective or even counterproductive.

The introduction of discrete steps, which depend on the achievement of sub-tasks, into the cost (or fitness) landscape does not prevent most optimization algorithms considered here to find candidate solutions competitive to the manually implemented controller. However, as expected, the introduction of these discrete steps reduces the performance of all algorithms with less competitive candidate solutions being found. We note, that MDPSO-OCBA seems slightly more robust to such cost landscapes in comparison to MADS and IRACE, finding solutions competitive to a manually designed controller more often. Given the additional difficulties imposed on the optimization process, one may argue that such landscapes should be avoided. However, allowing for such discrete steps in the cost function results in a significantly more efficient design phase, notably for scenarios involving sequential tasks, which greatly increases the attractiveness of distal control architectures.

It is further worth noting that, while we cannot claim that the results obtained here are valid in general, the abstraction of the chosen scenarios through the distal control architecture is thorough, providing a reasonable generality for sequential tasks, independent of the underlying hardware (the numerical values of the parameters might change, the optimization problem's landscape remains similar, however).

VI. Conclusion

In this work, we have shown that the challenge of automatically generating a FSM can be cast into a well-defined optimization problem. A Mixed-Discrete Particle Swarm Optimization algorithm (MDPSO) has been enhanced with an Optimal Computing Budget Allocation (OCBA) scheme in order to adequately address this noisy problem. A comparison with MDPSO showed that MDPSO-OCBA results not only in better performing FSMs, but also significantly better estimates of the ground truth performance achieved.

A comparison of MDPSO-OCBA with two different state-of-the-art optimization algorithms has been drawn for a benchmark function with different noise levels as well as problems concerned with the design of FSM-based robotic controllers for three different scenarios. We have shown that, without any meta-parameter tuning, MDPSO-OCBA is able to compete with, and even slightly outperform, IRACE, the only (to the best of our knowledge) algorithm previously used for automatic FSM generation. MADS, the other state-of-the-art mixed-discrete optimization algorithm considered, is instead significantly outperformed.

We believe that the good performance of MDPSO-OCBA in this type of noisy, mixed-integer optimization problems deserves additional analysis. Our current hypothesis is that the strong exploration aspect of MDPSO coupled with the powerful noise resistance including optimally guarantees provided by OCBA is an ideal combination for the considered problems. Future work will also aim at improving the optimization process in order to obtain an even higher rate of competitive candidate solutions.

References

[1] J. Pugh and A. Martinoli, "Distributed scalable multi-robot learning using particle swarm optimization," *Swarm Intelligence*, vol. 3, pp. 203–222, Sept. 2009.

[2] J. Thangavelautham, N. A. E. Samid, P. Grouchy, E. Earon, T. Fu, N. Nagrani, and G. M. T. D'Eleuterio, "Evolving multirobot excavation controllers and choice of platforms using an artificial neural tissue paradigm," in *IEEE International Symposium on Computational Intelligence in Robotics and Automation*, pp. 258–265, Dec. 2009.

[3] C. Szepesvári, *Algorithms for reinforcement learning*. Synthesis Lectures on Artificial Intelligence and Machine Learning, Jan. 2010.

[4] G. Francesca, M. Brambilla, A. Brutschy, V. Trianni, and M. Birattari, "AutoMoDe: a novel approach to the automatic design of control software for robot swarms," *Swarm Intelligence*, vol. 8, pp. 89–112, June 2014.

[5] G. Francesca, M. Brambilla, A. Brutschy, L. Garattoni, R. Miletitch, G. Podevijn, A. Reina, T. Soleymani, M. Salvaro, C. Pinciroli, F. Mascia, V. Trianni, and M. Birattari, "AutoMoDe-Chocolate: automatic design of control software for robot swarms," *Swarm Intelligence*, vol. 9, pp. 125–152, June 2015.

[6] K. Hasselmann, F. Robert, and M. Birattari, "Automatic design of communication-based behaviors for robot swarms," in *International Conference on Swarm Intelligence (ANTS)*, Lecture Notes in Computer Science, pp. 16–29, Springer International Publishing, 2018.

[7] D. Garzón Ramos and M. Birattari, "Automatic Design of Collective Behaviors for Robots that Can Display and Perceive Colors," *Applied Sciences*, vol. 10, p. 4654, Jan. 2020.
[8] E. Ferrante, E. Duenez-Guzman, A. Turgut, and T. Wenseleers, "GESwarm: grammatical evolution for the automatic synthesis of collective behaviors in swarm robotics," in *ACM Genetic and Evolutionary Computation Conference*, pp. 17–24, July 2013.
[9] A. Lampe and R. Chatila, "Performance measure for the evaluation of mobile robot autonomy," in *IEEE International Conference on Robotics and Automation*, pp. 4057–4062, May 2006.
[10] N. Munoz Ceballos, J. Valencia Velasquez, and N. Ospina, "Quantitative Performance Metrics for Mobile Robots Navigation," in *Mobile Robots Navigation*, IntechOpen, Mar. 2010.
[11] R. Bostelman, T. Hong, and J. Marvel, "Performance measurement of mobile manipulators," in *SPIE 9498, Multisensor, Multisource Information Fusion: Architectures, Algorithms, and Applications*, p. 94980E, May 2015.
[12] S. Chowdhury, W. Tong, A. Messac, and J. Zhang, "A mixed-discrete particle swarm optimization algorithm with explicit diversity-preservation," *Structural and Multidisciplinary Optimization*, vol. 47, pp. 367–388, Mar. 2013.
[13] C.-H. Chen, J. Lin, E. Yücesan, and S. Chick, "Simulation Budget Allocation for Further Enhancing the Efficiency of Ordinal Optimization," *Discrete Event Dynamic Systems*, vol. 10, July 2000.
[14] M. López-Ibáñez, J. Dubois-Lacoste, L. Pérez Cáceres, M. Birattari, and T. Stützle, "The IRACE package: iterated racing for automatic algorithm configuration," *Operations Research Perspectives*, vol. 3, pp. 43–58, Jan. 2016.
[15] C. Audet and W. Hare, *Derivative-free and blackbox optimization*. Springer International Publishing, Jan. 2017.
[16] S. Le Digabel, "Algorithm 909: NOMAD: nonlinear optimization with the MADS algorithm," *ACM Transactions on Mathematical Software*, vol. 37, pp. 44:1–44:15, Feb. 2011.
[17] C. Audet, S. Le Digabel, C. Tribes, and V. Rochon Montplaisir, "The NOMAD project." Software available at https://www.gerad.ca/nomad.
[18] C. Audet, A. Ihaddadene, S. Le Digabel, and C. Tribes, "Robust optimization of noisy blackbox problems using the mesh adaptive direct search algorithm," *Optimization Letters*, vol. 12, pp. 675–689, June 2018.
[19] E. Di Mario, Z. Talebpour, and A. Martinoli, "A comparison of PSO and reinforcement learning for multi-robot obstacle avoidance," in *IEEE Congress on Evolutionary Computation*, pp. 149–156, June 2013.
[20] J. Pugh, Y. Zhang, and A. Martinoli, "Particle swarm optimization for unsupervised robotic learning," in *IEEE Swarm Intelligence Symposium*, pp. 92–99, June 2005.
[21] E. Di Mario, I. Navarro, and A. Martinoli, "Distributed particle swarm optimization using optimal computing budget allocation for multi-robot learning," in *IEEE Congress on Evolutionary Computation*, pp. 566–572, May 2015.
[22] T. Tušar, D. Brockhoff, and N. Hansen, "Mixed-integer benchmark problems for single- and bi-objective optimization," in *ACM Genetic and Evolutionary Computation Conference*, pp. 718–726, July 2019.
[23] D. Floreano and F. Mondada, "Evolution of homing navigation in a real mobile robot," *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, vol. 26, pp. 396–407, June 1996.
[24] J. M. Soares, I. Navarro, and A. Martinoli, "The Khepera IV mobile robot: performance evaluation, sensory data and software toolbox," in *Second Iberian Robotics Conference*, vol. 417, pp. 767–781, 2016.
[25] T. Lochmatter, P. Roduit, C. Cianci, N. Correll, J. Jacot, and A. Martinoli, "SwisTrack - a flexible open source tracking software for multi-agent systems," in *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 4004–4010, Sept. 2008.
[26] O. Michel, "WebotsTM: professional mobile robot simulation," *International Journal of Advanced Robotic Systems*, vol. 1, pp. 39–42, Mar. 2004.
[27] R. C. Arkin, "Motor schema - based mobile robot navigation," *International Journal of Robotics Research*, vol. 8, pp. 92–112, Aug. 1989.

APPENDIX

A. Basic behaviors

The obstacle avoidance behavior is based on the Braitenberg algorithm [27]. Given N proximity sensor readings $s[N] = [s_1, ..., s_N]$ where each element $s_i \in [0,1]$ (with 1 corresponding to contact), the speed values v_l, v_r for left and right wheel respectively, expressed as a fraction of the maximum speed, are calculated through a weighted sum of the individual sensor values s_i through Algorithm 3.

By using the infrared sensors as ambient light sensors and reversing the way the speed delta is applied, Algorithm 3 is transformed into a light following behavior in Algorithm 4.

For our experiments using Khepera IV robots, we had $N = 8$ and the weights used for obstacle avoidance (w_{OA}) and light following (w_{LF}) behaviour are as follows:

$$w_{OA}[8] = [0, 1.2, 1.2, 0.2, -1.2, -1.2, -0, -0]$$

$$w_{LF}[8] = [60, 40, 20, 0, -20, -40, -60, -40]$$

Each behavior accepts two parameters, p_0 and $p_1 \in [0,1]$. We note, however, that p_1 is not used in Algorithm 4 and both p_0 and p_1 are unused in Algorithm 5. As a fail-safe, wheel speeds are bounded by [-1,1] for all behaviors.

Algorithm 3: Obstacle avoidance behavior

Input: N proximity sensor values $s[N] \in [0,1]$,
behavior parameters $p_0,\ p_1$,
weightset $w_{OA}[N]$
Output: Wheelspeeds $v_L, v_R \in [0,1]$

$offset = 0.25 - s[front]$
$delta = 0$
for $i = [1...N]$ **do**
  if $s[i] > p_1 \cdot 0.5$ **then**
    delta += $w_{OA}[i] \cdot s[i] \cdot p_0$
  end
end
$v_L = offset + delta$
$v_R = offset - delta$

Algorithm 4: Light following behavior

Input: N ambient light sensor values $s[N] \in [0,1]$,
behavior parameters $p_0,\ p_1(unused)$,
weightset $w_{LF}[N]$
Output: Wheelspeeds $v_L, v_R \in [0,1]$

$offset = 0.25 + s[front]$
$delta = 0$
for $i = [1...N]$ **do**
  delta += $w_{LF}[i] \cdot s[i] \cdot p_0$
end
$v_L = offset - delta$
$v_R = offset + delta$

Algorithm 5: Stop behavior

Input: Behavior parameters $p_0(unused),\ p_1(unused)$
Output: Wheelspeeds $v_L, v_R \in [0,1]$

$v_L = 0$
$v_R = 0$