An Experiment in Automatic Design of Robot Swarms

AutoMoDe-Vanilla, EvoStick, and Human Experts

Gianpiero Francesca[1], Manuele Brambilla[1], Arne Brutschy[1], Lorenzo Garattoni[1], Roman Miletitch[1], Gaëtan Podevijn[1], Andreagiovanni Reina[1], Touraj Soleymani[1], Mattia Salvaro[1,2], Carlo Pinciroli[1], Vito Trianni[3], and Mauro Birattari[1]

[1] IRIDIA, Université Libre de Bruxelles, Belgium
{gianpiero.francesca,mbiro}@ulb.ac.be
[2] Università di Bologna, Italy
[3] ISTC-CNR, Rome, Italy

Abstract. We present an experiment in automatic design of robot swarms. For the first time in the swarm robotics literature, we perform an objective comparison of multiple design methods: we compare swarms designed by two automatic methods—AutoMoDe-Vanilla and EvoStick—with swarms manually designed by human experts. AutoMoDe-Vanilla and EvoStick have been previously published and tested on two tasks. To evaluate their generality, in this paper we test them without any modification on five new tasks. Besides confirming that AutoMoDe-Vanilla is effective, our results provide new insight into the design of robot swarms. In particular, our results indicate that, at least under the adopted experimental protocol, not only does automatic design suffer from the reality gap, but also manual design. The results also show that both manual and automatic methods benefit from bias injection. In this work, bias injection consists in restricting the design search space to the combinations of pre-existing modules. The results indicate that bias injection helps to overcome the reality gap, yielding better performing robot swarms.

1 Introduction

Automatic design is an appealing way to produce robot control software. So far, evolutionary robotics [1] has been the approach of choice for the automatic design of robot swarms [2,3]. In evolutionary robotics, robots are controlled by a neural network, whose parameters are obtained via artificial evolution in simulation. The main issue of this approach is its difficulty to overcome the *reality gap* [4], that is, the unavoidable difference between simulation and reality.

Recently, Francesca *et al.* [5] proposed a novel approach: AutoMoDe, automatic modular design. AutoMoDe synthesizes control software in the form of a probabilistic finite state machine by selecting, assembling, and fine tuning pre-existing parametric modules. The rationale behind AutoMoDe lies in the machine learning concept of bias–variance tradeoff [6]: Francesca *et al.* [5]

M. Dorigo et al. (Eds.): ANTS 2014, LNCS 8667, pp. 25–37, 2014.

conjectured that the difficulty experienced by evolutionary robotics in overcoming the reality gap bears a resemblance to the generalization problem faced by function approximators in supervised learning. They argued that such difficulty derives from an excessive *representational power* of the control architecture that is typically adopted in evolutionary robotics to be able to fine-tune the dynamics of the robot–environment interaction. Thus, Francesca *et al.* [5] proposed a solution that is reminiscent of the bias injection advocated in the machine learning literature [7] to reduce the representational power of approximators and increase their generalization ability. By synthesizing control software on the basis of pre-existing modules, AutoMoDe reduces the design space. This corresponds to injecting bias, thus decreasing the variance of the design process. As a result, AutoMoDe is expected to overcome the reality gap successfully.

Francesca *et al.* [5] defined, implemented, and tested AutoMoDe-`Vanilla` (hereafter simply `Vanilla`) and `EvoStick`, two specific versions for the e-puck platform [8] of AutoMoDe and evolutionary robotics, respectively. Results obtained on two tasks, aggregation and foraging, indicate that AutoMoDe is a viable and promising approach: `Vanilla` produced better robot control software than `EvoStick`, and appeared to better overcome the reality gap [5].

In this paper, we use exactly the same implementations of `Vanilla` and `EvoStick` that have been previously published in [5], and we test them on five new tasks. In our analysis, we include also swarms designed manually by human experts and swarms synthesized manually starting from the same modules used by `Vanilla`. We perform all tests with a swarm of 20 e-puck robots.

In this paper, we give an original contribution to the swarm robotics literature because we perform the first objective assessment of an automatic method for the design of robot swarms. This is indeed the first work in which an automatic design method previously published and tested on some tasks is tested on new tasks strictly without any modification. The new tasks were proposed by researchers that had not been involved in the development of `Vanilla` and that, at the moment of proposing the tasks, had only a vague idea of its functioning. In particular, they knew that `Vanilla` assembles pre-existing modules, but they did not have any knowledge of the modules made available to the method. As a consequence, we can claim that the tasks have not been selected to favor or disfavor `Vanilla`, or any of the other design methods under analysis. This work is also the first one in the domain of swarm robotics in which automatic design and manual design are compared under controlled conditions.

The results presented in this paper confirm that AutoMoDe is a viable approach to the automatic design of robot swarms. They highlight the strengths of `Vanilla` and also a weakness, for which we suggest a possible solution. More generally, these results provide a new insight into the design of robot swarms. They show that, at least under our experimental protocol, manual design suffers from the reality gap to an extent comparable to that of automatic design. To the best of our knowledge, this has never been discussed in the literature and has never been observed in a controlled empirical study. Finally, contrary to

Table 1. Reference model

Sensor/Actuator	Variables
proximity	$prox_i \in [0,1]$, with $i \in \{1,2,\ldots,8\}$
light	$light_i \in [0,1]$, with $i \in \{1,2,\ldots,8\}$
ground	$gnd_i \in \{black, gray, white\}$, with $i \in \{1,2,3\}$
range and bearing	$n \in \mathbb{N}$ and $r_m, \angle b_m$, with $m \in \{1,2,\ldots,n\}$
wheels	$v_l, v_r \in [-\bar{v}, \bar{v}]$, with $\bar{v} = 0.16\,\mathrm{m/s}$

Period of the control cycle: 100 ms

our original expectations, the results presented in the paper show that human experts produce better swarms when they are constrained to use predefined modules rather then when their creativity is unconstrained.

2 Design Methods Considered

We consider four design methods: `Vanilla`, `EvoStick`, `U-Human`, and `C-Human`. These methods are intended to design control software for a swarm of e-puck robots [8] with Gumstix Overo Linux board, ground sensor, and range-and-bearing sensor—see [5] for a detailed description of the platform. To be more precise, the design methods are allowed to use a subset of the capabilities of this platform. Such subset of capabilities is formally described by the reference model reported in Table 1. The control software designed by the four methods can access sensors and actuators through suitable variables: $prox_i \in [0,1]$ are the readings of the eight proximity sensors; $light_i \in [0,1]$ are the readings of the eight light sensors; $gnd_i \in \{black, gray, white\}$ are the readings of the three ground sensors; n is the number of neighboring robots perceived via the range-and-bearing sensor; r_m and $\angle b_m$ are respectively the range and bearing of the m-th neighbor; finally, $v_l, v_r \in [-\bar{v}, \bar{v}]$, with $\bar{v} = 0.16\,\mathrm{m/s}$ represent the speed of the wheels. All these variables are updated with a period of 100 ms.

Two of the design methods under analysis are automatic methods: `Vanilla` and `EvoStick`. The other two are manual methods: `U-Human` and `C-Human`. `Vanilla` and `EvoStick` have been introduced by Francesca *et al.* [5] and are an implementation of AutoMoDe and evolutionary robotics, respectively. Concerning `U-Human` and `C-Human`, their main difference is that in `U-Human` the designer is *unconstrained*, that is, he is free to develop the control software in any way he prefers, whereas in `C-Human` the designer is *constrained* to develop a finite state machine using the same parametric modules available to `Vanilla`.

In the rest of this section, we introduce the four design methods featured in the experiment. For a thorough description of `Vanilla` and `EvoStick`, we refer the reader to the original publication [5].

`Vanilla` generates control software in the form of a finite state machine starting from a set of twelve pre-existing parametric modules: states are selected from a set of six low-level behaviors and transitions are defined on the basis of six parametric conditions. The low-level behaviors are: exploration, stop,

phototaxis, anti-phototaxis, attraction, and repulsion. With the exception of stop, these behaviors include an obstacle avoidance mechanism. The conditions are: black-floor, gray-floor, white-floor, neighbor-count, inverted-neighbor-count, fixed-probability. All these modules are based on the reference model given in Table 1. For a detailed description of the modules and their parameters, see [5]. `Vanilla` is constrained to create finite state machines composed of at most four states, each with at most four outgoing transitions. As an optimization algorithm, `Vanilla` adopts F-Race [9,10]. Specifically, it uses the implementation provided by the irace package [11] for R [12]. F-Race can be essentially described as a sample & select algorithm. As already pointed out by Francesca *et al.* [5], F-Race has been adopted in `Vanilla` due to its simplicity: the authors wished to keep the focus on the control architecture rather than on the optimization process. Within the optimization process, control software candidates are evaluated using the ARGoS multi-robot simulator [13].

`EvoStick` generates control software in the form of a feed-forward neural network without hidden nodes. Inputs and outputs of the network are defined on the basis of the variables given in the reference model of Table 1. To optimize the parameters of the neural network, `EvoStick` adopts a standard evolutionary algorithm. Within the optimization process, candidate parameter sets are evaluated using ARGoS.

`U-Human` is a manual method in which the human designer is left free to design control software as he deems appropriate. Within the control software, sensors and actuators are accessed via an API that implements the reference model given in Table 1. Within the design process, the designer tests his control software using ARGoS.

`C-Human` is a manual method in which the human designer is constrained to use `Vanilla`'s control architecture and modules. The designer does not directly write the control software: rather, he employs a software tool that allows him to specify a finite state machine, visualize it, and test it using ARGoS. In other words, the human designer takes the role of `Vanilla`'s optimization algorithm. As in `Vanilla`, the human is constrained to create finite state machines comprised of at most four states, each with at most four outgoing transitions.

3 Experimental Protocol

In the protocol we adopt, five researchers, hereinafter referred to as *experts*, play a central role. The experts are active in swarm robotics, have about two years of experience in the domain, and are familiar with ARGoS. These experts have not been involved in the development of `Vanilla` and `EvoStick` and, at the moment of participating in the experiment, they had only a vague idea of `Vanilla`: they knew that `Vanilla` operates on pre-existing parametric modules, but they were unaware of what modules are available.

The role of each expert is threefold: i) define a task; ii) solve a task acting as `U-Human`; and iii) solve a task acting as `C-Human`. Table 2 summarizes the role

Table 2. Role of the experts, anonymously indicated here by the labels E1 to E5

task		defined by	`U-Human`	`C-Human`
SCA	shelter with constrained access	E1	E5	E4
LCN	largest covering network	E2	E1	E5
CFA	coverage with forbidden areas	E3	E2	E1
SPC	surface and perimeter coverage	E4	E3	E2
AAC	aggregation with ambient cues	E5	E4	E3

played by each expert. The criteria and the restrictions that the experts had to follow in the definition of the tasks are presented in Sect. 4. When solving a task either as `U-Human` or `C-Human`, each expert worked for four consecutive hours. The involvement of each expert spanned two days: on day one, the expert acted as `U-Human` on the first task assigned to him; on day two, he acted as `C-Human` on the second task. In both cases, the expert came to know the definition of the task he had to solve only at the beginning of the four hours. During these four hours, the expert could test the control software in simulation using ARGoS, but was not allowed to test it in reality on the e-pucks.

Regarding the automatic design methods, `Vanilla` and `EvoStick` have been allowed a *design budget* of 200,000 simulations for each task. The design process has been conducted on a high performance computing cluster comprised of 400 opteron6272 cores. `Vanilla` and `EvoStick` produced the control software for each task in about 2 hours and 20 minutes.

To summarize, all methods under analysis i) produce control software for the same robotic platform formally described by the reference model given in Table 1; ii) complete the design process within 4 hours; and iii) use the same simulator to assess candidate control software during the design process. The protocol of the experiment does not allow any modification of the control software on the basis of its performance in reality on the e-pucks.

We used ARGoS to cross-compile the control software for the e-puck. We then uploaded it to the robots without any modification. We evaluated the control software generated by the four methods for each of the five tasks via 10 runs on the robots. We computed the performance of the robot swarm in an automatic way using a tracking system [14] that acquires images via a ceiling-mounted camera and records the position and orientation of all robots every 100 ms. To assess the impact of the reality gap on the four design methods, we performed also a set of 10 runs per task in simulation.

For each task, we report a notched box-and-whisker plot that summarizes the results: wide boxes represent data gathered with robots, narrow boxes data obtained in simulation. If notches of two boxes do not overlap, the observed difference is significant. We report also the results of a Friedman test [15] that aggregates the data gathered with the robots over the five tasks.

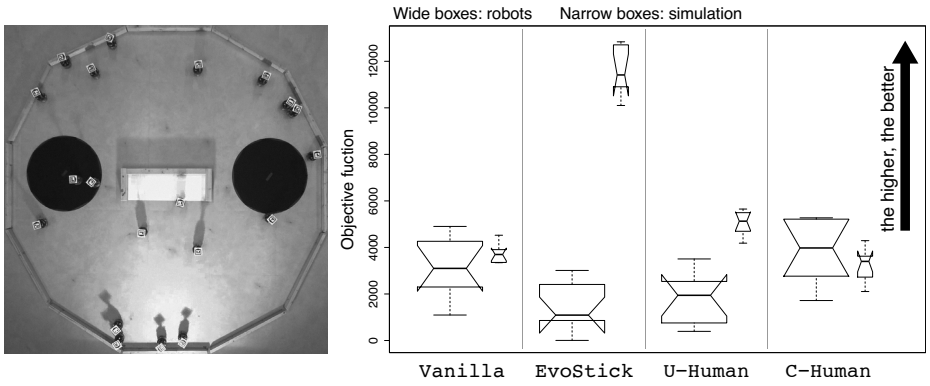

Fig. 1. SCA – arena and 20 e-pucks (left); results of the analysis (right). Robots should aggregate in the white shelter.

4 Tasks and Results

Each of the tasks has been independently defined by one of the experts. Each expert has been provided with the reference model of Table 1 and has been asked to conceive a task that he would be able to solve with a swarm of 20 robots characterized by the given reference model. The expert has also been provided with a list of constraints that the task definition must satisfy: The arena is a dodecagonal area of $4.91\,\mathrm{m}^2$ surrounded by walls. The floor of the arena is gray. The arena can contain up to 3 colored regions in which the floor can be either black or white. These regions can be either circular, with diameter of up to 0.6 m, or rectangular, with sides up to 0.6 m. The setup might include a single light source positioned outside the arena, at 0.75 m from the ground. It might include also up to 5 cuboidal obstacles of size $0.05\,\mathrm{m} \times 0.05\,\mathrm{m} \times L$, where $0.05\,\mathrm{m} \leq L \leq 0.80\,\mathrm{m}$. The swarm comprises 20 e-puck robots in the configuration described in Sect. 2. The duration of each run is $T = 120\,\mathrm{s}$. At the beginning of the run, the robots are randomly distributed in the arena. The task must be formally described by an objective function, which must be either maximized or minimized. The objective function must be defined on the basis of the position of the robots, evaluated with a period of 100 ms.

In the rest of this section, we describe the five tasks and we report the results obtained by the four design methods. For a more detailed description of the tasks and of their objective functions, see the online supplementary material [16].

SCA – Shelter with Constrained Access

In SCA, the goal of the swarm is to maximize the number of robots on an aggregation area. The aggregation area has a rectangular shape, is characterized by a white ground, and is surrounded by walls on three sides. The environment presents also a light source and two black regions that are positioned in front and aside the aggregation area, respectively—see Fig. 1.

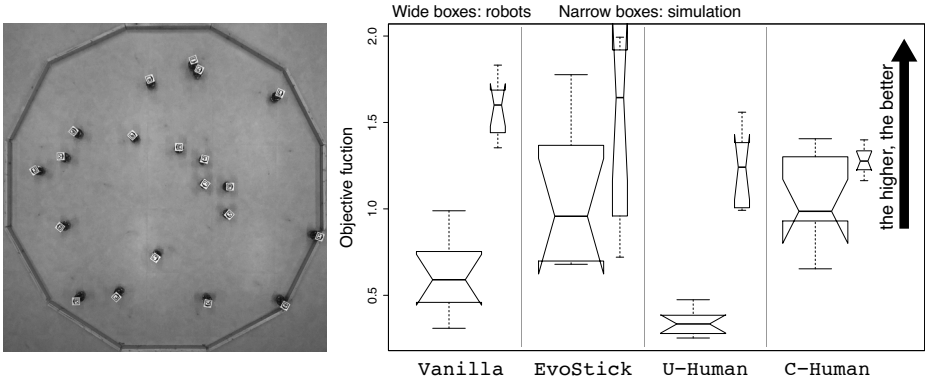

Fig. 2. LCN – arena and 20 e-pucks (left); results of the analysis (right). Robots should cover the largest possible area while maintaining connection with one another.

Formally, the problem is defined as the maximization of $F_{SCA} = \sum_{t=1}^{T} N_a(t)$, where $N_a(t)$ is the number of robots in the aggregation area at time t, and T is the duration of the run.

Results. `C-Human` and `Vanilla` perform better than the other methods. In particular, `Vanilla` is significantly better that `EvoStick`—this is indicated by the fact that, in Fig. 1, the notches of the respective boxes do not overlap. An interesting result is the inability of `EvoStick` to overcome the reality gap. The same observation can be made also for `U-Human`, even though to a far minor extent. In contrast, `C-Human` and `Vanilla` overcome the reality gap satisfactorily.

LCN – Largest Covering Network

In LCN, the robots must maintain connection with each other, while trying to cover the largest possible area—see Fig. 2 for a picture of the experimental setting. We assume that i) two robots are connected when their distance is less than 0.25 m, and ii) each robot covers a circular area of radius 0.35 m.

Formally, the problem is defined as the maximization of $F_{LCN} = A_{C(T)}$ where $C(T)$ is the largest group of connected robots at the end T of the run and $A_{C(T)}$ is the surface of the union of the areas covered by the robots in $C(T)$.

Results. `C-Human` and `EvoStick` achieve better performance compared to the other methods, with `Vanilla` performing slightly better than `U-Human`. The methods performing worse are those that encounter more difficulties in overcoming the reality gap: `U-Human` and `Vanilla`.

CFA – Coverage with Forbidden Areas

In CFA, the goal of the swarm is to cover the entire arena except a few forbidden areas characterized by a black ground—see Fig. 3.

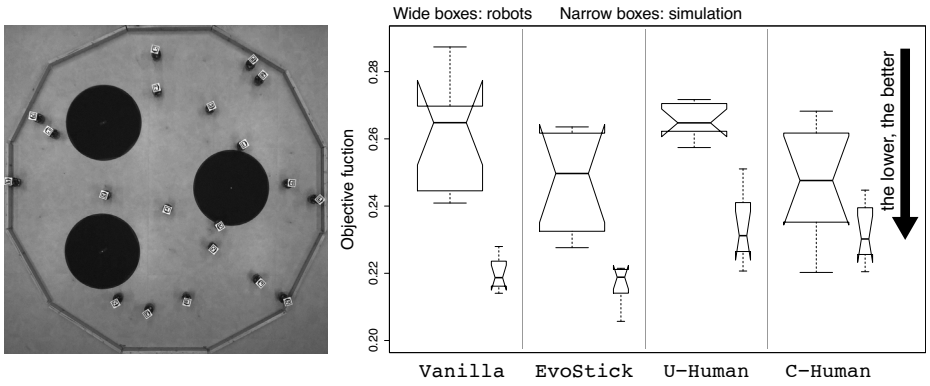

Fig. 3. CFA – arena and 20 e-pucks (left); results of the analysis (right). Robot should cover the arena except the forbidden black areas.

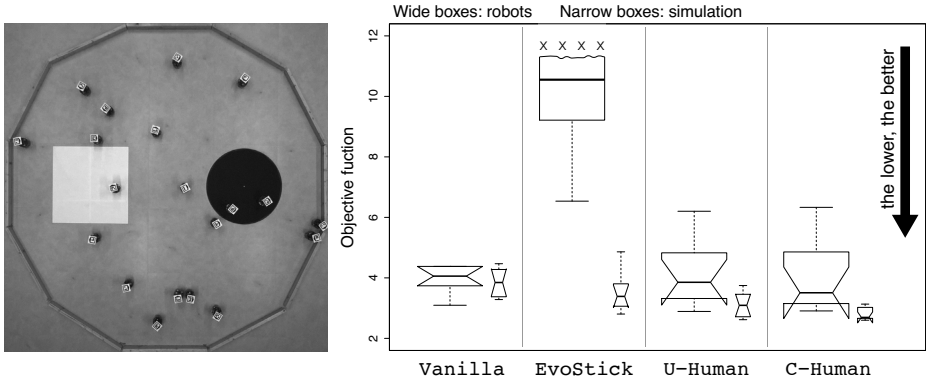

Fig. 4. SPC – arena and 20 e-pucks (left); results of the analysis (right). Robot should cover the surface of the white square and the perimeter of the black circle.

Formally, the problem is defined as the minimization of $F_{CFA} = E[d(T)]$, where $E[d(T)]$ is the expected distance, at time T, between a generic point of the arena and the closest robot that is not in a forbidden area. Distances are measured in meters.

Results. All methods perform more or less similarly. The results are all within a range of few centimeters, that is, less that half of the e-puck's diameter. Concerning the reality gap, for all methods we observe differences between simulation and reality, but these differences are small in absolute terms. Also in this case, they are within a range of few centimeters.

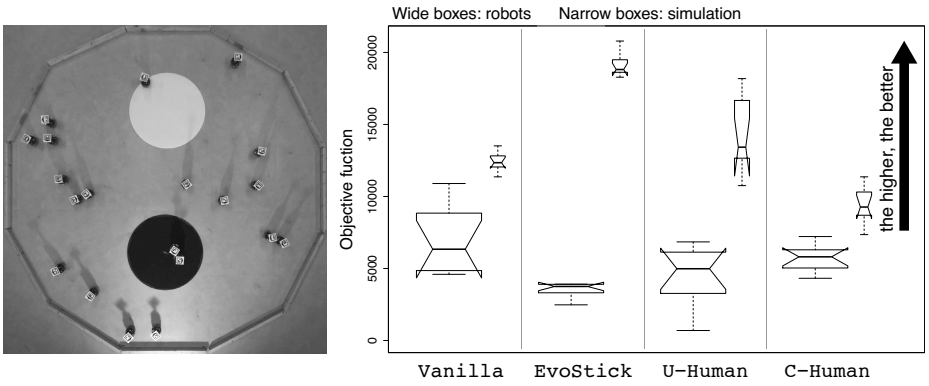

Fig. 5. AAC – arena and 20 e-pucks (left); results of the analysis (right). Robot should aggregate in the black region.

SPC – Surface and Perimeter Coverage

In SPC, the goal of the swarm is to cover the surface of a white square region and the perimeter of a black circular region—see Fig. 4.

Formally, the problem is defined as the minimization of $F_{SPC} = c_a E[d_a(T)] + c_p E[d_p(T)]$, where $E[d_a(T)]$ is the expected distance, at time T, between a generic point of the white region and the closest robot positioned on the white region itself; $E[d_p(T)]$ is the expected distance, at time T, between a generic point of the perimeter of the black region and the closest robot positioned on the perimeter itself; and c_a and c_p are normalization factors [16]. Failing to place at least a robot on the surface of the white region and/or on the perimeter of the black region is a major failure. In this case, $E[d_a(T)]$ and $E[d_p(T)]$ are undefined and we thus assign an arbitrarily large value to F_{SPC}.

Results. The most notable element is that `EvoStick` is not able to overcome the reality gap and achieves significantly worse results than the other methods. The four `X`s marked in the plot indicate four runs that resulted in a major failure. `Vanilla`, `U-Human`, and `C-Human` perform comparably well.

AAC – Aggregation with Ambient Cues

In AAC, the goal of the swarm is to maximize the number of robots on an aggregation area represented by a black region. Besides the black region, the environment comprises a white region and a light source that is placed south of the black region—see Fig. 5.

Formally, the problem is defined as the maximization of $F_{AAC} = \sum_{t=1}^{T} N_b(t)$, where $N_b(t)$ is the number of robots on the black region at time t.

Results. `Vanilla` performs slightly better than `U-Human` and `C-Human`, and significantly better than `EvoStick`. Concerning the manual methods, `C-Human` performs slightly better than `U-Human`. The greatest difference among the methods

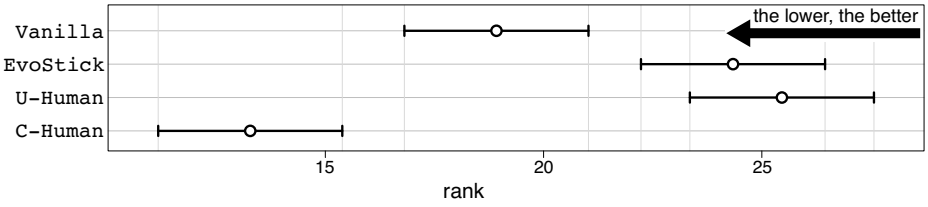

Fig. 6. Friedman test on aggregate data from the five tasks

lies in their ability to overcome the reality gap. In particular, `EvoStick` is the method that has the most severe difficulty in overcoming the reality gap, followed by `U-Human`. `Vanilla` and `C-Human` still present problems, but to a minor extent compared to the other methods.

5 Analysis and Discussion

To aggregate the results presented in Sect. 4, we perform a Friedman test [15], using the task as a blocking factor and considering 10 replicates per task. The outcome of the test is represented in Fig. 6. The plot represents the expected rank obtained by a design method in the robot experiments, together with a confidence interval. If the confidence intervals of two methods do not overlap, the difference between the expected rank of the two is statistically significant. The test indicates that `C-Human` perform significantly better than `Vanilla`, which, in turn, perform significantly better than `EvoStick` and `U-Human`.

These results confirm those obtained by Francesca *et al.* [5]: `Vanilla` produced swarms with significantly better performance than those produced by `EvoStick`. However, the results also highlight that `Vanilla` has a limitation: as already noted in Francesca *et al.* [5], F-Race, the optimization algorithm adopted in `Vanilla`, is not particularly powerful and is unable to fully exploit the potential of the available parametric modules—see the results of `C-Human`, which is based on the same modules. The results clearly suggest that `Vanilla` can be improved by adopting a more powerful optimization algorithm.

The analysis of the swarms produced by human experts are particularly interesting and informative on their own. First of all, our results show that, when it is not possible to modify the developed control software on the basis of its performance in reality, manual design suffers from the reality gap, as automatic design does. In other words, it is difficult for human experts to foresee whether the developed control software will work in reality as expected or not.

Moreover, we observed that, under the protocol we adopted, human experts produce better swarms when they are constrained to use predefined modules. This result was unexpected and appears counter-intuitive. We expected that the understanding and intuition of human experts would have produced excellent results in case their creativity had been left unconstrained. We expected that the restriction to use predefined modules would have prevented human experts

from fully expressing their potential. Our results proved us wrong: although the control software produced by `U-Human` scored well in simulation, it failed to be effective in reality. Our results clearly indicate that the restriction to use predefined modules enables `C-Human` to successfully overcome the reality gap.

Concerning the comparison between manual and automatic design, `Vanilla` produced swarms that are significantly better than those produced by `U-Human`, but worse that those produced by `C-Human`. This is a promising result. It proves that the core idea of AutoMoDe is valid: by constraining the design space to the control software that can be obtained assembling predefined modules, one effectively increases the ability to overcome the reality gap. This insight is valid for both automatic and manual design.

As the set of modules used by `C-Human` are the same used by `Vanilla`, the performance advantage of `C-Human` over `Vanilla` is to be fully ascribed to the limitations of `Vanilla`'s optimization algorithm, as already discussed above. The results obtained by `C-Human` and by `Vanilla` show that the set of modules is generally appropriate for tackling swarm robotics tasks with the robotic platform considered: they enabled the synthesis of control software that performed satisfactorily across all the five tasks.

6 Conclusions

In this paper, we presented an experiment in automatic design of robot swarms. This experiment introduces a number of novelties with respect to the literature. In particular, the two automatic methods under analysis—`Vanilla` and `EvoStick`—had been previously published [5] and have been used here strictly without any modification. In swarm robotics, this is the first time that i) an automatic design method is tested on as many as five tasks, without adapting it to each of them; ii) the tasks considered are different from the one for which the method has been originally proposed; and iii) the tasks are devised by researchers that had not been involved in the development of the method. Moreover, this is the first time that a comparison between automatic and manual design methods is performed under controlled conditions.

The results of the experiment are encouraging. First of all, they confirm previous results obtained on other tasks [5]: `Vanilla` performs better that `EvoStick`. Second, they show that, under the protocol we adopted, human experts produce better swarms when they are constrained to use pre-existing modules: `C-Human` outperforms `U-Human`. Together, the superiority of `Vanilla` over `EvoStick` and of `C-Human` over `U-Human` corroborate the core hypothesis behind AutoMoDe: by introducing a bias in the design process—that is, by restricting the design space—one obtains better robot swarms. Moreover, `Vanilla` outperformed `U-Human`, the unconstrained manual design: this is the first clear empirical evidence that the automatic design of robot swarms is a viable reality. On the other hand, we do not consider it a failure that `C-Human` scored better than `Vanilla`. As `C-Human` uses the same modules defined in `Vanilla`, differences are to be ascribed to the limitations of `Vanilla`'s optimization algorithm. The results indicate that

`Vanilla`'s module set is appropriate to solve swarm robotics tasks with the platform considered in our study.

Our short-term future work will focus on the development of an improved version of `Vanilla` that, taking into account the indications emerged from our results, will adopt a more powerful optimization algorithm. In the medium term, we will develop an instance of AutoMoDe for a more complex reference model.

Acknowledgments. We thank Maria Zampetti and Maxime Bussios for their help with the robots, and Marco Chiarandini for his implementation of the Friedman test. This research has received funding from the European Research Council under the European Union's Seventh Framework Programme—ERC grant agreement n. 246939. It received funding also from the European Science Foundation via the H2SWARM project. Vito Trianni acknowledges support from the Italian CNR. Arne Brutschy and Mauro Birattari acknowledge support from the Belgian F.R.S.-FNRS.

References

1. Nolfi, S., Floreano, D.: Evolutionary Robotics: The Biology, Intelligence, and Technology of Self-organizing Machines. MIT Press, Cambridge (2000)
2. Brambilla, M., Ferrante, E., Birattari, M., Dorigo, M.: Swarm robotics: A review from the swarm engineering perspective. Swarm Intelligence 7(1), 1–41 (2013)
3. Dorigo, M., Birattari, M., Brambilla, M.: Swarm robotics. Scholarpedia 9(1), 1463 (2014)
4. Trianni, V., Nolfi, S.: Engineering the evolution of self-organizing behaviors in swarm robotics: A case study. Artificial Life 17(3), 183–202 (2011)
5. Francesca, G., Brambilla, M., Brutschy, A., Trianni, V., Birattari, M.: AutoMoDe: A novel approach to the automatic design of control software for robot swarms. Swarm Intelligence 8(2), 89–112 (2014)
6. Geman, S., Bienenstock, E., Doursat, R.: Neural networks and the bias/variance dilemma. Neural Computation 4(1), 1–58 (1992)
7. Dietterich, T., Kong, E.B.: Machine learning bias, statistical bias, and statistical variance of decision tree algorithms. Technical report, Department of Computer Science, Oregon State University (1995)
8. Mondada, F., et al.: The e-puck, a robot designed for education in engineering. In: 9th Conf. on Autonomous Robot Systems and Competitions, Portugal, Instituto Politécnico de Castelo Branco, pp. 59–65 (2009)
9. Birattari, M., Stützle, T., Paquete, L., Varrentrapp, K.: A racing algorithm for configuring metaheuristics. In: Proc. of the Genetic and Evolutionary Computation Conference (GECCO 2002), pp. 11–18. Morgan Kaufmann, San Francisco (2002)
10. Birattari, M.: Tuning Metaheuristics. Springer, Berlin (2009)
11. López-Ibáñez, M., Dubois-Lacoste, J., Stützle, T., Birattari, M.: The irace package, iterated race for automatic algorithm configuration. Technical Report TR/IRIDIA/2011-004, IRIDIA, Université Libre de Bruxelles, Belgium (2011)
12. R Development Core Team: R: A language and environment for statistical computing. R Foundation for Statistical Computing (2008)

13. Pinciroli, C., Trianni, V., O'Grady, R., Pini, G., Brutschy, A., Brambilla, M., Mathews, N., Ferrante, E., Di Caro, G., Ducatelle, F., Birattari, M., Gambardella, L.M., Dorigo, M.: ARGoS: A modular, parallel, multi-engine simulator for multi-robot systems. Swarm Intelligence 6(4), 271–295 (2012)
14. Stranieri, A., Turgut, A., Francesca, G., Reina, A., Dorigo, M., Birattari, M.: IRIDIA's arena tracking system. Technical Report TR/IRIDIA/2013-013, IRIDIA, Université Libre de Bruxelles, Belgium (2013)
15. Conover, W.J.: Practical Nonparametric Statistics. Wiley, New York (1999)
16. Francesca, G., et al.: An experiment in automatic design of robot swarms. Supplementary Material (2014), `http://iridia.ulb.ac.be/supp/IridiaSupp2014-004`