Université Libre de Bruxelles

*Institut de Recherches Interdisciplinaires
et de Développements en Intelligence Artificielle*

On Mimicking the Effects of the Reality Gap with Simulation-only Experiments

A. Ligot and M. Birattari

IRIDIA – Technical Report Series

Technical Report No.
TR/IRIDIA/2018-003

March 2018

IRIDIA – Technical Report Series
ISSN 1781-3794

Published by:

IRIDIA, *Institut de Recherches Interdisciplinaires et de Développements en Intelligence Artificielle*
Université Libre de Bruxelles
Av F. D. Roosevelt 50, CP 194/6
1050 Bruxelles, Belgium

Technical report number TR/IRIDIA/2018-003

On Mimicking the Effects of the Reality Gap with Simulation-only Experiments*

Antoine Ligot and Mauro Birattari

IRIDIA, Université libre de Bruxelles, Belgium

Abstract. One issue in the automatic design of control software for robot swarms is the so-called reality gap—the difference between reality and the simulation models used in the automatic design process. It is commonly understood that the reality gap manifests itself as a drop in performance when control software developed in simulation is used to control physical robots. Yet, often disregarded is the relative nature of this performance drop: the reality gap does not affect equally all instances of control software. Indeed, one might observe a rank inversion: control software A might perform better than control software B in simulation, but perform worse on robots. The possibility of rank inversion undermines any performance comparison made in simulation. It would thus seem the only way to assess control software is in robot experiments, which are costly and time consuming.

We argue it is unnecessary to assume reality is more complex than simulation models for the effects of the reality gap to occur. Indeed, we show that performance drop and rank inversion can occur if one automatically designs control software in simulation using a model and then assesses it in simulation on another model—what we call a pseudo-reality. Our results suggest that an appropriately conceived pseudo-reality could be used to test automatically-generated control software for performance drop and rank inversion, without performing robot experiments.

1 Introduction

The reality gap is one of the main issues in the automatic design of robot swarms [16]. A robot swarm is a highly redundant, self-organized, and decentralized system [1,37,12]. Designing the individual rules that lead to the desired collective behavior is difficult. Methods to guide the designers exist for some specific collective behaviors and under some hypotheses [22,2,7,36]. Nonetheless, a generally applicable methodology is still missing.

Automatic design methods eliminate the burden of manually decomposing the desired global behavior into the appropriate microscopic behaviors of the individuals. By maximizing a mission-dependent performance measure, an optimization algorithm searches for an appropriate instance of control software to

* All experiments were performed by AL. The article was drafted by AL and revised by the two authors. The research was directed by MB.

be installed on each individual robot. Generally, the optimization process relies on simulation. Methods have been proposed that (could possibly or have been demonstrated to) operate directly on robot hardware [29,41,27,8,21,39,11]. Although these methods are promising to adapt/fine-tune behaviors to the environment, they do not appear to be an alternative to simulation-based design due to safety concerns and to the limited solution space they can explore [16]. When the design is performed in simulation, a resulting instance of control software is likely to be fine-tuned to the specific simulation model [14], which should not be expected to perfectly reproduce the real world. Due to the differences between simulation and reality, which are commonly referred to as the *reality gap* [9,26], a performance drop typically occurs when an instance of control software designed in simulation is assessed on physical robots.

An issue that is often overlooked is that the occurrence of performance drops due to the reality gap is a relative problem: each instance of control software might be affected to a different extent. The relative nature of performance drops might result in what we shall call a *rank inversion*: control software A outperforms control software B in simulation, but B outperforms A when assessed on the physical robots. Rank inversions can be observed when comparing instances of control software produced by different design methods [18], or by the same one at different steps along the optimization process [4]. Indeed, Birattari et al. [4] observed a phenomenon that they called *overdesign*: past an optimal number of steps of the optimization process, the performance obtained in reality diverges from the one obtained in simulation.

In the literature, performance drops due to the reality gap are commonly explained by saying that reality is more complex than simulations—or equivalently, that simulations are too simplistic [33,28].

In this work, we argue that it is not necessary to assume that reality is more complex than simulation for the effect of the reality gap to occur. More precisely, we contend that performance drops that lead to rank inversion can be observed even if the model under which control software is designed is not a simplistic version of the context/conditions under which it is eventually assessed. We support our contention with a set of simulation-only experiments in which we create an artificial reality gap.

Creating an artificial, simulation-only reality gap is not a novel contribution we make here for the first time. Koos et al. [28] already created a simulation-only reality gap between a simple simulator—used to design control software—and an accurate one—used for assessing it. The choice of creating a reality gap between a simple and a more complex simulator clearly reflects the common understanding discussed above, which is precisely what we challenge here. We maintain that it is not necessary to assume that control software is assessed under context/conditions that are more complex than those experienced in the design for the effects of the reality gap to manifest.

The artificial reality gap we create is based on two robot models: M_A and M_B. We design control software in simulation on model M_A and then we assess it, always in simulation, but relying on model M_B. We shall call a *pseudo-reality* any secondary model that we use for assessing control software—and that therefore plays the role of reality. Model M_A has been proposed by Francesca et al. [18] who used it to design control software that was eventually assessed on robots. We introduce here model M_B, which we conceived so that, when used as pseudo-reality to assess control software designed on M_A, it produces performance drops and rank inversions that are qualitatively similar to those observed by Francesca et al. [18].

A priori, it could be argued that M_A and M_B are equally complex as they share the same nature—see Section 3. Nonetheless, to completely exclude the possibility that the observed effects are the results of an undesired higher complexity of M_B, we consider both the case in which we use M_A for the design and M_B for the assessment, and the case in which we invert the roles of the two models. As we show in Section 4, qualitatively similar drops and inversions appear in both cases. This substantiates our contention, and indicates that the effects of the reality gap can manifest even when the design model is not a simplistic version of the one used in the assessment, possibly due to the fact that control software *overfits* the former.

Besides shedding further light on the nature of the reality gap, this study suggests that creating an artificial, simulation-only version of it could have useful practical implications. For example, it would dispense researchers from costly and time consuming robot experiments that, at the moment, are necessary to tell whether a design method is more prone than another one to performance drops, whether a rank inversion should be expected, or whether to stop an optimization process to prevent the *overdesign* phenomenon to occur.

2 Related work

The reality gap [9,26] is the differences between simulation and reality. Because of the reality gap, control software designed in simulation might fail to work properly on the real robotic platform. It is common to observe a performance drop when control software is transferred from simulation to reality. In the literature, it is assumed that this performance drop occurs because simulation is unable to reproduce the complexity of reality [33,28].

Several approaches have been proposed to cross the reality gap effectively—that is, to limit the performance drop of control software. However, none of these approaches have been studied in details, no extensive comparison has been made, and the reality gap remains a major issue in the automatic design of robot control software [16,38]. Approaches to cross the reality gap have mainly been proposed in the context of evolutionary robotics for single robots. Nonetheless, they are typically general enough to be relevant to any design method based on off-line simulation, both for single- and multi-robot systems.

Behind these approaches, we see two main lines of reasoning. On the one hand, some researchers aimed at reducing the differences between simulation and reality as much as possible [31,26,6,42,28]. They were driven by the assumption that a smooth transition from simulation to reality would occur if simulation

Table 1. Taxonomy of the most significant approaches proposed in the literature to cross the reality gap. We group the approaches according to the main line of reasoning followed in their development.

focus on	reducing differences between simulation and reality	enhancing robustness of control software
simulation models	Miglino et al. [31] Jakobi et al. [26] Bongard and Lipson [6] Zagal and Ruiz-Del-Solar [42]	Jakobi [24,25]
design methods	Koos et al. [28]	Floreano et al. [15,40,13] Francesca et al. [18,17]

reproduced relevant real-world dynamics accurately. On the other hand, other researchers strived to make control software robust to differences [24,25,40,18,17]. They were driven by the assumption that differences between simulation and reality are eventually unavoidable. Each of these lines of reasoning were developed with a focus either on simulation models [31,26,24,6,42] or on the design method [40,28,18,17]. In the first case, researchers focused on making simulation models more realistic or more general so as to render the design process more robust. In the second case, researchers focused on conceiving methods that either exploit regions of the search space that are accurately reproduced by the simulator or that are intrinsically more robust than traditional methods. See Table 1 for a taxonomy.

Reducing differences between simulation and reality—focus on simulation models. Miglino et al. [31] were the first to propose guidelines for reducing differences between simulation models and reality. They suggested to (i) use samples from the robot's sensors and actuators; (ii) add conservative noise to models; and (iii) continue the design process in reality, should an unacceptable performance drop be observed. Similarly, Jakobi et al. [26] insisted on the importance of adding appropriate levels of noise to models. Since then, using real data in simulation and fine-tuning noise models have become common practice [38]. Bongard and Lipson [6] proposed a method based on the co-evolution of control software and simulator. While optimizing the control software, the method improves the simulation models using sensor readings gathered in robot experiments. Zagal and Ruiz-Del-Solar [42] developed a method in which differences between performance observed in simulation and in reality are used to tune the parameters of the simulation.

Reducing differences between simulation and reality—focus on design methods. Koos et al. [28] proposed a multi-objective method that aims at constraining the design process to instances of control software whose behavior is accurately simulated. The method relies on a model to estimate the differences between performance in simulation and reality. The model is updated based on physical-robot evaluations of instances of control software generated by the design process. To assess the proposed method, the authors performed experiments with two different robotic platforms. They also performed experiments in a fully simulated setting in which the role of the physical-robot evaluations was played by highly-realistic simulations. In other terms, the authors artificially created a simulation-only reality gap problem between a simple and a more accurate simulator.

Enhancing robustness of control software—focus on simulation models. Jakobi [24,25] was the first to explicitly aim at producing control software that is robust to differences between simulation and reality. The method he proposed is based on two devices: (i) model only the robot-robot and robot-environment interactions that are meaningful to obtain the desired behavior, and (ii) apply random variations on all aspects of the simulation.

Enhancing robustness of control software—focus on design methods. Floreano et al. [15,40] applied an on-line adaptation mechanism to the parameters of a neuro-controller. The behavior developed was observed to transfer smoothly from simulation to reality [13]. Francesca et al. [18,17] observed that the reality gap resembles the generalization problem of supervised learning. They conjectured that evolutionary robotics could be affected by the reality gap because it overfits the conditions experienced during the design process due to an excessive representational power of neural networks. Guided by their conjecture, the authors developed a design method with restricted representational power and showed that the control software it produces crosses the reality gap satisfactorily.

3 Materials and methods

In this section, we describe the robots, the automatic design methods, the simulation models and the protocol used in our experiments.

Robots (simulated). We simulate an extended version of the e-puck robot [19,32] using the ARGoS3 simulator [34] (version 3.0.0-beta45). For the purpose of this study, we consider a subset of the sensors and actuators the robot is equipped with. The control software has access to variables that abstract sensors and actuators. These variables are updated every 100 ms. The reference model RM1.1 [23] of Table 2 formally defines the sensors and actuators and the corresponding variables.

The accessible sensors comprise eight infrared proximity sensors for detecting obstacles ($prox_i$) and for measuring ambient light ($light_i$), three ground sensors for sampling the grayscale color of the ground situated under the robot ($ground_i$), and a range-and-bearing board used for local communications between robots [20]. Upon reception of a message via the range-and-bearing board, an e-puck can estimate the relative distance and angle of the emitting robot. At

Table 2. Reference model RM1.1 [23]. Sensors and actuators of the extended version of the e-puck robot simulated in the experiments.

sensor/actuator	variables
proximity	$prox_i \in [0,1]$, with $i \in \{0, ..., 7\}$
light	$light_i \in [0,1]$, with $i \in \{0, ..., 7\}$
ground	$ground_i \in \{white, gray, black\}$, with $i \in \{0, ..., 2\}$
range-and-bearing	$n \in \{0, ..., 19\}$ and $V_d \in ([0, 0.7]\, m, [0, 2\pi])$
wheels	$v_l, v_r \in [-0.12, 0.12]\, m/s$

each time step, the relative distance and angle of all perceived neighbors are summed into a vector (V_d), which points to the center of mass of all detected robots. In addition to this direction vector V_d, the control software has also access to the number of perceived neighbors (n).

The control software also controls actuators: the motors of the wheels. The e-pucks are driven by the mean of a two-wheeled differential steering system. The control software dictates the displacement of the robot via two velocity variables (v_l and v_r).

Design methods. In this section, we briefly describe the three automatic design methods considered in the experiments: `EvoStick` [18], AutoMoDe-`Vanilla` [18], and AutoMoDe-`Chocolate` [17]. We refer the readers to the original papers for detailed descriptions of these methods.

`EvoStick` is an implementation of the classical evolutionary robotics setup. An evolutionary algorithm optimizes the parameters of a fully connected, feed-forward, neural network. The neural network comprises 24 input and 2 output nodes that are directly connected. The inputs and outputs are defined on the basis of the reference model RM1.1 (see Table 2). More precisely, the inputs are allocated as follows: 8 for the readings of the proximity sensors, 8 for the readings of the light sensors, 3 for the readings of the ground sensors, 1 for the number of neighbors, and 4 for the scalar projections of the vector V_d on four unit vectors distributed around the robot. The outputs of the neural network are the speed of the left and right wheels of the e-puck.

`Vanilla` produces control software in the form of probabilistic finite state machines by assembling preexisting modules. A module is either a *behavior* or a *transition*. A behavior is an action that can be performed by the robot, while a transition is a condition on the environment perceived by the robot. All modules operate on the variables presented in the reference model RM1.1 of Table 2, and some of the modules have parameters that adjust their functioning. In a probabilistic finite state machine, the transitions (i.e., the edges) regulate the succession of behaviors (i.e., states) that alternatively control the robot by determining the values of the output variables.

Similarly to `Vanilla`, `Chocolate` is a modular automatic design method. The methods differ by the optimization algorithm they use: `Vanilla` uses F-race [5,3] and `Chocolate` uses Iterated F-race [30]. In order to conceive probabilistic finite state machines, `Vanilla` and `Chocolate` have at their disposal the same set

Table 3. The two e-puck models.

sensor/actuator	M_A	M_B
proximity	$[-0.05, 0.05]$	$[-0.05, 0.05]$
light	$[-0.05, 0.05]$	$[-0.90, 0.90]$
ground	$[-0.05, 0.05]$	$[-0.05, 0.05]$
range-and-bearing	0.85	0.90
wheels	0.05	0.15

of preexisting modules: six behaviors and six transitions. In addition to the topology of the probabilistic finite state machine, `Vanilla` and `Chocolate` also tune the parameters of the modules. The design space explored by the two methods is restricted to all possible probabilistic finite state machines composed of up to four states (i.e., behaviors) and up to four edges (i.e., transitions) departing from each state. `Chocolate` has proved to outperform `Vanilla` [17].

Models. We use the two e-puck models, namely M_A and M_B, described in Table 3. In this table, the values for the proximity, light and ground sensors are the range of the uniform white noise added to the readings of the sensors. The value for the range-and-bearing sensor is the probability of failing to receive a message sent by a robot within communication range. Finally, the value for the wheels actuator is the standard deviation of Gaussian white noise added to the variables controlling the speed of the left and right wheels.

Model M_A is the same model used during the design process of the experiments ran by Francesca et al. [18]. We conceived model M_B with the purpose of obtaining performance drops that lead to a rank inversion when using model M_B as a pseudo-reality to assess the performance of control software automatically generated on the basis of model M_A.

Protocol. We consider two missions: *aggregation* and *foraging*. For each mission, we define an objective function to be maximized. The same objective function is used for both designing control software and assessing its performance. We run experiments in which the control software is designed by the three design methods described above: `EvoStick`, `Vanilla`, and `Chocolate`. We consider a homogeneous swarm composed of $N = 20$ robots operating in a dodecagonal arena for a time period of 250 s. The arena is delimited by walls and its surface is $4.91\,\mathrm{m}^2$.

For each mission, we consider two stages: S_{AB} and S_{BA}. In stage S_{AB}, each automatic design method produces control software via simulations based on model M_A; the control software is then assessed with simulations based on model M_B. To study the generalization capability of the control software produced, the performance evaluated on model M_B is compared to the one evaluated on model M_A. In stage S_{BA}, the roles of the two models are inverted: control software is produced on M_B and then assessed on M_A. Also in this case, the performance on M_A is compared to the one on M_B to study the generalization capability of

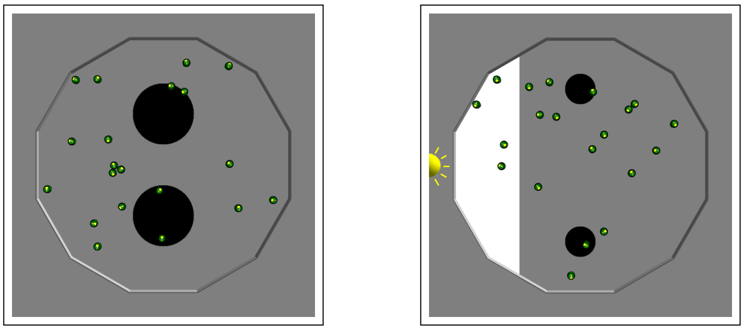

Fig. 1. Simulated environments: aggregation (left) and foraging (right).

the control software. In other terms, in stage S_{AB} the pseudo-reality is model M_B; whereas in stage S_{BA}, it is model M_A.

Each design method is run with a design budget of 200 000 simulations. For each mission and each stage S_{xy}—where by x and y we indicate A and B, or viceversa—each design method is run 20 times on model M_x and produces therefore a total of 20 instances of control software. For the assessment, each of these instances is evaluated 20 times on model M_x, and 20 times on model M_y to study their generalization capability.

We present the results by means of notched box-and-whiskers boxplots. The notches indicate the 95% confidence interval around the median. If the notches of two boxes do not overlap, the difference between their medians is significant [10]. Moreover, we aggregate the results of the two stages to estimate the performance drop experienced by each design method. For each method, we report a 95% confidence interval on the difference between the performance observed on models M_x and M_y.[1] We also highlight a lower bound D on the difference between the performance drop of `EvoStick` and `Vanilla`—confidence 95%. We focus on `EvoStick` and `Vanilla` as Francesca et al. [18] assessed their performances for the same mission in robot experiments.

4 Experiments

In this section, we provide details on the two missions considered and we report the results of our experiments. Figure 1 pictures the simulated environments in which the swarm operates. The missions have already been studied in [18]. We report in the following only the information that is strictly needed to understand the results. We refer the reader to the original article for the details.

[1] Confidence intervals are computed based on the statistic of the paired Wilcoxon signed rank test. The normal approximation is adopted as the sample size is larger than 50. The implementation used is the one of R's *stats* package [35].

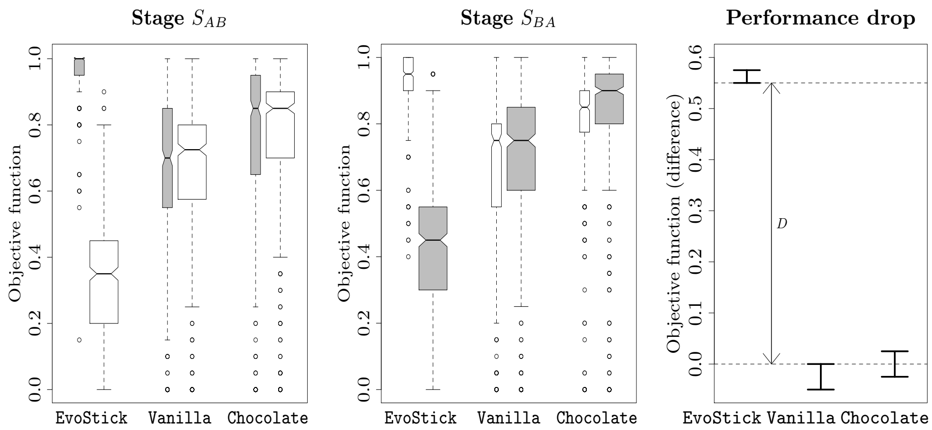

Fig. 2. Aggregation. In the left and center plots, narrow boxes represent the performance assessed on the model used during the design step; wide boxes represent the performance assessed in pseudo-reality. Gray boxes represent performance assessed on model M_A; white boxes represent performance assessed on model M_B. In the right plot, the segments represent a 95% confidence interval on the performance drop experienced by each method—aggregated across the two stages. D is a bound on the difference between the performance drop of `EvoStick` and `Vanilla`.

4.1 Aggregation

In this experiment, the swarm must aggregate on one of two black areas, named a or b. These black areas have a radius of 0.35 m. The performance of the swarm is measured via the following objective function:

$$F_{aggregation} = \max(N_a, N_b)/N, \tag{1}$$

where $N = 20$ is the total number of robots composing the swarm; and N_a and N_b are the number of robots that at the end of the experimental run are located on a and b, respectively. The objective function is maximized when, at the end of a run, all robots are either on a or on b.

The results of this experiment show a rank inversion—see Figure 2 (left and center). In each stage S_{xy}, `EvoStick` performs significantly better than both `Vanilla` and `Chocolate` when the performance of the control software they produced is assessed on model M_x. On the other hand, `EvoStick` performs significantly worse than both `Vanilla` and `Chocolate` when the performance is assessed on model M_y.

Indeed, the performance of the control software designed by `EvoStick` drops noticeably when assessed in pseudo-reality: the drop is at least 0.55 (confidence 95%). In the case of `Vanilla` and `Chocolate`, the drop is significantly smaller: at most 0.00 and 0.02 respectively (confidence 95%). See Figure 2 (right).

In both stages, the rank inversion between `EvoStick` and `Vanilla` is similar to the one observed by Francesca et al. [18] on the same mission. At least in

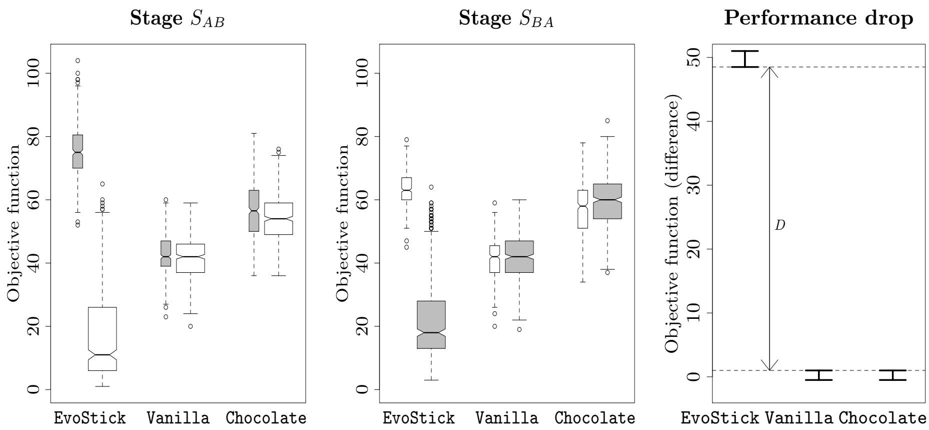

Fig. 3. Foraging. See caption of Figure 2 for the explanation of width and color of boxes.

this experiment, the artificial reality gap we created with the models M_A and M_B was able to qualitatively predict performance drop and rank inversion for `EvoStick` and `Vanilla`.

4.2 Foraging

In this experiment, the swarm must perform an idealized form of foraging. We consider that an individual robot has retrieved an object when it enters the nest after having visited a foraging source. Two sources are available, and are represented by black circular areas of radius 0.15 m. The nest is represented by a white area situated at a distance of 0.45 m from the two black areas. A light source is placed behind the nest to help the robots locate it.

The performance of the swarm is measured by the number of objects retrieved during the whole experimental run. It is computed via the following objective function:

$$F_{foraging} = N_o, \tag{2}$$

where N_o is the total number of objects retrieved.

Also in this experiment, we observe a rank inversion—see Figure 3 (left and center). `EvoStick` performs significantly better than `Vanilla` and `Chocolate` when the performance of the control software produced is assessed on model M_x, but significantly worse when the performance is assessed on model M_y.

When assessed in pseudo-reality, the performance of the control software designed by `EvoStick` drops by at least 48 objects (confidence 95%), whereas in the case of `Vanilla` and `Chocolate`, the drop is at most of 1 object (confidence 95%). See Figure 3 (right).

Also on this mission, the rank inversion between `EvoStick` and `Vanilla` is similar to the one observed by Francesca et al. [18]: our artificial reality gap yields good qualitative predictions.

5 Conclusion

With this article, we shed further light on the reality gap. Specifically, we investigated how and under what conditions the effects of the reality gap manifest. We contend that, for the effects of the reality gap to manifest, it is unnecessary to assume that the control software is assessed under context/conditions that are more complex than those experienced in the design.

To substantiate our contention, we conceived a set of simulation-only experiments in which we created an artificial reality gap based on two robot models M_A and M_B. We used M_A for the design and M_B for the assessment; we then inverted the role of the two models. In both cases, we observed performance drop and rank inversion: a design method performed significantly better than the others when the control software they produced was assessed on the same model used in the design, but significantly worse on the other one. Having observed performance drop and rank inversion both when (i) designing on M_A and assessing on M_B, and when (ii) designing on M_B and assessing on M_A, we can exclude that the effects of the reality gap emerge only due to the fact that the design is performed on a simplistic model that fails to reproduce the complexity of the environment in which the final assessment is performed.

Furthermore, our results indicate that simulation-only experiments could be used to tell whether and to what extent automatic design methods are prone to performance drop and rank inversion. This might have useful practical implications. Indeed, we foresee that an artificial, simulation-only reality gap could be used to *validate* automatically-generated control software and to predict its real-world performance. We have in mind here a development process that mimics the classical machine learning procedure based on training, validation, and test set. We imagine a development process in which control software is generated using a model, validated on another model to predict its ability to cross the reality gap, and eventually tested in the real world.

Future work will be dedicated to producing evidence that the artificial reality gap considered in our experiments can reliably predict real-world performance. Moreover, future work should be dedicated to defining reliable and meaningful ways to generate a pair of models that can properly serve as an artificial reality gap. In this work, we considered two models that differ in the noise level. Other differences between the models could be considered, which could be more appropriate. Finally, future research should be dedicated to quantifying the difference between two models. A quantity measuring the difference between two models could be used to characterize the severity of the artificial reality gap associated with them.

Acknowledgements. The project has received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (grant agreement No 681872). Mauro Birattari acknowledges support from the Belgian *Fonds de la Recherche Scientifique* – FNRS.

References

1. Beni, G.: From swarm intelligence to swarm robotics. In: Şahin, E., Spears, W.M. (eds.) Swarm Robotics, SAB 2004 International Workshop. LNCS, vol. 3342, pp. 1–9. Springer, Berlin Heidelberg, Germany (2004)
2. Berman, S., Kumar, V., Nagpal, R.: Design of control policies for spatially inhomogeneous robot swarms with application to commercial pollination. In: Zexiang, L. (ed.) 2011 IEEE International Conference Robotics and Automation (ICRA). pp. 378–385. IEEE Press, Piscataway, NJ (2011)
3. Birattari, M.: Tuning Metaheuristics: A Machine Learning Perspective. Springer, Berlin Heidelberg, Germany (2009)
4. Birattari, M., Delhaisse, B., Francesca, G., Kerdoncuff, Y.: Observing the effects of overdesign in the automatic design of control software for robot swarms. In: Dorigo, M., et al. (eds.) Swarm Intelligence, 10th International Conference, ANTS 2016. LNCS, vol. 9882, pp. 45–57. Springer, Cham, Switzerland (2016)
5. Birattari, M., Stützle, T., Paquete, L., Varrentrapp, K.: A racing algorithm for configuring metaheuristics. In: Langdon, W., et al. (eds.) GECCO 2002: Proceedings of the Genetic and Evolutionary Computation Conference. pp. 11–18. Morgan Kaufmann, San Francisco, CA (2002)
6. Bongard, J., Lipson, H.: Once more unto the breach: co-evolving a robot and its simulator. In: Pollack, J., et al. (eds.) Artificial Life IX: Proceedings of the Conference on the Simulation and Synthesis of Living Systems. pp. 57–62 (2004)
7. Brambilla, M., Brutschy, A., Dorigo, M., Birattari, M.: Property-driven design for swarm robotics: A design method based on prescriptive modeling and model checking. ACM Transactions on Autonomous and Adaptive Systems 9(4), 17.1–28 (2015)
8. Bredeche, N., Montanier, J.M., Liu, W., Winfield, A.F.: Environment-driven distributed evolutionary adaptation in a population of autonomous robotic agents. Mathematical and Computer Modelling of Dynamical Systems 18(1), 101–129 (2012)
9. Brooks, R.: Artificial life and real robots. In: Varela, F.J., Bourgine, P. (eds.) Towards a Practice of Autonomous Systems. Proceedings of the First European Conference on Artificial Life. pp. 3–10. MIT Press, Cambridge, MA (1992)
10. Chambers, J., Cleveland, W., Kleiner, B., Tukey, P.: Graphical Methods For Data Analysis. Wadsworth, Belmont, CA (1983)
11. Di Mario, E., Martinoli, A.: Distributed particle swarm optimization for limited-time adaptation with real robots. Robotica 32(02), 193–208 (2014)
12. Dorigo, M., Birattari, M., Brambilla, M.: Swarm robotics. Scholarpedia 9(1), 1463 (2014)
13. Floreano, D., Urzelai, J.: Evolution of plastic control networks. Autonomous Robots 11(3), 311–317 (2001)
14. Floreano, D., Husbands, P., Nolfi, S.: Evolutionary robotics. Handbook of Robotics pp. 1423–1451 (2008)
15. Floreano, D., Mondada, F.: Evolution of plastic neurocontrollers for situated agents. In: Maes, P., et al. (eds.) From animals to animats 4: Proceedings of the International Conference on Simulation of Adaptive Behavior. ETH Zurich, Switzerland (1996)
16. Francesca, G., Birattari, M.: Automatic design of robot swarms: achievements and challenges. Frontiers in Robotics and AI 3(29), 1–9 (2016)

17. Francesca, G., Brambilla, M., Brutschy, A., Garattoni, L., Miletitch, R., Podevijn, G., Reina, A., Soleymani, T., Salvaro, M., Pinciroli, C., Birattari, M.: AutoMoDe-Chocolate: automatic design of control software for robot swarms. Swarm Intelligence 9(2/3), 125–152 (2015)
18. 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)
19. Garattoni, L., Francesca, G., Brutschy, A., Pinciroli, C., Birattari, M.: Software infrastructure for e-puck (and TAM). Tech. Rep. TR/IRIDIA/2015-004, IRIDIA, Université Libre de Bruxelles, Belgium (2015)
20. Gutiérrez, Á., Campo, A., Dorigo, M., Donate, J., Monasterio-Huelin, F., Magdalena, L.: Open e-puck range & bearing miniaturized board for local communication in swarm robotics. In: Kosuge, K. (ed.) 2009 IEEE International Conference on Robotics and Automation (ICRA). pp. 3111–3116. IEEE Press, Piscataway, NJ (2009)
21. Haasdijk, E., Bredeche, N., Eiben, A.: Combining environment-driven adaptation and task-driven optimisation in evolutionary robotics. PloS one 9(6), e98466 (2014)
22. Hamann, H., Wörn, H.: A framework of space–time continuous models for algorithm design in swarm robotics. Swarm Intelligence 2(2–4), 209–239 (2008)
23. Hasselmann, K., Ligot, A., Francesca, G., Birattari, M.: Reference models for AutoMoDe. Tech. Rep. TR/IRIDIA/2018-002, IRIDIA, Université libre de Bruxelles, Belgium (2018)
24. Jakobi, N.: Evolutionary robotics and the radical envelope-of-noise hypothesis. Adaptive Behavior 6(2), 325–368 (1997)
25. Jakobi, N.: Minimal simulations for evolutionary robotics. Ph.D. thesis, University of Sussex, Falmer, UK (1998)
26. Jakobi, N., Husbands, P., Harvey, I.: Noise and the reality gap: the use of simulation in evolutionary robotics. In: Morán, F., et al. (eds.) Advances in Artificial Life. LNCS, vol. 929, pp. 704–720. Springer, London (1995)
27. König, L., Mostaghim, S.: Decentralized evolution of robotic behavior using finite state machines. International Journal of Intelligent Computing and Cybernetics 2(4), 695–723 (2009)
28. Koos, S., Mouret, J.B., Doncieux, S.: The transferability approach: crossing the reality gap in evolutionary robotics. IEEE Transactions on Evolutionary Computation 17(1), 122–145 (2013)
29. Lee, J.B., Arkin, R.C.: Adaptive multi-robot behavior via learning momentum. In: George Lee, C.S. (ed.) IEEE/RSJ International Conference on Intelligent Robots and Systems – IROS. pp. 2029–2036. IEEE Press, Piscataway, NJ (2003)
30. López-Ibáñez, M., Dubois-Lacoste, J., Pérez Cáceres, L., Birattari, M., Stützle, T.: The irace package: Iterated racing for automatic algorithm configuration. Operations Research Perspectives 3, 43–58 (2016)
31. Miglino, O., Lund, H., Nolfi, S.: Evolving mobile robots in simulated and real environments. Artificial life 2(4), 417–434 (1995)
32. Mondada, F., Bonani, M., Raemy, X., Pugh, J., Cianci, C., Klaptocz, A., Magnenat, S., Zufferey, J.C., Floreano, D., Martinoli, A.: The e-puck, a robot designed for education in engineering. In: Gonçalves, P., Torres, P., Alves, C. (eds.) Proceedings of the 9th Conference on Autonomous Robot Systems and Competitions. pp. 59–65. Instituto Politécnico de Castelo Branco, Portugal (2009)
33. Nolfi, S., Floreano, D., Miglino, G., Mondada, F.: How to evolve autonomous robots: different approaches in evolutionary robotics. In: Brooks, R.A., Maes, P.

(eds.) Artificial Life IV: Proceedings of the Workshop on the Synthesis and Simulation of Living Systems. pp. 190–197. MIT Press, Cambridge, MA (1994)
34. 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., Dorigo, M.: ARGoS: a modular, parallel, multi-engine simulator for multi-robot systems. Swarm Intelligence 6(4), 271–295 (2012)
35. R Development Core Team: R: A Language and Environment for Statistical Computing. R Foundation for Statistical Computing, Vienna, Austria (2008), `http://www.R-project.org`
36. Reina, A., Valentini, G., Fernández-Oto, C., Dorigo, M., Trianni, V.: A design pattern for decentralised decision making. PloS one 10(10), e0140950 (2015)
37. Şahin, E.: Swarm robotics: From sources of inspiration to domains of application. In: Şahin, E., Spears, W.M. (eds.) Swarm Robotics, SAB 2004 International Workshop. LNCS, vol. 3342, pp. 10–20. Springer, Berlin Heidelberg, Germany (2004)
38. Silva, F., Duarte, M., Correia, L., Oliveira, S., Christensen, A.: Open issues in evolutionary robotics. Evolutionary Computation 24(2), 205–236 (2016)
39. Silva, F., Urbano, P., Correia, L., Christensen, A.L.: odNEAT: An algorithm for decentralised online evolution of robotic controllers. Evolutionary Computation 23(3), 421–449 (2015)
40. Urzelai, J., Floreano, D.: Evolutionary robotics: coping with environmental change. In: Whitney, L.D., et al. (eds.) GECCO 2000: Proceedings of Conference on the Genetic and Evolutionary Computation Conference. pp. 941–948. Morgan Kaufmann, San Francisco, CA (2000)
41. Watson, R., Ficici, S., Pollack, J.: Embodied evolution: Distributing an evolutionary algorithm in a population of robots. Robotics and Autonomous Systems 39(1), 1–18 (2002)
42. Zagal, J.C., Ruiz-Del-Solar, J.: Combining simulation and reality in evolutionary robotics. Journal of Intelligent & Robotic Systems 50(1), 19–39 (2007)