







AutoMoDe-Arlequin: Neural Networks as Behavioral Modules for the Automatic Design of Probabilistic Finite-State Machines

Antoine Ligot , Ken Hasselmann , and Mauro Birattari  

IRIDIA, Université libre de Bruxelles, Brussels, Belgium
mbiro@ulb.ac.be

Abstract. We present *Arlequin*, an off-line automatic design method that produces control software for robot swarms by combining behavioral neural-network modules generated via neuro-evolution. The neural-network modules are automatically generated once, in a mission-agnostic way, and are then automatically assembled into probabilistic finite-state machines to perform various missions. With *Arlequin*, our goal is to reduce the amount of human intervention that is required for the implementation or the operation of previously published modular design methods. Simultaneously, we assess whether neuro-evolution can be used in a modular design method to produce control software that crosses the reality gap satisfactorily. We present robot experiments in which we compare *Arlequin* with *Chocolate*, a state of the art modular design method, and *EvoStick*, a traditional neuro-evolutionary swarm robotics method. The preliminary results suggest that automatically combining neural-network modules into probabilistic finite-state machines is a promising approach to the automatic conception of control software for robot swarms.

1 Introduction

Swarm robotics is an approach to controlling groups of autonomous robots [13]. A robot swarm is a decentralized system in which individual robots do not have predefined roles and act solely based on the local information collected through their sensors or shared by nearby peers. A collective behavior in a swarm emerges from the interactions between the robots, and between the robots and the environment. These interactions depend on how the system evolves and are therefore unknown at design time. Designing the individual behavior of the robots to obtain the desired collective behavior is a challenging task as there is no general methodology to do so [8].

For specific missions in specific cases, experts can use principled manual design methods to obtain the desired collective behavior [1, 2, 7, 25, 26, 31, 37, 42, 44]. In the general case, however, experts usually proceed by trial and error. An alternative to manual design exists: optimization-based design, which consists in searching among a set of possible individual behaviors the one that maximizes a

mission-dependent objective function that measures the performance of the swarm. These methods can be classified as online or offline [9, 18]: in the first case, the optimization is performed while the robots operate in the environment; in the second one, it is performed before deployment, typically using computer simulations. The work presented in this paper belongs in offline automatic design.

A popular approach to the offline automatic design of robot swarms is neuro-evolutionary swarm robotics [47], in which individual behaviors are artificial neural networks whose weights, and possibly their topologies, are fine-tuned by an evolutionary algorithm [5, 12, 18, 46, 48]. Unfortunately, neuro-evolutionary swarm robotics suffers from a major drawback: it typically does not cope well with the so-called *reality gap*, that is, the intrinsic difference between simulation and reality [10, 30, 46]. As a result, the performance of the generated control software is likely deceiving in reality and drops significantly with respect to the one observed in simulation [16, 41]. Despite the effort made to handle the reality gap [6, 17, 28–30, 32, 39], none of the ideas explored so far appears to be the ultimate solution [18, 35, 46]. Other approaches to the offline automatic design of robot swarms, based on modularity, have been proposed: they generate control software by assembling low-level behavioral modules [14, 15, 20]. In this paper, we present a novel automatic modular design method: *Arlequin*. This method belongs to the AutoMoDe family [19, 20, 27, 33, 45]. The novelty of *Arlequin* is that, contrarily to the previous instances of AutoMoDe that automatically combine behavioral modules conceived by hand, it automatically combines behavioral modules that were themselves automatically generated *a priori* via neuro-evolutionary swarm robotics. With *Arlequin*, our goal is two-fold: (i) to conceive a method that requires less human expertise during its implementation than the current instances of AutoMoDe, and (ii) further corroborate the conjecture of Francesca et al. [20] that lead to the creation of AutoMoDe.

Francesca et al. [20] conjectured that the reality gap problem faced in evolutionary swarm robotics bears a resemblance to the generalization problem of machine learning, and that the performance drop observed when porting control software to physical robots is due to a sort of *overfitting* of the conditions experienced during the design. According to the *bias/variance tradeoff* [22, 49], the expected generalization error of a learning algorithm can be decomposed into a bias and a variance factor. High-complexity learning algorithms have high variance and low bias, whereas low-complexity ones have low variance and high bias. For an increasing level of complexity, the generalization error typically first decreases then increases again. To minimize the generalization error, one must find the optimal level of complexity of the learning algorithm. Based on this reasoning, Francesca et al. conjectured that the difficulty of evolutionary swarm robotics to cross the reality gap is due to an excessively high representational power that entails a sort of overfitting of the idiosyncrasies of simulation [16, 41]. The authors therefore created AutoMoDe to have a higher bias than the neuro-evolutionary approaches in order to decrease the representational capability of the control architecture, and to hopefully reduce the performance drop experienced. In AutoMoDe, the bias is injected by restricting to the control software

to be a combination of pre-existing modules. So far, the empirical evidence indicates that manually conceiving modules in simulation and validating them on physical robots can effectively limit the overall performance drop caused by the module level. With **Arlequin**, we investigate whether the principles of modularity also hold true when the behavioral modules are automatically generated by a neuro-evolutionary method. That is, we investigate whether the bias injected by restricting the control software produced to be combination of neural-network modules is enough to cross the reality gap satisfactorily.

We created **Arlequin** to be similar in many aspects to **Chocolate**, a previously presented instance of AutoMoDe [19]. Indeed, the two methods only differ in the behavioral modules used. We did so to single out the aspect we wish to investigate: the relative advantages and disadvantages of generating behavioral modules automatically. **Chocolate** has at his disposal six hand-coded behavioral modules, which are replaced by six neural-network modules in **Arlequin**. To generate these neural-network modules, we inferred an objective function describing each of the six hand-coded behavioral modules of **Chocolate**, and fed these objective functions to a neuro-evolutionary design method called **EvoStick** [20]. Similarly to the modules of **Chocolate**, the neural-network modules are generated once, independently of the specific missions **Arlequin** will then solve. We evaluate the performance of **Arlequin** on two missions involving 20 e-puck robots. To assess whether the conjecture of Francesca et al. on the bias/variance tradeoff also holds true when the predefined behavioral modules are generated automatically via neuro-evolution, we compare the performance of **Arlequin** with the ones of **EvoStick** and **Chocolate**.

2 AutoMoDe-Arlequin

Arlequin generates control software for a version of the e-puck [40]—a small, circular, two-wheeled robot—equipped with a range-and-bearing board [24], a ground sensor module, and an Overo Gumstix board [21]. We considered a subset of the capabilities of the robot. In particular, the control software that can be generated has access to the ground sensor module to detect the color of the ground situated below the robot (i.e., black, gray, or white); the infrared sensor module to detect the presence of nearby obstacles and of a light source; the range-and-bearing module to detect the presence of peers within a range of approximately 0.7 m and to infer a vector V_d indicating their direction of attraction; and the wheels actuators to move the robot.

Arlequin generates control software by automatically combining predefined modules into probabilistic finite-state machines. The modules comprise six low-level behaviors (i.e., simple actions performed by the robot) and six conditions (i.e., situations experienced by the robot). The low-level behaviors are associated to states of the probabilistic finite-state machine, whereas the conditions are associated to transitions. The low-level behavior associated with the active state is executed as long as the conditions associated with all its outgoing transitions are evaluated as false. Once a condition associated with an outgoing transition

is evaluated as true, the active state is updated and the corresponding low-level behavior is executed. **Arlequin** has many commonalities with **AutoMoDe-Chocolate** [19]. The two methods adopt *irace* [4, 38] as optimization algorithm to select and combine the different modules into a probabilistic finite-state machine. The two methods also impose the same constraints on the probabilistic finite-state machines produced: they can comport up to four states with up to four outgoing transitions per states. Finally, **Arlequin** and **Chocolate** have at their disposal the same hand-coded condition modules. We refer the reader to the original description of these conditions [20].

Arlequin and **Chocolate** differ in the predefined behavioral modules adopted: **Chocolate** combines hand-coded parametric modules, whereas **Arlequin** combines neural-network modules generated by **EvoStick** [20]. **EvoStick** is a relatively simple implementation of the classical neuro-evolutionary robotics approach: it generates control software in the form of neural networks whose synaptic weights are obtained via an evolutionary process. In **EvoStick**, the produced neural networks are fully connected, do not contain hidden layers, and have 25 input and 2 output nodes. The neural networks are therefore characterized by a total of 50 parameters, each being a real value in $[-5, 5]$. The 25 input nodes are organized as follows: 3 are dedicated to the readings of the ground sensors, 8 to the readings of the proximity sensors, 8 to the readings of the light sensors, 5 to the readings of the range-and-bearing sensors (4 for the scalar projections of the vector V_d pointing to the neighboring peers on four unit vectors, and 1 for the number of detected robots), and one serves as bias. The 2 output nodes control the speed of the left and right wheels of the robot. **EvoStick** uses populations of 100 individuals and evaluates each individual 10 times per generation.

To obtain behaviors that are similar to the six hand-coded low-level behaviors of **Chocolate** via neuro-evolution, we inferred an objective function for each of them. We fed these objective functions to **EvoStick** to generate control software for a swarm of 20 simulated e-puck, and considered simulation runs of 120 s. For each of the low-level behaviors, **EvoStick** generated 10 instances of control software. We then evaluated each instance of control software 20 times in simulation using different initial conditions, and selected the ones with the highest average performance to be used as low-level behaviors for **Arlequin**. The design budget allocated to **EvoStick** is 20 000 execution runs, which corresponds to 20 generations. The six hand-coded low-level behaviors of **Chocolate** and the corresponding objective functions we devised to obtain the automatically generated modules of **Arlequin** are described in Sect. 2.1.

2.1 Low-Level Behaviors

Exploration: In **Chocolate**, the robot moves straight until an obstacle is perceived by its front proximity sensors, then turns on the spot for a random number of steps drawn in $\{0, 1, \dots, \pi\}$. The parameter $\pi \in \{0, 1, \dots, 100\}$ is meant to be afterwards tuned by the optimization algorithm on a per-mission basis. In **Arlequin**, the environment is discretized into a two-dimensional grid G , and the

objective function considered rewards the number of cells visited individually. The objective function, to be maximized, is $\sum_{r=1}^N \sum_{i=1}^X \sum_{j=1}^Y G_r[i][j]$, where $G_r[i][j] = 1$ if robot r visited cell $G_r(i, j)$ at least once, 0 otherwise; N is the number of robots in the swarm; and $X = 20$ and $Y = 20$ are the numbers of rows and columns in grid G , respectively. **Stop:** In *Chocolate*, the robot does not move. In *Arlequin*, the objective function penalizes the displacement of the individual robots. The objective function, to be minimized, is $\sum_{t=1}^T \sum_r^N \|P_r(t) - P_r(t-1)\|$, where $P_r(t)$ is the position of robot r at time t , and T is the duration of the experimental run. **Phototaxis:** In *Chocolate*, the robot moves towards the light, if perceived. Otherwise, the robot moves straight. In *Arlequin*, the objective function penalizes the distance between the individual robots and the light. The objective function, to be minimized, is $\sum_{t=1}^T \sum_{r=1}^N \|P_r(t) - P_{light}\|$, where $P_r(t)$ and P_{light} are the positions of robot r at time t and of the light, respectively. **Anti-phototaxis:** In *Chocolate*, the robot moves away from the light, if perceived. Otherwise, the robot moves straight. In *Arlequin*, the objective function rewards the distance between the individual robots and the light. The objective function, to be maximized, is $\sum_{t=1}^T \sum_{r=1}^N \|P_r(t) - P_{light}\|$, where $P_r(t)$ and P_{light} are the positions of robot r at time t and of the light, respectively. **Attraction:** In *Chocolate*, the robot moves towards the neighboring peers (V_d), if perceived. Otherwise, the robot moves straight. A parameter $\alpha \in [1, 5]$ controls the speed of convergence towards the detected peers and is meant to be afterwards tuned by the optimization algorithm on a per-mission basis. In *Arlequin*, the objective function penalizes the distance between each pair of robots within the swarm. The objective function, to be minimized, is $\sum_{t=1}^T \sum_{i=1}^{N-1} \sum_{j=i+1}^N \|P_i(t) - P_j(t)\|$, where $P_i(t)$ and $P_j(t)$ are the positions of robot i and j , respectively. **Repulsion:** In *Chocolate*, the robot moves away from the neighboring peers, if perceived. Otherwise, it moves straight. A parameter $\alpha \in [1, 5]$ controls the speed of divergence and is meant to be afterwards tuned by the optimization algorithm on a per-mission basis. In *Arlequin*, the objective function rewards, for each individual robot, the distance from its closest peer. The objective function, to be minimized, is $\sum_{t=1}^T \sum_{r=1}^N \|P_r(t) - P_{r_{min}}(t)\|$, where $P_r(t)$ is the position of robot r and $P_{r_{min}}(t)$ is the one of the robot closest to robot r at time t .

3 Experiments

We generated control software with *Arlequin*, *Chocolate*, and *EvoStick* for two missions: FORAGING and AGGREGATION-XOR [20]. We considered a swarm of 20 e-puck robots that operate in a dodecagonal arena of 4.91 m² delimited by walls. For each mission, the design budget allowed to each method is 200 000 simulation runs. For each mission, we executed each design method 10 times and collected the best instance of control software produced by each execution. We assessed the performance of each instance of control software twice: once in simulation and once on physical robots [3]. We present the results in the form of notched boxplots: the notches represent the 95% confidence interval on the position of the median. If the notches of two boxes do not overlap, the

difference between the respective medians is significant [11]. All simulation runs were performed with ARGoS [43], which allowed us to directly port the control software generated to the physical robots without any modifications. All the control software generated, the raw data collected, and the experimental runs recorded are available online as supplementary material [36] (Fig. 1).

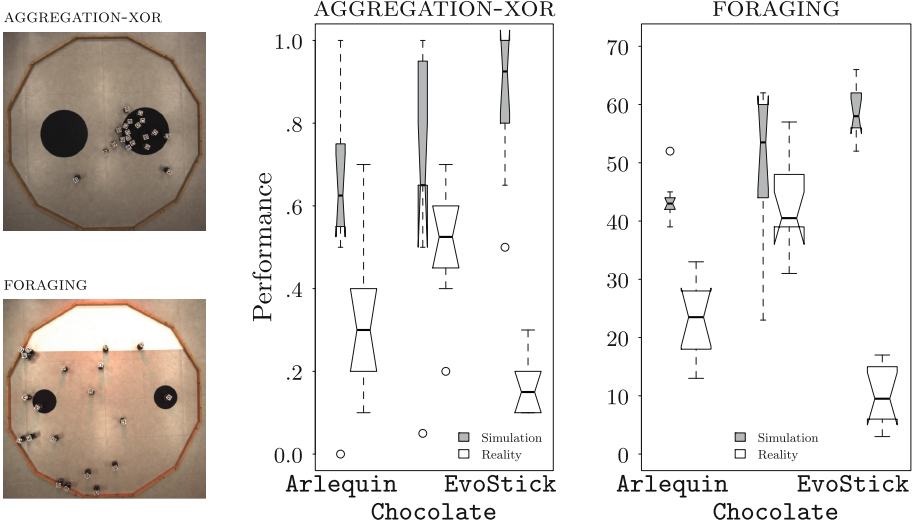


Fig. 1. The arenas and the results of the experiments.

AGGREGATION-XOR. The robots must aggregate on one of the two black areas. After 180s, the performance measured by the function $F_A = \max(N_l, N_r)/N$, where N_l and N_r are the number of robots located on each of the two black area; and N is the total number of robots. In simulation, **Arlequin** and **Chocolate** show similar performance, but **Arlequin** is outperformed by **EvoStick**. In reality, **Arlequin** and **EvoStick** suffer from a significant performance drop, with **EvoStick** suffering from the reality gap the most. Indeed, the drop experienced by **Arlequin** is at most 0.48, whereas the one experienced by **EvoStick** is at least 0.55, which makes the performance drop experienced by **Arlequin** significantly lower than the one experienced by **EvoStick** (95% confidence computed with a paired Wilcoxon test). **Chocolate** shows similar performance in simulation and in reality. The performance drop experienced by the three methods when crossing the reality gap is such that, in reality, **Arlequin** outperforms **EvoStick**, but is outperformed by **Chocolate**.

FORAGING. The robots must retrieve objects from two source areas (black circles) and deposit them in a nest (white area). The objects are virtual: a robot is deemed to carry an object after it enters one of the source areas and to retrieve the object when it then enters the nest. A light source is placed behind the nest.

The performance measured by the function $F_F = N_o$, where N_o is the total number of objects retrieved after 180 s. In simulation, **Arlequin** is outperformed by **EvoStick** and **Chocolate**. In reality, the three methods suffer from a significant performance drop, with **EvoStick** suffering the most, followed by **Arlequin**, then **Chocolate**. The drop of **Arlequin** is at most 26, whereas the one of **EvoStick** is at least 42, which makes the drop of **Arlequin** significantly lower than the one of **EvoStick** (95% confidence computed with a paired Wilcoxon test). As a result, **Arlequin** outperforms **EvoStick**, but is outperformed by **Chocolate**.

4 Conclusions

We presented **Arlequin**, a novel instance of AutoMoDe that differs from the previously presented ones by the nature of the predefined behavioral modules to be combined: **Arlequin** uses neural network modules generated via neuro-evolution, whereas the others use hand-coded ones. The behavioral modules of **Arlequin** were generated via **EvoStick**, a neuro-evolutionary method. We compared the performance of the control software generated by **Arlequin** with the one of **EvoStick** and **Chocolate** on two missions. In both missions, the control software produced by **Arlequin** suffered from a significant performance drop. However, the control software generated by **EvoStick** suffered from a significantly larger drop than the one produced by **Arlequin**, and as a result, **Arlequin** outperformed **EvoStick** in reality. This corroborates the conjecture of Francesca et al. [20]: restricting the control software to be a combination of low-level, simple behaviors yields better results in reality than the traditional neuro-evolutionary approach, despite being the other way around in simulation. Our results show that this holds true also when the low-level behaviors are neural networks.

Future work will explore different ways of generating and selecting the pool of modules to be combined into probabilistic finite-state machines (i.e., select the modules on the basis of their performance assessed in *pseudo-reality* [34,35] or on physical robots, generate them with the *transferability approach* [32]). Future work will also be dedicated to further reducing the human expertise required during the implementation of **Arlequin**. Recently, Gomes and Christensen [23] proposed an approach to conceive low-level behaviors in a completely automated fashion. Their approach is based on *repertoires* of behaviors obtained in a task-agnostic fashion with a diversity algorithm. We wish to investigate how one could automatically produce control software for swarm robotics by combining behavioral modules selected from these repertoires.

Acknowledgements. The experiments were conceived by the three authors and performed by AL and KH. The article was drafted by AL and revised by the three authors. The research was directed by MB.

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). MB acknowledges support from the Belgian *Fonds de la Recherche Scientifique* – FNRS.

References

1. Beal, J., Dulman, S., Usbeck, K., Viroli, M., Correll, N.: Organizing the aggregate: languages for spatial computing. In: Marjan, M. (ed.) *Formal and Practical Aspects of Domain-Specific Languages: Recent Developments*, pp. 436–501. IGI Global, Hershey (2012). <https://doi.org/10.4018/978-1-4666-2092-6.ch016>
2. Berman, S., Kumar, V., Nagpal, R.: Design of control policies for spatially inhomogeneous robot swarms with application to commercial pollination. In: *IEEE International Conference on Robotics and Automation, ICRA, Piscataway, NJ, USA*, pp. 378–385. IEEE (2011). <https://doi.org/10.1109/ICRA.2011.5980440>
3. Birattari, M.: On the estimation of the expected performance of a metaheuristic on a class of instances. How many instances, how many runs? Technical report TR/IRIDIA/2004-01, IRIDIA, Université libre de Bruxelles, Belgium (2004)
4. Birattari, M., Yuan, Z., Balaprakash, P., Stützle, T.: **F-race** and iterated **F-race**: an overview. In: Bartz-Beielstein, T., Chiarandini, M., Paquete, L., Preuss, M. (eds.) *Experimental Methods for the Analysis of Optimization Algorithms*, pp. 311–336. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-02538-9_13
5. Bongard, J.C.: Evolutionary robotics. *Commun. ACM* **56**(8), 74–83 (2013)
6. Bongard, J.C., Lipson, H.: Once more unto the breach: co-evolving a robot and its simulator. In: Pollack, J.B., Bedau, M.A., Husbands, P., Watson, R.A., Ikegami, T. (eds.) *Artificial Life IX: Proceedings of the Conference on the Simulation and Synthesis of Living Systems*, pp. 57–62. MIT Press, Cambridge (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 Trans. Auton. Adapt. Syst.* **9**(4), 17:1–17:28 (2014). <https://doi.org/10.1145/2700318>
8. Brambilla, M., Ferrante, E., Birattari, M., Dorigo, M.: Swarm robotics: a review from the swarm engineering perspective. *Swarm Intell.* **7**(1), 1–41 (2013). <https://doi.org/10.1007/s11721-012-0075-2>
9. Bredeche, N., Haasdijk, E., Prieto, A.: Embodied evolution in collective robotics: a review. *Front. Robot. AI* **5**, 12 (2018). <https://doi.org/10.3389/frobt.2018.00012>
10. Brooks, R.A.: 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 (1992)
11. Chambers, J.M., Cleveland, W.S., Kleiner, B., Tukey, P.A.: *Graphical Methods For Data Analysis*. CRC Press, Belmont (1983)
12. Doncieux, S., Mouret, J.-B.: Beyond black-box optimization: a review of selective pressures for evolutionary robotics. *Evol. Intell.* **7**(2), 71–93 (2014). <https://doi.org/10.1007/s12065-014-0110-x>
13. Dorigo, M., Birattari, M., Brambilla, M.: Swarm robotics. *Scholarpedia* **9**(1), 1463 (2014). <https://doi.org/10.4249/scholarpedia.1463>
14. Duarte, M., et al.: Evolution of collective behaviors for a real swarm of aquatic surface robots. *Plos One* **11**(3), e0151834 (2016). <https://doi.org/10.1371/journal.pone.0151834>
15. Duarte, M., Oliveira, S.M., Christensen, A.L.: Evolution of hierarchical controllers for multirobot systems. In: Sayama, H., Rieffel, J., Risi, S., Doursat, R., Lipson, H. (eds.) *Artificial Life 14. Proceedings of the Fourteenth International Conference on the Synthesis and Simulation of Living Systems*, pp. 657–664. MIT Press, Cambridge (2014). <https://doi.org/10.7551/978-0-262-32621-6-ch105>

16. Floreano, D., Husbands, P., Nolfi, S.: Evolutionary robotics. In: Siciliano, B., Khatib, O. (eds.) *Springer Handbook of Robotics*, pp. 1423–1451. Springer, Heidelberg (2008). <https://doi.org/10.1007/978-3-540-30301-5.62>
17. Floreano, D., Mondada, F.: Evolution of plastic neurocontrollers for situated agents. In: Maes, P., Matarić, M.J., Meyer, J.A., Pollack, J.B., Wilson, S.W. (eds.) *From Animals to Animats 4: Proceedings of the Fourth International Conference on Simulation of Adaptive Behavior (SAB)*, pp. 402–410. MIT Press, Cambridge (1996)
18. Francesca, G., Birattari, M.: Automatic design of robot swarms: achievements and challenges. *Front. Robot. AI* **3**(29), 1–9 (2016). <https://doi.org/10.3389/frobt.2016.00029>
19. Francesca, G., et al.: AutoMoDe-chocolate: automatic design of control software for robot swarms. *Swarm Intell.* **9**(2–3), 125–152 (2015). <https://doi.org/10.1007/s11721-015-0107-9>
20. 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 Intell.* **8**(2), 89–112 (2014). <https://doi.org/10.1007/s11721-014-0092-4>
21. Garattoni, L., Francesca, G., Brutschy, A., Pinciroli, C., Birattari, M.: Software infrastructure for e-puck (and TAM). Technical report TR/IRIDIA/2015-004, IRIDIA, Université libre de Bruxelles, Belgium (2015)
22. Geman, S., Bienenstock, E., Doursat, R.: Neural networks and the bias/variance dilemma. *Neural Comput.* **4**(1), 1–58 (1992). <https://doi.org/10.1162/neco.1992.4.1.1>
23. Gomes, J., Christensen, A.L.: Task-agnostic evolution of diverse repertoires of swarm behaviours. In: Dorigo, M., Birattari, M., Blum, C., Christensen, A.L., Reina, A., Trianni, V. (eds.) *ANTS 2018. LNCS*, vol. 11172, pp. 225–238. Springer, Cham (2018). <https://doi.org/10.1007/978-3-030-00533-7.18>
24. 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.) *IEEE International Conference on Robotics and Automation, ICRA, Piscataway, NJ, USA*, pp. 3111–3116. IEEE (2009). <https://doi.org/10.1109/ROBOT.2009.5152456>
25. Hamann, H.: *Swarm Robotics: A Formal Approach*. Springer, Cham (2018). <https://doi.org/10.1007/978-3-319-74528-2>
26. Hamann, H., Wörn, H.: A framework of space-time continuous models for algorithm design in swarm robotics. *Swarm Intell.* **2**(2–4), 209–239 (2008). <https://doi.org/10.1007/s11721-008-0015-3>
27. Hasselmann, K., Robert, F., Birattari, M.: Automatic design of communication-based behaviors for robot swarms. In: Dorigo, M., Birattari, M., Blum, C., Christensen, A.L., Reina, A., Trianni, V. (eds.) *ANTS 2018. LNCS*, vol. 11172, pp. 16–29. Springer, Cham (2018). <https://doi.org/10.1007/978-3-030-00533-7.2>
28. Jakobi, N.: Evolutionary robotics and the radical envelope-of-noise hypothesis. *Adapt. Behav.* **6**(2), 325–368 (1997). <https://doi.org/10.1177/105971239700600205>
29. Jakobi, N.: Minimal simulations for evolutionary robotics. Ph.D. thesis, University of Sussex, Falmer, UK (1998)
30. Jakobi, N., Husbands, P., Harvey, I.: Noise and the reality gap: the use of simulation in evolutionary robotics. In: Morán, F., Moreno, A., Merelo, J.J., Chacón, P. (eds.) *ECAL 1995. LNCS*, vol. 929, pp. 704–720. Springer, Heidelberg (1995). <https://doi.org/10.1007/3-540-59496-5.337>
31. Kazadi, S.: Model independence in swarm robotics. *Int. J. Intell. Comput. Cybern.* **2**(4), 672–694 (2009). <https://doi.org/10.1108/17563780911005836>

32. Koos, S., Mouret, J.B., Doncieux, S.: The transferability approach: crossing the reality gap in evolutionary robotics. *IEEE Trans. Evol. Comput* **17**(1), 122–145 (2013). <https://doi.org/10.1109/TEVC.2012.2185849>
33. Kuckling, J., Ligot, A., Bozhinoski, D., Birattari, M.: Behavior trees as a control architecture in the automatic modular design of robot swarms. In: Dorigo, M., Birattari, M., Blum, C., Christensen, A.L., Reina, A., Trianni, V. (eds.) *ANTS 2018*. LNCS, vol. 11172, pp. 30–43. Springer, Cham (2018). https://doi.org/10.1007/978-3-030-00533-7_3
34. Ligot, A., Birattari, M.: On mimicking the effects of the reality gap with simulation-only experiments. In: Dorigo, M., Birattari, M., Blum, C., Christensen, A.L., Reina, A., Trianni, V. (eds.) *ANTS 2018*. LNCS, vol. 11172, pp. 109–122. Springer, Cham (2018). https://doi.org/10.1007/978-3-030-00533-7_9
35. Ligot, A., Birattari, M.: Simulation-only experiments to mimic the effects of the reality gap in the automatic design of robot swarms. *Swarm Intell.* **14**(1), 1–24 (2019). <https://doi.org/10.1007/s11721-019-00175-w>
36. Ligot, A., Hasselmann, K., Birattari, M.: AutoMoDe-Arlequin: neural networks as behavioral modules for the automatic design of probabilistic finite state machines: supplementary material (2020). <http://iridia.ulb.ac.be/supp/IridiaSupp2020-005/index.html>
37. Lopes, Y.K., Trenkwalder, S.M., Leal, A.B., Dodd, T.J., Groß, R.: Supervisory control theory applied to swarm robotics. *Swarm Intell.* **10**(1), 65–97 (2016). <https://doi.org/10.1007/s11721-016-0119-0>
38. López-Ibáñez, M., Dubois-Lacoste, J., Pérez Cáceres, L., Birattari, M., Stützle, T.: The irace package: iterated racing for automatic algorithm configuration. *Oper. Res. Perspect.* **3**, 43–58 (2016). <https://doi.org/10.1016/j.orp.2016.09.002>
39. Miglino, O., Lund, H.H., Nolfi, S.: Evolving mobile robots in simulated and real environments. *Artif. Life* **2**(4), 417–434 (1995). <https://doi.org/10.1162/artl.1995.2.4.417>
40. Mondada, F., et al.: 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, Castelo Branco (2009)
41. Nolfi, S., Floreano, D.: *Evolutionary Robotics: The Biology, Intelligence, and Technology of Self-Organizing Machines*. MIT Press, Cambridge (2000)
42. Pinciroli, C., Beltrame, G.: Buzz: a programming language for robot swarms. *IEEE Softw.* **33**(4), 97–100 (2016). <https://doi.org/10.1109/MS.2016.95>
43. Pinciroli, C., et al.: ARGoS: a modular, parallel, multi-engine simulator for multi-robot systems. *Swarm Intell.* **6**(4), 271–295 (2012). <https://doi.org/10.1007/s11721-012-0072-5>
44. Reina, A., Valentini, G., Fernández-Oto, C., Dorigo, M., Trianni, V.: A design pattern for decentralised decision making. *PLOS ONE* **10**(10), e0140950 (2015). <https://doi.org/10.1371/journal.pone.0140950>
45. Salman, M., Ligot, A., Birattari, M.: Concurrent design of control software and configuration of hardware for robot swarms under economic constraints. *PeerJ Comput. Sci.* **5**, e221 (2019). <https://doi.org/10.7717/peerj-cs.221>
46. Silva, F., Duarte, M., Correia, L., Oliveira, S.M., Christensen, A.L.: Open issues in evolutionary robotics. *Evol. Comput.* **24**(2), 205–236 (2016). https://doi.org/10.1162/EVCO_a_00172
47. Trianni, V.: *Evolutionary Swarm Robotics*. Springer, Berlin (2008). <https://doi.org/10.1007/978-3-540-77612-3>

48. Trianni, V.: Evolutionary robotics: model or design? *Front. Robot. AI* **1**, 13 (2014). <https://doi.org/10.3389/frobt.2014.00013>
49. Wolpert, D.: On bias plus variance. *Neural Comput.* **9**, 1211–1243 (1997). <https://doi.org/10.1162/neco.1997.9.6.1211>