2023 IEEE International Conference on Robotics and Automation (ICRA 2023)
May 29 - June 2, 2023. London, UK

Show me what you want: Inverse reinforcement learning to automatically design robot swarms by demonstration

Ilyes Gharbi[1], Jonas Kuckling[1], David Garzón Ramos[1], and Mauro Birattari[1]

Abstract— **Automatic design is a promising approach to generating control software for robot swarms. So far, automatic design has relied on mission-specific objective functions to specify the desired collective behavior. In this paper, we explore the possibility to specify the desired collective behavior via demonstrations. We develop Demo-`Cho`, an automatic design method that combines inverse reinforcement learning with automatic modular design of control software for robot swarms. We show that, only on the basis of demonstrations and without the need to be provided with an explicit objective function, Demo-`Cho` successfully generated control software to perform four missions. We present results obtained in simulation and with physical robots.**

I. INTRODUCTION

Swarm robotics is an approach to control large groups of autonomous robots [1]–[3]. It is considered a prominent research direction [4] and has attained a notable position in the literature [5]–[12]. A robot swarm is a decentralized system and consists of relatively simple robots that can perceive and interact with the environment only in their local neighborhood. A swarm is a self-organizing system, that is, its collective behavior emerges from the interactions of its individual robots. The design challenge in swarm robotics is to program the individual robots so that a desired collective behavior emerges. Several methods have been proposed for specific classes of missions [13]–[21]. Yet, due to the many unpredictable interactions within the swarm, no generally-applicable and principled method exists to design a desired collective behavior [22]–[24].

Automatic off-line design has proven to be a viable approach to designing control software for robot swarms [25]–[29]—other related approaches exist [30]–[32]. In automatic off-line design, an optimization algorithm searches the space of possible instances of control software to find one that maximizes a given mission-specific objective function, which measures the performance of the swarm. The objective function is typically assessed through simulations. The selected instance of control software is then uploaded to real robots, which are eventually deployed in the target environment to perform the mission. Notably, no human intervention beyond the specification of the mission takes place [32]. Defining an objective function—which is part of the formal specification of a mission—is challenging, and requires to be familiar with mathematical modeling. This is a task that requires the attention of a skilled professional and could not be performed by an untrained end user.

The problem of defining an objective function is similar to the one of defining a reward function in reinforcement learning. Inverse reinforcement learning is an approach to address this problem: instead of learning a policy that maximizes a given reward function, inverse reinforcement learning algorithms learn a reward function from demonstrations of an optimal behavior. The learned reward function can then be used to generate a policy that reproduces the demonstrated behavior. Inverse reinforcement learning is motivated by the fact that, for some classes of problems, demonstrating an optimal behavior is easier than defining a reward function [33], [34]. One of the earliest proposed approaches to inverse reinforcement learning is *apprenticeship learning* [34]. Given demonstrations of the desired behavior, the apprenticeship learning algorithm iterates between i) learning a policy based on an intermediate reward function and ii) learning a new intermediate reward function based on the behavior of the previously generated policies. The algorithm stops when the behavior of the current policy is sufficiently close to the provided demonstrations.

We contend that inverse reinforcement learning can be adopted in the framework of the automatic design of control software for robot swarms: instead of defining a mission-specific objective function, we can provide demonstrations of the desired swarm behavior and let an inverse reinforcement learning algorithm infer an objective function to automatically generate the control software that produces the desired behavior itself. In this work, we focus on desired behaviors that can be described through the final position of the robots.

II. RELATED WORK

Inverse reinforcement learning has already found application in robotics: Krishnan et al. proposed SWIRL, an inverse reinforcement learning algorithm to learn various robot tasks, including parallel parking and surgical cutting along a line [35]. The robot successfully learned the tasks from demon-

[1]IG, JK, DGR, and MB are with IRIDIA, Université libre de Bruxelles, Brussels, Belgium. {ilyes.gharbi, jonas.kuckling, david.garzon.ramos, mauro.birattari}@ulb.be. As of April 15, 2023, JK is with the Department of Computer and Information Science, University of Konstanz, Germany.

The project has received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (DEMIURGE: grant agreement No. 681872) and from Belgium's Wallonia-Brussels Federation through a ARC Advanced Project 2020 (Guaranteed by Optimization). JK and MB acknowledge support from the Belgian Fonds de la Recherche Scientifique-FNRS. DGR acknowledges support from the Colombian Ministry of Science, Technology and Innovation – Minciencias.

The experiments were designed by IG, JK, and DGR and performed by IG and JK. The paper was drafted by IG and JK and edited by IG, JK, DGR, and MB; all authors read and commented the final version. The research was directed by MB.

strations and the learned policies were robust to perturbations, such as different initial positions.

Inverse reinforcement learning was also studied in the scope of multi-agent systems. Natarajan et al. used inverse reinforcement learning to develop a centralized system that coordinates multiple traffic lights [36]. Song et al. used inverse reinforcement learning to design policies in general Markov games [37].

In swarm robotics, Šošic et al. used inverse reinforcement learning to learn swarm policies from trajectories obtained from simulations of two particle models [38]. The results show that the swarm was able to replicate the behavior of both particle models. However, the design process required the complete behavior to be already pre-implemented so as to serve as a demonstration.

Besides inverse reinforcement learning, other approaches have been adopted in swarm robotics to learn collective behaviors from demonstrations. Li et al. proposed Turing learning, a method that enables robots to imitate the behavior of other pre-programmed robots, without the need to manually specify the set of features that describe the desired behavior [39]. However, the approach assumes that an implementation of the desired behavior exists and can be used to generate demonstrations. Alharthi et al. extracted swarm behaviors from video demonstrations and used evolutionary algorithms to synthesize control software in the form of behavior trees [40]. Also in this case, the approach requires that an implementation of the desired behavior exists.

III. APPRENTICESHIP LEARNING

Reinforcement learning problems are commonly modelled as a Markov decision process $M = (S, A, T, \gamma, R)$ [41]. A reinforcement learning algorithm learns a policy $\pi : S \rightarrow A$ that maximizes the expected sum of discounted rewards: $E_{s_0}[V_M^\pi(s_0)] = E_{s_0}[\sum_t \gamma^t R(s_t)|\pi]$, with $s_0, \ldots, s_t \in S$.

In inverse reinforcement learning, the reward function R is not provided. Instead, demonstrations of the desired behavior are given in the form of sequences of states. It is assumed that a "true" reward function R^* exists and it is such that the policy π^* that maximizes the value function based on R^* would generate the given demonstrations.

In apprenticeship learning [34], it is furthermore assumed that there exists some function $\phi : S \rightarrow [0, 1]^k$ that maps the states of the system to a k-dimensional vector of features. The "true" reward function R^* is assumed to be a linear combination of the features: $R^*(s) = w^* \cdot \phi(s)$, where $w^* \in \mathbb{R}^k$ and $s \in S$. For every policy π, a feature expectation can be defined as $\mu(\pi) = E_{s_0}[\sum_t \gamma^t \phi(s_t)|\pi] \in \mathbb{R}^k$. It follows that, for R^*, $E_{s_0}[V_M^\pi(s_0)] = w^* \cdot \mu(\pi)$. When the expectation cannot be computed formally, it can be replaced by an empirical estimate $\hat{\mu}(\pi)$ computed on the basis of sampled trajectories. With μ_E, we indicate the feature expectation of the provided demonstrations.

Algorithm 1 shows the pseudo-code of the apprenticeship learning algorithm. Given the mapping ϕ and the feature expectation μ_E of the demonstrations, the algorithm iteratively refines the vector of weights w, until the observed feature expectation μ_i approximates μ_E. At every iteration, a support vector machine [42] is fitted on μ_E and all encountered μ_i. Its coefficients are used as w_{i+1}, the vector of weights that defines the reward function. A new policy π_{i+1} is learned on $R_{i+1}(s) = w_{i+1} \cdot \phi(s)$ and its feature expectation μ_{i+1} is added to the set of feature expectations used to fit the support vector machine in the following iteration. The algorithm terminates when a stopping criterion is met—for example, after a given number of iterations or when a criterion of similarity between the demonstrated and generated behavior is met.

Algorithm 1 Apprenticeship learning [34]

Given: ϕ, μ_E
  Select a random initial policy π_0
  Compute $\mu_0 := \mu(\pi_0)$
  repeat
    Compute w_{i+1} by fitting a SVM on μ_E and all μ_i
    Learn policy π_{i+1} on rewards $R_{i+1}(s) = w_{i+1} \cdot \phi(s)$
    Compute $\mu_{i+1} := \mu(\pi_{i+1})$
  until Stopping criterion met
  return w_{i+1} as w^*

IV. DESIGNING ROBOT SWARMS BY DEMONSTRATION

As shown in Section II, all demonstration-based methods proposed in swarm robotics so far require that at least some robots exist that can demonstrate the desired behavior. This clearly prevents the existing approaches from being used to generate new behaviors. Specifically, we focus here on the class of missions in which what the robots should accomplish is to position themselves in the environment according to a desired distribution. In this case, a demonstration is the desired goal state expressed as the final position of the robots. Although this class of missions does not cover all possible missions of interest in swarm robotics, it includes a large share of the missions that have been studied in the literature [43], [44].

We propose Demo-`Cho`, an automatic design method that combines apprenticeship learning (see Section III) with `Chocolate`, a state-of-the-art automatic off-line design method to generate control software for robot swarms [12], [45]. Demo-`Cho` generates control software for the e-puck robot, a two-wheeled robot [46], [47], extended by an Overo Gumstix [48] and a range-and-bearing board [49] (see Figure 1). Its sensors and actuators were formalized through the reference model RM1.1 [50]. According to RM1.1, the robot is endowed with 8 proximity sensors that can perceive obstacles and other robots, 8 light sensors that can perceive a light source, 3 ground sensors that can detect if the floor is white, black or gray, and a range-and-bearing board that provides the number of neighbors perceived and a vector pointing to their center of mass. The robot is also endowed with two wheels whose velocity can be independently controlled. We assume that the robots operate in a bounded arena in which the floor is gray and some regions might be white or black.

Sensors
Promixity
Light
Ground
Range-and-bearing
Actuators
Wheels

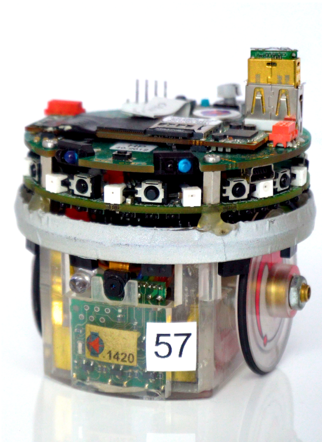


Fig. 1: The e-puck robot and its reference model RM1.1.

Outside the arena, there is a light source that is on in some missions and off in others.

In Demo-`Cho`, the end user can provide demonstrations of the desired final positions of the robots.[2] Demo-`Cho` then uses the apprenticeship learning algorithm to iteratively generate objective functions and `Chocolate` to produce control software by optimizing them. Demo-`Cho` stops after a fixed number of iterations.

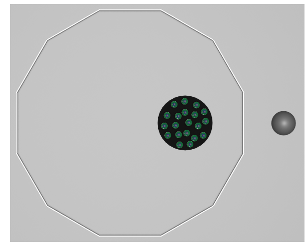
Concerning the feature mapping ϕ, the features we adopted to describe the final position of the robots are based on the distance of each robot from relevant landmarks. Notably, we consider two classes of landmarks: black or white regions and the nearest peer of each robot. We scale distances to the interval $[0, 1]$ according to $10^{-2x/d}$ where d is the arena's diameter and x is the distance to the landmark. Concerning the distance from the regions, if the shortest straight path between the robot and the region is obstructed by a wall, the feature value is set to 0. It is worth noting that the set of features is mission-dependent, as the number of black and white regions possibly varies between missions. Yet, the construction of this mapping is mission-independent, fully automatic, and does not require the intervention/analysis of a human expert. Because all robots of the swarm are interchangeable, the features form an unordered set. To cast them into a vector in a meaningful way so that the apprenticeship learning algorithm can operate on them, we sort them first by the landmark and then in descending order. To give an example, in the feature vector $(\phi_{l1,1}, \phi_{l1,2}, ..., \phi_{l1,n}, \phi_{l2,1}, ...)$, $\phi_{l1,1}$ is the feature corresponding to the distance of the nearest robot to landmark $l1$, $\phi_{l1,2}$ is the one corresponding to the distance of the second nearest robot to $l1$, etc.

V. EXPERIMENTAL SETUP

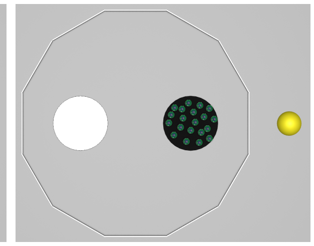
A. Design methods

To appraise the performance of the control software generated by Demo-`Cho`, we present also the results obtained by `Chocolate` and `EvoStick`. `Chocolate` designs control

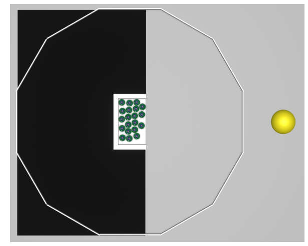
[2]See the supplementary material at `https://iridia.ulb.ac.be/supp/IridiaSupp2022-003/`



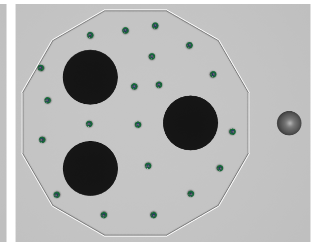
(a) HOMING



(b) AGGREGATION WITH AMBIENT CUES (AAC)



(c) SHELTERING WITH AMBIENT CUES (SAC)



(d) COVERAGE WITH FORBIDEN AREAS (CFA)

Fig. 2: Missions and an example of a demonstration.

software in the form of a probabilistic finite-state machine, assembled from behavioral and conditional modules that are hand-crafted once and for all in a mission-agnostic way [45]. `EvoStick` is an implementation of the classical neuro-evolutionary approach and designs control software in the form of a feed-forward artificial neural network [27]. Notably, both `Chocolate` and `EvoStick` require the actual objective function, whereas Demo-`Cho` does not.

B. Missions

We assess Demo-`Cho` on four missions that were already studied in the literature. For each of them, an objective function is available because it was defined as part of their specifications in the original works that introduced them. We report the original objective functions here and we assume that they are accurate representations of the desired collective behaviors.

All mission take place in the same dodecagonal arena of approximately $5\,\mathrm{m}^2$. For all missions, the swarm size is fixed to 20 robots.

In HOMING [12], the swarm must explore the arena and aggregate in the home area represented by a circular black region with radius of $30\,\mathrm{cm}$ (see Figure 2a). The original objective function is $F_{Homing} = N(T)$, where $N(t)$ is the number of robots in the home area at time t and $T = 180\,\mathrm{s}$ is the mission duration.

In AAC [45] (aggregation with ambient cues), the swarm must aggregate as quickly as possible in a target area represented by a circular black region with radius of $30\,\mathrm{cm}$. Additionally, the arena contains one white circular region with radius of $30\,\mathrm{cm}$, and a light source is placed outside of the arena (see Figure 2b). The original objective function is $F_{AAC} = \sum_{t=1}^{T} N(t)$, where $N(t)$ is the number of robots in the target area at time t and $T = 180\,\mathrm{s}$ is the mission duration.

In SAC [51] (shelter with ambient cues), the swarm must aggregate as quickly as possible in a shelter that can only be accessed from one side. The shelter is indicated by a white rectangular area of $25\,\mathrm{cm}$ by $15\,\mathrm{cm}$ and delimited by three walls, leaving an opening only on one side. The floor in the arena behind the opening of the shelter is black and a light source is placed outside the arena, facing the open side of the shelter (see Figure 2c). For technical reasons regarding the encoding of the environment in the simulator, the black region is composed by three contiguous rectangular sub-regions, one behind the shelter and one on each of its sides. The original objective function is $F_{SAC} = \sum_{t=1}^{T} N(t)$, where $N(t)$ is the number of robots in the shelter at time t and $T = 180\,\mathrm{s}$ is the mission duration.

In CFA [45] (coverage with forbidden areas), the swarm must spread through the arena while avoiding the forbidden areas represented by three black circular regions with radii of $30\,\mathrm{cm}$ (see Figure 2d). The original objective function is $E[d(T)]$, the expected distance between a generic point in the arena and the closest robot not on a forbidden area, at the end of T, and $T = 180\,\mathrm{s}$ is the experiment duration. To be consistent with the other missions in which the objective function is to be maximized, we reformulate the objective function as $F_{CFA} = 250 - E[d(T)]$ where 250 is the theoretical maximum value of $E[d(T)]$.

C. Protocol

For each mission, we provided five demonstrations of the final position of the robot swarm to be used by Demo-`Cho`—see the supplementary material.[2] We ran 10 independent design processes for each of the three design methods under analysis. All design methods adopt the same simulator: ARGoS3 [52]. Demo-`Cho` was run for 50 iterations, each iteration with a budget of $10\,000$ simulation runs per iteration. `Chocolate` and `EvoStick` were run with a design budget of $10\,000$ simulation runs and optimize the original objective function. All in all, this grants Demo-`Cho` a budget that is fifty times larger than the one of `Chocolate` and `EvoStick`. The goal of this protocol is not to achieve a fair comparison between the three design methods, which could be a rather complex endeavour—see the discussion in Section VII. Indeed, `Chocolate` and `EvoStick` have the clear advantage of being fed with an objective function; the larger budget allocated to Demo-`Cho` is intended to compensate somehow for the fact that Demo-`Cho` has to infer the objective function from the given demonstrations. In this context, we felt that the primary concern was to provide an appropriate budget to each automatic design process: the one performed by `Chocolate` and `EvoStick`, and each of the 50 ones performed within each execution of Demo-`Cho`. Following our previous experience, we allocated to each of these design processes a budget of $10\,000$ simulations. Concerning the choice of the number of iterations to be taken as a stopping criterion for Demo-`Cho`, as no previous literature exist on this issue, we fixed this to a sufficiently large number to make sure that the algorithm had time to converge to a meaningful solution—see the discussion in Section VI where we comment *a posteriori* on this choice, in the light of the results obtained through the present study.

We assessed the resulting instances of control software once in simulation and once in reality. In the experiments with the robots, performance was measured automatically using a tracking system [53]. We provide both a qualitative and a quantitative assessment of the performance of the swarms generated by the three methods under analysis. The qualitative assessment is based on visual inspection of the generated behaviors. The quantitative assessment is based on the mission-specific objective function, the same one that `Chocolate` and `EvoStick` optimize within the design process. For a detailed discussion of this choice, we refer the reader to Section VII.

We report the results in the form of notched boxplots. In the boxplots, the upper and lower hinges correspond to the first and third quartiles. The whiskers extend to the largest value of the sample but no further than 1.5 times the interquartile range from the hinge. Data beyond the whiskers are outliers and are represented by points. We also report the median of the sample, represented by a line in the box, and a 95% confidence interval, represented by notches extending from the median line. If the notches of two boxplots do not overlap, we can conclude that the difference between the medians of the two samples is statistically significant.

The source code, experiment files, and results of all experiments are available as supplementary material.[2]

VI. EXPERIMENTAL RESULTS

Figure 3 shows the boxplots of the results obtained in simulation and reality. The three design methods achieved similar performance in simulation across the four missions, despite the fact that Demo-`Cho`, contrary to `Chocolate` and `EvoStick`, did not have access to the objective function at design time. Visual inspection of the generated behaviors in simulation shows that those generated by Demo-`Cho` match the expectations that one might have on the mission at hand: the robots behave in a meaningful way in all four missions—see supplementary videos.[2] It has to be noted that in the two missions AAC and SCA, the original objective function does not evaluate only the final position—i.e., the one illustrated by the demonstrations provided to Demo-`Cho`—but is computed cumulatively over the whole duration of an experimental run. Yet, the performance of Demo-`Cho` was not worse than the one of `Chocolate` or `EvoStick`.

The experiments allowed us to gain some insight on the number of iterations needed by Demo-`Cho` to converge to a meaningful solution. All in all, the selected number of 50 iterations appears to be a reasonably appropriate choice—see the supplementary material.[2] Typically, after the first 10 iterations, the behavior found already reproduces well the given demonstrations. Further improvement can be observed in the following iterations to then become rare after 40 iterations. Future work should be devoted to gain a deeper

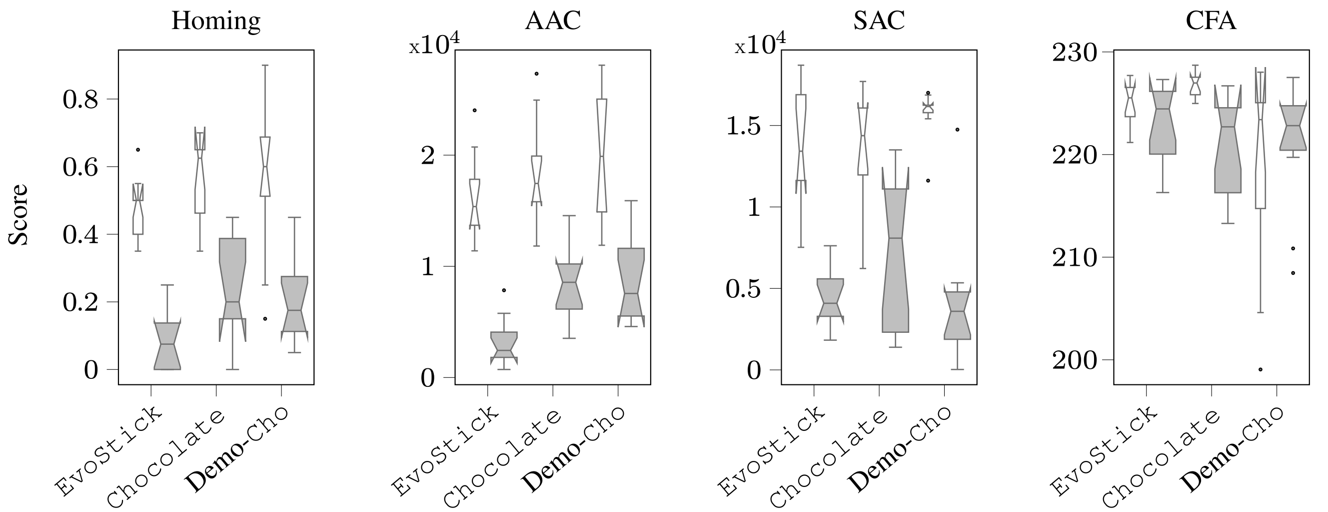
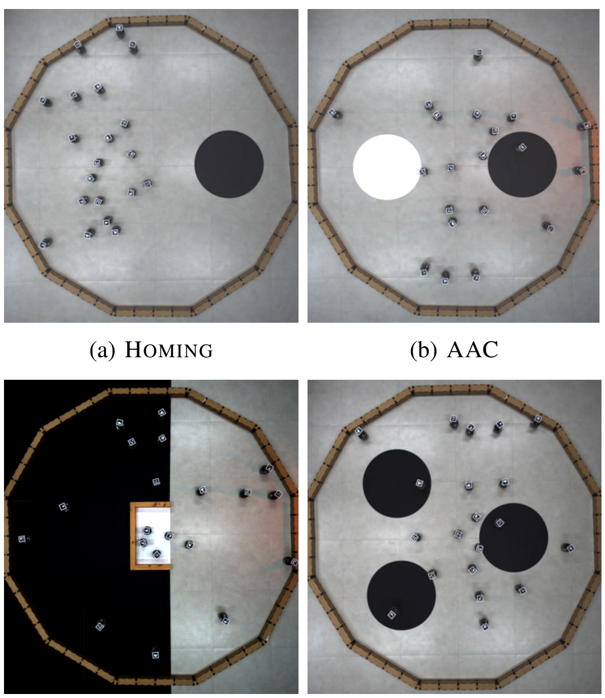


Fig. 3: Experimental results obtained in simulation (narrow white boxes) and reality (wide gray boxes). See Fig. 2 for acronyms.



(a) HOMING (b) AAC

(c) SAC (d) CFA

Fig. 4: The four missions in reality. See Fig. 2 for acronyms.

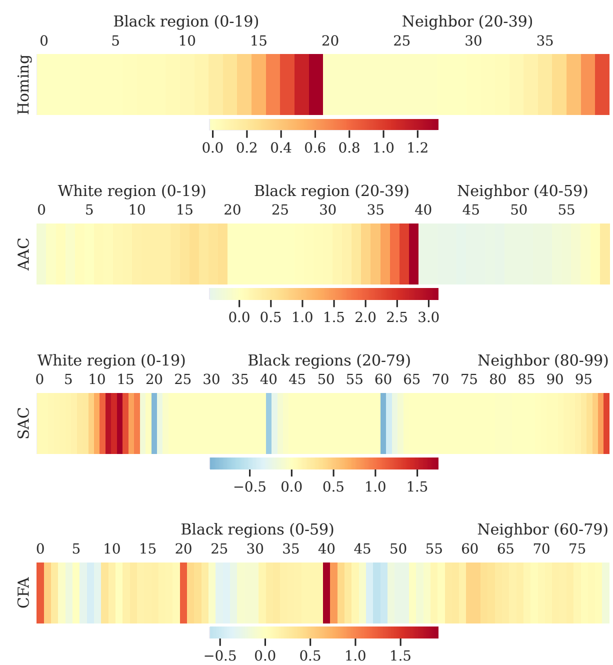


Fig. 5: Heat maps of the average weight vectors learned by Demo-`Cho`. See Fig. 2 for acronyms.

insight in the issue by observing the development of the improvement over an even larger number of iterations.

When assessed in reality, all three methods showed a drop in performance—as it is often the case in off-line automatic design [54]. In the missions HOMING, SAC, and CFA, the three design methods achieved similar performance in reality. In AAC, Demo-`Cho` and `Chocolate` achieved similar performance in reality and outperformed `EvoStick`. On the basis of these results, we can argue that learning from demonstrations—as opposed to optimizing a given objective function—does not appear to have any major impact on the ability of a modular design method to cross the reality gap.

Figure 5 shows the weights w learned by Demo-`Cho`, averaged per mission. Some general observations can be made for the four missions. For each group of features—those relating to the same landmark—Demo-`Cho` tends to put larger weights on the feature of lower value, that is, those corresponding to the robots that are the farthest from the landmark. Indeed, minimizing the distance of the farthest robots guarantees that the distance of all the others is minimized. Concerning the weights for the specific missions, we can observe that: In HOMING, the distance to the black region was selected by Demo-Cho as the most important feature. Albeit to a lesser extent, also the distance to the nearest neighbor was considered important. Thus, the design process rewarded behaviors that aggregate tightly in the home area. Also in AAC, Demo-Cho selected the distance to the black region as the most important feature. Unlike in HOMING, however, the distance to the nearest neighbor was not considered important, neither was the one to the white region. For this mission, the design process rewarded behaviors that aggregate in the target area. The tightness of the aggregation possibly resulted implicitly, as all robots must fit in the target area. In SAC, the design process selected two important features: the distance to the white region and the one to the nearest peer. The selection of these two features can be interpreted to describe an aggregation behavior in the shelter. Curiously, unlike for the other features, Demo-Cho assigned the highest weight to the feature associated with the sixth farthest robot from the white region, rather than the feature associated with the farthest one. This might be explained by the fact that it is unlikely that all the robots eventually reach the shelter and five robots outside the shelter at the end of the experimental run is a common outcome. Additionally, we observe three features that Demo-Cho penalizes through the assignment of a negative weight: the distance of the nearest robot to each of the black regions. Maximizing the distance between the nearest robot and a landmark guarantees that the distance of all robots is maximized. In CFA, Demo-Cho selected three groups of features as important: the distance to each of the black regions. In this case, the weights were selected to favor the presence of the robots nearby each of the black regions: the highest weight is associated with the feature corresponding to the distance of robot closest to the landmark. Additionally, Demo-Cho slightly penalizes the features corresponding to the distances from the landmark of the fifth to eighth nearest robots. As a result, the design process aimed to keep the robots close to the forbidden areas without favoring an aggregation. Additionally, some importance is placed on the features describing the inter-robot distance: a slightly positive weight is associated to the distance of nearest peers.

The interpretation of the weights is straightforward for HOMING, AAC, and SAC, while it is less intuitive for CFA. Indeed, in CFA, one could have expected more emphasis on the inter-robot distance and the penalization of the distance to the forbidden areas. Nonetheless, excluding two outliers, the performance achieved by Demo-Cho in this mission is satisfactory and the behavior of the robots appears to be meaningful at visual inspection—see supplementary videos.[2]

VII. CONCLUSIONS

In this work, we presented Demo-Cho, an automatic method for designing control software of robot swarms that combines inverse reinforcement learning with automatic modular design. Instead of optimizing an explicitly defined objective function, Demo-Cho generates control software based on provided demonstrations. In our experiments, Demo-Cho was able to create satisfactory behaviors to perform four missions that were previously studied in the literature. Expressing a desired outcome in terms of a mathematical function is unintuitive and requires the attention of an expert. Specifying desired behaviors through demonstrations is natural and intuitive and could allow even end users without any technical expertise to specify their desired behaviors.

In the experiments presented in this paper, we accept the original assumption made by the proponents of the missions that the objective function accurately specifies the desired behavior. We therefore use this objective function for the final assessment of the behaviors produced by Demo-Cho on the basis of the given demonstrations. However, this way of assessing performance is viable only for missions that already have been specified via the definition of an objective function. A general protocol to assess behaviors generated from demonstrations could be defined on the basis of an appropriate metric that measures the degree of similarity between the given demonstrations and the generated behavior. Yet, the goal would not be to reproduce the demonstrations but to generalize with respect to them. An appropriate protocol could take inspiration from the classical cross-validation and leave-one-out procedures typically adopted in machine learning.

A protocol should also be defined to compare in a fair way methods based on demonstrations with traditional ones that optimize a given objective function. The latter clearly have an advantage on the former, which have to infer an objective function from the given examples. An appropriate protocol should test also traditional methods on an objective function other than the one they used at design time. For example, two experts might define one objective function each. One of these objective functions could be used by the traditional methods in the design phase; and the other could be used to test both traditional methods and demonstration-based ones. This would put the two methods on the same foot for what concerns the evaluation.

In the future, we will extend Demo-Cho to missions that can be represented through the final position of elements other than the robots—e.g., objects to be clustered, gathered, spread in the environment. We also anticipate that some missions will have to be represented via a non-linear combination of features, which the apprenticeship learning method based on the maximum-margin principle is not be able to address straightforwardly. To overcome this limitation, we will investigate other IRL methods such as those based on the maximum-entropy principle [55], which can produce non-linear objective functions. Additionally, we will investigate the minimum number of demonstrations necessary to design a desired behavior and more generally, the impact the number of demonstrations and their diversity have on the quality of the behaviors that can be obtained.

References

[1] G. Beni, "From swarm intelligence to swarm robotics," in *SAB 2004*, ser. LNCS, E. Şahin and W. M. Spears, Eds., vol. 3342. Berlin, Germany: Springer, 2005, pp. 1–9.

[2] E. Şahin, "Swarm robotics: from sources of inspiration to domains of application," in *SAB 2004*, ser. LNCS, E. Şahin and W. M. Spears, Eds., vol. 3342. Berlin, Germany: Springer, 2005, pp. 10–20.

[3] M. Dorigo, M. Birattari, and M. Brambilla, "Swarm robotics," *Scholarpedia*, vol. 9, no. 1, p. 1463, 2014.

[4] G.-Z. Yang, J. Bellingham, P. E. Dupont, P. Fischer, L. Floridi, R. Full, N. Jacobstein, V. Kumar, M. McNutt, R. Merrifield, B. J. Nelson, B. Scassellati, M. Taddeo, R. Taylor, M. Veloso, Z. L. Wang, and R. Wood, "The grand challenges of Science Robotics," *Sci. Robot.*, vol. 3, no. 14, p. eaar7650, 2018.

[5] M. Rubenstein, A. Cornejo, and R. Nagpal, "Programmable self-assembly in a thousand-robot swarm," *Science*, vol. 345, no. 6198, pp. 795–799, 2014.

[6] J. Werfel, K. Petersen, and R. Nagpal, "Designing collective behavior in a termite-inspired robot construction team," *Science*, vol. 343, no. 6172, pp. 754–758, 2014.

[7] L. Garattoni and M. Birattari, "Autonomous task sequencing in a robot swarm," *Sci. Robot.*, vol. 3, no. 20, p. eaat0430, 2018.

[8] I. Slavkov, D. Carrillo-Zapata, N. Carranza, X. Diego, F. Jansson, J. Kaandorp, S. Hauert, and J. Sharpe, "Morphogenesis in robot swarms," *Sci. Robot.*, vol. 3, no. 25, p. eaau9178, 2018.

[9] J. Yu, B. Wang, X. Du, Q. Wang, and L. Zhang, "Ultra-extensible ribbon-like magnetic microswarm," *Nat. Commun.*, vol. 9, no. 1, p. 3260, 2018.

[10] S. Li, R. Batra, D. Brown, H.-D. Chang, N. Ranganathan, C. Hoberman, D. Rus, and H. Lipson, "Particle robotics based on statistical mechanics of loosely coupledomponents," *Nature*, vol. 567, no. 7748, pp. 361–365, 2019.

[11] H. Xie, M. Sun, X. Fan, Z. Lin, W. Chen, L. Wang, L. Dong, and Q. He, "Reconfigurable magnetic microrobot swarm: multimode transformation, locomotion, and manipulation," *Sci. Robot.*, vol. 4, no. 28, p. eaav8006, 2019.

[12] K. Hasselmann, A. Ligot, J. Ruddick, and M. Birattari, "Empirical assessment and comparison of neuro-evolutionary methods for the automatic off-line design of robot swarms," *Nat. Commun.*, vol. 12, p. 4345, 2021.

[13] H. Hamann and H. Wörn, "A framework of space–time continuous models for algorithm design in swarm robotics," *Swarm Intell.*, vol. 2, no. 2–4, pp. 209–239, 2008.

[14] S. Kazadi, "Model independence in swarm robotics," *Int. J. Intell. Comput. Cybern.*, vol. 2, no. 4, pp. 672–694, 2009.

[15] S. Berman, V. Kumar, and R. Nagpal, "Design of control policies for spatially inhomogeneous robot swarms with application to commercial pollination," in *ICRA 2011*. Piscataway, NJ, USA: IEEE, 2011, pp. 378–385.

[16] J. Beal, S. Dulman, K. Usbeck, M. Viroli, and N. Correll, "Organizing the aggregate: languages for spatial computing," in *Formal and Practical Aspects of Domain-Specific Languages: Recent Developments*, M. Marjan, Ed. Hershey, PA, USA: IGI Global, 2012, pp. 436–501.

[17] M. Brambilla, A. Brutschy, M. Dorigo, and M. Birattari, "Property-driven design for swarm robotics: a design method based on prescriptive modeling and model checking," *ACM Tran. Auton. Adap.*, vol. 9, no. 4, pp. 17:1–17:28, 2014.

[18] A. Reina, G. Valentini, C. Fernández-Oto, M. Dorigo, and V. Trianni, "A design pattern for decentralised decision making," *PLOS ONE*, vol. 10, no. 10, p. e0140950, 2015.

[19] Y. K. Lopes, S. M. Trenkwalder, A. B. Leal, T. J. Dodd, and R. Groß, "Supervisory control theory applied to swarm robotics," *Swarm Intell.*, vol. 10, no. 1, pp. 65–97, 2016.

[20] C. Pinciroli and G. Beltrame, "Buzz: a programming language for robot swarms," *IEEE Softw.*, vol. 33, no. 4, pp. 97–100, 2016.

[21] H. Hamann, *Swarm robotics: a formal approach*. Cham, Switzerland: Springer, 2018.

[22] M. Brambilla, E. Ferrante, M. Birattari, and M. Dorigo, "Swarm robotics: a review from the swarm engineering perspective," *Swarm Intell.*, vol. 7, no. 1, pp. 1–41, 2013.

[23] M. Dorigo, G. Theraulaz, and V. Trianni, "Reflections on the future of swarm robotics," *Sci. Robot.*, vol. 5, p. eabe4385, 2020.

[24] ——, "Swarm robotics: past, present, and future [point of view]," *Proc. IEEE*, vol. 109, no. 7, pp. 1152–1165, 2021.

[25] G. Baldassarre, V. Trianni, M. Bonani, F. Mondada, M. Dorigo, and S. Nolfi, "Self-organized coordinated motion in groups of physically connected robots," *IEEE Trans. Syst. Man Cybern. B*, vol. 37, no. 1, pp. 224–239, 2007.

[26] M. Gauci, J. Chen, W. Li, T. J. Dodd, and R. Groß, "Self-organized aggregation without computation," *Int. J. Robot. Res.*, vol. 33, no. 8, pp. 1145–1161, 2014.

[27] G. Francesca, M. Brambilla, A. Brutschy, V. Trianni, and M. Birattari, "AutoMoDe: a novel approach to the automatic design of control software for robot swarms," *Swarm Intell.*, vol. 8, no. 2, pp. 89–112, 2014.

[28] M. Duarte, V. Costa, J. Gomes, T. Rodrigues, F. Silva, S. M. Oliveira, and A. L. Christensen, "Evolution of collective behaviors for a real swarm of aquatic surface robots," *PLOS ONE*, vol. 11, no. 3, p. e0151834, 2016.

[29] S. Jones, M. Studley, S. Hauert, and A. Winfield, "Evolving behaviour trees for swarm robotics," in *DARS 13*, ser. SPAR, R. Groß, A. Kolling, S. Berman, E. Frazzoli, A. Martinoli, F. Matsuno, and M. Gauci, Eds., vol. 6. Cham, Switzerland: Springer, 2018, pp. 487–501.

[30] G. Francesca and M. Birattari, "Automatic design of robot swarms: achievements and challenges," *Front. Robot. AI*, vol. 3, no. 29, pp. 1–9, 2016.

[31] N. Bredeche, E. Haasdijk, and A. Prieto, "Embodied evolution in collective robotics: a review," *Front. Robot. AI*, vol. 5, p. 12, 2018.

[32] M. Birattari, A. Ligot, and K. Hasselmann, "Disentangling automatic and semi-automatic approaches to the optimization-based design of control software for robot swarms," *Nat. Mach. Intell.*, vol. 2, no. 9, pp. 494–499, 2020.

[33] S. Russel, "Learning agents for uncertain environments (extended abstract)," in *COLT 1998*, P. Bartlett and Y. Mansour, Eds. New York, NY, USA: ACM, 1998, pp. 101–103.

[34] P. Abbeel and A. Y. Ng, "Apprenticeship learning via inverse reinforcement learning," in *ICML 2004*, C. Brodley, Ed. New York, NY, USA: ACM, 2004, p. 1.

[35] S. Krishnan, A. Garg, R. Liaw, B. Thananjeyan, L. Miller, F. T. Pokorny, and K. Goldberg, "SWIRL: a sequential windowed inverse reinforcement learning algorithm for robot tasks with delayed rewards," *Int. J Robot. Res.*, vol. 38, no. 2–3, pp. 126–145, 2019.

[36] S. Natarajan, G. Kunapuli, K. Judah, P. Tadepalli, K. Kersting, and J. Shavlik, "Multi-agent inverse reinforcement learning," in *Ninth Int. Conf. Mach. Learn. Appl.*, S. Draghici, T. M. Khoshgoftaar, V. Palade, W. Pedrycz, M. A. Wani, and X. Zhu, Eds. Los Alamitos, CA, USA: IEEE Computer Society, 2010, pp. 395–400.

[37] J. Song, H. Ren, D. Sadigh, and S. Ermon, "Multi-agent generative adversarial imitation learning," in *Adv. N.A. Inf. Process. Syst. 31: 32nd Conf. N.A. Inf. Process. Syst. (NeurIPS 2018)*, S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, Eds. Red Hook, NY, USA: Curran Associates, 2018, pp. 402–408.

[38] A. Šošić, W. R. Khuda Bukhsh, A. M. Zoubir, and H. Koeppl, "Inverse reinforcement learning in swarm systems," in *AAMAS 2017*. Richland, SC, USA: International Foundation for Autonomous Agents and Multiagent Systems (IFAAMAS), 2017, pp. 1413–1421.

[39] W. Li, M. Gauci, and R. Groß, "Turing learning: a metric-free approach to inferring behavior and its application to swarms," *Swarm Intell.*, vol. 10, pp. 211–243, 2016.

[40] K. Alharthi, Z. S. Abdallah, and S. Hauert, "Automatic extraction ofnderstandable controllers fromideo observations ofwarm behaviors," in *ANTS 2022*, M. Dorigo, H. Hamann, M. López-Ibáñez, J. García-Nieto, A. Engelbrecht, C. Pinciroli, V. Strobel, and C. Camacho-Villalón, Eds., vol. 13491. Cham, Switzerland: Springer, 2022, pp. 41–53.

[41] L. P. Kaelbling, M. L. Littman, and A. W. Moore, "Reinforcement learning: a survey," *J. Artif. Intell. Res.*, vol. 4, pp. 237–285, 1996.

[42] M. A. Hearst, S. T. Dumais, E. Osuna, J. C. Platt, and B. Schölkopf, "Support vector machines," *IEEE Intell. Syst. App.*, vol. 13, no. 4, pp. 18–28, 1998.

[43] L. Garattoni and M. Birattari, "Swarm robotics," in *Wiley Encyclopedia of Electrical and Electronics Engineering*, J. G. Webster, Ed. Hoboken, NJ, USA: John Wiley & Sons, 2016, pp. 1–19.

[44] M. Schranz, M. Umlauft, M. Sende, and W. Elmenreich, "Swarm robotic behaviors and current applications," *Front. Robot. AI*, vol. 7, p. 36, 2020.

[45] G. Francesca, M. Brambilla, A. Brutschy, L. Garattoni, R. Miletitch, G. Podevijn, A. Reina, T. Soleymani, M. Salvaro, C. Pinciroli, F. Mascia, V. Trianni, and M. Birattari, "AutoMoDe-Chocolate: automatic

design of control software for robot swarms," *Swarm Intell.*, vol. 9, no. 2–3, pp. 125–152, 2015.

[46] F. Mondada, M. Bonani, X. Raemy, J. Pugh, C. Cianci, A. Klaptocz, S. Magnenat, J.-C. Zufferey, D. Floreano, and A. Martinoli, "The e-puck, a robot designed for education in engineering," in *ROBOTICA 2009*, P. Gonçalves, P. Torres, and C. Alves, Eds. Castelo Branco, Portugal: Instituto Politécnico de Castelo Branco, 2009, pp. 59–65.

[47] L. Garattoni, G. Francesca, A. Brutschy, C. Pinciroli, and M. Birattari, "Software infrastructure for e-puck (and TAM)," IRIDIA, Université libre de Bruxelles, Brussels, Belgium, Tech. Rep. TR/IRIDIA/2015-004, 2015.

[48] GCtronic, "Overo extension," https://www.gctronic.com/doc/index.php/Overo_Extension, 2018.

[49] Á. Gutiérrez, A. Campo, M. Dorigo, J. Donate, F. Monasterio-Huelin, and L. Magdalena, "Open e-puck range & bearing miniaturized board for local communication in swarm robotics," in *ICRA 2009*, K. Kosuge, Ed. Piscataway, NJ, USA: IEEE, 2009, pp. 3111–3116.

[50] K. Hasselmann, A. Ligot, G. Francesca, D. Garzón Ramos, M. Salman, J. Kuckling, F. J. Mendiburu, and M. Birattari, "Reference models for AutoMoDe," IRIDIA, Université libre de Bruxelles, Brussels, Belgium, Tech. Rep. TR/IRIDIA/2018-002, 2018.

[51] K. Hasselmann, A. Ligot, and M. Birattari, "SAC: shelter with ambient cues," Personal communication, 2022.

[52] C. Pinciroli, V. Trianni, R. O'Grady, G. Pini, A. Brutschy, M. Brambilla, N. Mathews, E. Ferrante, G. A. Di Caro, F. Ducatelle, M. Birattari, L. M. Gambardella, and M. Dorigo, "ARGoS: a modular, parallel, multi-engine simulator for multi-robot systems," *Swarm Intell.*, vol. 6, no. 4, pp. 271–295, 2012.

[53] G. Legarda Herranz, D. Garzón Ramos, J. Kuckling, M. Kegeleirs, and M. Birattari, "Tycho: a robust, ROS-based tracking system for robot swarms," IRIDIA, Université libre de Bruxelles, Brussels, Belgium, Tech. Rep. TR/IRIDIA/2022-009, 2022.

[54] A. Ligot and M. Birattari, "Simulation-only experiments to mimic the effects of the reality gap in the automatic design of robot swarms," *Swarm Intell.*, vol. 14, pp. 1–24, 2020.

[55] B. D. Ziebart, A. Maas, J. A. Bagnell, and A. K. Dey, "Maximum entropy inverse reinforcement learning," in *Proc. of the 23rd Nat. Conf. on Art. Int.*, ser. AAAI'08, vol. 3. Chicago, Illinois: AAAI Press, 2008, pp. 1433—1438.