Iterative Learning for Model Reactive Control: Application to autonomous multi-agent control

Omar Shrit, David Filliat, Michele Sebag

Iterative Learning for Model Reactive Control: Application to autonomous multi-agent control

Omar Shrit
Université Paris Saclay
omar.shrit@université-paris-saclay.fr

David Filliat
École Nationale Supérieure de Techniques Avancées
david.filliat@ensta-paris.fr

Michèle Sebag
Université Paris Saclay
michele.sebag@université-paris-saclay.fr

***Abstract*—In this paper, a decentralized autonomous controller aimed to control a fleet of quadrotors is designed, based on the iterative generation and exploitation of logged traces. The presented approach, inspired by model predictive control, aims to maintain the geometrical configuration for a set of quadrotors led by remotely controlled leaders. The novelty of this approach is to rely on inexpensive commercial off-the-shelf sensors (as opposed to positioning systems and/or cameras) that only measure the distance among quadrotors. In the first phase (trace generation) quadrotors are operated using randomized controllers based on domain knowledge, and their trajectories are registered. In the exploitation phase, a policy is learned from the traces generated in the previous phase, and the policy is iteratively refined, to achieve a robust reactive control of each quadrotor agent. Extensive experiments using RotorS, a Software In the Loop (SITL) framework in Gazebo simulator demonstrates the efficiency of the approach, and its ability to preserve the flocking structure of the quadrotors, following the (remotely and independently controlled) leaders.**

***Index Terms*—Quadrotors, leader-follower, machine learning, iterative learning, model predictive control, neural networks**

Simulation videos

Available at https://www.dropbox.com/sh/97ixrp0ayejvim0/AADdIRkRWxtexlCP2C-nFGQEa?dl=0

I. Introduction

Since the 80s, Multi-Agent Systems (MAS) have been thoroughly studied [1]: they aim to execute complex tasks in a robust way, through the cooperation of simple agents. A main challenge is to design decentralized agent controllers, such that their execution on each independent agent enforces their cooperation and achieves the target complex task. Indeed, decentralized multi-agent control is known to be more difficult by one or several orders of magnitude compared to the control of a single-robot [2].

This paper focuses on multi-robot system and specifically on quadrotor fleets. Quadrotors are appreciated for both their ability to realize vertical take off, fly at low speed or to be stationary in the air and their low price.
The last decades have shown a considerable development of small aerial robots [3], as they tackle applications that are out of reach for quadruped and humanoid robots, e.g., building construction [4], delivering packages [5], providing on-demand wireless network [6] monitoring agriculture [7] or execute search and rescuing missions [8]. Most applications rely on manually designed controllers, specifically designed for the task at hand and operating in a centralized or decentralized way. Accordingly, they might face some limitations when dealing with stochastic environments, and more generally hardly adapt themselves to properly reflect changes in the environment. In order to address these limitations, some online optimization schemes have been proposed such as Model Predictive Control (MPC) [9], [10] or Decentralized Model Predictive Control (DMPC) [11] (more in section III). Such schemes however also suffer from some limitations. On the one hand, MPC and DMPC require heavy computational resources to solve the objective function with respect to the agent situation (set of constraints), precluding their use in real-time. On the other hand, the task at hand and/or the environment might be too complex to support the manual design of the MPC model. In such cases, an alternative is offered by machine learning, and typically (deep) reinforcement learning [12].

This paper presents a new machine learning approach called IL4MRC for *Iterative Learning for Model Reactive Control*, aimed to address the decentralized controller design challenge. IL4MRC has been designed, deployed and validated within the RotorS framework [13]. This framework pertains to the *Software In The Loop* (SITL) frameworks, specifically designed to run the autopilot software as an independent process on the host machine (as opposed to, on a real quadrotor hardware). By allowing the use of the same software infrastructure for training/design and deployment/validation, the RotorS framework offers a fair and unbiased platform to study high level tasks such as path planning, collision avoidance, or flocking behaviors. This alternative to real-world test-beds has been extensively investigated in the context of quadrotor fleets [14], and its robustness has been demonstrated [15], showing that the trained controllers (learned or optimized in simulation) can be directly transferred and deployed in a real-world setting, with same performances.

The objective of IL4MRC decentralized controller is to achieve specific flocking behavioral patterns. Specifically, the considered set of quadrotors consists of: i) *leaders*, assumed to be externally operated and synchronized; ii) *followers*. The latter quadrotors, operated by the IL4MRC controller,

are required to maintain general flocking behavioral patterns consistent with the leaders. The technical difficulty is to achieve such a behavior under severe resource constraints, in terms of computational abilities and sensing capacities.

Taking inspiration from MPC [16], IL4MRC extracts and gradually refines models, guiding the decentralized controller operated on each follower. Formally, these models are learned along an 3-step iterative supervised learning process. In each iteration, as a first step a randomized controller is ported on each quadrotor and used to generate logs, reporting the quadrotor state (vector of sensor values) and action in each time step. As a second step, the logs are used as in imitation reinforcement learning [17] to learn a policy and a forward models. In a third step, these models are used along a greedy MPC to define a refined controller, and this controller is used in the next iteration to generate new logs.

The empirical validation of the approach is conducted in the context of micro-aerial vehicles (MAV). The considered Software In The Loop (SITL) framework [13] relies on the physics-compliant Gazebo simulator [18].

As said, the requirements on the controller are twofold: i) it must allow each follower to flexibly accommodate the changing directions and dynamics of the leaders, and it must allow the set of followers as a whole to preserve a flocking behavior; ii) it must do so under severe resource constraints in terms of computational abilities and sensors. Compared to the state of the art in MPC [16], the contribution is threefold. On the one hand, the quadrotor controller only relies on cheap cues (WiFi signal strength) and does not require heavy positional or visual sensors (GPS or cameras). On the other hand, the exploitation of a complex model (as in [16]) is replaced by iteratively learning and refining simple models, in a way that can be thought of as a self-play setting [19]. Lastly, compared to a full-scale reinforcement learning (RL) approach [20] IL4MRC offers an agile alternative inspired by inverse reinforcement learning [21], with a much lower sample complexity.

This paper is organized as follows. Section II briefly reviews and discusses related work. Section III gives a formal background, details the underlying assumptions and presents a general overview of IL4MRC. Section IV proposes a proof of concept to validate the proposed method. Sections V and VI respectively present the experimental setting and reports the empirical evidence obtained for this proof of concept. The paper concludes with a discussion and some perspectives for further research.

II. Related work

This section briefly discusses the state of the art in multi-agent controller design. Model Predictive Control, being the method most related to the IL4MRC approach, will be presented in more detail in section III.

In the literature, most of the individual controllers for multi-agent systems rely on manual design, along two main approaches. In the centralized setting, a controller has an omniscience of all the states of the agents, but the space and action space dimensions are multiplied by the number of the agents. For example, Lupashin *et al*. [22] create a centralized controller that operates on an independent command center that send these commands to each quadrotor in order to create the desired behavior. In the decentralized setting where the goal is to design a controller independently operating each agent [23], the state of each agent is partially observed (the state / intentions of the other agents are most generally unknown), and the controller design requires to solve a Partially Observable Markov Decision Problem.

Another alternative to manual design investigated in the last twenty years, multi-agent learning was mainly focused on reinforcement learning [24], [25] or genetic algorithms [26]. For example, genetic algorithms have been applied in RoboCup [27] competition in order to make a team of small robots to play soccer, while reinforcement learning have been applied to resolve the keep away sub-problem [28]

In the context of machine learning applied to quadrotors, to our knowledge most of existing solutions are effective to a single quadrotor. Molchanov *et al*. [29] propose a neural network flight controller based on reinforcement learning for a single quadrotors where the model is trained in simulator and then transferred into a real-world platform. In [30], machine learning is used to achieve quadrotors navigation; the authors train quadrotors to navigate through a trail in a challenging environment (forests) using a single camera. On the same line [31], the authors uses supervised learning on a single quadrotor to accomplish specific tasks such as navigation or grasping [32]. CAD^2RL achieves autonomous flight in indoor environment [33], using deep reinforcement learning instead of SLAM (Simultaneous Localization And Mapping) in a simulated environment to train vision based navigation policy for a flying robot with monocular camera; the learned policy is transferred to real world to avoid obstacles and achieve collision-free flight.

In the context of several quadrotors, Hock *et al*. [34] propose an iterative learning control approach to a set of quadrotors that follow the desired trajectory proposed by so-called the virtual leader, while preserving the group formation. However, their system depends on the communications to provide positional information to each agent between of them. Likewise, Ekaterina *et al*. [35] have developed a decentralized controller for quadrotors based on Graph Neural Network (GNN). The objective is to learn a local controller for each quadrotor that uses communications and exchange local information with neighbors in order to achieve consistent group behaviour (flocking problem). Otherwise, the same problem has been addressed differently by [14], in which communications have been replaced by the use of visual sensors that provide the full state of the neighbors of each agent and the GNN is replaced by supervised learning neural network.

III. Overview of IL4MRC

For the sake of self-containedness, this section first briefly presents Model Predictive Control [36] and discusses its

strengths and weaknesses. The IL4MRC approach, aimed to overcome these weaknesses, is then described.

A. Decentralized Model Predictive Control

Most generally, MPC considers a dynamic system with input (interchangeably referred to as command, or action in the following) and output (referred to as response or state) respectively noted u_t and y_t. Denoting respectively $\mathbf{u}_{1:t}$ and $\mathbf{y}_{1:t}$) the sequence of commands and output of the system along time, MPC relies on a model of the system at hand, describing how the output of the system depends on its past input and output:

$$y(t) = \mathcal{F}(\mathbf{y}_{1:t-1}, \mathbf{u}_{1:t-1})$$

or $y(t) = \mathcal{F}(\mathbf{y}, \mathbf{u}, t)$ for short. In the linear case, an example of such a behavioral model is given by [37]:

$$y_t = -a_1 y_{t-1} - a_2 y_{t-2} + b_1 u_{t-1} + b_2 u_{t-2} \tag{1}$$

MPC goals is to find the command law yielding a desired response in a finite horizon. For instance, denoting $\mathbf{y}^*_{1:T}$ the desired response for $t = 1 \dots T$, the sought command $\mathbf{u}^*_{1:T}$ satisfies:

$$\mathbf{u}^*_{1:T} = \arg\min_u \left\{ \sum_{t=1}^{T} (y^*_t - \mathcal{F}(\mathbf{y}, \mathbf{u}, t))^2 + \sum_{t=2}^{T} (u_t - u_{t-1})^2 \right\} \tag{2}$$

where the first term enforces the fact that the actual trajectory of the agent matches the desired one, and the second term serves as a regularizing term, requiring the command to be as smooth as possible. MPC is widely used for the control of industrial complex systems (see [38], [39] among many others) and DMPC provides control on large scale systems [11], though it faces some theoretical and algorithmic issues. Typically, in the general case of non-linear behavioral models, Eq. 2 cannot be solved in closed form and resorts to numerical approximate optimization. Secondly, the $\mathcal{F}$ model itself might be ridden with uncertainties, particularly when considering non-deterministic systems [40]. Thirdly, MPC fails computationally when the system is too large. In this case MPC is replaced by DMPC. Nevertheless the deployment of DMPC should consider the computational capacity of the agent.

Recent approaches build upon i) acquiring empirical data reporting the system behavior under various command laws; ii) using supervised machine learning (e.g. neural nets) to approximate the general behavioral law from these empirical data; and iii) approximating the sought command law by minimizing Eq. 2 based on the trained model [38], [39].

a) Discussion: The shortcomings of MPC can be analyzed as follows: On the one hand, i) in order for model $\mathcal{F}$ to be very accurate, it must be trained from extensive data; ii) a very accurate $\mathcal{F}$ might define a hard optimization landscape whenever complex systems are considered. On the other hand, an approximate $\mathcal{F}$ model can easily be learned; but it leads to a very sub-optimal command law.

After the above analysis, the most severe issue for ML-based MPC approaches [38], [39] is to gather training examples and refine model $\mathcal{F}$ in regions where there exists but a small margin between optimal and sub-optimal commands (with respect to the sought target behavior). If such critical regions are not properly sampled in the training set, the learned model is prone to harmful mistakes, and therefore cannot support an effective optimization process.

From a machine learning perspective, the difficulty is twofold: firstly, such critical regions can hardly be identified *a priori*; secondly and most importantly, such critical regions might be hard to sample (e.g. have a very small measure). When this is the case, great amounts of empirical data are needed to gather sufficient evidence.

B. An iterative learning strategy

The proposed IL4MRC approach addresses the above difficulties under the following assumptions:

- Assuming that the property is satisfied in the initial state of the system. The goal is to preserve a property of the system. Under this assumption, the desired command law boils down to: "In each time step, select the action most appropriate to avoid violating the property".
  This first assumption implies that the control problem can be tackled in terms of reaction (as opposed to, planning).
- The command or action space noted $\mathcal{U}$ is discrete.
  This assumption makes it easier to verify if the control problem can be solved in all possible destination for the robot.

Note that these assumptions hold true for this specific applications: the property of the system to be preserved is the initial geometric shape. Likewise, the action space of the robot is all the possible direction of the robot represented as a discrete set (e.g., Forward, Backward, Left, Right, Up, Down).

Under these assumptions, IL4MRC proceeds along an iterative 4-step process, starting from an initial random controller $u^{(0)}$. At the i-iteration:

1) trajectories defined as sequence of state and actions $\{(y_t, u^{(i)}(y_t)), t = 1 \dots T\}$ are recorded, where $u^{(i)}(y_t)$ is the action selected in state y_t by the controller learned in iteration $i-1$ (see below);
2) training set $\mathcal{E}_i$ is built, with
   $$\mathcal{E}_i = \{(y_t, u^{(i)}(y_t), \ell_t), t = 1, \dots T\}$$
   where label ℓ_t is positive or negative depending on whether the sought property of the system still holds at step $t+1$;
3) model $\mathcal{F}_i$ is learned from $\mathcal{E}_i$. Denoting $\mathcal{Y}$ (respectively $\mathcal{U}$) the response or state space (resp. the action space),
   $$\mathcal{F}_i : \mathcal{Y} \times \mathcal{U} \mapsto \mathcal{R}$$
   where $\mathcal{F}_i(y, u)$ is expected to be positive if selecting action u in state y leads to satisfy the sought property.
4) controller $u^{(i+1)}$ is defined as:
   $$u^{(i+1)}(y_t) = \arg\min_{u \in \mathcal{U}} \{\mathcal{F}_i(y_t, u)\} \tag{3}$$

Compared to the state of the art, the originality of the IL4MRC approach is twofold. On the one hand, the command law based on Eq. (3) is simple and computationally frugal, with complexity $\mathcal{O}(|\mathcal{U}|)$. On the other hand, the stress is put on visiting the (state, action) space based on the current model and gradually learning where this model needs to be refined. The complexity thus is shifted from the optimization task to the learning task, and from the learning task to the data acquisition task. A main benefit is that the model trained from the data reflects by construction the various sources of uncertainty and biases of the task at hand, either due to command imprecision, or to non-deterministic aspects of the system, or related to the inaccuracies of the current model.

IV. A PROOF OF PRINCIPLE OF IL4MRC: APPLICATION TO QUADROTORS CONTROL

This section presents a proof of principle of IL4MRC,[1] applied to the control of a set of quadrotors. After describing the position of the problem, the algorithmic pipeline (data acquisition phase, training of the model, exploitation of the model) is detailed.

A. Position of the problem

Considering a set of quadrotors with two leaders and several followers, the goal is to gradually build a controller for each follower, knowing that each follower has very minimal sensing capabilities, such as measuring the distance to its neighbors. The controller should enable the follower to autonomously follow the leader, such that all the quadrotors are able to collectively preserve their initial geometric pattern.

The objective of this controller is not to achieve flocking or swarming for the followers. Instead, the objective of this study is to establish a proof of principle for a controller that preserve the initial geometrical pattern. This work can be considered as a first block towards achieving the flocking behavior based on extreme minimal sensing capacity for the studied agent.

The idea behind this study is the cost of the sophisticated sensors. Since it is a key issue for multi-agent system in both terms of energy consumption (to carry e.g. heavy cameras or GPS) and algorithmic complexity (to exploit fine-grained information). Consider a pilot with a set of a cheap COTS (Commercial off-the-shelf) quadrotors with no communication at all among them. Today, it is practically impossible to allow for this pilot to control the entire set of quadrotors at the same time for a specific application. Therefore, to control such a system, the pilot can have direct communication with the two leaders and the remaining followers are equipped with the trained controller and the small sensing capacities.

In the following, each quadrotor is only endowed with WiFi sensors. Such sensors measure the signal strength of the radio link between robots. The distance to a neighbor robot can thus be estimated from the signal strength via a propagation model given by [41]:

$$PL(d, f) = 10\alpha \log_{10}(d) + \beta + 10\gamma \log_{10}(f) + N(0, \sigma)$$

[1]https://github.com/shrit/MagicFlock

where, f is the radio link frequency, d is the direct 3D distance between the transmitter and the receiver. α is related to the increase of the path loss with distance, while γ is related to the increase of the path loss with frequency. β is the offset value, $N(0, \sigma)$ is a gaussian random variable with a standard deviation of σ

It is known that signal strength of the radio link can be noisy, thus to reduce the effect of the noise we apply a first-order infinite impulse response filter known as exponential weighted moving average filter, given by [42]:

$$s_t^f = \begin{cases} s_0 & : t = 0 \\ \alpha s_t + (1 - \alpha) s_{t-1}^f & : t > 0 \end{cases}$$

Where s_t is the signal strength measured at the receiver. s_t^f is the smoothed value by the filter, and $0 < \alpha < 1$ is the smoothing coefficient. The quadrotors only sense the received signal strength, there is no exchange of information between quadrotors.

One might think that these sensors make the problem of maintaining the initial geometrical pattern an algorithmically easy task. Indeed our first attempt was to directly write the algorithm. Complementary experiments (Table 1) however show that the distance traveled by the quadrotor significantly varies depending on its direction. A first lesson learned from these experiments is that coding common sense geometric reasoning based on *in situ* sensor data is a deceptive task. A second lesson is that synthetic data sets (e.g., generated from models) hardly capture the actual behavior of the system.

Formally, two settings are defined to test the scalability of the approach.

a) 3-quadrotor settings: In the first setting, the formation includes two *leaders*, remotely operated using a same random controller and one *follower*. The goal is to train the follower controller which is embedded on the follower and to enable it to maintain the initial shape of the formation, set to a equilateral triangle.

b) 4-quadrotor settings: In the second setting, the formation includes two *leaders* likewise remotely operated using a same random controller; and two independent *followers*. The goal is to train an embedded controller for each one of the true followers, enabling both of them to maintain with the true leaders the initial shape of the formation, set to a parallelogram (rhombus). Both leaders are respectively situated at the extreme north and south of the rhombus.

B. Data acquisition

During the data acquisition, the state of each follower is recorded along a set of episodes. Each episode starts with quadrotors taking off in the initial shape pattern (equilateral triangle in 3-quadrotor, and rhombus in 4-quadrotor). The episode ends when the current geometric pattern of the formation is too far from the initial one (see below).

The action space $\mathcal{U}$ is made of 7 actions: Forward, Backward, Left, Right, Up, Down, NoMove.

At time $t = 0$, the leaders uniformly select a same action u_t^* in $\mathcal{U}$ and keep it constant for 10 time steps; another action

is selected at time $t = 10$ and kept for 10 time steps, and so forth.

At each time step $t \geq 0$, each follower independently and uniformly selects an action u_t in $\mathcal{U}$.

For each follower, its state $\mathbf{y}_t$ at time t is the 2-dimensional vector made of its distance to each one of the two leaders. Eventually, the data set attached to each follower reports 1,500 episodes, where each episode is a varying length sequence $(\mathbf{y}_0, u_0, \mathbf{y}_1, \ldots, \mathbf{y}_t)$ and $\mathbf{y}_t$ is a terminal state iff $\|\mathbf{y}_t - \mathbf{y}_0\| > \epsilon$, with $\epsilon > 0$ a tolerance parameter. On average, 1,500 episodes correspond to 7,000 pairs $u_t, \mathbf{y}_{t+1}$.

C. Forward model

The data set is exploited to learn a forward model, estimating the next state of the quadrotor based on its two last states and actions.

Formally, the data set is decomposed as a set of pairs $(X = (\mathbf{y}_{t-1}, u_{t-1}, \mathbf{y}_t, u_t); Z = \mathbf{y}_{t+1})$ and a mainstream supervised learning algorithm is used to train a function $\mathcal{F}$ such that $\mathcal{F}(X) = Z$ from 80% of the data.

The quality of the learned model $\mathcal{F}$ is estimated on the remaining 20% of the data. Further work will be focused on self-adapting the ϵ tolerance parameter depending on the mean square error of $\mathcal{F}$.

D. IL4MRC *controller*

At production time, model $\mathcal{F}$ is used to support a controller, defined as:

$$u^*_{t+1} = \arg\min_{u \in \mathcal{U}} \left\{ \|\mathbf{y}_0 - \mathcal{F}(\mathbf{y}_{t-1}, u_{t-1}, \mathbf{y}_t, u_t)\| \right\}$$

More precisely, the quadrotor uses its initial state $\mathbf{y}_0$ to define the target property to be preserved (section III-B). Based on its past state and action, and its current state, it determines the best action according to the forward model $\mathcal{F}$, that is, the action more amenable to bring it in the target state.

It is emphasized that each quadrotor is associated to a specific forward model, inducing a specific controller. This strategy is meant to address the fact that actual quadrotors might have slightly different behaviors, e.g. due to fatigue effects.

The quadrotors are operated at production time very similarly as in the data acquisition phase. In each episode, the quadrotors take off, the leaders are randomly operated (with their action persisting for 10 consecutive time steps) and the episode ends when the current geometric pattern of the formation is too far from the target one, that is, when at least one follower is too far from its target state. The difference is that in each time step, each follower executes the action according to its own controller (as opposed to, a random action in the data acquisition phase).

V. EXPERIMENTAL SETTING

This section describes the goal of experiments and the experimental setting used to validate the IL4MRC approach.

A. Goals of experiments

As said, an episode starts with the quadrotors taking off in the target geometric pattern (equilateral triangle in setting 1, and rhombus in settings 2).

The straightforward performance indicator is the average number of time steps this geometric pattern is preserved up to the tolerance ϵ. A high variability of the quadrotors behavior was however observed, due to several factors: the random moves of the leaders; the drifting noise of each quadrotor and the variability of the traveled distance by each quadrotor (see Table I).

A more reliable performance indicator is thus retained, namely the cumulative distribution of the episode length, reporting for each number of time steps t the fraction $z(t)$ of the episodes with a length less than t.

B. Baselines

The performance of IL4MRC is assessed comparatively to two baselines. The simplest baseline is the random controller (as used during the data set acquisition). A refined baseline is based on a k-nearest neighbor. Formally, this baseline exploits the same data set $\mathcal{E}$ as the one used to train the forward model. To each triplet $(\mathbf{y}_{t-1}, u_{t-1}, \mathbf{y}_t)$ is associated its k-nearest neighbors (with $k = 4$ in the experiments), where the distance is set to the Euclidean distance on the state space and the Hamming distance on the action space. Letting $(\mathbf{y}^{(i)}_{t-1}, u^{(i)}_{t-1}, \mathbf{y}^{(i)}_t, u^{(i)}_t, \mathbf{y}^{(i)}_{t+1})$ respectively denote the fragments of trajectories including these nearest neighbors, the selected action is $u^{(i)}_t$ such that it brings the quadrotor in state $\mathbf{y}^{(i)}_{t+1})$ as close as possible to the target state:

$$u_t = \arg\min_{i=1\ldots4} \left\{ \|\mathbf{y}_0 - \mathbf{y}^{(i)}_{t+1}\| \right\}$$

C. Simulation platform

In the first (respectively, second) setting, the system includes three (resp. four) robots of same type, the IRIS quadrotor designed by 3DR[2], with height 0.11m, width 0.47m and weight 1.5 kg.

The quadrotors are simulated using the software in the loop simulation (SITL) [13] integrating the Gazebo simulator [3]. Each quadrotor uses PX4 [4] as an autopilot software.

The velocity of each robot is 1m/s, the duration of one time step and the duration of each action both are 1 second. The take-off altitude is 25 m. The tolerance ϵ on the deviation from the target pattern is 2. Roughly speaking, the pattern is broken if one of the quadrotors stays motionless for 3 time steps while the leaders move.

D. Learning of the forward model

As said, the training data set records 1,500 episodes, totaling circa 7,000 time steps on average. The forward model is implemented as a neural net, using mlpack [43]. The neural architecture is a 2-hidden layers, with 200 neurons on each

[2]https://3dr.com/

[3]http://gazebosim.org/

[4]https://px4.io/

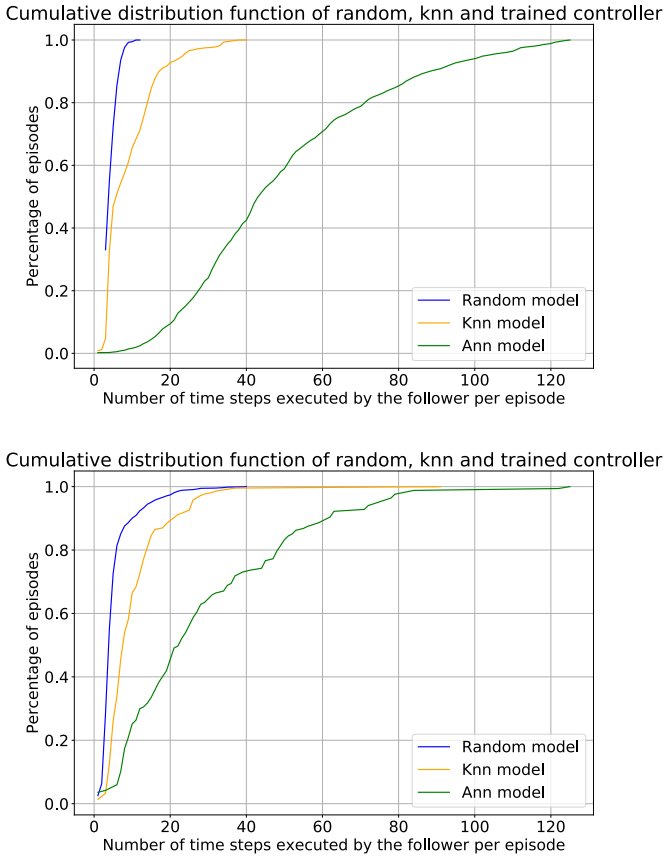

Figure 1. Performance of IL4MRC on the 3-quadrotor setting (top) and the 4-quadrotor setting (bottom), compared to the random and the k-nearest neighbor baselines. For each controller and each time step t on the horizontal axis, is indicated the fraction $z(t)$ of the episodes terminated before t time steps (see text).

layer, with Leaky ReLU as activation function [44]. The training uses Glorot initialization [45], with .5 Dropout and batch size 32; the hyper-parameters are adjusted using Adam [46] with $\beta_1 = 0.9$, $\beta_2 = 0.999$, $\varepsilon = 10^{-8}$, and initial learning rate $\alpha = 0.001$.

VI. Empirical Validation

The performance of IL4MRC is displayed in Fig. 1, reporting the cumulative distribution of the episode lengths in the 3-quadrotor (Fig. 1, top) and 4-quadrotor (Fig. 1, button) settings. The difficulty of the problems is evidenced as the random controller loses track of the leaders after 10 time steps on circa 90% of the episodes in the 3-quadrotor setting, and almost 100% of the episodes in the 4-quadrotor setting. The k-nn controller significantly improves on the random controller: circa 10% of the episodes last more than 20 time steps. Surprisingly, its performances are quite similar in both settings. Complementary experiments (omitted for space limitations) show that the performance of the k-nn can be improved by increasing the size of the data set, although this adversely affects the controller speed and increases the latency among the quadrotors.

Finally, IL4MRC significantly improves on the k-nn controller: 60% (respectively 30%) of the trajectories last more than 40 time steps in the 3-quadrotor (resp. the 4-quadrotor) setting. The enhanced performance of IL4MRC compared to the k-nn can be easily understood from the generalization effect: the k-nn model cannot provide reliable estimates in regions which have not been visited in the training set and more generally its accuracy is limited by the size of the data set. Qn the opposite, under the assumption that the target behavior is sufficiently smooth, the IL4MRC model can estimate the behavior of the quadrotor in regions which have been rarely visited in the training set.

VII. Conclusion

Inspired from the Model Predictive Control setting, this paper presents a new approach for controller design, under two assumptions: the fact that the target behavior can be cast as a property-preserving task on the one hand, and the fact that the action space is discrete on the other hand.

Under these assumptions, the burden of controller design is shifted toward data acquisition: the gathered data enables to learn a simple and short-sighted forward model, and this forward model i) enables the fast optimization of the next action; ii) seamlessly handles the uncertainty about the current state and the actual behavior of the agent.

A proof of principle is presented to illustrate the approach, on the difficult task of maintaining a swarm with neither sophisticated sensors (e.g. GPS or collision detector) nor extensive computational resources. The empirical evidence on this task suggests that a somewhat sophisticated behavior can be achieved by learning offline and exploiting online a basic forward model.

Further research should develop the transfer of the neural controller from the simulated environment to real quadrotors. The distance between the agents will be estimated using the signal strength of any radio link between them, using the appropriate propagation model. A most interesting question concerns the adaptation of the presented approach to learn and adjust a simplified propagation model.

References

[1] S. Russell and P. Norvig, *Artificial Intelligence: A Modern Approach*, 3rd ed. USA: Prentice Hall Press, 2009.

[2] K. H. Wray and S. Zilberstein, "Generalized controllers in pomdp decision-making," in *2019 (ICRA)*, May 2019, pp. 7166–7172.

[3] Y. Mulgaonkar and V. Kumar, "Towards open-source, printable pico-quadrotors," 2014.

[4] Q. Lindsey, D. Mellinger, and V. Kumar, "Construction of cubic structures with quadrotor teams." in *Robotics: Science and Systems*, H. F. Durrant-Whyte, N. Roy, and P. Abbeel, Eds., 2011. [Online]. Available: http://dblp.uni-trier.de/db/conf/rss/rss2011.html#LindseyMK11

[5] (2019, Jan.) Wing project. https://x.company/projects/wing/.

[6] V. Sharma, M. Bennis, and R. Kumar, "Uav-assisted heterogeneous networks for capacity enhancement," *IEEE Communications Letters*, vol. 20, no. 6, pp. 1207–1210, 2016.

[7] C. Zhang and J. M. Kovacs, "The application of small unmanned aerial systems for precision agriculture: a review," *Precision Agriculture*, vol. 13, no. 6, pp. 693–712, Dec 2012. [Online]. Available: https://doi.org/10.1007/s11119-012-9274-5

Actions	forward	backward	left	right	up	down
mean	0.699932	0.713274	0.694962	0.704504	1.02292	1.00244
std dev.	0.00236473	0.00247612	0.00238012	0.00234223	0.00319365	0.00318693

Table I

AVERAGE TRAVELED DISTANCE AND STANDARD DEVIATION DEPENDING ON THE MOVE DIRECTION, OPERATED FOR 1 SECOND WITH SPEED 1 M/S. MOST UNEXPECTEDLY, THE AVERAGE TRAVELED DISTANCE DEPENDS ON THE MOVE DIRECTION, AND THEIR DISTRIBUTIONS ARE STATISTICALLY SIGNIFICANTLY DIFFERENT.

[8] K. P. Valavanis and G. J. Vachtsevanos, *Handbook of Unmanned Aerial Vehicles*. Springer Publishing Company, Incorporated, 2014.

[9] M. Saska, Z. Kasl, and L. Přeucil, "Motion planning and control of formations of micro aerial vehicles," *IFAC Proceedings Volumes*, vol. 47, no. 3, pp. 1228 – 1233, 2014, 19th IFAC World Congress.

[10] M. Saska, T. Krajník, V. Vonásek, P. Vaněk, and L. Přeučil, "Navigation, localization and stabilization of formations of unmanned aerial and ground vehicles," in *2013 ICUAS*, May 2013, pp. 831–840.

[11] L. Dai, Q. Cao, Y. Xia, and Y. Gao, "Distributed mpc for formation of multi-agent systems with collision avoidance and obstacle avoidance," *Journal of the Franklin Institute*, 2017.

[12] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, "Continuous control with deep reinforcement learning," 2015.

[13] F. Furrer, M. Burri, M. Achtelik, and R. Siegwart, *RotorS—A Modular Gazebo MAV Simulator Framework*. Cham: Springer International Publishing, 2016, pp. 595–625. [Online]. Available: https://doi.org/10.1007/978-3-319-26054-9_23

[14] F. Schilling, J. Lecoeur, F. Schiano, and D. Floreano, "Learning vision-based cohesive flight in drone swarms," *CoRR*, vol. abs/1809.00543, 2018. [Online]. Available: http://arxiv.org/abs/1809.00543

[15] ——, "Learning vision-based flight in drone swarms by imitation," *CoRR*, vol. abs/1908.02999, 2019. [Online]. Available: http://arxiv.org/abs/1908.02999

[16] T. Baca, D. Hert, G. Loianno, M. Saska, and V. Kumar, "Model predictive trajectory tracking and collision avoidance for reliable outdoor deployment of unmanned aerial vehicles," in *2018 (IROS)*, Oct 2018.

[17] R. Kralev, R. Mendonca, A. Zhang, T. Yu, A. Gupta, P. Abbeel, S. Levine, and C. Finn, "Learning to reinforcement learn by imitation," 2019. [Online]. Available: https://openreview.net/forum?id=HJG1Uo09Fm

[18] J. Tan, T. Zhang, E. Coumans, A. Iscen, Y. Bai, D. Hafner, S. Bohez, and V. Vanhoucke, "Sim-to-real: Learning agile locomotion for quadruped robots," *Robotics: Science and Systems XIV*, Jun 2018. [Online]. Available: http://dx.doi.org/10.15607/RSS.2018.XIV.010

[19] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. van den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, S. Dieleman, D. Grewe, J. Nham, N. Kalchbrenner, I. Sutskever, T. Lillicrap, M. Leach, K. Kavukcuoglu, T. Graepel, and D. Hassabis, "Mastering the game of go with deep neural networks and tree search," *Nature*, vol. 529, pp. 484–503, 2016. [Online]. Available: http://www.nature.com/nature/journal/v529/n7587/full/nature16961.html

[20] H. Zhang, R. Cao, S. Zilberstein, F. Wu, and X. Chen, "Toward effective soft robot control via reinforcement learning," in *Intelligent Robotics and Applications*, 2017.

[21] A. Y. Ng and S. J. Russell, "Algorithms for inverse reinforcement learning," in *ICML*, ser. ICML '00. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2000, p. 663–670.

[22] S. Lupashin, M. Hehn, M. W. Mueller, A. P. Schoellig, M. Sherback, and R. D'Andrea, "A platform for aerial robotics research and demonstration: The flying machine arena," *Mechatronics*, vol. 24, no. 1, pp. 41 – 54, 2014.

[23] C. Forster, S. Lynen, L. Kneip, and D. Scaramuzza, "Collaborative monocular slam with multiple micro aerial vehicles," *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 3962–3970, 2013.

[24] L. Busoniu, R. Babuska, and B. De Schutter, "A comprehensive survey of multiagent reinforcement learning," *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, vol. 38, no. 2, pp. 156–172, 2008.

[25] K. Hwang, W. Jiang, and Y. Chen, "Model learning and knowledge sharing for a multiagent system with dyna-q learning," *IEEE Transactions on Cybernetics*, vol. 45, no. 5, pp. 978–990, 2015.

[26] P. Valencia, P. Lindsay, and R. Jurdak, "Distributed genetic evolution in wsn," ser. IPSN '10. Association for Computing Machinery, 2010. [Online]. Available: https://doi.org/10.1145/1791212.1791215

[27] S. Luke, "Genetic programming produced competitive soccer softbot teams for RoboCup97." Morgan Kaufmann, 22-25 Jul. 1998, pp. 214–222.

[28] P. Stone and D. McAllester, "An architecture for action selection in robotic soccer," ser. AGENTS '01. New York, NY, USA: Association for Computing Machinery, 2001. [Online]. Available: https://doi.org/10.1145/375735.376320

[29] A. Molchanov, T. Chen, W. Honig, J. A. Preiss, N. Ayanian, and G. S. Sukhatme, "Sim-to-(multi)-real: Transfer of low-level robust control policies to multiple quadrotors," *2019 IROS*, Nov 2019.

[30] A. Giusti, J. Guzzi, D. Ciresan, F.-L. He, J. P. Rodriguez, F. Fontana, M. Faessler, C. Forster, J. Schmidhuber, G. Di Caro, D. Scaramuzza, and L. Gambardella, "A machine learning approach to visual perception of forest trails for mobile robots," *IEEE Robotics and Automation Letters*, 2016.

[31] Y. Zhu, R. Mottaghi, E. Kolve, J. J. Lim, A. Gupta, L. Fei-Fei, and A. Farhadi, "Target-driven visual navigation in indoor scenes using deep reinforcement learning," *CoRR*, vol. abs/1609.05143, 2016.

[32] L. Pinto and A. Gupta, "Supersizing self-supervision: Learning to grasp from 50k tries and 700 robot hours," *CoRR*, vol. abs/1509.06825, 2015. [Online]. Available: http://arxiv.org/abs/1509.06825

[33] F. Sadeghi and S. Levine, "(cad)^{2}rl: Real single-image flight without a single real image," *CoRR*, vol. abs/1611.04201, 2016. [Online]. Available: http://arxiv.org/abs/1611.04201

[34] A. H. . A. P. Schoellig, "Distributed iterative learning control for multi-agent systems," *Autonomous Robots*, 2019.

[35] E. V. Tolstaya, F. Gama, J. Paulos, G. J. Pappas, V. Kumar, and A. Ribeiro, "Learning decentralized controllers for robot swarms with graph neural networks," *ArXiv*, vol. abs/1903.10527, 2019.

[36] J. Rawlings and D. Mayne, *Model Predictive Control: Theory and Design*, 01 2009.

[37] S. Piché, J. Keeler, G. Martin, G. Boe, D. Johnson, and M. Gerules, "Neural network based model predictive control," in *NIPS 99*, ser. NIPS'99. Cambridge, MA, USA: MIT Press, 1999, pp. 1029–1035. [Online]. Available: http://dl.acm.org/citation.cfm?id=3009657.3009802

[38] I. S. Mohamed, S. Rovetta, T. D. Do, T. Dragicević, and A. A. Z. Diab, "A neural-network-based model predictive control of three-phase inverter with an output lc filter," *IEEE Access*, vol. 7, pp. 124 737–124 749, 2019.

[39] J. Carius, F. Farshidian, and M. Hutter, "Mpc-net: A first principles guided policy search," 2019.

[40] D. Mayne, "Nonlinear model predictive control:challenges and opportunities," in *Nonlinear Model Predictive Control*, F. Allgöwer and A. Zheng, Eds. Basel: Birkhäuser Basel, 2000, pp. 23–44.

[41] I. T. Union, "Propagation data and prediction methods for the planning of outdoor radiocommunication systems and radio local area networks in the frequency range 300 mhz to 100 ghz," Tech. Rep. P.1411-9, 2017.

[42] R. G. Brown, *Smoothing, Forecasting and Prediction of Discrete Time Series*. Dover Publications; Dover Phoenix Ed edition, 2004.

[43] R. R. Curtin, J. R. Cline, N. P. Slagle, W. B. March, P. Ram, N. A. Mehta, and A. G. Gray, "Mlpack: A scalable c++ machine learning library," *J. Mach. Learn. Res.*, vol. 14, no. 1, pp. 801–805, Mar. 2013.

[44] V. Nair and G. E. Hinton, "Rectified linear units improve restricted boltzmann machines," in *ICML-10, June 21-24, 2010, Haifa, Israel*, 2010, pp. 807–814.

[45] X. Glorot and Y. Bengio, "Understanding the difficulty of training deep feedforward neural networks," in *AISTATS'10*, 2010.

[46] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," 2014, cite arxiv:1412.6980Comment: Published as a conference paper at the 3rd ICLR, San Diego, 2015.