LSTM Path-Maker : a new LSTM-based strategy for Multiagent Patrolling *
LSTM Path-Maker : une nouvelle stratégie pour la patrouille multiagent basée sur l'architecture LSTM

Mehdi Othmani-Guibourg[1,2] Amal El Fallah-Seghrouchni[2] Jean-Loup Farges[1]

[1] ONERA, 31000, Toulouse, France
[2] Sorbonne Université - Faculté des Sciences
CNRS, UMR 7606, LIP6,
F-75005, Paris, France

{prénom}.{nom}@onera.fr
{prénom}.{nom}@lip6.fr

Résumé

Depuis plus d'une décennie, la tâche de la patrouille multiagent a attiré l'attention de la communauté multiagent de manière croissante, en raison de son grand nombre d'applications potentielles. Cependant, les algorithmes basés sur des méthodes d'apprentissage profond pour traiter cette tâche sont à ce jour peu développés. Dans cet article, nous proposons d'intégrer un réseau de neurone récurrent à une stratégie de patrouille multiagent. Ce faisant, nous avons proposé un modèle formel de stratégie d'agent basée sur l'architecture LSTM, que nous avons nommé LSTM-Path-Maker. Le réseau LSTM est entraîné sur des traces de simulation d'une stratégie coordonnée et centralisée, puis embarqué dans chaque agent en vue de patrouiller efficacement sans communication. Enfin, cette nouvelle stratégie basée sur l'architecture LSTM est évaluée en simulation et comparée d'une part à une stratégie coordonnée et d'autre part à une stratégie réactive. Les résultats préliminaires indiquent que la stratégie proposée est meilleure que la stratégie réactive.

Mots Clef

Systèmes multiagents, Réseaux de neurones artificiels, Réseaux récurrents à mémoire court et long terme

Abstract

For over a decade, the multi-agent patrol task has received a growing attention from the multi-agent community due to its wide range of potential applications. However, the existing patrolling-specific algorithms based on deep learning algorithms are still in preliminary stages. In this paper, we propose to integrate a recurrent neural network as part of a multi-agent patrolling strategy. Hence we proposed a formal model of an LSTM-based agent strategy named LSTM Path Maker. The LSTM network is trained over simulation traces of a coordinated strategy, then embedded on each agent of the new strategy to patrol efficiently without communicating. Finally this new LSTM-based strategy is evaluated in simulation and compared with two representative strategies : a coordinated one and a reactive one. Preliminary results indicate that the proposed strategy is better than the reactive.

Keywords

Multiagent Systems (MAS), Artificial Neural Networks (ANN), Long Short-Term Memory (LSTM)

1 Introduction

The generic task of patrolling is by nature conveniently well-suited for being shared in space and time by several agents. There is a wide variety of tasks that may be formulated as particular multi-agent patrolling (MAP) problem. As a concrete example, the task consisting in monitoring an area by a swarm of drones faces with the problem of coordinating them to patrol that area. Area monitoring is useful as part of crises, for example in order to detect a start of fire in a forest, but also to provide an alert, either to save people or to detect the presence of intruders as part of a complex humanitarian mission in a conflict area.

A fully-fledged feature of patrolling and other complex systems is the difficulty to derive analytical results from their system-wide equations. Thereby it appears that the only method enabling to predict their behaviour is to simulate the local interactions of their components : this is exactly the main purpose of agent-based simulation. Thus, the quality of a patrolling strategy is evaluated in simulation by using different measures, each one measuring a specific property of distributions of visits generated by strate-

*Paper presented at the 52nd Hawaii International Conference on System Sciences (HICSS52 2019), titre, résumé et mots-clés en français ajoutés.

gies. Informally, it is consensual that a good strategy is one that minimises the time lag between two passages on the same place and for all places.

For over fifteen years different types of agent strategy for the (MAP) were proposed : centralised [1], emergent [1], idleness-based [1], heuristic (idleness and distance) with pathfinding [2], hamiltonian-cycle-based [3], TSP-heuristic-based [4], reinforcement-learning-based [5] and even auctions-based [6] strategies. In this context, Almeida et al. [2] defined two main types of agent : reactive agents that act only according to their perception, and cognitive agents that can pursue a goal.

Until now, as part of the cooperative multi-agent learning, few works concentrate on the problematic of using Artificial Neural Networks (ANNs) for the multi-agent patrolling. For example, a few of these studied non-hierarchical neural network-based methods for planning a complete coverage patrolling path where each neuron encodes a specific region of the space, such as Guo et al. [7] or even a cooperative multi-agent learning where each robot is endowed with a neural network directly connected to nodes of others robots' internal neural network whose weights of connections are evolved, with D'Ambrosio et al. [8]. However, none tackles the advantages that may be afforded by deep artificial neural networks in order to outperform the previous strategies. *This paper thereupon proposes the use of the ANN architecture Long Short-Term Memory (LSTM) for the multi-agent patrolling problem. The Recurrent Neural Networks (RNN), as machines to learn temporal series, are well adapted to this problem to the extent that they can be viewed as a temporal decision problem. In this way, a new strategy based on the LSTM architecture is introduced where the ANN is used as a path generation device by non-communicating agents to navigate as optimally as possible through the graph. To that end each neural network architecture is trained offline over data generated for this purpose, then embedded in the agents which will use it to select the next node to visit with respect to the previous ones. Finally, the performances are evaluated according to the usual evaluation criteria used until now in this field of study.*

The Section 2 presents the background on the multi-agent patrolling and the LSTM networks useful to understand proposed developments as well as the previous works using the ANNs for the MAP. Then, Section 3 introduces LSTM Path-Maker (LPM), the new strategy for the multi-agent patrolling based on the LSTM architecture. In Section 4 the learning results are analysed and and the new strategies evaluated. Finally, Section 5 draws some conclusions, shows certain boundaries for this new strategy and indicates directions for further works.

2 Background

This section presents the background on multi-agent patrolling and the LSTM architecture.

2.1 Multi-agent patrolling

Formal definition. The MAP model consists formally of a society of agents noted $\mathbf{A}$, able to move in an environment with the same mobility parameters, and a graph noted $\mathbf{G} = (\mathbf{V}, \mathbf{E})$ as an abstraction representing a discretisation of the area to patrol. Here, $card(\mathbf{V}) = N$ and $\mathbf{V} = \{1, .., N\}$ is the set of nodes identified by their indexes and standing for the places to visit. $\mathbf{E}$, which is included in the set of 2-element subsets of $\mathbf{V}$, is the set of edges of $\mathbf{G}$ accounting for the paths between the places. With each edge $\{v, w\}$ corresponds a *transit time* $c_{v,w}$ representing the travel time of the edge $\{v, w\}$. At the beginning of an instance of a patrol task, agents are positioned on nodes of $\mathbf{G}$. To each node is associated a dynamic variable named *idleness*, indicating the time elapsed since it has not been visited by any agent[4]. The idleness of a node v at time t, noted $i_t(v)$, is defined as being the amount of time elapsed since that node has received the visit of an agent. The idleness of all nodes at the beginning of the patrolling task is set to 0. Finally, each time an agent arrives at a node v, it shall decide, among the edges including v, namely all the edges $\{v, w\}$, the next edge to travel.

Strategies. A *strategy* of agent is an information processing method, or algorithm, allowing each agent to take a decision each time it arrives at a node. In the MAP, whatever the strategy considered, each agent intends actions based on its appropriated perceptions from the environment and its knowledge about idlenesses of nodes. Among the wide family of strategies, two are relevant for this work as representative strategies : *Conscientious Reactive* (CR) and *Heuristic Pathfinder Cognitive Coordinated* (HPCC).
The algorithm of CR is to select the next node to visit as the one with the highest idleness in its neighbourhood. There is no communication between agents : idlenesses are estimated by each agent on the basis of its own path. CR can be thought of as a good representative and thereby a comparison strategy for the reactive and decentralised ones. Note that for CR as well as for any decentralised strategy, the considered idlenesses are estimated by each agent from its own visits to nodes. This estimation is an overestimation of the actual value of idlenesses because visits of other agents are neglected.
For HPCC, there is a perfect communication between agents : idlenesses are estimated by a coordinator on the basis of all paths of agents. The decision process includes two steps :

- — selection of a target node that is not necessary in the neighbourhood,
- — computation of a path between the current position of the agent and the target node previously selected.

The selection of the target node takes into account not only the normalised idleness, but also the normalised *time to go* of a candidate goal node from the agent's current position. The time to go between two nodes of $\mathbf{V}$ corresponds here to the shortest path between these two nodes. Idleness and time-to-go are normalised by scaling from 0 to 1. A zero

normalised value is attributed to a the maximum idleness - encouraging the agent to traverse nodes with high idleness, since the objective function will be minimized - whereas a value equal to 1 is attributed to the minimum idleness. Intermediary values are calculated by means of proportions as shown in **(1)** :

$$\begin{aligned} &\text{If } min_{v\in\mathbf{V}}\{i_t(v)\} \neq max_{v\in\mathbf{V}}\{i_t(v)\}, \forall v_0 \in \mathbf{V} \\ &\bar{i}_t(v_0) = \frac{max_{v\in\mathbf{V}}\{i_t(v)\} - i_t(v_0)}{max_{v\in\mathbf{V}}\{i_t(v)\} - min_{v\in\mathbf{V}}\{i_t(v)\}} \end{aligned} \tag{1}$$

where $i_t(v)$ and $\bar{i}_t(v)$ are the global and normalised idleness, respectively.
Normalised time to go is calculated similarly. For that purpose, at the minimum time to go is attributed a zero normalised value - encouraging the agent to traverse edges with short distance, since the objective function will be minimized - whereas at the maximum time to go is attributed a value equal to 1. Intermediary values are calculated by means of proportions as shown in **(2)** :

$$\begin{aligned} &\forall d(v_0, v) \text{ a time-to-go from } v_0 \text{ to } v, \\ &\bar{d}(v_0, v) = \frac{d(v_0, v) - min\{d\}}{max\{d\} - min\{d\}} \end{aligned} \tag{2}$$

where $max\{d\}$ and $min\{d\}$ are the maximum and the minimum time-to-go respectively, over all the $v, w \in \mathbf{V} : v \neq w$.
Finally, for an agent at the position v_0 at time t, the values associated to nodes are given by **(3)** :

$$\begin{aligned} &\forall v \in \mathbf{V}, val_{r_H}(v, t) = \\ &\qquad r_H \times \bar{i}_t(v) + (1 - r_H) \times \bar{d}(v_0, v) \end{aligned} \tag{3}$$

where the weighting factor $r_H \in [0, 1]$ must be chosen by the strategy designer. Minimising the node values according to that expression i.e. selecting the nodes with the minimum value, allows agents to visit nearby nodes with higher idleness first and foremost. Moreover, there is a mechanism forbidding the coordinator to select nodes that are currently assigned to other agents.
The path computation takes into account the idleness of the nodes between the current location and the goal to compute the best path leading there. For that, it weights the edges as shown in **(4)** :

$$\begin{aligned} &\forall e \in \mathbf{E} : e = \{v, w\}, \\ &\qquad c_{r_P}(e) = r_P \times \bar{i}_t(w) + (1 - r_P) \times \bar{c}_{v,w} \end{aligned} \tag{4}$$

where the weighting factor $r_P \in [0, 1]$ must be chosen by the strategy designer. In that case, it is the normalised transit time $\bar{c}_{v,w}$ of edge and not the normalised time-to-go $\bar{d}(i, j)$ of path that is used to value edges :

$$\forall \{v, w\} \in \mathbf{E}, \bar{c}_{v,w} = \frac{c_{v,w} - min\{c\}}{max\{c\} - min\{c\}} \tag{5}$$

where $max\{c\}$ and $min\{c\}$ are the maximum and the minimum transit times, respectively.
Minimising the edge weights according to that expression allows agents as well to visit nearby nodes with higher idleness first and foremost.
HPCC as a communicating, fully-informed, coordinated, and thereby centralised strategy is one of the best online - namely without pre-calculation of paths - strategy. It can be then regarded as a representative and thereupon a comparison strategy for the coordinated and centralised ones. Note also that HPCC uses actual idlenesses because visits of nodes by all agents are analysed by the coordinator in a centralised manner.

Evaluation criteria. Sampaio et al. [9] introduced evaluation criteria, relevant to establish aggregation measures not based on idleness but on the intuitive concept of *interval between visits* to the node. In this class of evaluation criteria, the size of intervals between visits at each node is calculated by registering the value of idleness just before each visit by an agent. All intervals for all nodes are used to make an aggregated calculation. The two interval-based evaluation criteria we selected are the *Mean Interval* (MI) and the *Quadratic Mean Interval* (QMI), the mean and the root mean square respectively, on all intervals between visits of a mission execution. QMI as quadratic mean, takes better into account the difference of time interval between the nodes and thus, measures the tendency of nodes to be equitably visited through a simulation run.
In order to better evaluate the contribution of each agent when the population size varies, these evaluation criteria are normalised by multiplying values by the number of agents.

2.2 LSTMs

Recurrent Neural Networks (RNNs) are neural networks that process an input sequence one element at a time, maintaining in their hidden units - neurons in the *hidden layers* - a *state vector* called *hidden state*, containing information about the history of the sequence's past elements. Each output of the hidden units h_t, depends upon the hidden state h_{t-1}. This hidden state can be viewed as a *memory*. Indeed, adding memory to a neural network allows to process information of the sequence itself : the sequential information is preserved in the hidden state that enable to find correlations between events separated by several time steps. This memory is contained in the *hidden layers* which have a *feedback loop*, and therefore they constitute *recurrent layers*.
LSTM are a special kind of RNN introduced and designed to take into account long-term dependencies. They have the same general chain structure as the RNNs except that the repeating module has a different structure as shown in **Fig. 1**. In the first place, as stated by Hochreiter et al. [10], an LSTM network was a RNN with one input layer, one fully self-connected hidden layer containing purpose-built *memory cells*, *gate units*, and an output layer. This memory unit corresponds to a neuron with a recurrent self-

connection. Thereby a cell referred originally to an object with a single scalar output. The activations of those neurons within the memory units constitute the *state* noted c_t, sometimes called *cell state*, of the LSTM network.

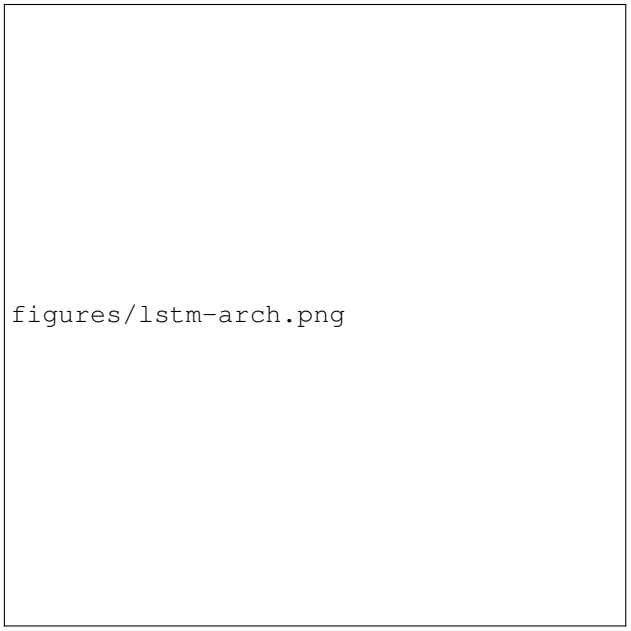

FIGURE 1 – Layered LSTM unit : the core composant of the LSTM architecture.

As stated by Graves et al.[11] an *LSTM layer* consists of a set of recurrently connected blocks, known as *memory blocks*, which in turn consists of cells. One cell, as a neuron, outputs one scalar. Originally, each memory block has contained one or more layered recurrently connected neurons called memory cells and sharing the same three multiplicative units : i_t the *input gate*, o_t the *output gate* and f_t the *forget gates*, i.e. all the cells of a memory block are connected to the same gate units. The gate units provide continuous analogies of write, read and reset operations for the cells. A memory block of size 1 is then a simple memory cell[10] connected to $tanh$ activations. These blocks, can be thought of as a differentiable version of the memory chips in a digital computer. In doing so, it follows the network can only interact with the cells via the gates. Besides, the memory block and the gates form the *LSTM unit* as shown in the **Fig. 1**, which corresponds to a repeating module. The state is thereupon, the memory accumulated by the LSTM through time by using its forget, input and output gates. However, unlike the base RNN model in which it cover the same concept, the cell state c_t must here not be confused with the hidden state h_t, the former being the cell output while and the latter the output of the hidden layers.

Also, it should be emphasised that the hidden state, respectively the cell state, noted h_t, respectively c_t, of an LSTM network, must be distinguished from the hidden state, respectively the cell state, of the layer l (for a multi-layer LSTM) noted h_t^l, respectively c_t^l.

For some years and hitherto, most implemented LSTM architectures contain only one cell in their LSTM units. The LSTM units of a same layer can thereupon be "layered" into only one LSTM unit where for all t a time step, i_t, f_t, o_t and c_t, the input gate, forget gate, output gate and cell activation turn into vectors with the same size as the hidden vector h_t ; hence the element-wise multiplication $*$. In that context, an LSTM layer can be viewed as a vectorial LSTM unit and thereby the vectorial cell and gates compose a layer. It follows that defining the size of a layer's cell defines that of its memory cell block and that of its hidden state in cascade.

The hidden state output from an LSTM layer l is then computed from the following composite function :

$$i_t^l = \sigma(W_{xi}^l\, x_t^l + W_{hi}^l\, h_{t-1}^l + b_i^l) \tag{6}$$

$$f_t^l = \sigma(W_{xf}^l\, x_t^l + W_{hf}^l\, h_{t-1}^l + b_f^l) \tag{7}$$

$$o_t^l = \sigma(W_{xo}^l\, x_t^l + W_{ho}^l\, h_{t-1}^l + b_o^l) \tag{8}$$

$$c_t^l = f_t^l * c_{t-1}^l + i_t^l * \tanh(W_{xc}^l\, x_t^l + W_{hc}^l\, h_{t-1}^l + b_c^l) \tag{9}$$

$$h_t^l = o_t^l * tanh(c_t^l) \tag{10}$$

The parameters of an LSTM layer that must be learned for a layer l are thereby :

— $W_{xi}^l, W_{hi}^l, W_{xf}^l, W_{hf}^l, W_{xo}^l, W_{ho}^l, W_{xc}^l$ and W_{hc}^l

— b_i^l, b_f^l, b_o^l and b_c^l

The structure corresponding to several memory blocks in a layer l can be derived from its more general architecture by setting to 0 the elements of W_{hi}^l, W_{hf}^l, W_{ho}^l which are not block-diagonal.

Deep LSTMs combine the multiple levels of representation that have proved so effective in deep networks with the flexible use of long-range context that empowers RNNs. The architecture of the deep LSTMs is the same as that presented previously apart from the fact that there are several LSTM layers.

2.3 Related works

Few works addressed the problematic of the use of ANNs in the context of the MAP. Among related works, Guo et al. [7] studied the use of ANN-based methods for planning a complete coverage patrolling path. In that work, the area to patrol is discretised into fixed radius disks that can be thought of as nodes to visit. Then, each neuron, as a state variable, represents a region activated negatively or positively in function of either the presence of obstacle, or the absence of a visit by the patrol, respectively. Finally, the activities of all the neurons compose a dynamic landscape such that the non-visited regions globally attract the robot in the entire space, and the obstacle locally repel the robot to avoid collisions. Even though being an original and interesting approach, the type of neural network used in this work is not relevant to our problematic laid down. Indeed, the latter consists of learning temporal sequences

which corresponds here to paths in the graph. Also, in that work only one neural network was used concomitantly by all agents, while in our current framework agents do not communicate, and instead, act in a decentralised way.

D'Ambrosio et al. [8] developed a new communication scheme they called the *hive brain*, as part of the cooperative multi-agent learning. In this scheme, each robot is endowed with a neural network directly connected to nodes of others robots' internal ANN, whose weights of connections are evolved. As stated by the authors, this technique is drawn from an interesting physical phenomenon called *odd sympathy* [12], which is the tendency of pendulum clocks to synchronise when mounted near each other due to a small amount of physical information transferred between the pendulums. Thereby they elaborated the hive brain in an analogical way where the robots learn to synchronise by training their respective ANN in a robot simulator; the training is performed here using an evolutionary algorithm. In our perspective, this work presents the same problem than the foregoing, namely the implicit use of communications in the simulator between agents to feed the *brain* of one agent from another one. However, it inspired our new strategy of agent to the extent that each agent embeds its individual ANN. Also, although in our work agents are currently embedded with the same trained ANN whose the parameters remain unchanged during a mission execution, in a not too distant future it will bee valuable to draw from this work to synchronise the agents' ANNs' state and thereby improve our new strategy's performances.

Finally, the works of Sales et al. [13] is also related to our problematic to some extent. Indeed, they developed an autonomous patrolling system composed of four intelligent robots that can freely move through an indoor environment and detect intruders. The robots are endowed with a localisation/navigation system composed of an ANN used in combination with a Finite State Machine (FSM), whose the states correspond to the key features of the environment. The FSM associates a sequence of actions to execute with a sequence of states. The ANNs process the sensors data to identify and classify the FSM states (current and transitions), and to determine the actions to perform. After being trained offline to identify the key features of the environment such as corridors, intersections and turns, they are fed into data obtained from robots' sensors, then they output the FSM states. From this work we retained the method to train the ANNs offline in order to set them for a specific mission leading agents to navigate as efficient as possible without communicating. Lastly, in this work, each robot calculates the shortest path by using A* to reach the intruder's position when detected taking into account its teammates, while in ours, the network is used to select the next node in the neighbourhood with respect, on the one hand, to the previous ones, and on the other hand, to what was learned during the offline training stage.

3 An LSTM-based strategy

This section presents our contribution, that is LPM, a new LSTM-network-based decentralised agent strategy, which learns to navigate the nodes composing the area to patrol, from series of histories collected upon numerous simulation executions of a fully-informed and coordinated strategy : the HPCC strategy. The first assumption has been that if agents learn in average to behave in the same way as the coordinator, which is fully-informed, then they may approach performances reached by the coordinated strategy. The main goal of this work is to use LSTM networks to perform that.

3.1 Formal definition

The LPM strategy is an ANN-based strategy : the decision-making process is carried out by means of an LSTM network which outputs the next node from the current node provided as input of the network. This strategy can be thought of as a reactive strategy using an artefact for guidance through the area to patrol, such as a compass, which takes implicitly into account the idleness of nodes and the agents' positions. In our context, the temporal series representing the successive visited nodes by an agent is called a *path*. Any vertex of a path, has for subsequent vertex one of its neighbours.
For a given scenario, the LSTM network temporally learns the next node to visit v_{t+1} from a *model strategy*, according to the previous ones $v_t, v_{t-1}, ..., v_0$ in the path and that for all paths : each path, as a temporal series accounting for the path of an agent over the graph, is fed into the network node after node. It follows that, with defining f as the decision procedure of the model strategy - and thereby the strategy itself -, the LSTM network of the scenario that approximates f can be defined as follows :

Let $I_t = \{i_t(v_0), i_t(v_1), ..., i_t(v_N)\}$ being the set of shared idlenesses at the time t and v_a the node from which a next node to visit, noted $\bar{v}$, must be selected as a decision process, by an agent a. Then, the next node to visit $\bar{v}$ will be selected from the procedure f, the *requesting node* at the time t $v_t \in \mathbf{V}$ which corresponds to the node visited by an agent a at the time $t \in \mathbf{T}$, and the set of shared idlenesses I_t such as :

$$\forall t \in \mathbf{T}, \bar{v} = f(v_t, I_t) \tag{11}$$

Thus, with considering $\tilde{f}$ the vertex-purpose-LSTM-network-based decision procedure as a function approximator of f, and v_t we have :

$$\forall t \in \mathbf{T}, v_{t+1} = \tilde{f}(v_t, ..., v_1) \tag{12}$$

This equation pertains to the formal definition of an LSTM network : each output depends upon the previous outputs.
Let N the number of nodes in the graph. With the aim of feeding the LSTM network with the most appropriate and relevant information about the nodes, each node has been

encoded as a N-dimensional one-hot vector : for the vertex v_i, all the coordinates of the vector will be set to 0 except the i-th coordinate which will be set to 1. The output of the network thereupon is an N-dimensional vector whose the values are in $[0, 1]$; these values can be regarded as probabilities. Thus, to ensure that all values are positive and their sum equal to one, the output layer of the networks is a softmax layer. The node represented by the maximum output vector's coordinate should be selected as the next one to visit.

Let (L, H) the *profile of parameters* of an LSTM architecture so that L and H stand for the number of layers and the number of hidden units (or memory cells) per layer respectively, of a given LSTM architecture. Formally, by defining $b : \mathbf{V} \rightarrow \mathbf{V_{bin}}$ as being the function mapping all the indices of nodes into their one-hot representation, the proposed architecture can then be described with :

$$x_t^1 = b(v_t) \tag{13}$$

$$\text{If } L > 1, \forall l \in \{1, ..., L-1\}\ x_t^{l+1} = h_t^l \tag{14}$$

$$L_{net} = softmax(W \cdot h_t^L) \tag{15}$$

where $dim(h) = H$ and W is a $card(V) \times H$-matrix of parameters.

Finally, upon training stage's completion, each LPM agent will be endowed with the same parametrised LSTM network.

3.2 Network training

The training of the LSTM network is performed from logged paths of any high-performance strategy f. Generally, the high-performance strategies make use of communications and centralised decision-making process. The purpose here, is then to approach the performances of these strategies without communicating and thereupon *distributing* and *decentralising* the decision-making process. Indeed, for example in the context of a drones' reconnaissance mission or even silent bots penetrating a network, communications may be impossible or discouraged. Such a strategy to learn will be called the *model strategy* or simply the *model* if that does not lead to confusion.

For each *scenario* $\{f, \mathbf{G}, N_a\}$, also called *simulation configuration* or simply *configuration*, with N_a the number of agents, whether it does not cause any confusion, the LSTM network is first pre-trained with the purpose of learning the topology i.e. the structure of the graph representing the area. Thereafter, the network is trained over all the paths retrieved from the executions of configuration for $\{f, \mathbf{G}, N_a\}$, so that it learns to output with the highest probability the next node to visit in the path. The process described here can be thought of as performing sequence modelling where the sequence is a path of nodes ; here the sequence modelling corresponds to a *path generation*.

As aforementioned in the **Subsection 3.1**, the network's output layer is a softmax layer. It can be interpreted as a probability distribution. Thus path generation aims at learning a probability distribution over paths by minimising the cross-entropy of a model given a set of N training sequences of length T :

$$\min_{\boldsymbol{\theta}} -\sum_{n=1}^{N}\sum_{t=1}^{T} log\ p(v_t^n | v_1^n, ..., v_{t-1}^n; \boldsymbol{\theta}) \tag{16}$$

where $\boldsymbol{\theta}$ is the set of the model's parameters, whose the dimension is $dim(H) = 4(2L - 1)H^2 + (4L + 5Card(V))H$, and p is the predicted probability for the current element of the observed sequence (v_t^n).

3.3 Decision

Generally, despite of the pre-training stage, the network may output the highest probability for a node that is not in the neighbouring of the one given in input. In doing so, the decision shall be made only among the output probabilities standing for the neighbour nodes. It follows that each time the one-hot vector of the current vertex is presented to the network, the decision procedure $\tilde{f}$ concerning the next node to visit consist of selecting the next node among the neighbours of the current vertex with the maximum probability. This can be mathematically rewritten as bellow :

Let :

- $\mathbf{V_{bin}} \subset \{0, 1\}^{card(V)}$ be the set of nodes formatted into one-hot vectors,
- $L_{net} : \{0, 1\}^{card(\mathbf{V})} \rightarrow [0, 1]^{card(\mathbf{V})}$ the function represented by the LSTM network used here,
- $Ng : [0, 1]^{card(\mathbf{V})} \times \mathbf{V} \rightarrow [0, 1]^{card(\mathbf{V})}$, the function setting to zero the values of the coordinates not corresponding to the neighbours of a given node's one-hot vector.

Then,

$$\begin{aligned} &\forall t \in \mathbf{T}, \forall v_t \in \mathbf{V_{bin}}, \\ &\quad v_{t+1} = argmax(\ Ng(L_{net}(b(v_t)), v_t)\ ) \end{aligned} \tag{17}$$

It then follows that :

$$\begin{aligned} &\forall t \in \mathbf{T} :\ v_t \in \mathbf{V}, \\ &\quad \tilde{f}(v_t, ..., v_1) = argmax(\ Ng(\ L_{net}(b(v_t)), v_t\ )\ ) \end{aligned} \tag{18}$$

with $\tilde{f}$ depending upon $v_1, ..., v_t$ due to their relevant features being stored in the memory of L_{net}.

An alternative way to use the output of the LSTM would be to compute a path leading to the node output by the network and to provide the first node of this path as v_{t+1}. This procedure was omitted because the network learns to predict only neighbour nodes.

Finally, the first experiments showed that using LSTM network as it stands, tends to lead agents to converge indefinitely towards a small set of nodes, leaving thereupon others nodes non-visited until the end of the execution. In doing so, the decision procedure was slightly improved : henceforth, with the aim to make the system more robust, the

next vertex to visit from the current one is randomly selected according to the distribution of probability output by the LSTM network, normalised over the neighbourhood of the current vertex using the Bayes' theorem over the distribution of neighbours. This new procedure enables therefore to add a little randomness in the decision process when selecting the next node in the neighbourhood, leading to increase the robustness of the system, and thereby to avoid agents to visit only a restricted set of nodes. This new resulting strategy was called *Random-Next-Neighbour-LSTM-Path-Maker*, abbreviated *RLPM*.

4 Experiments and results

LPM was tested and compared to HPCC and CR. This section presents the results pertaining to.

4.1 Scenarios

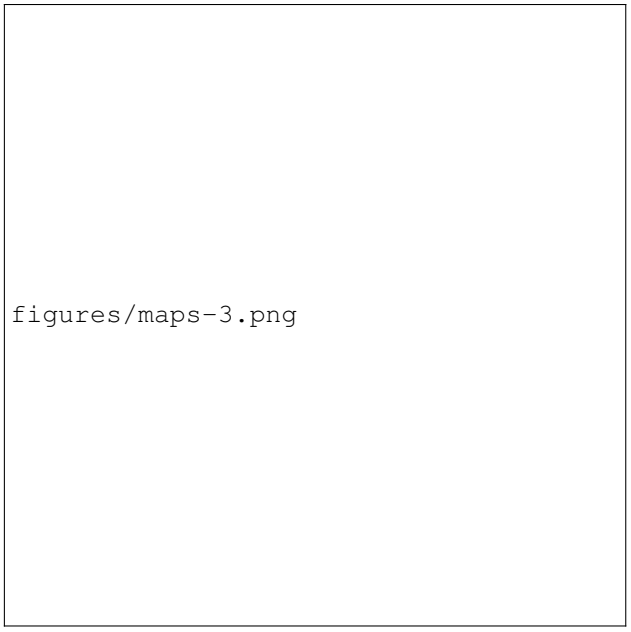

FIGURE 2 – Graphs used for assessment.

In order to align with previous works carried out in this field [14] and pursue the experiments in a comparative way, three different graphs were selected to evaluate the strategies CR and HPCC as a benchmark for the MAP : *Islands*, *Grid* and *A*, as shown in the **Figure 2**. Considering the different structures of these graphs, each one can be thought of as the representative of a class of graphs, hence the choice.

For each graph we tested in simulation the strategies CR, HPCC with a value of 0.2 for r_H and r_P, RLPM was trained from HPCC's simulation with the same value for r as well, then tested. To that end we used *Pytrol*, a new Python MAP-purpose simulator that we specially designed for this purpose, while a MAP-specific training program was coded using the deep learning library *PyTorch* to train the ANNs. These tests were performed over population sizes of 1, 5, 10, 15 and 25 agents and for each size we selected 100 random starts, also called executions. For each start, each strategy was tested over 3000 time steps. For any topology, each execution in simulation on an high-performance server takes approximately between 20 seconds and 130 seconds for 1 and 25 agents, respectively. This time complies with real life applications involving drones equipped with more basic computers and patrolling an area. Considering that each move takes exactly one period in the proposed model, after excluding the moves upon edges, an agent visits in average 600 nodes during one execution of 3000 time steps. In doing so, the paths used to train the LSTM networks have approximately a length of 600 nodes.

4.2 Training results

For each scenario, we trained seven architectures with profiles of parameters, defined as (L, H) in **Section 3.1** : $(1, 1)$, $(2, 2)$, $(4, 10)$, $(1, 50)$, $(2, 50)$, $(3, 50)$, and $(50, 2)$, with an end-to-end training i.e. a non truncated back-propagation through time. For any architecture we trained over 10000 epochs one LSTM network for each simulation configuration in two stages using the PyTorch library. First, the network is pre-trained over 2×10^6 epochs to capture as far as possible the structure of the topology, with 2-length series which stand for the edges of the graph. Then, the network is trained over the paths of agents with parameters initialised with the values learnt during the pre-training stage, over 10000 epochs. The **Figure 3** shows the initial and final values of the cost, that is the cross-entropy, during the validation stage for each architecture, averaged over the maps and numbers of agents. Here the initial cost corresponds to the validation cost after the first epoch. This figure shows that the better networks are $(2, 50)$, $(1, 50)$ and $(4, 10)$. Interestingly, the initial and final cost for the architecture $(50, 2)$ are almost identical and it has the worst cost with a value of 3.87. This result tends to show that the network's parameters converged very quickly, that is in 1 epoch. The number of parameters for $(1, 1)$, $(2, 2)$ and $(50, 2)$ are 258, 564 and 2484 respectively. It seems that those numbers are too low for a satisfactory approximation of the sequences. At the opposite, $(3, 50)$ has 63100 parameters. It is likely that this number is too large to avoid overfitting and a relatively bad performance in term of validation cost. Indeed, for one agent the size of the training data is approximately 50000, that is lower than the number of parameters of the $(3, 50)$ network.

Considering the bad final validation costs of $(50, 2)$, $(1, 1)$ and $(2, 2)$, of 3.87, 3.03, and 2.08 respectively, we tested and evaluated the four LSTM architectures : $(4, 10)$, $(1, 50)$, $(2, 50)$, $(3, 50)$. Thus, each architecture has given rise to four variants of RLPM named RLPM-L-H.

4.3 Performance results

To evaluate their performances, the RLPMs were tested and compared with CR, the reactive strategy, and HPCC, the cognitive one wherefrom they were trained. We used normalised *MI* and *QMI* as evaluation criteria, also referred to

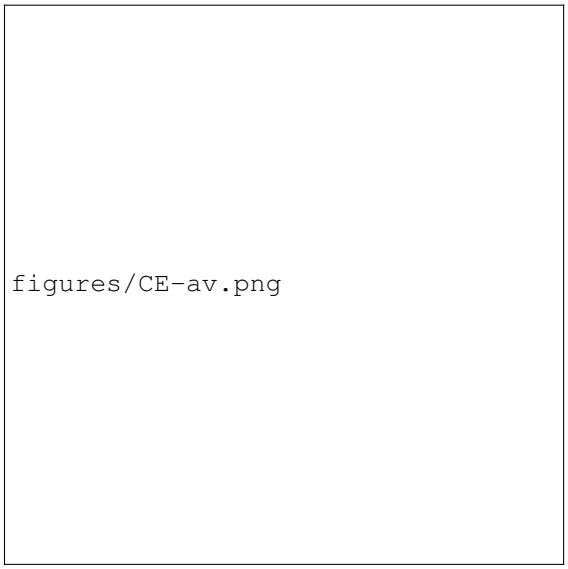

FIGURE 3 – Costs averaged over the maps and numbers of agents for each architecture.

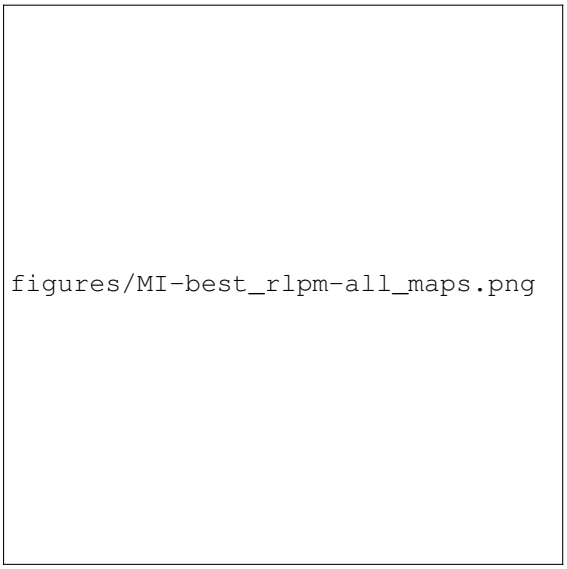

FIGURE 4 – Normalised MI of the evaluated strategies in y-axis for the three maps and the population sizes of agents in x-axis.

as metrics.

Fig. 4 shows all the results from our experiments for the normalised MI. For the sake of clarity, we show the RLPM version with the best value over this criterion , i.e. with the lowest value of MI, and that for each configuration. The best corresponding version for each configuration is denoted upon the graph. Not surprisingly HPCC always outperformed all the other strategies (CR, and the RLPMs) on all the maps and for all the population sizes of agents, except for the map A where RLPM-3-50 barely outperforms HPCC for 1 agent with a MI of 210 against 212 for HPCC. For all the maps the RLPMs overwhelmingly outperform the reactive strategy CR. For all the sizes of agent societies, the RLPMs are very close from HPCC, especially for 1 agent where their difference over MI ranges from -2, as previously stated for the map A where RLPM-3-50 is even better than HPCC, to 11 for the map $Grid$ with a value of 251 for RLPM-1-50 against 240 for HPCC. Besides, the evolution of RLPMs' performances over MI with respect to the number of agents fit rather well the HPCC's ones with an average difference of MI over all the population sizes for the three maps of 15.

For the maps Islands and A the architecture $(2, 50)$ is always the best. However, for the map Grid the architecture $(1, 50)$ is the best for 1, 5, 10 and 25 agents, but for 15 agents it is $(4, 10)$. Further analyses showed that the architecture $(1, 50)$ for 15 agents is only worse than $(4, 10)$ of 1 time step, 297 against 296 for $(4, 10)$. RLPM-1-50 is thus globally the best strategy for this map. Also, for the same map the average difference of performances over the population sizes between the best architectures previously enumerated and the architecture $(2, 50)$ is only of 2 time steps. This leads to consider RLPM-2-50 as being globally the best RLPM strategy for MI.

The **Fig. 5** shows the results for the normalised QMI. As for MI, for the sake of clarity, it is only showed the RLPM version with the best value, i.e. with the lowest value of QMI, that for each configuration. As well, the best corresponding version for each configuration is denoted upon the graph. For the map Islands, the QMI of the best RLPM is worse than HPCC and CR for all the numbers of agents, except for 25 agents where CR is worse of 28. Also, it must be pointed out that for 1 agent CR is a little better than HPCC, 290 against 320. This result can be explained by the topology of the map in combination with the HPCC's decision-making rule regarding the next node to visit : HPCC takes into account the distance from the agent's current node while CR chooses its next node to visit as the one having the greatest idleness in its neighbour. The best architectures are $(2, 50)$ for 1 agent, $(1, 50)$ for 5, 15 and 25 agents and $(4, 10)$ for 10 agents. In average, over the whole population sizes and the three maps, for the map Islands the best RLPMs are worse than HPCC of 256 periods with a significant difference of 533 for 15 agents. For the map A, the RLPMs are always better than CR but worse than HPCC, and except for 25 agents where RLPM-3-50 is the best RLPM strategy, RLPM-1-50 is always the best one. However, RLPM-2-50 turns out to be the best strategy for QMI when averaging over the population sizes with a value of 481 periods. Also, in average the best RLPMs are worse than HPCC of 152 periods. Lastly, for the map *Grid*, the RLPMs are worse than HPCC, but better than CR except for 1 agent where CR is better than the RLPMs of 32 periods. The best architectures are $(2, 50)$ for 1 agent and $(1, 50)$ for 5, 10, 15 and 25 agents. As well as for the A map, in average over the population sizes, RLPM-2-50 is the best strategy and the best RLPMs are worse than HPCC

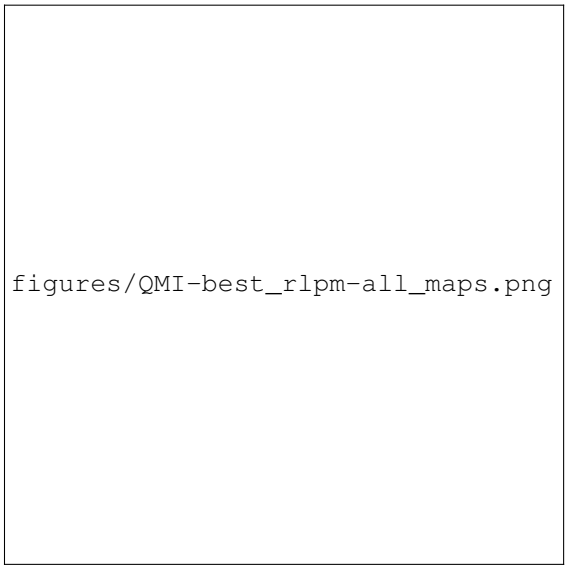

FIGURE 5 – Normalised QMI of the evaluated strategies in y-axis for the three maps and the population sizes of agents in x-axis.

of 105 periods.
Finally, the architecture $(2, 50)$ tends to be the best RLPM strategy for MI, except for the map Grid where $(1, 50)$ is slightly better, while for QMI, $(1, 50)$ is irremediably and globally the best strategy.

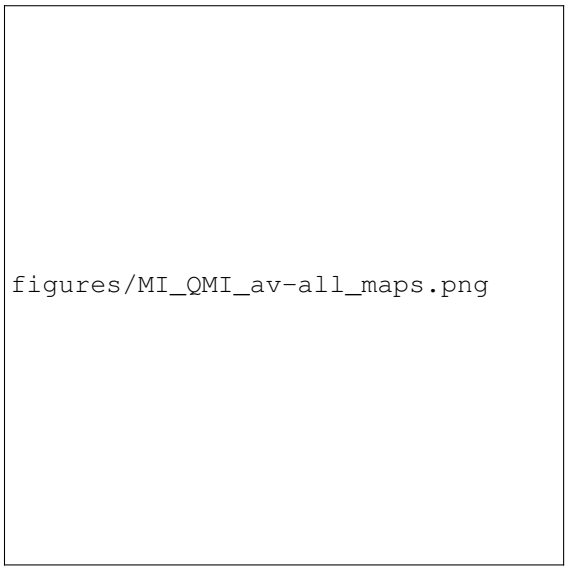

FIGURE 6 – Criterion space of MI and QMI.

The **Fig. 6** represents the criterion space for MI and QMI with the results of RLPM strategies averaged over the different numbers of agents. Here the different RLPM strategies and thus the LSTM architectures makes up the decision space. For the map Islands, it exists two Pareto optimal solutions in the decision space : the architectures $(2, 50)$ and $(1, 50)$ where the former is the best for MI with a value of 211, while the latter is the best for QMI with a value of 629. For the map A, both are also the only Pareto optimal solutions where the former is still the best for MI with a value of 234, while the latter is still the best for QMI with a value of 480. Finally for the map Grid, the architecture $(2, 50)$ is the only Pareto optimal solution with a value of $(290, 517)$. This analysis thereby tends to confirm our preliminary and foregoing assumption regarding the architecture $(2, 50)$ and $(1, 50)$ as globally the best ones pertaining to our problematic, where the former tends to be better for MI and the latter better for QMI.
As well, the architecture $(3, 50)$ is the worst strategy for the QMI criterion. QMI as quadratic mean, takes better into account the difference of time interval between the nodes and thus measures the tendency of nodes to be equitably visited through a run. Indeed, it penalises strategies that leave nodes unvisited (or which produces wide intervals between visits) during the simulation run[9]. Therefore, it provides an additional precision upon the distribution of visits over the nodes : one node with wide intervals have a little impact upon MI while it has upon QMI. Similarly, the architecture $(4, 10)$ is most of the time the worst strategy for MI. The performances over QMI of these strategies tends to show that they visit perpetually the same little set of nodes, whereupon the visits are poorly distributed over the nodes. It is likely that $(4, 10)$ presents a too small number of parameters to learn the behaviour of the HPCC startegy and, conversely, $(3, 50)$ a too large number of parameters to avoid over-fitting.

5 Conclusion and perspectives

In this paper we proposed and evaluated a new strategy for the multi-agent patrolling problem, based on the LSTM network architecture. To that end, we reminded the model underlying the multi-agent patrolling problem as well as the LSTM architecture. Then, we formally defined the new proposed LSTM-based strategy, wherefore the LSTM network was trained from the traces of a high-performance strategy. Seven architectures of LSTM were analysed in this work. Finally, we developed a new fully-fledged simulator in Python, specially designed for the multi-agent patrolling ; this simulator, that we named *Pytrol*, allowed to gather data to learn, test and evaluate the new strategies which were confronted to the reactive and cognitive standard strategies.
The evaluation demonstrated that RLPM-2-50 and RLPM-1-50, the strategies set from the LSTM architectures with 2 layers and 50 neurons, and 1 layers and 50 neurons respectively, are globally the best. RLPM-2-50 is the best upon MI - a central tendency measure - while RLPM-1-50 is the best upon QMI - measure that tends to emphasise the node with long times without visits.
These first experiments show good results as far as for each topology the proper architecture is selected. It has been showed that in an extreme situation where communications are prohibited, a learning strategy based on the

LSTM architecture can perform missions in a context of crisis with good performances, even better than the reactive and decentralised representative CR. The latter result show thereupon that a supervised-learning-based strategy with directed randomness is better than a reactive one and close to HPCC the cognitive representative for the criterion MI, although RLPM does not communicate, given that CR and HPCC are good representatives for the reactive and cognitive strategies respectively. Moreover, CR and RLPM are decentralised strategies, by design. However, RLPM was obtained by adding randomness in the decision procedure, otherwise the system being too much rigid tends to lead agents to converge indefinitely towards a small set of nodes. This entails that the learning system resting upon the LSTM architecture used here is not adaptive. A preliminary avenue to explore would be to use a new cost function to optimise, instead of the cross-entropy, to train the models in a different way, what could improve QMI by increasing the variability of the learned distribution. Also, in order to exploit the potential of the LSTM networks for the generation of paths in the multi-agent patrolling, new deeper and more complex architecture will be implemented and evaluated in the future, as well as other ANN architectures to improve the distribution of agents over the nodes and thereby QMI, but also the performances more generally. In fact, results presented here are based on LSTM predictor, but the method proposed here is generic and can be applied using any type of temporal series predictor.

Markov Decision Process (MDP), Decentralised Markov Decision Process (DEC-MDP) or Decentralised Partially Observable Markov Decision Process (DEC-POMDP) models could have been considered to model MAP. However, in a centralised perspective, that is with centralised strategies, the general MAP model used here is equivalent to a MDP model where the state corresponds to both positions of agents and global idlenesses. In that, it would be a determinist MDP. For decentralised strategies, in the considered model each agent has an overestimation of idlenesses, but there is not any probabilist distribution over them. This model can then not be regarded as a DEC-POMDP. The extension of this work to any MDP model is thereby limited by what has been stated before.

Finally, in order to bring RLPM in real life, several steps remain to be performed. First, simulation tests using a graph constructed from geographical data of an area to be patrolled shall be conducted. Then validation in field tests with actual drones will be possible.

References

[1] A. Machado, G. Ramalho, J.-D. Zucker, and A. Drogoul, "Multi-agent patrolling : An empirical analysis of alternative architectures," in *International Workshop on Multi-Agent Systems and Agent-Based Simulation*, pp. 155–170, Springer, 2002.

[2] A. Almeida, P. Castro, T. Menezes, and G. Ramalho, "Combining idleness and distance to design heuristic agents for the patrolling task," in *II Brazilian Workshop in Games and Digital Entertainment*, pp. 33–40, 2003.

[3] Y. Elmaliach, N. Agmon, and G. A. Kaminka, "Multi-robot area patrol under frequency constraints," *Annals of Mathematics and Artificial Intelligence*, vol. 57, no. 3-4, pp. 293–320, 2009.

[4] Y. Chevaleyre, "Theoretical analysis of the multi-agent patrolling problem," in *Intelligent Agent Technology, 2004.(IAT 2004). Proceedings. IEEE/WIC/ACM International Conference on*, pp. 302–308, IEEE, 2004.

[5] H. Santana, G. Ramalho, V. Corruble, and B. Ratitch, "Multi-agent patrolling with reinforcement learning," in *Proceedings of the Third International Joint Conference on Autonomous Agents and Multiagent Systems-Volume 3*, pp. 1122–1129, IEEE Computer Society, 2004.

[6] T. Menezes, P. Tedesco, and G. Ramalho, "Negotiator agents for the patrolling task," in *Advances in Artificial Intelligence-IBERAMIA-SBIA 2006*, pp. 48–57, Springer, 2006.

[7] Y. Guo, L. E. Parker, and R. Madhavan, "Collaborative robots for infrastructure security applications," in *Mobile robots : the evolutionary approach*, pp. 185–200, Springer, 2007.

[8] D. B. D'Ambrosio, S. Goodell, J. Lehman, S. Risi, and K. O. Stanley, "Multirobot behavior synchronization through direct neural network communication," in *International Conference on Intelligent Robotics and Applications*, pp. 603–614, Springer, 2012.

[9] G. Sampaio, G. Ramalho, and P. Tedesco, "A technique inspired by the law of gravitation for the timed multi-agent patrolling," in *22nd IEEE International Conference on Tools with Artificial Intelligence (ICTAI)*, vol. 1, pp. 113–120, 2010.

[10] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[11] A. Graves and J. Schmidhuber, "Framewise phoneme classification with bidirectional lstm networks," in *Proceedings. 2005 IEEE International Joint Conference on Neural Networks, 2005.*, vol. 4, pp. 2047–2052, IEEE, 2005.

[12] M. Bennett, M. F. Schatz, H. Rockwood, and K. Wiesenfeld, "Huygens's clocks," *Proceedings of the Royal Society of London. Series A : Mathematical, Physical and Engineering Sciences*, vol. 458, no. 2019, pp. 563–579, 2002.

[13] D. O. Sales, D. Feitosa, F. S. Osório, and D. F. Wolf, "Multi-agent autonomous patrolling system using ann and fsm control," in *2012 Second Brazilian Conference on Critical Embedded Systems*, pp. 48–53, IEEE, 2012.

[14] M. Othmani-Guibourg, A. El Fallah-Seghrouchni, J.-L. Farges, and M. Potop-Butucaru, "Multi-agent patrolling in dynamic environments," in *(ICA), 2017 IEEE International Conference on Agents*, pp. 72–77, IEEE, 2017.