Respuestas, 23 (2) July - December 2018, pp. 53-61, ISSN 0122-820X

Respuestas
Journal of Engineering Sciences

RESPUESTAS

Original Article

https://doi.org/10.22463/0122820X.1738

Multiagent reinforcement learning using Non-Parametric Approximation

Aprendizaje por reforzamiento para sistemas multiagentes utilizando Aproximación No Paramétrica

David Luviano Cruz[1*], Francesco José García-Luna[2], Luis Asunción Pérez-Domínguez[3]

[1]Doctor en Ciencias, david.luviano@uacj.mx, Orcid: 0000-0002-4778-8873, Universidad Autónoma de Ciudad Juárez, Juárez, México.*
[2]Doctor en Ciencias, francesco.garcia@uacj.mx, Orcid: 0000-0002-8571-914X, Universidad Autónoma de Ciudad Juárez, Juárez, México.
[3]Doctor en Ciencias en Ingeniería, luis.dominguez@uacj.mx, Orcid: 0000-0003-2541-4595, Universidad Autónoma de Ciudad Juárez, Juárez, México.

How to cite: D.L. Cruz, F.J. García-Luna, L.A. Pérez-Domínguez, "Multiagent reinforcement learning using Non-Parametric Approximation", *Respuestas,* vol. 23, no. 2, pp. 53-61, 2019

Received on January 10, 2018 - Approved on May 29, 2018.

ABSTRACT

This paper presents a hybrid control proposal for multi-agent systems, where the advantages of the reinforcement learning and nonparametric functions are exploited. A modified version of the Q-learning algorithm is used which will provide data training for a Kernel, this approach will provide a sub optimal set of actions to be used by the agents. The proposed algorithm is experimentally tested in a path generation task in an unknown environment for mobile robots.

Keywords:

Multiagent systems, nonparametric approximator, reinforcement learning, trajectory generation

RESUMEN

En este artículo se presenta una propuesta híbrida de algoritmo de control para sistemas multiagentes, en donde se aprovechan las ventajas del aprendizaje por reforzamiento y de las funciones de aproximación no paramétricas. Se utiliza una versión modificada del algoritmo Q-learning la cual proveerá de datos de entrenamiento para un Kernel, el cual ofrecerá una aproximación sub-óptima de acciones a realizar por los agentes. El algoritmo propuesto es probado experimentalmente en una tarea de generación de trayectoria en un entorno desconocido para robot móviles.

Palabras clave:

Sistemas multiagentes, aproximador no paramétrico, aprendizaje por reforzamiento, generación de trayectorias

Introduction

There are Research related to multiagent systems (MAS) is an emerging subfield of distributed artificial intelligence, which aims to provide principles for building complex systems through the integration of multiple agents.

There are features in MAS that distinguish it from a single-agent control system. First the agents are considered partially autonomous, this is due to the fact that the agents do not have available all the global information and therefore of the work environment, reason why they can only have access to a limited information, second in the MAS an individual agent cannot decide an optimal action only using his local knowledge [1].

***Corresponding author.**
E-mail address: david.luviano@uacj.mx (David Luviano Cruz)

Multi-agent systems have found applications in a wide variety of fields such as robot teams playing soccer, distributed control, unmanned aerial vehicles, training control, resource and traffic management, systems support, data mining, design engineering, intelligent search, medical diagnostics, product delivery, among others. The agents that make up a MAS have to deal with environments that can be static or dynamic, deterministic (an action has a single effect) or non-deterministic, discrete (there is a finite number of actions and states) or continuous [2].

For example, most existing artificial intelligence techniques for individual agents have been developed in static media as they are easier to handle and allow for rigorous mathematical treatment. In MAS, with the mere presence of multiple agents they make a static environment appear as dynamic from the point of view of other agents.

Although traditional control approaches seek to equip agents with MAS with pre-programmed or pre-designed behaviors, agents often need to learn new behaviors online so that MAS performance gradually improves. This is because the complexity of the working environment in which the agents operate and the tasks assigned to them make an a priori design of the control laws difficult or even impossible.

An agent that learns through Reinforcement Learning (RL) acquires knowledge through interaction with the dynamic environment where it performs its assigned task. In each step of time, the agent perceives the state of the environment and executes a determined action, which generates that the environment transitions to a new state. A reward signal scale evaluates the quality of each state transition so the agent must maximize the reward accumulated during the interaction with the environment, it is important to mention, the agents are not told what action to take, so they must explore the environment to find the actions that provide a greater reward in the long term [3].

One area where learning by reinforcement has been successful is trajectory planning for mobile robots, where a trajectory is generated from a starting point to an end point respecting certain restrictions imposed on movement, such as obstacles, delimitation of the trajectory area.

The problems addressed by RL are generally limited to problems with discrete states and with a finite number of actions available to agents. This is due to the so-called curse of dimensionality, which is the exponential growth of state-action pairs to learn as the number of states and actions increase in the problem, leading to an increase in computation time and the amount of memory needed to store the data associated with the algorithm [4].

Therefore, it is necessary to incorporate an additional strategy to learning by reinforcement, which offers the opportunity to generalize the results obtained. There are two approaches to be used to approximate action-state values in RL, one of them is the parametric approximators, where the functional form of the mapping and the number of parameters are designed beforehand and do not depend on the data [5], on the other hand, in non-parametric approximators, the number of parameters and the shape of the approximator are derived as a function of the available data.

The article proposes a methodology that takes advantage of the RL along with a non-parametric approximator in the form of Kernel, the algorithm is integrated by two stages of learning, the first will use the Q-learning algorithm, where the model of the task is known, at this stage the agent will explore the task environment in order to collect information states-actions, that is to say which is the optimal action to take each one of the explored states, in the second stage, the information obtained by the algorithm of RL will be used to adjust the weights of the Kernel, which will offer us the optimal actions to take by the agent (robot) in states that were not explored in the first stage of learning.

The proposed algorithm is validated by means of simulation, where the generation of optimal trajectories for mobile robots is sought under a cooperative task scheme.

Materials and methods

Reinforcement learning for multi-agent Systems

The generalization of Markov's decision-making process in learning by reinforcement in multi-agent systems (MARL) is the so-called stochastic game [7]. The stochastic game is defined as a tuple

$$\left(S, A_1, A_2, ..., A_n, f, \rho_1, \rho_2, ..., \rho_n\right)$$

$$\rho_i \ : \ S \times \mathbf{A} \times S \to \mathrm{R}$$
$$f \ : \ S \times \mathbf{A} \times S \to [0,1]$$
$$\mathbf{A} = A_1 \times A_2 .... \times A_n$$

where n is the number of agents, S is the finite set of states in the environment, $A_i \quad i = 1,...,n$ are the finite sets composed of the available actions of each agent, producing the set of joint actions $A = A_1 \times A_2 \times ... \times A_n$, ρ_i is the function of reward for the agent which is considered limited and the function of transition probability of states being f. In MAS state transitions are the result of joint actions taken by all agents in dișcrete step time. t :

$$\mathbf{a}_t = \left[a_{1,t}^T, ..., a_{n,t}^T\right]^T, \mathbf{a}_t \in \mathbf{A}, a_{i,t} \in A_i$$

Policies

$$\pi_i = S \times A_i \to [0,1]$$

Together they form the policy of joint action $\boldsymbol{\pi}$.

Since the reward signals $r_{i,t+1}$ of the agents depends on a joint action carried out by all the agents, their return R for a multi-agent system depends on the joint policy:

$$R_i^{\boldsymbol{\pi}}(x) = E\left\{\sum_{t=0}^{\infty} \gamma^t r_{\ i,t+1} \middle| s_0 = s, \boldsymbol{\pi}\right\}$$

The function Q of each agent depends on joint action and joint action policy:

$$Q_i^{\boldsymbol{\pi}} \ : \ S \times \mathbf{A} \to \mathrm{R}$$
$$Q_i^{\boldsymbol{\pi}}(s, \mathbf{a}) = E\left\{\sum_{k=0}^{\infty} \gamma^k r_{\ i,k+1} \middle| s_0 = s, \mathbf{a}_0 = \mathbf{a}, \boldsymbol{\pi}\right\}$$

In entirely cooperative stochastic games, the reward functions are the same for all agents $\rho_1 = \rho_2 = ... = \rho_n$ which implies that the returns R are also the same $R_1^{\pi} = R_2^{\pi} = ... = R_n^{\pi}$ Therefore, all agents have the same objective which is to maximize the common return.

The optimal function Q is defined as Q^*

$$Q^*(s,a) = \max_{\pi} Q^{\pi}(s,a)$$

Which satisfies Bellman's optimization equation.:

$$Q^*(s,a) = \sum_{s \in S} f\left(s,a,s^{'}\right)\left[\rho\left(s,u,s^{'}\right) + \gamma \max_{a^{'}} Q^*\left(s^{'}, a^{'}\right)\right]$$

for all $s \in S \ \ a \in A$

Once Q^* is available, an optimal stock policy can be calculated by choosing in each state a stock with the highest value Q optimal:

$$\overline{\pi}^*(x) = \arg\max_{a} Q^*(s,a)$$

A generalized approach used to solve the problem of coordination in MAS is to make sure that any decision situation is solved in the same way by all the agents using some type of negotiation. In our proposal, implicit coordination is used, where agents learn to choose actions in a coordinated way through trial and error.

Q-learning algorithm

There are a large number of algorithms available for learning by reinforcement, one of the most popular methods in RL is the Q-Learning algorithm, which uses an iterative approach procedure [8]. Q-Learning begins with an arbitrary function Q, observe transitions $\left(s_t, a_t, s_{t+1}, r_{t+1}\right)$ and after each transition updates the function Q with:

$$Q_{t+1}(s_t,a_t)=Q_{t+1}(s_t,a_t)+\alpha\left[r_{t+1}+\gamma\max_{a'}Q_t\left(s_{t+1},a'\right)-Q_t\left(s_t,a_t\right)\right]$$

The term in square brackets is called a temporal difference. The learning parameter $\alpha\in(0,1]$ can be variant in time and usually decreases with time.

The sequence Q_t converges on Q^* under the following conditions [8]:

- Different function values Q are saved and updated for each action-state.
- The summation $\sum_{t=0}^{\infty}\alpha$ is finite.
- Asymptotically all action-state pairs are visited infinitely.

The third point can be satisfied if agents are kept trying all actions in all states that have non-zero probability of happening. This requirement is called exploration, which can be done in several ways, one of them is choosing in each step a random action with probability. $\varepsilon\in(0,1)$ and choosing a greedy action with probability $\varepsilon-1,$ this way we get a greedy exploration.

Learning by means of Non-Parametric Kernel approximator

The algorithms that use RL obtain a policy of optimal actions from the optimal values obtained during the learning process, most of these methods are based on discrete considerations of the environment and a limited number of states, actions and agents in order to avoid the problem of dimensionality.

Since most real applications have a large number of states and the Q-Learning algorithm is based on search tables, the non-parametric approximation method based on Kernel is used to approximate the unknown states that are not visited when the RL algorithm is carried out, also to make generalizations when the environment has been slightly modified, in both cases avoiding the need to recalculate the optimal policies.

In the approach phase we assume that we have a collection of data s_t coming from the observation of the agents, taken in the interval $t\in[\tau_1,\tau_2],$ these data come from the model: $\hat{s}_{t+1}=\phi(s_t)+\varepsilon$.

where $\phi(s_t)$ is an unknown soft response curve and ε is the mistake. The goal is to find an estimate of Kernel $\hat{\phi}$ at some point in a pre-specified time *t*. The Kernel is simply a weighted average of all data points:

$$\hat{\phi}(s_t)=N^{-1}\sum_{k=\tau_1}^{\tau_2}W_t s_t a(t)$$

Where $N=\tau_2-\tau_1+1,$ W_k is the sequence of weights. The estimated state is denoted by: $\hat{s}_{t+1}\in R^n$.

A conceptually simple representation of the sequence of weights W_t , is by describing the shape of the weight function by means of a density function with a scaling parameter that has the function to adjust the size and shape of the weights near the data points s_t. It is common to refer to this shaping function as a kernel $K[z]$. The kernel is a real, continuous, dimensioned and symmetrical function which can be integrated into one $\int K[z]dz=1$

The sequence of weights for the kernel is defined by:

$$W_t s_t=K_t[s(t)]/\hat{f}_k(s_t)$$

where $\hat{f}(s_t)=N^{-1}\sum_{k=\tau_1}^{\tau_2}K_k[s(t)],$ $K_k[s(t)]$ is a Gaussian function:

$$K_t[s(t)]=a\exp\left(\frac{\|s_t-\mathbf{c}_t\|^2}{2\sigma^2}\right),\quad a>0$$

The regression carried out by the Kernel for the data $[a(k),s(t)]$ according to Nadaraya-Watson [10], where $t\in[\tau_1,\tau_2]$ is:

$$\hat{\phi}(s(t))=\frac{\sum_{t=\tau_1}^{\tau_2}K_k[s(t)]a(t)}{\sum_{t=\tau_1}^{\tau_2}K_k[s(t)]}$$

In automatic control applications all random variables have a constant probability density function, with this

type of random variables, the summation implied by the kernel function is equivalent to a Monte Carlo integration:

$$K_t[s(t)] \to \frac{N}{\tau_2}$$

Therefore, the regression proposed by Nadaraya used in the kernel can be replaced by the regression proposed by Priestley-Chao [9], which is defined as:

$$\hat{\phi}(s(t)) = \frac{\tau_2}{N}\sum_{t=\tau_1}^{\tau_2} K_t[s(t)]a(t) \qquad (1)$$

Given the above the expression (1) is used to model the unknown or unexplored states of the environment. One possible way is to design an expanded kernel function in order to encompass a larger amount of data by means of a full amplitude:

$$\sigma_{\max} = 2\sqrt{2\ln(2)}\sigma$$

where $\sigma_{\max}$ is chosen on the basis of the total number of data obtained, e.g. $\sigma_{\max} = \frac{N}{10}$ the amplitude parameter of the Gaussian function is given by:

$$\sigma = \frac{N}{20\sqrt{2\ln(2)}}$$

Where the total data available is.

Proposed learning algorithm

Proposed learning flow is show in the figure 1.

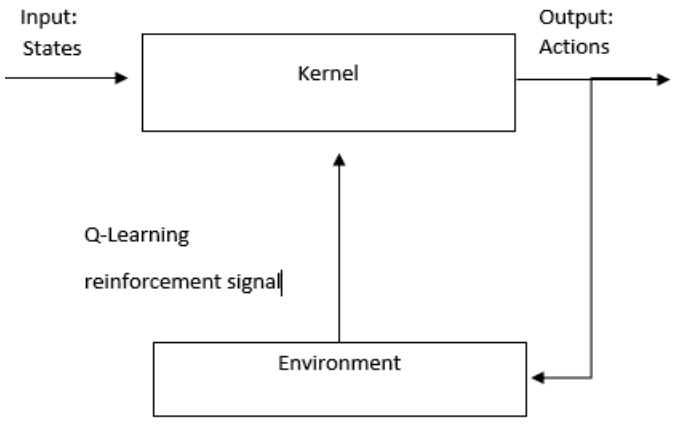

Figure 1. Proposed learning flow

At every discreet step of time t, the states where the agents are located are observed and these are referred to a table of states-actions called Q -tabla. The Q-learning algorithm is used to obtain optimal actions, from the data of Q^* which is obtained at the end of the convergence of the algorithm.

When the states where the agents are located are not available in the Q-table, either because they were not explored during the learning algorithm by reinforcement or by the occurrence of small changes in the environment, the actions to be performed by the agents will be approximate Kernel. The new approach will be sent to the agents in order to continue the task entrusted.

The previously trained approximator generates as output a sub-optimal action for each agent in the environment, thus avoiding to run the RL algorithm again when the agents face an unknown state.

The proposed method can be listed in the following steps:

1) *The initial states of each training cycle of the RL Q-learning algorithm are captured:* The current state of the agents' state with respect to the environment is captured through sensors.

2) *Limit the number of captured states:* The limitation of captured states reduces the set of states agents require to complete the task which saves time and computing power.

3) *Establish the actions available to the agents:* At each moment the agents are required to carry out an action with a degree of coordination, therefore, it is necessary to select in advance the most reliable actions to be carried out by the agents, with the aim of keeping the space for actions minimized and avoiding dimensionality problems.

4) *Estimate the Q-state-action values of each agent*: The numerical reward of each action is calculated and given to the agent after a joint action is performed, the values obtained are saved in a search table called Q-table.

5) *Repeat steps 2-4 until the agents reach the target*: The training cycle ends if the agents reach the final objective or if an established limit of iterations is reached.

6) *Repeat steps 2-5 until the Q values converge*: This happens when the values remain unchanged or they are below a predetermined level.

7) *Obtaining final Q-table of action-states:* The final table of optimal states-actions is fine-tuned for the selection of the optimal actions by means of the location of the action that will generate the maximum value Q in every state.

8) *Kernel training phase*: The table of values is used Q obtained by the Q-Learning algorithm to train the kernel, each column of the table Q represents a state, which is entered as input and the optimal actions found as outputs of the system.

Once the kernel has been trained, it will provide a joint approximate action that the agents will implement when they are facing unknown states that had not been explored or learned in the previous stages of learning.

Results and Discussion

In order to validate the performance of the proposed method, two Khepera IV mobile robots are used, whose objective is to generate a trajectory from an initial point to a goal. The software used was Matlab, the robots were used in slave mode with a bluetooth connection at 115200 bauds, the exchange of information between the robots Khepera and Matlab was through ASCII code. The task must be completed in a minimum time avoiding obstacles and coordinating among them, it is assumed that the agents have no prior knowledge about the position and shape of the obstacles present in the environment, the configuration of the working environment is shown in figure 2.

Figure 2. Task Configuration

The initial position of the agents is randomly selected and 50 learning steps are carried out, if this limit is reached, the experiment is stopped and restarted. As soon as the agents find the optimal trajectory without colliding with obstacles or other agents it will be said that the values of the Q-table have converged, so the learning stage will be stopped by reinforcement.

In order to complete the assigned cooperative task, each agent is required to choose one action from the 4 available actions

- Move forward 5 cm.
- Turn around 25° in the direction of the clock.
- Turn around 25° in a counterclockwise direction.
- Don't move.

The reward function $\rho(x,u)$ is given by:

$r_{k+1} = 1$ If the agent takes the target.

$r_{k+1} = 10$ If the agents take the target to the base position

$r_{k+1} = 0$ in any other situation

The reward function is designed in such a way that by assigning the numeric value 1 when the agent takes the target, the numeric reward of 10 for when the agent reaches the target prevents agents from finding suboptimal states motivated by intermediate rewards.

The convergence of the algorithm is shown in figure 3. In this figure it is shown that the algorithm converges after 16 trials, the duration of each trial depends on the number of steps that the agents perform before stopping the trial or reaching the learning objective.

The set of data that will be used as training samples for the Kernel are taken from the optimal Q-table generated by the Q-learning algorithm, Table I. Each column of the Q-table represents a state and the output of the approximator will be the joint action that the agents must execute.

In order to compare the performance of the algorithm, we choose as initial position for the agents, an unknown state which is not in the Q-table, under this situation it would not be possible to provide the optimal actions to the agents, so the kernel will offer a coordinated suboptimal action for each agent, example of these situations are shown in figures 4 and 5, where the agents have initial position in states that are not in the Q-table.

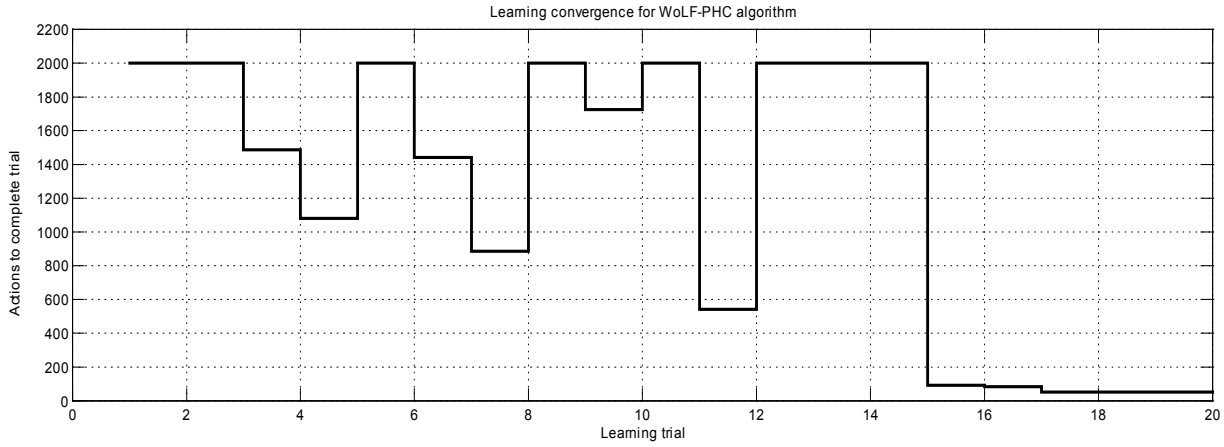

Figure 3. Convergence of the opposed algorithm

Table I. Q-table with optimal actions

Position	Rewards depending on action					
	Left	**Right**	**Up**	**Down**	**Y max**	**Best action**
Q11	0,4902706	7,130905	7,130905	0,4902706	7,13090502	UP & RIGHT
Q12	3,3002035	9,9684807	2,1309055	5,4902701	9,96848068	RIGHT
Q13	0,4902706	0,4902706	0,4902706	0,4902706	0,49027057	N/A
Q14	7,2929408	13,772154	10,682144	7,2929408	13,7721544	RIGHT
Q15	5,9475735	12,31808	5,6821441	12,29294	12,3180798	RIGHT
Q21	5,4902701	9,9684807	9,9684807	3,3002035	9,96848068	UP & RIGHT
Q22	8,300203	5,5598197	12,0893	8,300203	12,0892996	UP
Q23	7,2929408	13,772154	13,772154	10,559819	13,7721544	UP & RIGHT
Q24	12,29294	15,53604	12,31808	12,29294	15,5360397	RIGHT
Q25	10,947573	7,3180803	7,3180803	13,905507	13,9055071	DOWN
Q31	8,300203	12,0893	4,9684811	5,5598197	12,0892996	RIGHT
Q32	0,4902706	0,4902706	0,4902706	0,4902706	0,49027057	N/A
Q33	12,29294	15,53604	15,53604	8,9055076	15,5360397	UP & RIGHT
Q34	13,905507	18,05877	10,53604	13,905507	18,0587703	RIGHT
Q35	0,4902706	0,4902706	0,4902706	0,4902706	0,49027057	N/A
Q41	10,559819	10,739142	13,772154	7,2929408	13,7721544	UP
Q42	8,9055076	12,341102	15,53604	12,29294	15,5360397	UP
Q43	13,905507	13,916723	18,05877	13,905507	18,0587703	UP
Q44	15,975216	14,036066	21,826641	15,975216	21,8266407	UP
Q45	15,490269	15,490269	15,490269	15,490269	15,4902692	ALL
Q51	12,29294	5,9814957	12,341102	5,9814957	12,3411022	UP
Q52	13,905507	7,4598222	13,916723	10,981495	13,9167228	UP
Q53	15,975216	8,9167232	8,9167232	12,459822	15,9752158	LEFT
Q54	0,4902706	0,4902706	0,4902706	0,4902706	0,49027057	N/A
Q55	23,143842	14,10045	14,10045	14,10045	23,1438419	LEFT

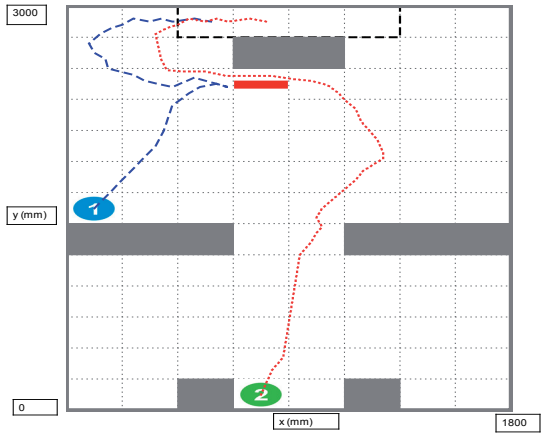

Figure 4. First path found by the Kernel starting from an unknown state

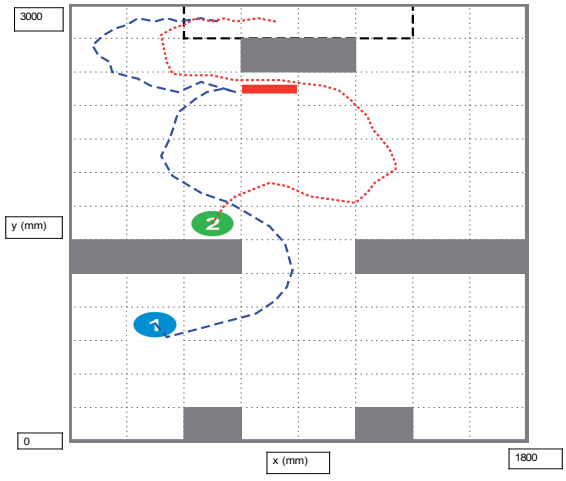

Figure 5. Second path found by the Kernel starting from an unknown state

The state vector $x=[x_1,x_2,x_3,x_4]^T$ and the control signal $u=[u_1,u_2]$ of agent 1 is shown in Figure 6, where x_1 is the position in the x, x_2 is the position on the y-axis, x_3 is the speed on the axis x, x_4 is the speed on the y-axis. u_1 is the force applied to the shaft x, u_2 is the force applied to the y-axis.

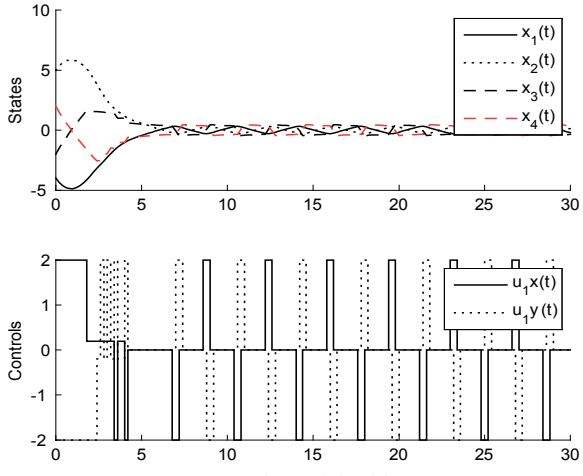

Figure 6. Status and control signal for agent 1

The state vector $x=[x_1,x_2,x_3,x_4]^T$ and the control signal $u=[u_1,u_2]$ of agent 2 is shown in Figure 7.

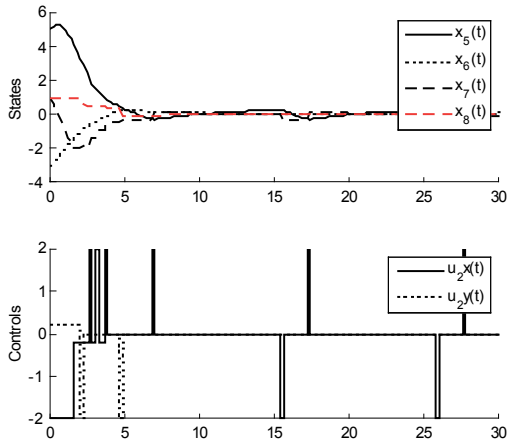

Figure 7. Status and control signal for agent 1

Conclusions

A proposal for integration between 2 control strategies is presented. In a first stage MARL is used as a means to obtain data and information of the task and the environment in which the agents are developed, later these data are used to train a non-parametric approximator.

The experimental results confirm the reliability. and robustness of the controller for path planning when agents face unknown states, overcoming the need to re-execute the learning algorithm by reinforcement, which leads to time savings and computational power.

It should be noted that when using a non-parametric approximator, the number of weights to be tuned in the kernel increases as the size of the data available for training increases, so a balance should be sought between simplicity and accuracy of the approximator.

In addition, it is necessary to emphasize that the proposed method uses a model of discrete states of the system, this is possible due to the quantization of the states in the environment. The minimum number of data captured by the algorithm should be sufficient to describe the dynamics of the system and together with the design of an appropriate reward function ensure that there is a local maximum in the return function. The choice of data captured will depend on prior knowledge of the problem to be solved. The

Respuestas, 23 (2) July - December 2018, pp. 53-61, ISSN 0122-820X

natural direction in the study of this topic would be to look for techniques where multi-agent systems with continuous states could be dealt with.

Acknowledgements

The authors extend their thanks to the Mexican Ministry of Public Education for funding this work through Research Agreement 511-6 / 17-7605.

References

[1] P. Stone, M. Veloso, "Multiagent systems: A survey from machine learning perspective", *Autonomous Robots*, vol.8, no.3, pp. 345-383, 2000.

[2] M. Wooldridge, *An Introduction to Multi Agent Systems*, Baffins Lane, Chichester, England: John Wiley & Sons. 1992.

[3] L. Busoniu, R. Babuska and B. De Schuttert, "Multi-agent Reinforcement Learning: An Overview", Delf Center for System and Control, Delf University of Technology, pp. 183-221, 2010.

[4] J.M. Vidal, "Learning in multiagent systems: An introduction from a game-theoretic perspective", In: Alonso E., Kudenko D., Kazakov D. (eds) Adaptive Agents and Multi-Agent Systems. Lecture Notes in Computer Science, vol. 2636. Springer, Berlin, Heidelberg, pp. 202-215, 2003.

[5] R. Postoyan, L. Busoniu, D. Nesic and J. Daafouz, "Stability Analysis of Discrete-Time Infinite-Horizon Optimal Control with Discounted Cost". *IEEE Transactions on Automatic Control*, vol. 62, no. 6, pp. 2736–2749, 2017

[6] C. Watkins, P. Dayan, "Q Learning: Technical Note", *Machine Learning*, vol.8, pp. 279-292, 1992.

[7] C. Boutilier, "Planning Learning and Coordination in Multiagent Decision Processes", In *Proceedings of the Sixth Conference on Theoretical Aspects of Rationality and Knowledge (TARK96)*, 1996, pp. 195-202, 1996.

[8] Y. Ishiwaka, T. Sato and Y. Kakazu, "An approach to the pursuit problem on a heterogeneous multiagent system using reinforcement learning", *Robotics and Autonomous Systems,* vol. 43, no. 4, pp. 245-256, 2003.

[9] A. Nadaraya, "On Estimating Regression", *Theory of Probability and its Applications*, vol. 9, no.1, pp. 141-142, 1964.