A Probabilistic Automata Framework for Behavioral Recognition

José L. Montaña[1], Cristina Tîrnăucă[1], Carlos Ortiz–Sobremazas[1], and Santiago Ontañón[2]

[1] Universidad de Cantabria, Santander, Spain
{joseluis.montana, cristina.tirnauca, carlos.ortizsobremazas}@unican.es
[2] Drexel University, Philadelphia, USA
santi@cs.drexel.edu

Abstract. We propose a framework based on probabilistic automata for the problem of behavioral recognition. This framework has a general purpose and the only hypothesis is that we can observe and represent the environment and the actions describing a behavior using a finite alphabet of symbols. The experiments performed support the idea that a probabilistic automaton with fully observable states can recognize behaviors that have some degree of randomness, but is unable to discriminate between strategies with some degree of randomness and their underlying deterministic variant.

1 Introduction

Imagine some agent (human, robot,...) who performs a task following some planned strategy. In our formulation of the problem of behavioral recognition we want to discover the task and/or the strategy the agent is using just by observing the actions it performs. We think that this point of view is general enough to deal with relevant specific applications such as: masquerade detection in computer intrusion, analysis of the task performed by the user in some e-learning activity, classification and prediction of the user's behavior in a web-user interaction process and, more generally, *activity recognition*. The aim of activity recognition is to recognize the actions and tasks of one or several agents taking as input a sequence of observations of their states and actions. Most research in activity recognition concentrates in the recognition of human activities. One goal of *human activity recognition* is to provide information on a user's behavior that allows computing systems to proactively assist users with their tasks. Since the early 1990s, this research field has been applied to many different areas such as medicine, human-computer interaction, psychology and sociology (see [1] for a tutorial-survey).

We propose to deal with the behavioral recognition problem using the framework and technical machinery of Learning from Observation (LfO) (see [2] for an unified presentation of the subject). Probably one of the first papers in this field is due to Bauer: in [3] he learns programs from example executions, which

basically amounts to learning strategies to perform abstract computations by demonstration, an approach that was especially popular in robotics [4]. Another early mention of LfO comes from Michalski et al. [5], who defines it merely as unsupervised learning. Gonzalez et al [6] discussed LfO at length, but provided no formalization nor suggested an approach to realize it algorithmically. More recent work on the more general LfO subject came nearly simultaneously but independently from Sammut et al [7] and Sidani [8]. Fernlund et al. [9] used LfO to build agents capable of driving a simulated automobile in a city environment. Pomerleau [10] developed the ALVINN system that trained neural networks from observation of a road-following automobile in the real world. Moriarty and Gonzalez [11] used neural networks to carry out LfO for computer games. Könik and Laird [12] introduced LfO in complex domains with the SOAR system by using inductive logic programming techniques. Other significant work done under the label of learning from demonstration has emerged recently in the case-based reasoning community. Floyd et al. [13] present an approach to learn how to play RoboSoccer by observing the play of other teams. Ontañón et al. [14] use learning from demonstration in the context of real-time strategy games in the context of case-based planning. Finally, another related area is that of inverse reinforcement learning [15], where the focus is on reconstructing the reward function given optimal behavior (i.e., given a policy, or a set of trajectories). One of the main problems here is that different reward functions may correspond to the observed behavior, and heuristics need to be devised to only consider families of reward functions that are interesting.

2 Behavioral Recognition from Observation

In behavioral recognition from observation there is a learning agent A that observes one or several actors performing a task T in an environment E and recording the actor's behavior in the form of *traces*. Then, those traces are used to discover some properties of the behavior, for instance the particular task T or, for the same task, the kind of strategy the actor is using to perform T. Most LfO work assumes that the agent does not have access to a description of T during learning, and thus, the features of the task and the way it is achieved must be learned purely by *unobtrusive observation* of the behavior of the actor. Let B_C be the *behavior* of an actor C. By behavior, we mean the control mechanism, policy, or algorithm that an actor or learning agent uses to determine which actions to execute over time. Our formalization is founded on the principle that behavior can be modeled as a *stochastic process*, and its elements as random variables depending on time. Our model includes the following variables: the learning agent observes the environment state via an input random variable X_t. The actions executed at time t are represented by a control random variable Y_t. We will use the following convention: if X_t is a variable, then we will use a calligraphic $\mathcal{X}$ to denote the set of values it can take, and lower case $x \in \mathcal{X}$ to denote specific values it takes. Depending on the type of the task to be recognized, the actions of the actor can vary in nature. In our framework, we assume that the random vari-

ables X_t and Y_t are multidimensional discrete variables. The behavior B_C of the actor C can be interpreted, therefore, as a stochastic process $I = \{I_1, ..., I_t, ...\}$, with state space $\mathcal{I} = \mathcal{X} \times \mathcal{Y}$, where $I_t := (X_t, Y_t)$. In a particular execution, a behavior B_C is manifested as the series of actions that the actor executes over time, which we call a *learning trace* (LT). An LT $O = [(x_1, y_1), ..., (x_m, y_m)]$ observed by the learning agent A can be seen as the realization – also trajectory or sample path – of the stochastic process corresponding to the behavior of the actor C. The pair of variables X_t and Y_t represent the *observation* of the learning agent A at time t.

2.1 Probabilistic Automata (PAs)

Although PAs have been introduced since the 60s by M. O Rabin (see [16]), they are still used in several fields of science and technology for modeling stochastic processes such as DNA sequencing analysis, image and speech recognition, human activity recognition and environmental problems among others. A reference covering the basic PA properties and explaining the relations with other Markovian models is [17].

Formally, a PA (with finite states) is a 5-tuple $\mathcal{A} = (\Sigma, Q, \phi, \iota, \gamma)$ where Σ is a finite alphabet (that is, a discrete set of symbols), Q is a finite collection of states, $\phi : Q \times \Sigma \times Q \longrightarrow [0, 1]$ is a function defining the transition probability (i.e., $\phi(q, a, q')$ is the probability of emission of symbol a while transitioning to state q' from state q), $\iota : Q \longrightarrow [0, 1]$ is the initial probability function and $\gamma : Q \longrightarrow [0, 1]$ is the final probability function. In addition, the following functions, defined over words $\alpha = (a_1, \ldots, a_m) \in \Sigma^*$ and state paths $\pi = (q_1, \ldots, q_m) \in Q^*$, must be probability distributions (Equation (1) when using final probabilities and Equation (2) otherwise):

$$\mathcal{P}_{\mathcal{A}}(\alpha, \pi) = \iota(q_1) \left(\prod_{i=1}^{m-1} \phi(q_i, a_i, q_{i+1}) \right) \gamma(q_n) \tag{1}$$

$$\hat{\mathcal{P}}_{\mathcal{A}}(\alpha, \pi) = \iota(q_1) \prod_{i=1}^{m-1} \phi(q_i, a_i, q_{i+1}) \tag{2}$$

This implies in particular that the two following functions are probability distributions over Σ^*:

$$\mathcal{P}_{\mathcal{A}}(\alpha) = \sum_{q,q'} \iota(q) \phi(q, \alpha, q') \gamma(q') \tag{3}$$

$$\hat{\mathcal{P}}_{\mathcal{A}}(\alpha) = \sum_{q,q'} \iota(q) \phi(q, \alpha, q') \tag{4}$$

Here $\phi(q, \alpha, q')$ is the extension of function ϕ to words with the obvious meaning: the probability of reaching state q' from state q while generating word α, $\mathcal{P}_{\mathcal{A}}$ is the probability of generating word α and $\hat{\mathcal{P}}_{\mathcal{A}}$ is the probability of generating a

word with prefix α (see [17] for a detailed explanation of Equations (3) and (4)). In many real situations we are interested in PAs with no final probabilities, and in this case we use Equation (4).

2.2 Behavioral Recognition Methodology based on PAs

Suppose that $\mathbf{T} = \{T_1, T_2, ..., T_n\}$ is a set of tasks that can be performed by an actor executing actions chosen from a finite set $\mathcal{Y}$ in a given environment $\mathcal{X}$. For each task T_k, we have an LT O_k with m_k observations. A behavioral recognition problem can be defined as the identification of the task performed by the actor given the LT O_k.

We propose to train a PA $\mathcal{A}_k = (\Sigma, Q, \phi, \iota, \gamma)$ for each task T_k. We define Σ to be the set of all actions $\mathcal{Y}$ that the actor performs, and Q the Cartesian product $\mathcal{X}^a \times \mathcal{Y}^b$, where $0 \leq b < a \leq b+2$ and $M := a+b$ is a measure of the amount of memory used by the automaton $\mathcal{A}_k$. In our experiments, M takes values in the set $\{1, 2, 3\}$. The case $M = 1$ models a purely *reactive behavior* (or Markovian behavior): the action depends only on the current state; the case $M = 2$ corresponds to a behavior in which the action depends on the current and previous states; finally, in the case $M = 3$, the action depends on the current state and the previous action/state pair. In general, for $M = 2l$ or $M = 2l + 1$, the current action depends on the current state and the previous l pairs of actions and states, and therefore it corresponds to a Markovian behavior of order l. In other words, it formalizes the dependence $Y_t^k = F_k(X_t^k, X_{t-1}^k, Y_{t-1}^k, \ldots, X_{t-l}^k[, Y_{t-l}^k])$ (notice that the action Y_{t-l}^k does not appear if M is an even number).

Training the automaton A_k from a trace O_k consists in determining the parameters ι, ϕ and γ). These parameters determine the probability functions defined in Equations (1), (2), (3) and (4). If our automaton is memory based on fully observable states, as described before, we can estimate the parameters by an easy counting argument. For any state $q \in Q$, let $count(q)$ be the number of occurrences of symbol $q \in Q$ in trace O_k. If $a \in \Sigma$, $count(q, a)$ has the obvious meaning, and also $count(q, a, q')$. We use a model with no final probabilities (we deal with Equations (2) and (4)). Using Laplace smoothing we can estimate the parameters as follows:

$$\iota(q) := \frac{count(q) + 1}{m_k + |Q|} \quad \phi(q, a, q') := \frac{count(q, a, q') + 1}{count(q) + |Q| \cdot |\Sigma|} \tag{5}$$

Once the parameters ι and ϕ have been determined, the function $\mathcal{P}_{A_k}$, defined as in Equation (2), can be used to compute the probability of any trace O generated according to some unknown behavior B. If we have access only to traces composed by actions (non observable environment), we can also compute the probability of a sequence of actions $\hat{\mathcal{P}}_{A_k}$ using Viterbi's algorithm, where $\hat{\mathcal{P}}_{A_k}$ is defined by Equation (4). In this last situation, the Baum-Welch's Expectation-Maximization algorithm could be used to discover the transition probabilities $\phi(q, a, q')$ (see [18]).

Suppose now that we are given a trace $O = [(x_1, y_1), \ldots, (x_m, y_m)]$ corresponding to some target task T and the objective is to infer the value i

such that $T = T_i$. To this end, we compute the value $P_{\mathcal{A}_k}(\alpha, \pi^M)$ for each $k \in \{1, \ldots, n\}$ and return $i := \text{argmax}_k\{P_{\mathcal{A}_k}(\alpha, \pi^M)\}$, where $\alpha = (y_1, \ldots, y_m)$ and $\pi^M = (q_1^M, \ldots, q_m^M)$. Note that the value of q_i^M depends on the amount M of memory used: $q_i^1 = x_i$, $q_i^2 = (x_i, x_{i-1})$ and $q_i^3 = (x_i, x_{i-1}, y_{i-1})$.

Moreover, we can measure the distance between an unknown behavior B_C exhibited by actor C and a given behavior B_A modeled by automaton $\mathcal{A}_k$ by computing the negative log-probability

$$R^M_{\mathcal{A}_k}(O) := -\frac{1}{m} \log P_{\mathcal{A}_k}(\alpha, \pi^M), \tag{6}$$

where O is the trace corresponding to behavior B_C and m is the number of observations in O. This value can be interpreted as a Monte Carlo approximation of the crossed entropy between behaviors B_C and B_A, known in the literature as Vapnik's risk (see [2]).

3 Experiments

We have run our experiments with a simulator of a simplified version of a Roomba, which is a series of autonomous vacuum cleaners sold by iRobot[3] . The original Roomba vacuum cleaner uses a set of basic sensors that helps it perform tasks. For instance, it is able to change direction whenever it encounters an obstacle. It uses two independently operating wheels that allow 360 turns in place. Additionally, it can adapt to perform other more creative tasks using an embedded computer in conjunction with the Roomba Open Interface.

In our implementation, the robot can only move `Up`, `Down`, `Left` and `Right` while there is no obstacle in front of it. Although it is possible for the agent to start anywhere, the traces we generate are always with the agent starting in the top-left corner of the map.

3.1 Training Maps

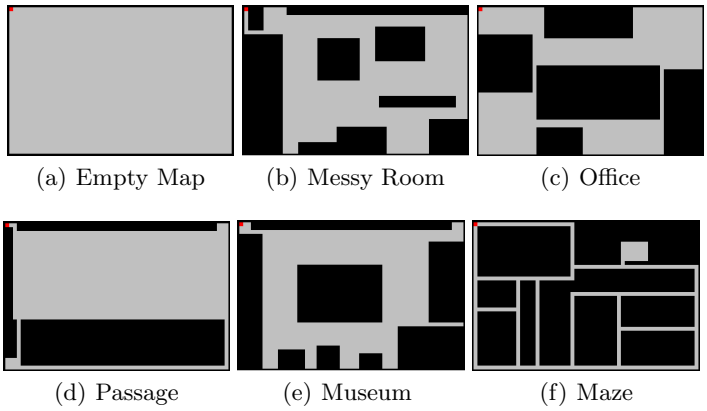
The environment in which the agent moves is a 40 x 60 rectangle surrounded by walls, which may contain all sorts of obstacles. For testing, we have randomly generated obstacles on an empty map. In the sequel, we briefly explain the six maps used in the training phase. Each of them is meant to represent a real-life situation, as indicated by their title (the list is by no means exhaustive).

Empty Map. The empty map consists of a big empty room with no obstacles.

Messy Room. The messy room simulates an untidy teenager bedroom, with all sorts of obstacles on the floor, and with a narrow entry corridor that makes the access to the room even more challenging for any "hostile intruder".

[3] According to wikipedia, "iRobot Corporation is an American advanced technology company founded in 1990 and incorporated in Delaware in 2000. Roomba was introduced in 2002. As of Feb 2014, over 10 million units have been sold worldwide".

The Office. The office map simulates a space in which several rooms are connected to each other by small passages. In this case, obstacles are representing big furniture such as office desks or storage cabinets.
The Passage. The highlight of this map is an intricate pathway that leads to a small room. The main room is huge and does not have any obstacle in it.
The Museum. This map is intended to simulate a room from a museum, with the main sculpture in the center, and with several other sculptures on the four sides of the room, separated by small spaces.
The Maze. The most part of this map consists of narrow pathways with the same width as the agent. There is also a little room which is difficult to find.



(a) Empty Map (b) Messy Room (c) Office

(d) Passage (e) Museum (f) Maze

Fig. 1. Training maps

3.2 Agent Strategies

We have designed a series of strategies with different complexities. When describing a strategy, we must define the behavior of the agent in a certain situation (which defines its state X_t) that depends on the configuration of obstacles in its vicinity (prefix Rnd is used for stochastic strategies).
Walk. The agent always performs the same movement in a given state. As an example, a possible strategy could be to go `Right` whenever there are no obstacles, and `Up` whenever there is only one obstacle to the right (stationary deterministic of type 2, it only depends on current state X_t).
Rnd_Walk. In this strategy, the next move is picked up randomly from the set of available moves. For example, an agent that has obstacles to the right and to the left can only move `Up` or `Down`, but there is no predefined choice between those two (stationary stochastic of type 2, it only depends on current state X_t).
Crash. In this strategy the robot should perform the same action as in the previous cell while there exists this possibility. Whenever it encounters a new

obstacle in its way, the agent must choose a certain predefined action. Therefore, it needs to have information about its previous action in order to know where to move (stationary deterministic of type 3, it depends on current state X_t and previous action Y_t).
RND_CRASH. This strategy is similar to the previous one: the agent maintains its direction while possible. The main difference is that the action to perform when the agent encounters an obstacle is chosen randomly (stationary stochastic of type 3, it depends on current state X_t and previous action Y_{t-1}).
ZIGZAG. It consists of different vertical movements in two possible directions, avoiding the obstacles. It has an internal state which tells the robot if it should advance towards the left or the right side with this vertical movements: it initially goes towards the right side, and once it reaches one of the right corners, the internal state changes so that the robot will start moving toward the left side (stationary deterministic of type 3, it depends on current state X_t, previous action Y_{t-1} and internal state C_t).
RND_ZIGZAG. This strategy is similar to the previous one, with the only difference that once it reaches a corner the internal state could either change its value or not, and this is randomly assigned (stationary stochastic of type 3, it depends on current state X_t, previous action Y_{t-1} and internal state C_t).

3.3 Traces and Performance Evaluation

In our evaluation the simulation time is discrete, and at each time step, the vacuum cleaner can take one of these 5 actions: up, down, left, right and stand still, with their intuitive effect (if it tries to move into an obstacle, the effect is equivalent to the stand still action). So the control variable Y can take 5 different values: `Up`, `Down`, `Left`, `Right`, `Stand`. The vacuum cleaner perceives the world through the input variable X having 4 different binary components: up, down, left, right, each of them identifying whether the vacuum cleaner can see an obstacle in that direction.

In the training phase, we have generated a trace of 1500 observations for each pair map/strategy. Then, we used for each strategy a single file obtained by concatenating the traces from the above mentioned six maps to train its corresponding probabilistic automaton, resulting in six PAs: $\mathcal{A}_1, \ldots, \mathcal{A}_6$.

In order to evaluate our behavior recognition system, we performed two experiments. We have randomly generated three maps (Testing Group 1) for the first experiment and 100 maps (Testing Group 2) for the second experiment. Then, we generated a family of traces (each of them containing 1500 observations) for each pair strategy/map: $(O_n^i)_{i\in\{1,2,3\},n\in\{1,\ldots,6\}}$ for the first experiment and $(\bar{O}_n^i)_{i\in\{1,\ldots,100\},n\in\{1,\ldots,6\}}$ for the second one.

For the first experiment, we computed the log-normalized distance $R^M_{\mathcal{A}_m}(O_n^i)$ between the observation trace O_n^i and the automaton $\mathcal{A}_m$, as described in Equation (6). The average value $R^M_{m,n} = \sum_{i=1}^{3} R^M_{\mathcal{A}_m}(O_n^i)/3$ of these distances is reported in Figure 2, in the (m,n)-th cell of Table M. Our system classifies the testing task represented by column n as being generated by the automaton $\mathcal{A}_k$ such that $k = \text{argmin}_m\, R^M_{m,n}$ (minimizing distance maximizes trace probability).

Table 1: $R^M_{m,n}$ values for $M = 1$						
9000/1500 obs	Walk	Rnd_Walk	Crash	Rnd_Crash	ZigZag	Rnd_ZigZag
Walk	**8.8447**	9.0881	9.554	7.2567	8.998	7.9354
Rnd_Walk	9.6389	**3.8179**	3.9744	3.661	4.6092	4.5729
Crash	8.8812	6.4605	9.2126	5.1413	4.4686	4.1559
Rnd_Crash	8.782	4.186	**3.4454**	**3.454**	4.9792	4.4259
ZigZag	10.0427	7.1226	11.6125	4.1299	4.0198	4.0415
Rnd_ZigZag	9.4172	6.9799	11.6288	3.7951	**3.9144**	**3.6616**

Table 2: $R^M_{m,n}$ values for $M = 2$						
9000/1500 obs	Walk	Rnd_Walk	Crash	Rnd_Crash	ZigZag	Rnd_ZigZag
Walk	10.009	9.8039	9.9791	8.7494	9.8476	8.8655
Rnd_Walk	**10.0088**	**4.8185**	**4.9955**	5.6685	6.1171	6.8666
Crash	10.009	8.0111	10.3333	7.0744	6.612	6.303
Rnd_Crash	**10.0088**	5.7422	5.1717	**5.2526**	6.9289	6.5429
ZigZag	**10.0088**	8.1036	11.6281	7.9913	5.7136	6.1689
Rnd_ZigZag	**10.0088**	8.0421	11.6978	7.8478	**5.7096**	**5.7097**

Table 3: $R^M_{m,n}$ values for $M = 3$						
9000/1500 obs	Walk	Rnd_Walk	Crash	Rnd_Crash	ZigZag	Rnd_ZigZag
Walk	**12.0089**	11.8429	11.9808	10.6123	12.0257	10.9446
Rnd_Walk	**12.0089**	**6.9452**	7.2812	8.0741	8.2668	9.1078
Crash	**12.0089**	11.3821	11.938	8.6991	8.3253	7.8532
Rnd_Crash	**12.0089**	10.7444	**6.458**	**6.5235**	8.3757	8.0651
ZigZag	**12.0089**	11.2901	11.9402	9.3499	6.7775	7.5367
Rnd_ZigZag	**12.0089**	11.2492	11.9295	9.2055	**6.7384**	**6.9907**

Fig. 2. Distance Matrix for Testing Group 1

The results of this testing group show that the PA recognition system is able to correctly recognize the three random strategies (Rnd_Walk, Rnd_Crash and Rnd_ZigZag). However, the system fails when recognizing the respective underlying deterministic strategies (Walk, Crash and ZigZag). Also note that the deterministic versions of the random behaviors (Walk, Crash and ZigZag) are not confused one each other but each of them is most of the times classified by the system as its corresponding non deterministic version (Crash is classified as Rnd_Crash, ZigZag as Rnd_ZigZag, etc.).

For the second experiment, the numerical value $C^M_{m,n}$ placed in the (m, n)-th cell of the confusion matrix is the empirical probability of the n-th task to be classified as the m-th task, that is, the percentage of the learning traces produced using strategy n that are recognized as being produced by strategy m.

$$C^M_{m,n} = \frac{|\{i \in \{1, \ldots, 100\} \mid m = \operatorname{argmin}_k R^M_{\mathcal{A}_k}(\bar{O}^i_n)\}|}{100}$$

The diagonal of this matrix reflects the empirical probabilities of right classification and the sum of the other rows different from the diagonal element is the probability of error. We observe that this second experiment confirms the conclusions of the first one.

Table 1: $C^M_{m,n}$ values for $M = 1$						
9000/1500 obs	Walk	Rnd_Walk	Crash	Rnd_Crash	ZigZag	Rnd_ZigZag
Walk	0.13	0	0	0	0	0
Rnd_Walk	**0.27**	**1**	0.07	0	0	0
Crash	0.2	0	0.2	0	0	0
Rnd_Crash	0.4	0	**0.53**	**1**	0	0
ZigZag	0	0	0	0	0	0
Rnd_ZigZag	0	0	0.2	0	**1**	**1**

Table 2: $C^M_{m,n}$ values for $M = 2$						
9000/1500 obs	Walk	Rnd_Walk	Crash	Rnd_Crash	ZigZag	Rnd_ZigZag
Det_Walk	0.27	0	0	0	0	0
Rnd_Walk	0.27	**1**	0.27	0	0	0
Crash	**0.53**	0	0.2	0	0	0
Rnd_Crash	0.33	0	**0.47**	**1**	0	0
ZigZag	0.13	0	0	0	0.2	0
Rnd_ZigZag	0.13	0	0.07	0	**0.8**	**1**

Table 3: $C^M_{m,n}$ values for $M = 3$						
9000/1500 obs	Walk	Rnd_Walk	Crash	Rnd_Crash	ZigZag	Rnd_ZigZag
Walk	0.33	0	0	0	0	0
Rnd_Walk	0.27	**1**	0	0	0	0
Crash	**0.6**	0	0.2	0	0	0
Rnd_Crash	0.4	0	**0.73**	**1**	0	0
ZigZag	0.13	0	0	0	0.2	0
Rnd_ZigZag	0.13	0	0.07	0	**0.8**	**1**

Fig. 3. Confusion Matrix for Testing Group 2

4 Conclusions

We have proposed a model for behavioral recognition. Behaviors are identified with tasks (or strategies for solving a given problem). Our system uses probabilistic automata and correctly identifies tasks performed by the actor whenever those tasks have a certain random component. A remarkable characteristic of our model is the difficulty to distinguish between a certain strategy and a similar strategy perturbed with some degree of randomness. The inference technique is based on the greatest likelihood probability value generated by the PAs of the model. The major computational limitation in our fully observable state space is the amount of memory required by the trained automaton. A possible solution is to employ only a few amount of non-observable internal states. Future work also contemplates the usage of probabilistic transducers, to take into account the input-output (state-action) nature of the observations composing the learning traces in the observation scenario.

Acknowledgments. The authors acknowledge the financial support of project BASMATI (TIN2011-27479-C04-04) of Programa Nacional de Investigación and project PAC::LFO (MTM2014-55262-P) of Programa Estatal de Fomento de la

Investigación Científica y Técnica de Excelencia, Ministerio de Ciencia e Innovación (MICINN), Spain.

References

1. Bulling, A., Blanke, U., Schiele, B.: A tutorial on human activity recognition using body-worn inertial sensors. ACM Comput. Surv. **46**(3) (2014) 33:1–33:33
2. Ontañón, S., Montaña, J.L., Gonzalez, A.J.: A dynamic-bayesian network framework for modeling and evaluating learning from observation. Expert Systems with Applications **41**(11) (2014) 5212–5226
3. Bauer, M.A.: Programming by examples. Artificial Intelligence **12**(1) (1979) 1–21
4. Lozano-Pérez, T.: Robot programming. In: Proceedings of IEEE. Volume 71. (1983) 821–841
5. Michalski, R.S., Stepp, R.E.: Learning from observation: Conceptual clustering. In Michalski, R.S., Carbonell, J.G., Mitchell, T.M., eds.: Machine Learning: An Artificial Intelligence Approach. Tioga (1983) 331–364
6. Gonzalez, A.J., Georgiopoulos, M., DeMara, R.F., Henninger, A., Gerber, W.: Automating the cgf model development and refinement process by observing expert behavior in a simulation. In: Proceedings of The 7th Conference on Computer Generated Forces and Behavioral Representation. (1998)
7. Sammut, C., Hurst, S., Kedzier, D., Michie, D.: Learning to fly. In: Proceedings of the Ninth International Workshop on Machine Learning (ML 1992). (1992) 385–393
8. Sidani, T.: Automated Machine Learning from Observation of Simulation. PhD thesis, University of Central Florida (1994)
9. Fernlund, H.K.G., Gonzalez, A.J., Georgiopoulos, M., DeMara, R.F.: Learning tactical human behavior through observation of human performance. IEEE Transactions on Systems, Man, and Cybernetics, Part B **36**(1) (2006) 128–140
10. Pomerleau, D.: Alvinn: An autonomous land vehicle in a neural network. In Touretzky, D., ed.: Advances in Neural Information Processing Systems 1, Morgan Kaufmann (1989)
11. Moriarty, C.L., Gonzalez, A.J.: Learning human behavior from observation for gaming applications. In: FLAIRS Conference. (2009)
12. Könik, T., Laird, J.E.: Learning goal hierarchies from structured observations and expert annotations. Mach. Learn. **64**(1-3) (2006) 263–287
13. Floyd, M.W., Esfandiari, B., Lam, K.: A case-based reasoning approach to imitating robocup players. In: Proceedings of the Twenty-First International Florida Artificial Intelligence Research Society (FLAIRS). (2008) 251–256
14. Ontañón, S., Mishra, K., Sugandh, N., Ram, A.: On-line case-based planning. Computational Intelligence Journal **26**(1) (2010) 84–119
15. Ng, A.Y., Russell, S.: Algorithms for Inverse Reinforcement Learning. In: in Proc. 17th International Conf. on Machine Learning. (2000) 663–670
16. Rabin, M.O.: Probabilistic automata. Information and Control **6**(3) (1963) 230–245
17. Dupont, P., Denis, F., Esposito, Y.: Links between probabilistic automata and hidden markov models: probability distributions, learning models and induction algorithms. Pattern Recognition **38**(9) (2005) 1349–1371
18. Rabiner, L.R.: A tutorial on hidden markov models and selected applications in speech recognition. In: Proceedings of the IEEE. (1989) 257–286