On the automatic Entropy-based construction of Probabilistic Automata in a Learning Robotic Scenario

Sergio Roa and Geert-Jan Kruijff
German Research Center for Artificial Intelligence / DFKI GmbH
{sergio.roa,gj}@dfki.de

***Abstract*— When a robot interacts with the environment producing changes through its own actions, it should find opportunities for learning and updating its own models of the environment. A robot that is able to construct discrete models of the underlying dynamical system which emerges from this interaction can guide its own behavior and adapt it based on feedback from the environment. Thus, the induction of probabilistic automata from this sensorimotor loop might be useful for planning/learning tasks. These probabilistic automata can be used as a prediction tool, as a means to assess the uncertainty or predictability of specific action consequences and thus, as a tool for an active learning method.**

I. Introduction

In the face of continuously changing environments, a robot needs to learn from the world by interacting with it. A robot should then accurately predict the consequences of actions on an object given its own body configuration. With the acquired skills knowledge, it can then solve more complex tasks in a hierarchical manner. We consider here the task of predicting consequences of pushing simple geometrical objects called *polyflaps*. Polyflaps have been proposed to design simple learning scenarios [1].

In this paper we discuss a scenario where a simulated robotic arm interacts with a polyflap. In the implementation we use the NVidia® PhysX™ library that allows us to perform realistic physical simulations and to obtain 3-dimensional feature vectors. The learning machines we use are able to process spatio-temporal features. Specifically, we use the Crystallizing Substochastic Sequential Machine Extractor (CrySSMEx) [2] algorithm for the extraction of probabilistic finite state automata. The robotic arm pushes the object and then a sequence of polyflap poses is stored, encoded as rigid body transformations during a certain time interval. To reduce the space- and time complexity of the problem, we select a discrete set of possible actions and starting positions for the arm to start the pushing movement. This reduction of dimensionality affords us also to evaluate and analyse more easily and carefully the inference algorithm and its corresponding outcomes. In general, *sliding* and *flipping* affordances are obtained by applying pushing actions. The experiments show that the machines are able to model the tuples ⟨action, state, output, next state⟩.

But how can we find state abstractions? Since we want to understand the behaviour of an object given a specific action, it is reasonable to encode a state on the basis of polyflap poses (rigid body transformations). A sequence of polyflap poses will then be encoded as a sequence of states (not necessarily a 1 to 1 relation), where transitions depend on the corresponding action. Moreover, state abstractions can possibly be obtained not only from quantized vectors but also from the output dynamics. Given the complexity of these physical processes, transition probabilities between states are also quite possible and an information-theoretic measure is used to help quantize the space and to generate the transitions by using the CrySSMEx algorithm. The extracted automata are useful as a verification tool, that can be employed by active learning techniques in order to evaluate uncertainty in specific actions.

II. Learning Scenario

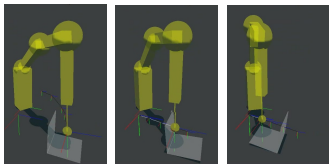

Fig. 1. Learning scenario with a polyflap

The learning scenario is shown in Fig. 1. The simulated arm corresponds to a Neuronics® Katana 6M™ arm with a ball as a simple finger. In order to simulate a pushing action we apply a linear trajectory over a specified time period until it reaches the desired pose. The arm has 6 joints, including the last joint for the finger which is static. The representation of object poses are in Euler angles with respect to a reference frame which is the origin in the scene (6-D pose).

The features corresponding to the arm are a starting 6-D pose vector for the finger arm $\mathbf{f}$, a feature value s encoding 3 velocity values (low, medium, high), obtained from applying 3 different movement durations, and an integer value denoting a direction angle Θ ranging from 60 to 120 degrees, parallel to the ground plane in the direction to the center of the standing polyflap side. Together, these features form the motor command feature vector denoted as $\mathbf{m}_t$ at time t. The values are all normalized to obtain vectors with mean 0 and standard deviation 1.0. A 6-D pose vector corresponding to the polyflap pose is denoted as $\mathbf{p}_t$. In order to perform preliminary experiments and avoid ambiguities and difficulties in analyzing data (cf. section III), we artificially enumerate (discretize) the set of possible actions. Therefore, we obtain a symbolic feature m_t denoting the motor information. In the enumeration, we encode the time step t into the symbol m_t in order to discretely

represent the finger transformations (change of poses) through an action sequence. Moreover, CrySSMex requires that we define an output label associated to each state or transition. This feature is very useful for evaluation and also for the convergence of the CrySSMEx algorithm itself (cf. section III). We then define the set $Y = \{-1, 1, 0, -0.5, 0.5\}$ of possible values for an output symbol y_t, denoting respectively when θ polyflap angle decreases (this happens when the polyflap tilts but does not completely flips over, so that it returns to the original angle), θ angle increases (polyflap falling down), polyflap does not move, polyflap moves backwards (negative X direction without angle change) and polyflap moves forwards. Thus, the tuples $\langle m_t, \mathbf{p}_t, y_t \rangle_{t=1}^{n}$ encode the rigid body transformations of polyflaps through these n steps and also encode the given robot control command and abstract behaviour after the pushing movement. In order to discretize and reduce the dimensionality of the task, we only used 18 different starting positions for the arm to start the pushing movement.

III. Automata induction method

Let us define a substochastic sequential machine (SSM) as a quadruple $\langle Q, M, Y, \mathcal{P} = \{p(q_j, y_l | q_i, m_k)\}\rangle$ where Q is a finite set of state elements (SEs), M is a finite set of input symbols, i.e. our motor command representation, Y is a finite set of output symbols, and $\mathcal{P}$ is a finite set of conditional probabilities (cf. explanation in [2] and eqs.1-3) where $q_i, q_j \in Q, m_k \in M$ and $y_l \in Y$. In practice, CrySSMEx can automatically induce discretizations of the input and output spaces by means of quantizer functions on these spaces but we will avoid this step in order to not introduce more difficulties in analyzing data and to increase the chance of convergence of the CrySSMEx algorithm. Thus, we assume the artificial enumeration for input and output spaces described in section II and we use the vector quantization method described in [2] for state discretization. A SSM models a situated discrete time dynamical system for which input (M), output (Y) and state (P) spaces are defined and a transition function $\gamma : P \times M \rightarrow P \times O$. A stochastic dynamical model of such a system is a joint probability mass function p_Ω induced from a transition event set Ω, and quantizers Λ_y, Λ_m and Λ_p for output, input and state spaces respectively. Ω consists of selected transition events recorded from a given set of input sequences. Thus, the joint probabilities of observed and quantized transitions (p_Ω) are translated into joint probabilities of SSM transitions according to $\mathcal{P}$. As already mentioned, we define $\Lambda_m(\mathbf{m}_t)$ and $\Lambda_y(t)$ according to the discretization described above and $\Lambda_p(\mathbf{p}_t)$ using the vector quantization method. Thus, we have:

$$p(q_i, m_k, y_l, q_j) = p_\Omega(\Lambda_p(\mathbf{p}_t) = i, \Lambda_m(\mathbf{m}_t) = k, \Lambda_y(t+1) = l, \Lambda_p(\mathbf{p}_{t+1}) = j) \quad (1)$$

The conditional probability is calculated with:

$$p(q_i, m_k) = \sum_{j=1}^{|Q|} \sum_{l=1}^{|Y|} p(q_i, m_k, y_l, q_j) \quad (2)$$

$$p(q_j, y_l | q_i, m_k) = \begin{cases} \frac{p(q_i, m_k, y_l, q_j)}{p(q_i, m_k)} & \text{if } p(q_i, m_k) > 0 \\ 0 & \text{if } p(q_i, m_k) = 0 \end{cases} \quad (3)$$

The substochasticity of the extracted machines is due to the possibility that the sample of input sequences in Ω will not necessarily provide examples of all possible input symbols in all possible enumerations of the quantized space of the dynamical system. As a consequence, the probability distributions can become substochastic [2]. The details of the procedure for extracting substochastic sequential machines is described in [2]. In summary, there is a recursive state splitting starting from only one SE. Then, a decision to split data into different SEs is based primarily on the output entropy $H(Y|Q = q_i, M = m_k) = H(\mathcal{P}_y(q_i, m_k))$ and then on the next state entropy $H(Q|Q = q_i, M = m_k) = H(\mathcal{P}_q(q_i, m_k))$, i.e., choosing state vectors that convey the most information (i.e., highly indeterministic) [2], where $H(\mathcal{P}) = -\sum_{i=1}^{n} p_i \log p_i$ and $p(q_{t+1}) = \mathcal{P}_q(q_i, m_k)$ and $p(y_{t+1}) = \mathcal{P}_y(q_i, m_k)$ are marginal distributions of $\mathcal{P}$. Additionally, states are possibly merged if there exists an equivalence relation between two states based on determining when two SEs are not equivalent if they, in their outgoing transitions, share some input symbols and transitions that lead to discrepancies in the future output. The procedure finishes when the machine is deterministic, i.e., when the entropies for all states equal to 0.

IV. Preliminary experimental results

We extracted an event set consisting of 90 actions. The algorithm converges after 45 iterations, when 74 states were obtained. Due to space constraints we do not show the extracted automaton, but for illustration purposes we show an automaton (Moore machine) extracted from 5 actions (cf. fig. 2) after convergence. Not all possible input symbols (in the boxes) are shown.

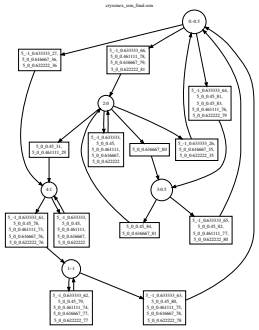

Fig. 2. Extracted finite state automaton from 5 actions. Circles are states and boxes contain input symbols

References

[1] A. Sloman, "Polyflaps as a domain for perceiving, acting and learning in a 3-D world," in *Position Papers for 2006 AAAI Fellows Symposium*. Menlo Park, CA: AAAI, 2006. [Online]. Available: http://www.cognitivesystems.org/publications/Fellows16.pdf

[2] H. Jacobsson, "The crystallizing substochastic sequential machine extractor - `CrySSMEx`," *Neural Computation*, vol. 18, no. 9, pp. 2211–2255, 2006.