WCCI 2012 IEEE World Congress on Computational Intelligence
June, 10-15, 2012 - Brisbane, Australia IJCNN

Learning Skills from Play: Artificial Curiosity on a Katana Robot Arm

Hung Ngo, Matthew Luciw, Alexander Forster, and Juergen Schmidhuber
IDSIA / SUPSI / USI
6928, Galleria 1, Manno-Lugano, Switzerland
Email: {hung, matthew, alexander, juergen}@idsia.ch

***Abstract*—Artificial curiosity tries to maximize learning progress. We apply this concept to a physical system. Our Katana robot arm curiously plays with wooden blocks, using vision, reaching, and grasping. It is intrinsically motivated to explore its world. As a by-product, it learns how to place blocks stably, and how to stack blocks.**

I. Introduction

How can embedded robots learn skills to act successfully in situations that their designers did not anticipate, ideally without a teacher?

The answers could lie in how children learn skills: by *intrinsically motivated* playing: "Children do not play for a reward-praise, money, or food. They play because they like it" [20] (p28). There are different types of playing (e.g., physical, social, following rules of games, etc. [9]); we focus on *constructive play*, where children create things by manipulating their environment, for example, by building towers from blocks. Constructive play gives children the opportunity to *experiment* with the *objects* in the environment, discovering what works and what doesn't. It leads to basic *knowledge* and *skills*, such as block stacking. Playing is recognized as being critical to future success. How can robots learn skills in a similar fashion?

There is a mathematical theory of how artificial systems can develop skills through playing [14], [15], based on two decades of work on artificial curiosity [15] (AC). According to this theory of curiosity and intrinsic motivation (IM), a creative agent needs two learning components: a general reinforcement learner and an adaptive world model (i.e., predictor/compressor of the agent's growing history of perceptions and actions). The learning progress or expected improvement of the model becomes an intrinsic reward for the reinforcement learner. Hence, to achieve high intrinsic reward, the reinforcement learner is motivated to create new experiences such that the model makes quick progress.

Here we apply principles of AC to a robot (a Katana robot arm), in an environment with a few manipulable objects (blocks). The robot plays with the objects in its environment, through an implementation of artificial curiosity. Through playing, the robot learns skills. However, the general usefulness or value of the skills is not apparent to the robot during learning. It is only driven to improve its understanding of the world as quickly as possible. The most interesting (to us) skill learned as a by-product of its curiosity-satisfying drive is a block stacking skill.

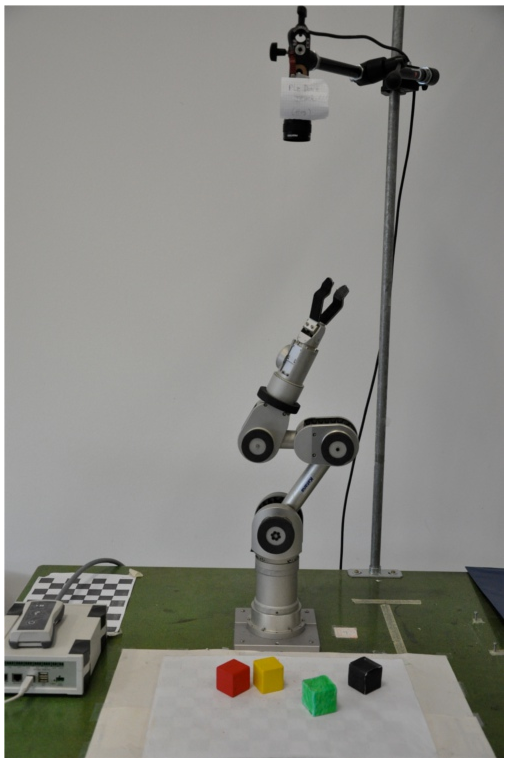

Fig. 1. The Katana robot arm in its block-world environment. It reaches for, grasps, and places blocks. The sensory input is an image from the overhead camera. There is a Markov Decision Process environment representation (discrete states and actions). Reinforcement learning updates a policy for action selection. The only reward is intrinsic: learning progress of a set of predictors (one for each state) that predict the outcome of that action. At any time, it prefers to try the state-action pairs that are associated with higher progress.

The rest of the paper is organized as follows. Section II outlines the high-level design of the system that enables skill learning through play. It describes a few challenges faced in this real-world implementation. Section III walks through the entire architecture in detail. Section IV presents experiments and results. Section V concludes the paper.

U.S. Government work not protected by U.S. copyright

II. System Design: Challenges

Our robot, a Katana arm [13], and its environment setting, which we simply call block-world, are shown in Fig 1. We would like the robot to play with the blocks and learn some useful skills as a byproduct. Two skills we desire the robot to learn: 1. how to reliably place a block stably, 2. how to reliably stack a few blocks. A more developmental system would not be designed such that we (the designers) know what the robot will eventually learn. We hope lessons learned from this system can inform an improved system so that the robot eventually learns things that even we do not expect.

Artificial curiosity couples reinforcement learning (RL) techniques with an intrinsic reward estimator. The intrinsic reward is (generally) the expected learning progress of some adaptive world model. We have to decide what the world model will be, and, given that, how to define the intrinsic reward (improvement). We can take inspiration from some previous work.

Storck *et al.* [16], albeit in a grid-world simulation, had an implementation of AC on top of an improving model of the environment, represented as a Markov Decision Process (MDP) [18]. An MDP is a four-tuple: $(\mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R})$, where $\mathcal{S}$ is a set of states, $\mathcal{A}$ is a set of actions, $\mathcal{P}^a_{s,s'}$ is the probability of transitioning from state $s \in S$ to s' when taking action a, and $\mathcal{R}^a_s$ is the expected immediate reward when taking action a in state s.

In the system of Storck *et al.*, the agent learns the transition probabilities $\mathcal{P}$ through experience. It updates an approximation of $\mathcal{P}$ after it executes some action in some state and it observes the outcome state. These probabilities are the agent's model of the world. The intrinsic reward r associated with pair (s, a) is in proportion to the Kullback-Leibler (KL) distance between the transition probabilities before and after when the agent took action a in state s. The agent's reward is based on how much the probability estimates *changed* after the new experience.

Does change represent learning progress? Yes, since a property of the probability learner is that it is guaranteed to converge to the true probabilities with enough experience. Since there is no forgetting, the agent will not get reward for learning things it knew earlier, but forgot.

The *policy* π gives action selection probabilities conditioned on the current state. Through policy iteration [3], a reinforcement learner, updates the policy using the current approximation of $\mathcal{P}$. Accurate transition probabilities allow an agent to optimally select actions to move around in the MDP. The *skills* this agent is driven to quickly learn are how to travel from some state to another.

The above system's AC is totally contained within the framework of MDPs since the intrinsic reward is based on the adaptive model of $\mathcal{P}$. Another type of approach bases its intrinsic reward on a learning system "outside" of the MDP. Kompella *et al.* [10] couple an MDP — where each state is a range of joint angles on a single joint of a robot arm, and the two actions are angle changes, e.g., go left or right — with multiple predictors. Each predictor tries to predict the output of a different sensory feature. The intrinsic reward is associated with the expected decrease in prediction error on the features. Expectations of intrinsic reward are calculated on the (recent) samples collected at each state-action.

We can say the external (outside the MDP framework) learner represents *knowledge* about some property of the world, and the AC system drives the agent to create policies (upon the MDP) that will improve the agent's knowledge (most quickly improve the performance of the predictors).

As in both systems, we'll make the world an MDP — we'll design what is meant by state and what is meant by action. The potential downside of this approach is that some information is lost — not accounted for within the state-actions we defined — thus limiting the limits of exploration. Our system has a set of predictors outside the MDP ("knowledge"), which will be the basis for intrinsic reward. Our predictors try to predict a basic physical concept — whether the block it places will stay there after it releases it. This corresponds to two classes: "stable" or "falling". There are separate predictors, one for *each* state. The transitions in the MDP must be learned from experience ("skills"), and their improvements could be a basis for intrinsic reward. Learning these probabilities corresponds to learning skills. The skills are *useful* for achieving improvement in knowledge — for improving the classification ability of the associate predictors.

The elephant in the room at this point is the issue of representation. The input from the camera is 640×480 pixels. The commands to the robot are based on the continuous joint angles, which is a point in six dimensional joint-space. It is intractable to apply standard RL directly on such high-dimensional continuous spaces. What are the states? What are the actions?

To simplify the robot's world, the top-down image is not handled holistically, but instead scanned over locally using much smaller "receptive fields". Each receptive field can be thought of as a window into a world where the robot can explore. Yet, what the robot learns in one world applies to all other worlds. Block stacking can be thought of as mostly invariant to global position in the workspace. Further, it is not important how the other blocks, which have no chance of interacting with the block the robot is placing, are positioned. This is why the system prominently uses receptive-fields — an attention module. When it is holding a block and figuring out where to place it, the robot can only "consider" a small part of the workspace, small enough to include the blocks that would matter if the block it is holding were placed there.

For some block configuration, each world-under-consideration boils down into a single state and action in an MDP that applies to all the worlds. The state is the height of the stack of blocks at this location. The action is designed to capture potential stability. Details are given in an example, provided in the next section. The resulting MDP becomes a basis for tractable exploration and captures the skills of stable block placement and block stacking.

Another challenging issue emerged in development on a real

robotic platform: there is a limited time for learning. In real-world systems, time and the number of repeated executions needed are limited resources. Each environment interaction takes several seconds to complete. First, as mentioned above, the MDP world constrains the amount of stuff the robot can do, ensuring the robot will get to something interesting in the limited time we have. Second, we use RL techniques based on dynamic programming to quickly formulate policies. Such techniques are *global*, as they can update all values at once, which is essential for systems driven by intrinsic reward. We use Least Square Policy Iteration (LSPI) [12]. LSPI greatly reduces the number of samples needed for learning.

Now we can discuss our working definition of skill: a combination of policy and approximate transition model $\mathcal{P}$. Due to approximate dynamic programming, if $\mathcal{P}$ (or the relevant part thereof) is good enough, the robot can formulate the policy (instantiate the skill) all at once. At this point, we can say the robot has learned that skill. A skill is for reaching some state from the current state. The original purpose of the skill is to improve the robot's knowledge (improve the predictor as quickly as possible). After the intrinsic rewards have been exhausted — once all the predictors are as good as possible — we can say the robot has learned all the skills. These skills may become useful for another purpose later (i.e., external reward).

Another challenging issue was the robot's physical limitations. Actions that might lead to awkward collisions must be avoided, since, in the worst case, they could break the robot. For example, the gripper in our Katana arm is not amenable to the robot forcing it forward onto a surface, yet it does not give force feedback signals, which we might like to use as negative reward (pain). In our system, there is a teacher, who plays a transparent but essential role, which is that of a "safety monitor". His job is to prevent catastrophic physical collisions of the robot. This involves, for example, moving the block the robot is reaching for slightly if the gripper is about to collide with it. In childrens' play, the teacher has a central role [11]. The teacher often shapes the environment so learning is more likely or easier, related to the scaffolding concept of Vygotsky [4].

III. Details of the System

A. Architecture

Fig. 2 shows the system architecture.

Perception. There are four different colored blocks somewhere in the robot's play area. The perception module detects the center positions and orientations of those which are visible from above. It takes a high-dimensional camera image as input, does an $L.A.B$ color space conversion, thresholds to four binary images through different rule sets for each block color (each block has a different color), fills in small holes, and takes the largest connected component in each image as the block. It outputs the visible blocks' centroids and orientations, in pixel coordinates.

Memory. This module maintains a list of estimated block positions, orientations, and heights. It uses input from the perception module and some basic built-in inferences, to update the world model. We need this to keep track of the blocks underneath other blocks. We also need this to estimate the height at different locations.

The memory module updates the world model after each block placement using an outcome perception. This model update takes as input: the previous world model, the ID (color) of the last placed block, the detected blocks from the overhead image, and the inferred outcome of the placement. The outcome is one of two options: "stable" or "falling". In one case, the block stays where it is placed, in which case the color at this location becomes that of the placed block and the height increases. In the other case, the resulting height of the placed color is less than expected. The first case corresponds to stable, and the second corresponds to unstable.

Attention. This module breaks the image into smaller overlapping (40×40 pixels) receptive fields. Each receptive field subimage corresponds to a location of potential placement. Every subimage is encoded as a discrete *state-action* pair. The *state* at this location is the maximum height $\in \{0, 1, 2, 3, ...\}$ of any existing stack here. The *action* $a \in \{0, 1, 2, 3, 4, 5\}$ encodes properties of the surface upon where the robot will place the block (explained further below).

Using the memory module's information about the occluded blocks, it converts the subimage to a binary image, with each bit set to 1 if it is a block pixel of *all* blocks beneath, and 0 otherwise. A feature detection function outputs a 1 if the center of the receptive field image is inside the convex hull constructed from all the connected pixels within this binary image, and 0 otherwise. To take into account the noise in its vision and movement system, the system applies this function centered at four other points; here we let it move three pixels along four directions (Up, Down, Left, Right).

We might use each possible binary vector as a unique action, leading to $2^5 = 32$ actions. But we can remove redundancy due to symmetry of the binary features. They are further grouped into 6 groups based on number of "on" bits. There are 6 possible actions. We think an example would be useful at this point — see Figs. 3 and 4.

The attention module maps the current situation to a set of the possible transitions in the global MDP model.

It is computationally heavy to extract the state-action at all receptive-field positions. We designed a biased-search mechanism, which is informed by the value system and the memory system. Its memory gives it the heights of the blocks in the workspace, which informs it of the possible state-actions currently available. It then sorts the values of these possible state-actions to get a desired state-action. The memory also tells it the locations where the blocks currently are. It will preferentially search around these locations — unless the most desired transition is to place a block at height zero. As soon as the robot finds the desired state-action, the search halts.

Cognition. This includes the two predictive models of how the environment reacts to the robots actions: an MDP model, which predicts the next state from the current state and action, and the sensory predictors — one associated with each state.

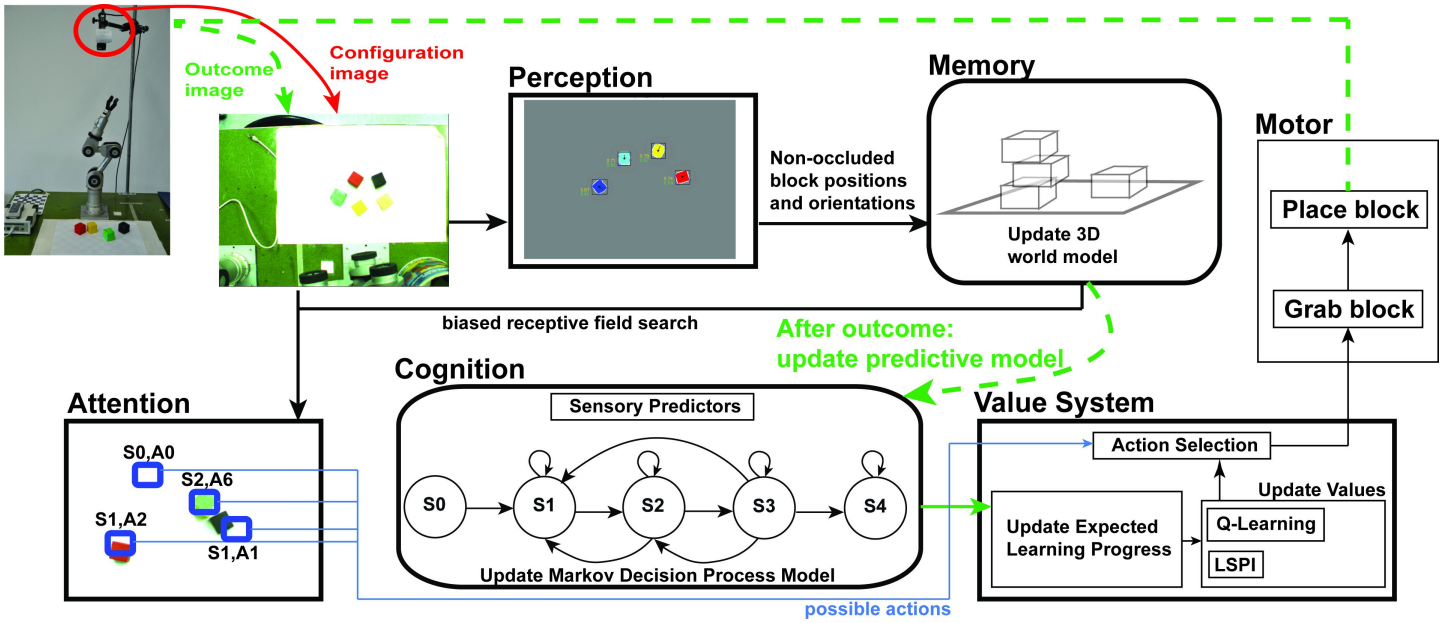

Fig. 2. System Architecture. See text for details.

We do this since the dynamics are different at different block-stack heights (noise accumulation). For a given state-action, the appropriate predictor predicts whether the placed block will stay where it is placed (stable/unstable). We want to approximate the underlying dynamics of block-world as a linear system on the extracted features — basically, the more bits set, the more stable it should be.

The state-action encoding leads to a learnable transition model $\mathcal{P}^a_{s,s'}$. Since we have a small number of states and actions, values of $\mathcal{P}$ are stored in a table. The model is updated using basic probability estimation from samples.

Each sensory predictor is realized by a *Regularized Least Squares* [8], [19] binary classifier. A measure of confidence improvement in the predictive estimates of the classifier is used as an intrinsic reward.

Value. The value system measures learning progress as a function of how the models improve after each update. RL method — LSPI — assigns value to all state-actions.

Instead of the agent being in a certain state, and selecting from all actions (typical in RL), at any time our system could potentially be in several possible states, each of which has just a subset of actions available to it. These possible transitions inform the action selection module inside the value system, which will select a transition with the highest value.

Motor. The overhead camera has been calibrated [5], and the system can convert pixel coordinates to the robot's arm-centered coordinates. When picking a block up, the robot arm selects one randomly. For placing, the motor module is advised by the action selection of the value system. To perform a place action, the robot selects a single receptive field, corresponding to the highest value of the corresponding state-action pair. The corresponding pixel-coordinates and height (we fixed the orientation) are given to the motor module, and the robot places the block at that location. It places gently upon the highest surface at that point (using input from the memory module). After releasing the block, it moves aside so as not to block the view of the overhead camera.

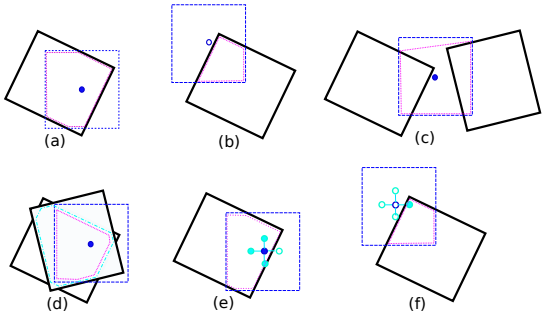

Fig. 3. Feature extraction from the attention module. The thick lines represent the boundaries of block pixels. The receptive fields are represented by dotted squares, centered at the selected placement locations (blue small circles). The pink lines represent the convex hulls of the block pixels inside the receptive fields. A feature is set to a) one if the placement location is inside the convex hull, or b) zero otherwise. Note the case c) where the selected placement position is outside any block underneath the receptive field, but the placement location is inside the convex hull, and thus the feature is still set to one. For stacks of several blocks as in d), the intersection of all the block pixels are constructed first, on which the receptive field is then applied to construct the convex hull. To account for the noise, the receptive field is shifted around the selected placement location to extract an extended set of features. In our implementation we shift the receptive field in 4 directions, so the set has 5 features. With placement location as in e) the set of features has 4 bits set to one, and as in f) only one bit set.

B. Procedure: Curious Behavior in the Block-World

1) The camera takes a snapshot.

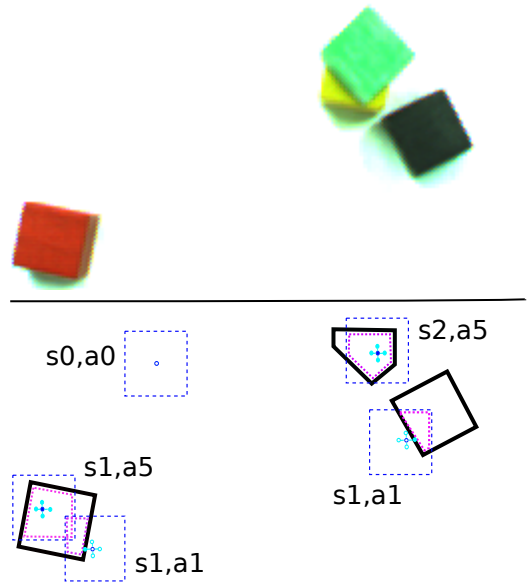

Fig. 4. An illustration of states and actions with a block configuration having one stack of height 2, and two other blocks on the table. Below: several possible placement locations, with associated state and action represented by the height and feature set of the blocks underneath the receptive fields centered at those placement locations. Note that this formulation allows generalization: placing on top of a block with set of features having one bit set is exactly the same (s_1, a_1) for both the red block and the black block.

2) The Perception System detects the positions and orientations of the blocks on top.
3) The Memory System updates the 3D world model (position, orientation, height of each block in the world).
4) The Value System decides among the current top blocks, which state (height) and action it should try.
5) Selective Attention searches in the current configuration among receptive fields for a good placement location, biased by the desired state-action selected above.
6) Selective Attention searches for any available block *outside* the selected placement location above, to pick up.
7) The Motor System uses action primitives (e.g. reach, grasp, move up, move down, release, move out of camera view) to pick up the block at place at the selected placement location.
8) The associated sensory predictor predicts the outcome (Stable/Falling).
9) Do steps 1-3 again.
10) The actual outcome (Stable/Falling) is inferred from the changes in the 3D world model. It then updates its predictive model associated with this state-action, and calculates its learning progress.
11) Value System uses the learning progress to update its Q-values and update the policy.
12) Return to step 1.

C. Learning (Model, Value, and Policy Updating)

Updating the MDP. For a transition update, we need s, a, s'. Both s and a are given by the attention module and the action selection. The outcome state s' will be the height of the stack after block placement. To get the outcome height, a new snapshot is taken, and the memory module updated. The transition probabilities from state s under action a are represented by a vector $\mathbf{c}_{s,a}^{state}$ where the i-th entry in the vector represents the probability of transitioning to the i-th internal state. These probabilities can be updated at once via vector addition:

$$\mathbf{c}_{s,a}^{state} \leftarrow (1-\mu)\mathbf{c}_{s,a}^{state} + \mu \mathbf{y}', \tag{1}$$

where $\mathbf{y}'$ is a vector whose entries are 0 except for the entry corresponding to the outcome state s' where it is 1. The learning rate μ decreases (inversely proportional) with the number of transitions [2].

Updating the Appropriate Sensory Predictor. The sensory predictor associated with the current state predicts the result of the placement in terms of the result — either stable or unstable.

Each predictor is an online linear regressor implemented using the well-known *Regularized-Least-Square* (RLS) algorithm [8], [19], which operates similarly to 2nd-order (or ridge regression)-like algorithms. The RLS algorithm is a well-studied algorithm with a provably optimal online regret bound [19], and it is efficient in implementation.

The predictor takes as input the attended binary feature vector $\mathbf{x}_{t(s)} \in \{0,1\}^7$. See Fig. 4. Each of the first six components corresponds to each of the six actions, and the last bit is a bias (always on). If all the five dots are filled, for example, then $\mathbf{x}_{t(s)} = (0,0,0,0,0,1,1)$. $t(s)$ is the number of training samples this state's predictor has encountered. In what follows we will omit the s, but one should note that each different predictor is treated completely separate from the others, meaning there are completely separate counters, weights and model parameters.

The current state-action corresponds to the current input feature vector: $\mathbf{x}_t$. The predictor tries to guess whether the result is stable or falling: $\hat{y}_t = \text{sign}(\mathbf{w}_{t-1}^\top \mathbf{x}_t)$. After observing the true outcome $y_{t(s)}^s \in \{-1, 1\}$, the weights $\mathbf{w}_t$ are updated using Algorithm 1.

Intrinsic Reward Calculation. How can we assign an expected reward to the state-action pairs? The "true" learning progress would be the reduction in deviation between the estimated decision boundary and the optimal Bayesian decision boundary from the previous time to the current. Whatever state-action is associated with the current time would be assigned intrinsic reward equal to this deviation reduction.

But we cannot compute the true learning progress directly since the Bayesian optimal decision boundary is unknown to the robot. As an approximate solution, we approximate the learning progress using the reduction in confidence interval of the RLS margin estimate corresponding to each action, representing progress towards the optimal decision boundary.

Algorithm 1: RLS-UPDATE$(\alpha, t, \mathbf{x}_t, y_t, \mathbf{A}_{t-1}^{-1}, \mathbf{b}_{t-1})$

```
//t: number of updates for this predictor
//x_t: input (action representation)
//y_t: output (stable/falling)
if t = 0 then
    A_0^{-1} ← (1/α) I // d × d matrix
    w_0 ← 0 // d × 1 vector
    b_0 ← 0 // d × 1 vector
else
    //Sherman-Morrison update
    A_t^{-1} ← A_{t-1}^{-1} − ((A_{t-1}^{-1} x_t)(A_{t-1}^{-1} x_t)^⊤) / (1 + x_t^⊤ A_{t-1}^{-1} x_t)
    b_t ← b_{t-1} + y_t x_t
    //Update model
    w_t^⊤ ← b_t^⊤ A_t^{-1}
end
return {A_t^{-1}, b_t, w_t}
```

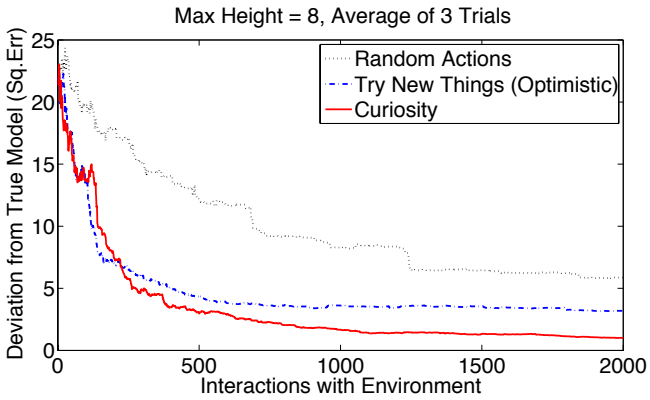

Fig. 5. Comparison of learning efficiency (deviation between current predictor weights and the optimal ones) in a simulated version of block-world (see text for details). The comparison is between a purely random policy, a policy for trying things not tried before — which acts randomly once all things have been tried — and our method of learning-progress-based artificial curiosity.

The idea of exploiting second order information in reward distribution estimates as a bonus for balancing exploration-exploitation has been applied quite successfully in the RL literature (e.g., [1], [17]), but, as far as we know, never in a purely curious system. The current confidence interval is

$$\eta_t = \sqrt{\mathbf{x}_t^\top A_t^{-1} \mathbf{x}_t}. \tag{2}$$

and the previous one,

$$\eta_t^- = \sqrt{\mathbf{x}_t^\top A_{t-1}^{-1} \mathbf{x}_t}, \tag{3}$$

is calculated using A_{t-1}^{-1} *prior* to incorporating input data $\mathbf{x}_t$. Then the learning progress (intrinsic reward) for the state-action pair (s, a) — which was attempted at time t — be

$$\mathcal{R}_s^a(t) = \eta_t^- - \eta_t \tag{4}$$

Two important properties of η_t are worth mentioning: it diminishes as the number of observations of each action increases; and it also has the potential to diminish for a particular action when other actions are taken (e.g., if they are correlated). See appendix for more information.

Updating the Values and Policy. Values are initialized optimistically. After each block placement, the system updates all Q-values at once using the version of LSPI that uses the MDP model [12]. In our case, it uses the approximation of $\mathcal{P}$, and the learning progress at each state-action as expected reward $\mathcal{R}$. Through LSPI-enabled policy iteration, the policy updates.

IV. Experiments and Results

A. In Simulation

Environment. We designed a simulation in order to compare the learning efficiency of our method to a few other possible methods. In simulation, hundreds of trials can be run, which would take far too long on the real robot. Of course we cannot capture all aspects of the real-world robot setting, but we try to capture enough so that we can compare a few different methods of generating policies.

The simulated environment allows us to analyze performance for any number of blocks (corresponding to maximum tower height) — here, we use eight. Each height up to eight has a different weight vector $\mathbf{u}$, as the "true model" for this height, which enables us to generate simulated block placement outcomes in lieu of real physics. For each height, the components of $\mathbf{u}$ are randomly generated between -1 and 1.

At any time, the eight blocks' configuration is represented by vector $\mathbf{q}$. Each element $|q_j|$ is the height of the corresponding block j. We set $-1 = sign(q_j)$ if that block is occluded from top view (stacked upon), or $1 = sign(q_j)$ means block j is on top. The set of different positive elements of $\mathbf{q}$ constitute the set of current available states (heights to place upon) — in addition to height zero, which is always available.

As an example $\mathbf{q} = (-1, -2, -3, 4, -1, 4, -2, -3)$ is a configuration with two different stacks of height four, having block IDs 4 and 6 on top of the two stacks. The set of available states will be height zero and height four. All six actions are available for each available state. The available transitions are pairs from the available states and available actions. Some policy will be used to select one of the available transitions to execute.

After selecting the state and action, the agent picks an "available" — meaning it will not disrupt execution of the selected transition — top block, and "places it". The outcome (stable/falling) is generated using the corresponding height's true (probabilistic) model, where the actual outcome label $\text{sign}(\mathbf{u}^T \mathbf{x}_t)$ is flipped with probability $\frac{1-|\mathbf{u}^T \mathbf{x}_t|}{2}$.

If the outcome turns out to be unstable, the placed block goes to height one as a top block, and one block in the stack with a lower height (randomly selected) becomes a top block. The values of other, higher blocks in the same stack are set

to $+1$.

Experiment. We compare our method with purely random action selection and a method of "optimistic initialization", which simply only assigns expected reward (=1) to state-actions not tried before (similar to R-max [6]). Other expected rewards are zero. Once the optimistic system exhausts all state-actions, it acts randomly. In both our method and the optimistic initialization method (until it visits everywhere), LSPI for planning is used at every time step. In our method, we use regularization parameter (for RLS) $\alpha = 1$, discount factor $\gamma = 0.9$ and learning rate 0.1.

To compare the effectiveness of these methods, we measure the sum of squared distances between estimated weight vectors $\mathbf{w}_t^i$ and their ground-truth $\mathbf{u}^i$ for all $i = 1, ..., 8$.

Result. Figure 5 shows the distance to the true model over time for the three methods. Our method explores the environment more efficiently in the long run, but early on the optimistic initialization method is actually more efficient.

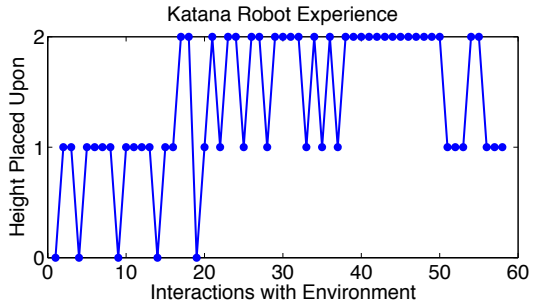

Fig. 6. Developmental stages of the curious robot arm.

B. On the Real Robot

Environment. Now we analyze the curious behavior of the system on the real Katana robot arm. There are four cubic wooden blocks of edge size 25mm with different colors. With tower height four, the robot arm does not have many feasible workspace points for the pick and place task. Hence we limit the maximum height to three.

Experiment. We initially set out the blocks equally spaced, and all at height one. Our system was then started. We collected data from a run of 56 environment interactions (picking up a block and placing it). At one point, the robot's action failed and the experiment had to be restarted, with the blocks and the learning system both in the same configuration as before.

Result. The left of figure 7 shows the learning progress for all actions at heights one and two (we omit the trivial learning progress at heights 0 and 3). On the left we plot time-varying learning progress for each input vector representing one action. We observe sharp drops of learning progress over time, corresponding to when the robot tries this state-action.

The right of Figure 7 shows the predictive distribution of the outcomes at these state-actions (less than 0.5 means unstable). These estimations initially change rapidly as new data comes in, but most seem to be converging to a sensible result. This is the nature of the RLS predictor: the more training data, the more confident the estimates. The result of action five on height two is not what we expected since four bits being on should mean the result would be stable. It turns out there was some problems with sensing the green block from vision. Since this result comes from a single run on the robot, these errors cause the problem with this plot.

Figure 6 shows the "developmental stages" of the system. Initially, it focuses attention on placing blocks on the floor. Then, it becomes interested in placing them on top of another single block (height one). Even when it occasionally observes a stable situation (leading to height two) it still wants to continue exploring at height one. Once the expected learning progress at height one gets small (see Figure 7), the system shifts its attention to exploring at height two, which it now knows how to reach through skills learned to achieve knowledge at the lower height.

After learning, the robot has gained the skills of how to place a block stably, and how to stack blocks. Please refer to our videos[1] of the real experiment.

V. Conclusions

We addressed challenges to implementing artificial curiosity on real robots. Our Katana robot interacts with its block-world through vision, reaching and grasping. It plays, via a reinforcement learning system with intrinsic reward based on expected learning progress, and, as a byproduct, learns skills such as stable placing of blocks, and stacking.

Appendix

Let's show that Eq. 2 represents the confidence interval. Let $\widehat{\Delta_t} = \mathbf{w}_t^\top \mathbf{x}_t$ be the margin parameter computed via the RLS estimate on sample t's action. The margin parameter on each action can be viewed as a measure of confidence for the classification of this action's stability. We'd like to quantify the uncertainty in these estimates. To do so, we introduce bias and variance bounds, B_t and $\mathbf{v}_t$. The bias is

$$B_t = \mathbf{u}^\top A_1 A_t^{-1} \mathbf{x}_t, \tag{5}$$

where $\mathbf{u}$ are the optimal weights (for the Bayes decision boundary) — $\Delta_t = \mathbf{u}^\top \mathbf{x}_t$ is the margin parameter of the Bayes-optimal classifier, parameterized by $\mathbf{u}$. The variance is

$$\mathbf{v}_t = S_{t-1}^\top A_t^{-1} \mathbf{x}_t, \tag{6}$$

$S_t = [\mathbf{x}_1, ..., \mathbf{x}_t]$ is the design matrix. It has been proven [8] that, for the bias bound:

$$|B_t| \leq \|\mathbf{u}\| \sqrt{\eta_t^2 + \eta_t^2}, \tag{7}$$

and for the variance bound:

$$\|\mathbf{v}_t\|^2 \leq \eta_t^2. \tag{8}$$

[1] A video of the system can be found here: http://www.idsia.ch/~ngo/ijcnn2012/katana_curiosity.html.

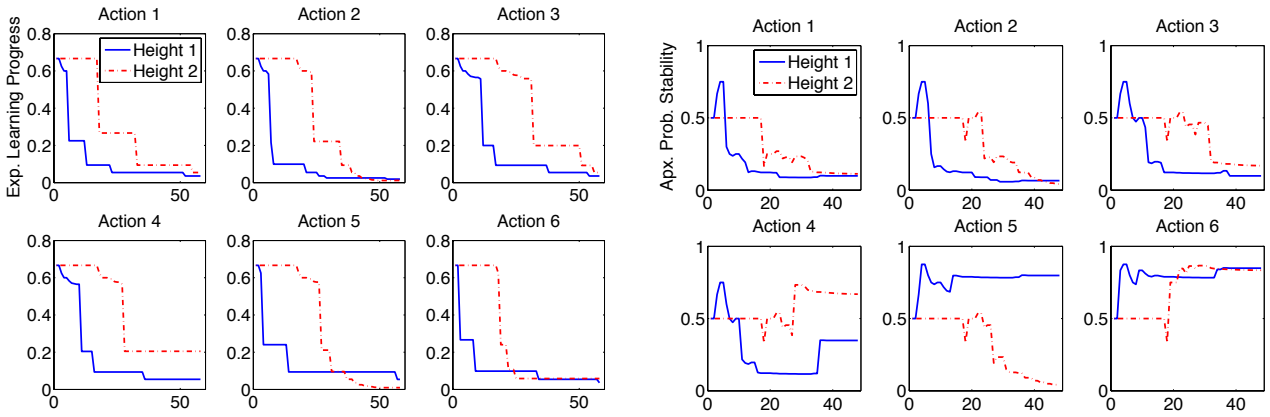

Fig. 7. Left: intrinsic reward diminishes rapidly as environment regularities are learned. Right: Predictive knowledge gained through playing experience.

Further, we note the following standard properties of the RLS estimate [7], [8]):

$$\mathbb{E}[\widehat{\Delta_t}] = \Delta_t - B_t \tag{9}$$

and for $\forall \epsilon > 0$,

$$\mathbb{P}\left(|\widehat{\Delta_t} + B_t - \Delta_t| \leq \epsilon \middle| \mathbf{x}_1, ..., \mathbf{x}_{t-1} \right) \leq 2 \exp\left(-\frac{\epsilon^2}{2\eta_t^2} \right). \tag{10}$$

We can see that η_t^2 represents the confidence intervals of the algorithm corresponding to each particular input $\mathbf{x}_t$; the more observations of a particular input $\mathbf{x}_t$ the algorithm gets, the higher its confidence on the predictive margin $\widehat{\Delta_t}$ is, hence the smaller (non-negative) η_t becomes.

The two properties above show that at the current time step t, a small confidence interval η_t indicates that the RLS margin $\widehat{\Delta_t}$ is close to the Bayes optimal decision boundary Δ_t with high probability, since the bias B_t and variance bound $\|\mathbf{v}_t\|^2$ of this estimate are bounded by (functions of) this confidence interval.

Acknowledgment

This work was funded through the 7th framework program of the EU under grants #231722 (IM-Clever project) and #270247 (NeuralDynamics project), and by Swiss National Science Foundation grant CRSIKO-122697 (Sinergia project).

The authors would like to thank Mark Ring, Faustino Gomez, Vincent Graziano, Sohrob Kazerounian, Varun Kompella, and Vien Ngo for constructive comments and encouragement.

References

[1] P. Auer. Using confidence bounds for exploitation-exploration trade-offs. *The Journal of Machine Learning Research*, 3:397–422, 2003.

[2] K.S. Azoury and M.K. Warmuth. Relative loss bounds for on-line density estimation with the exponential family of distributions. *Machine Learning*, 43(3):211–246, 2001.

[3] R. Bellman. Adaptive control processes: a guided tour. *Princeton University Press*, 1:2, 1961.

[4] L.E. Berk, A. Winsler, et al. *Scaffolding children's learning: Vygotsky and early childhood education*. National Association for the Education of Young Children Washington, DC, 1995.

[5] J.Y. Bouguet. Camera calibration toolbox for matlab. (2009).

[6] R.I. Brafman and M. Tennenholtz. R-max-a general polynomial time algorithm for near-optimal reinforcement learning. *The Journal of Machine Learning Research*, 3:213–231, 2003.

[7] N. Cesa-Bianchi, C. Gentile, and F. Orabona. Robust bounds for classification via selective sampling. In *Proceedings of the 26th Annual International Conference on Machine Learning*, pages 121–128. ACM, 2009.

[8] N. Cesa-Bianchi, C. Gentile, and L. Zaniboni. Incremental algorithms for hierarchical classification. *The Journal of Machine Learning Research*, 7:31–54, 2006.

[9] J.E. Johnson, J.F. Christie, T.D. Yawkey, and F.P. Wardle. *Play and early childhood development*. Scott, Foresman & Co, 1987.

[10] V.R. Kompella, L. Pape, J. Masci, M. Frank, and J. Schmidhuber. Autoincsfa and vision-based developmental learning for humanoid robots. In *Humanoid Robots (Humanoids), 2011 11th IEEE-RAS International Conference on*, pages 622–629. IEEE, 2011.

[11] S. Kontos. Preschool teachers' talk, roles, and activity settings during free play. *Early Childhood Research Quarterly*, 14(3):363–382, 1999.

[12] M.G. Lagoudakis and R. Parr. Least-squares policy iteration. *The Journal of Machine Learning Research*, 4:1107–1149, 2003.

[13] AG Neuronics. Katana user manual and technical description.

[14] J. Schmidhuber. Developmental robotics, optimal artificial curiosity, creativity, music, and the fine arts. *Connection Science*, 18(2):173–187, 2006.

[15] J. Schmidhuber. Formal theory of creativity, fun, and intrinsic motivation (1990-2010). *IEEE Transactions on Autonomous Mental Development*, 2(3):230 –247, 2010.

[16] J. Storck, S. Hochreiter, and J. Schmidhuber. Reinforcement driven information acquisition in non-deterministic environments. In *Proceedings of the International Conference on Artificial Neural Networks, Paris*, volume 2, pages 159–164, 1995.

[17] A.L. Strehl and M.L. Littman. Online linear regression and its application to model-based reinforcement learning. *Advances in Neural Information Processing Systems*, 20:1417–1424, 2008.

[18] R.S. Sutton and A.G. Barto. *Reinforcement learning: An introduction*. The MIT press, 1998.

[19] V. Vovk. Competitive on-line statistics. *International Statistical Review*, 69(2):213–248, 2001.

[20] F. Wardle. Getting back to the basics of childrens play. *Child Care Information Exchange*, 57:27–30, 1987.