Learning Elementary Movements Jointly with a Higher Level Task

Jens Kober and Jan Peters

***Abstract*— Many motor skills consist of many lower level elementary movements that need to be sequenced in order to achieve a task. In order to learn such a task, both the primitive movements as well as the higher-level strategy need to be acquired at the same time. In contrast, most learning approaches focus either on learning to combine a fixed set of options or to learn just single options. In this paper, we discuss a new approach that allows improving the performance of lower level actions while pursuing a higher level task. The presented approach is applicable to learning a wider range motor skills, but in this paper, we employ it for learning games where the player wants to improve his performance at the individual actions of the game while still performing well at the strategy level game. We propose to learn the lower level actions using Cost-regularized Kernel Regression and the higher level actions using a form of Policy Iteration. The two approaches are coupled by their transition probabilities. We evaluate the approach on a side-stall-style throwing game both in simulation and with a real BioRob.**

I. Introduction

In many motor skill tasks, the agents attempt to improve their motor skills at the same time as performing the task. In these settings it is important to balance learning of the individual actions by practicing them while at the same time, focusing on the overall performance in order to achieve the complete skill. Prominent examples are leisure time activities such as sports or motor skill games. For example, when playing darts with friends, you will neither always attempt the lowest risk action, nor always try to practice one particular throw, which will be valuable when mastered. Instead, you are likely to try plays with a reasonable level of risk and rely on safe throws in critical situations. This exploration is tightly woven into higher order dart games.

Optimal strategies often greatly differ according to the skill level of the player. For example, in the dart game the optimal target for obtaining the maximum number of points with a single throw differs greatly depending on the accuracy of the player [1]. For a professional player the best option is to try to hit the triple 20. For a beginner, who exhibits a high variation in throws, this target poses a high risk as the low scoring fields 1 and 5 are right next to it. For a beginner the best option is to aim at the left side of the 25 ("bull") ring which has the neighbors "double bull", 14, 11, and 8. This type of game can be modeled by a Markov decision process (MDP) and solved using traditional reinforcement learning methods [2].

In this paper, we address the problems the player faces when learning during a game. In particular, the player has to decide for each throw if he wants to try to improve his level of performance for the chosen action by exploration or go for a more certain reward exploiting his abilities acquired so far. As the skill level of the player constantly improves also the higher level strategy needs to be constantly adapted. This type of learning is a special form of hierarchical learning, where improving the individual throws corresponds to a lower level of learning, and the game strategy to a higher level. In the notions common in reinforcement learning, one could consider the lower level behaviors a type of option of the strategy level.

Hierarchical learning for robotics has been investigated in the supervised setting [3]–[9] as well as in the reinforcement learning setting [10]–[14]. Our approach uses a two-level hierarchy and we introduce novel concepts for the lower level learning and the interaction between the levels. We propose to learn the lower level behaviors using Cost-regularized Kernel Regression [15]. This paper demonstrates how the two levels are coupled by the transition probabilities of the higher level, which can be estimated from the learned parameters of the lower level. Our approach cannot discover options and assumes that these are non-interruptible.

In Section II, we will discuss the components of the framework. In Section III, we will evaluate the resulting framework with a side-stall-like throwing game in simulation and on a real BioRob.

II. Two-Level Learning Approach

Our framework considers a hierarchy of two levels: a strategy level and a behavior level. The strategy level determines the strategy for the high-level moves, here termed "behaviors", of the game. The behavior level deals with executing these behaviors in an optimal fashion. The strategy level chooses the next behavior, which is then executed by the behavior level. Upon completion of the behavior, the strategy level chooses the next behavior.

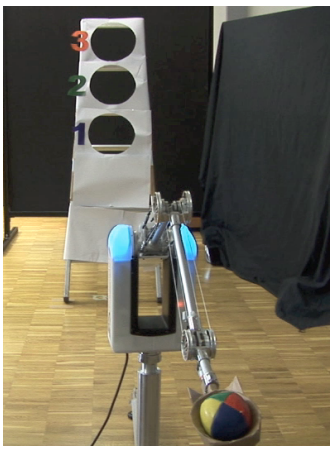


Fig. 1. This figure illustrates the setup of the robot evaluation.

We assume that the game has discrete states $s \in \mathbb{S}$ and discrete behaviors $b \in \mathbb{B}$. In the dart setting a behavior could be attempting to hit a specific field and the state could correspond to the current score. Given the current state, each

Both authors are with the Max Planck Institute for Intelligent Systems, Department of Empirical Inference, Spemannstr. 38, 72076 Tübingen, Germany and the Technische Universität Darmstadt, Intelligent Autonomous Systems Group, Hochschulstr. 10, 64289 Darmstadt, Germany {firstname.lastname}@tuebingen.mpg.de

behavior has an associated expected outcome $o \in \mathbb{O}$. The role of the behavior level is to determine the parameters required to achieve this outcome. For example, the behavior "throw at field X" has the outcome "change score by X" as a result of hitting field X. The transition probabilities of the strategy level would express how likely it is to hit a different field. We assume to have an episodic game with a finite horizon.

In the behavior level we augment the state space with continuous states that describe the robot and the environment to form the combined state space $\mathbf{s}$. This state space could, for example, include the position and velocity of the arm, the position of the dart board as well as the current score. The actions are considered to be continuous and could, for example, be the accelerations of the arm. As the strategy level has to wait until the behavior is completed the behaviors need to be of episodic nature as well.

The objective of the strategy level is to maximize the expected return

$$J(\pi) = E\left[\sum_{t=1}^{T} \mathcal{R}_t^{\pi}\right],$$

of policy π, where $E[\cdot]$ denotes the expectation, T is the length of the episode, and $\mathcal{R}_t^{\pi}$ is the immediate reward at time-step t. In the strategy level each time-step t corresponds to executing one behavior b. Alternatively the problem can be formulated as an infinite horizon problem where we define an absorbing terminal state in which all actions receive an immediate reward of 0. Using this formulation the expected return is $J(\pi) = V(s_0)$, with the value function $V(s) = \sum_{s'} \mathcal{P}_{ss'}^{\pi(s)}[\mathcal{R}_{ss'}^{\pi(s)} + V(s')]$, where s is the current state, s' the next state, $\mathcal{P}_{ss'}^{\pi(s)}$ the transition probability given the current policy π, and $\mathcal{R}_{ss'}^{\pi(s)}$ the immediate reward. Depending on the specific setting many algorithms can be applied, including dynamic programming, Q-learning, SARSA [2], or REPS [16].

As we will discuss in Section II-B we can estimate the transition probabilities for the behaviors parameters learned in the behavior level. The rewards associated to state transitions are known from the rules of the game. Thus, we have a complete model of the strategy level and can employ model-based reinforcement learning to solve the task. In this paper, we employ Policy Iteration [2], as illustrated in Algorithm 1.

Algorithm 1: Policy Iteration [2]

Initialization:
$V(s) \in \mathbb{R}$ and $\pi(s) \in \mathbb{B}(s)$ arbitrarily for all $s \in \mathbb{S}$

Policy Evaluation:
Repeat
$\Delta \leftarrow 0$
For each $s \in \mathbb{S}$:
$v \leftarrow V(s)$
$V(s) \leftarrow \sum_{s'} \mathcal{P}_{ss'}^{\pi(s)} \left[\mathcal{R}_{ss'}^{\pi(s)} + V(s')\right]$
$\Delta \leftarrow \max(\Delta, \|v - V(s)\|)$
Until $\Delta < \epsilon$ (a small positive number)

Policy Improvement:
policy-stable$\leftarrow$*true*
For each $s \in \mathbb{S}$:
$b_{old} \leftarrow \pi(s)$
$\pi(s) \leftarrow \arg\max_b \sum_{s'} \mathcal{P}_{ss'}^{b} \left[\mathcal{R}_{ss'}^{b} + V(s')\right]$
If $b_{old} \neq \pi(s)$ then *policy-stable*$\leftarrow$*false*
If *policy-stable*, then stop; else go to **Policy Evaluation**.

The objective of the behavior level is to find a behavior policy π_b that maximizes the expected return of the behavior. In order to make the learning tractable we rely on a parameterized stochastic policy $\pi_b = p(\boldsymbol{\gamma}|\mathbf{s})$, where $\boldsymbol{\gamma}$ is the set of parameters. This parametrization is identical for all behaviors b. Hence, the problem of behavior learning is to find a stochastic behavior-policy π_b that maximizes the expected return

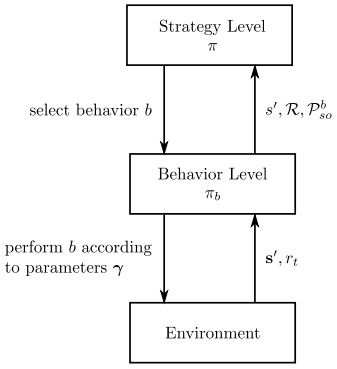


Fig. 2. This figure illustrates the setup of the roles of the different levels.

$$J(\pi_b) = \int p(\mathbf{s}) \int \pi_b(\boldsymbol{\gamma}|\mathbf{s}) R(\mathbf{s}, \boldsymbol{\gamma}) d\boldsymbol{\gamma}\, d\mathbf{s},$$

where the return of an episode with T steps is $R(\mathbf{s}, \boldsymbol{\gamma}) = T^{-1} \sum_{t=1}^{T} r_t$ where r_t denotes the rewards. We discuss our approach in Section II-A in more detail.

For our choice of a behavior level the transition probabilities for the strategy level can be approximated. Details are given in Section II-B.

When combining the three components of behavior learning (Section II-A), strategy learning (Algorithm 1), and determining the transition probabilities (Section II-B), we get a complete two-level learning system. The behavior level optimizes the parameters of the behavior to achieve the desired transitions. The parameters determine the transition probabilities, which need to be updated after each behavior learning step. The strategy learning decides which behaviors and associated explorations are safe in the current state of the game. The complete approach is illustrated in Algorithm 2 and Figure 2.

A. Behavior Level

We assume that the behaviors are parametrized with a small set of parameters $\boldsymbol{\gamma}$ that define the global behavior of the movement. For example, in the dart throwing task we would only consider the release position and velocity. The actual throwing movement of going backwards and accelerating forward on an arc would be described by an even lower level in the hierarchy. Here, we assume this level to be

Algorithm 2: Two-Level Learning

Strategy Level:
Determine transition probabilities using Equation 1
Determine optimal policy π using **Algorithm 1**.
Behavior Level:
Determine optimal policy π_b using **Algorithm 3**.
Perform behavior $b = \pi(s)$ using π_b
Determine cost c and next strategy-state s'

fixed. We have a stochastic policy $\pi_b = \mathcal{N}(\boldsymbol{\gamma}|\bar{\boldsymbol{\gamma}}(\mathbf{s}), \boldsymbol{\sigma}^2(\mathbf{s}))$, which is a Gaussian distribution with mean $\bar{\boldsymbol{\gamma}}(\mathbf{s})$ and variance $\boldsymbol{\sigma}^2(\mathbf{s})$. This problem can be solved using the Cost-regularized Kernel Regression [15].

The mean of the policy π_b is

$$\bar{\boldsymbol{\gamma}}_i(\mathbf{s}) = \mathbf{k}(\mathbf{s})^{\mathrm{T}} \left(\mathbf{K} + \lambda_i \mathbf{C}\right)^{-1} \boldsymbol{\Gamma}_i,$$

where $\boldsymbol{\Gamma}_i$ is a vector containing the training examples $\boldsymbol{\gamma}_i$ of the parameter component i, $\mathbf{C} = \mathbf{R}^{-1} = \mathrm{diag}(R_1^{-1}, \ldots, R_n^{-1})$ is a cost matrix, λ_i is a ridge factor, and $k(\mathbf{s}_1, \mathbf{s}_2)$ is a kernel. The kernel corresponds to a metric that measures the distance between the states $\mathbf{s}_1$ and $\mathbf{s}_2$. In this paper we employ a Gaussian kernel $k(\mathbf{s}_1, \mathbf{s}_2) = \exp(-\|\mathbf{s}_1 - \mathbf{s}_2\|^2/\sigma_k^2)$, where σ_k is the kernel width. The matrix $\mathbf{S}$ contains the states of all training examples. The vector $\mathbf{k}(\mathbf{s}) = [k(\mathbf{s}, [\mathbf{S}]_1), k(\mathbf{s}, [\mathbf{S}]_2), \ldots]$ measures the distance between the query state $\mathbf{s}$ and all training examples $\mathbf{S}$. The matrix $\mathbf{K}$ expresses the pairwise distance between all combinations of training examples. The variance

$$\sigma_i^2(\mathbf{s}) = k(\mathbf{s}, \mathbf{s}) + \lambda_i - \mathbf{k}(\mathbf{s})^{\mathrm{T}} \left(\mathbf{K} + \lambda_i \mathbf{C}\right)^{-1} \mathbf{k}(\mathbf{s}),$$

corresponds to the uncertainty of parameters. If the cost is high, the variance is high and we assume that we are still far from an optimal solution and, therefore, have to continue exploring. This effect is achieved by drawing the parameters $\gamma \sim \mathcal{N}(\boldsymbol{\gamma}|\bar{\boldsymbol{\gamma}}(\mathbf{s}), \boldsymbol{\sigma}^2(\mathbf{s}))$ from the stochastic policy for each episode. The complete algorithm is given in Algorithm 3.

B. Connecting the Levels

The rewards for the strategy learning are fixed by the rules of the game. The possible states and behaviors also result from the way the game is played. The missing piece for the strategy learning is the transition probabilities. For example when playing darts the behavior "throw at field X" will change the state according to the points associated with X, but if the variance is high the player will hit a neighboring field and, thus, have a different transition. The behavior learning associates each behavior with a variance. Each of these behaviors correspond to an expected change in state, the outcome o. For example "aim at 10" corresponds to "increase score by 10". However, the parameter function does not explicitly include information regarding what happens if the expected change in state is not achieved. We assume that there is a discrete set of outcomes $o \in \mathbb{O}$ (i.e., change in state) for all behaviors b for a certain state s. For example in a *501* game [17] in darts (the goal is to reach a score of 0 first) hitting each field, and missing the dart board, is associated with either lowering the player's, winning or to bust (i.e., going below zero and continuing on with the previous score the next turn). With the parameter function, we can calculate the overlaps of the ranges of possible parameters for the different behaviors. These overlaps can then be used to determine how likely it is to end up with a change of state associated with a behavior different from the desired one. This is illustrated in Figure 3. There we have three different behaviors and their associated parameters. The red and the blue behaviors have the same mean and, thus, it is very likely that the transitions resulting from these behaviors will be similar. Picking the green behavior is likely to result in the predicted outcome, as its associated parameters are fairly distinct from the two other behaviors. This approach relies on the assumption that we know for each behavior the associated range of parameters and their likelihood.

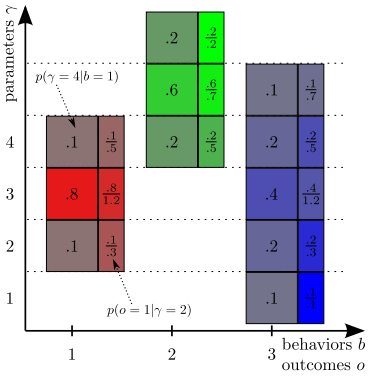


Fig. 3. This figure illustrates how the transition probabilities are calculated. We have three different behaviors (red, green, and blue) with their associated parameter value distributions (square boxes). The parameter values are discretized. The probability of ending up in the next state corresponding to an outcome when using the parameter bin value is given in the narrower boxes.

The parameters are drawn according to a normal distribution, thus the overlap has to be weighted accordingly. This is illustrated in Figure 4. The probability of the outcome o when performing behavior b can be calculated as follows:

$$\mathcal{P}_{so}^b = \int p^b(\gamma) \frac{p^o(\gamma)}{\sum_{k \in \mathbb{O}} p^k(\gamma)} d\gamma, \tag{1}$$

where γ is the parameters, $p^b(\gamma)$ is the probability of picking the parameter γ when performing behavior b, $p^o(\gamma)$ is the probability of picking the parameter γ when performing the action associated to the considered outcome o, and

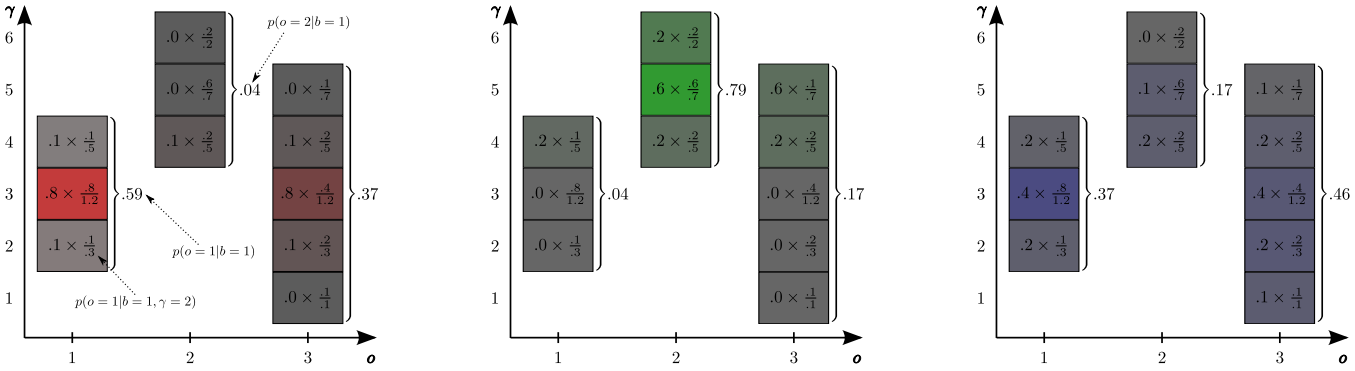
Algorithm 3: Cost-regularized Kernel Regression [15]

Initialization:

- Learn the behavior by imitation and/or reinforcement learning.
- Determine initial state $\mathbf{s}^0$, parameters $\boldsymbol{\gamma}^0$, and cost C^0 corresponding to the initial behavior.
- Initialize the corresponding matrices $\mathbf{S}, \boldsymbol{\Gamma}, \mathbf{C}$.
- Choose a kernel k, $\mathbf{k}$, $\mathbf{K}$.
- Set a scaling parameter λ.

For all iterations j**:**

- Determine the state $\mathbf{s}^j$ specifying the current situation.
- Calculate the parameters $\boldsymbol{\gamma}^j$ by:
  - Determine the mean of each parameter i $\gamma_i(\mathbf{s}^j) = \mathbf{k}(\mathbf{s}^j)^{\mathrm{T}} \left(\mathbf{K} + \lambda \mathbf{C}\right)^{-1} \boldsymbol{\Gamma}_i$,
  - Determine the variance $\sigma^2(\mathbf{s}^j) = k(\mathbf{s}^j, \mathbf{s}^j) - \mathbf{k}(\mathbf{s}^j)^{\mathrm{T}} \left(\mathbf{K} + \lambda \mathbf{C}\right)^{-1} \mathbf{k}(\mathbf{s}^j)$,
  - Draw the parameters from a Gaussian distribution $\boldsymbol{\gamma}^j \sim \mathcal{N}(\boldsymbol{\gamma}|\boldsymbol{\gamma}(\mathbf{s}^j), \sigma^2(\mathbf{s}^j)\mathbf{I})$.
- Execute the primitive using the new parameters.
- Calculate the cost c^j at the end of the episode.
- Update $\mathbf{S}, \boldsymbol{\Gamma}, \mathbf{C}$ according to the achieved result.



(a) Behavior 1 selected ($b = 1$). (b) Behavior 2 selected ($b = 2$). (c) Behavior 3 selected ($b = 3$).

Fig. 4. This figure illustrates how the transition probabilities are calculated. The probabilities from Figure 3 need additionally to be weighted with the probabilities of choosing this value when selecting one of the behaviors. E.g., Figure 4(a) illustrates the result of picking behavior 1. The next state is going to be the one associated to the left behavior in 59% of the cases, with the middle next state with a probability of 4% but in the right next state in 37%. In contrast, if we pick behavior 2 (Figure 4(b)) we are going to get its associated transition in 79% as its parameter set is very different to the other ones.

$\sum_{k \in \mathbb{O}} p^k(\gamma)$ is the normalizing factor.

III. Evaluations

A scenario where our framework is applicable is funfair games. These games usually have the setup that you perform some motor skill game and, if you were successful, you are awarded a prize. Examples of funfair games include throwing balls at coconuts, basket ball hoops, buckets, or tin cans, shooting with rifles or darts, as well as fishing or bowling like activities [18], [19]. Most players are not willing to spend a lot of money to improve their game but rather use a safe strategy. For our evaluation we require a game that has several higher level actions and we have chosen a game inspired by a side-stall game and blackjack, see Section III-A for details. We evaluate this game both in simulation (Section III-B) and on a real BioRob (Section III-C). The transition probabilities were obtained using numerical integration.

A. Description of the Side-Stall-Style Game

The game is reminiscent of blackjack as the goal is to collect as many points as possible without going over a threshold. The player throws a ball at three targets. The three rewards of one, two, and three are assigned to one target each. If the ball lands in the target, the player receives the corresponding number of points. The player starts with zero points if he gets more than 10 points he "busts" and incurs a loss of -10. The player has the option to "stand" (i.e., stop throwing and collect the accumulated number of points) at all times. Missing all targets does not entail a cost.

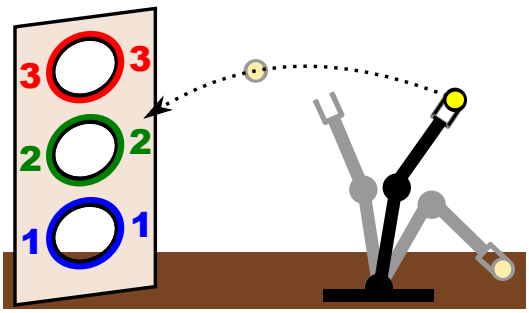


Fig. 5. This figure illustrates the side-stall game. The player throws the ball and if it lands in the target (illustrated by a wall with target holes) gets the number of points written next to it. Missing the targets is not punished, however, going over ten points leads to a loss of ten points.

The game can be modeled as an MDP where the states consist of the number of accumulated points (zero to ten) and two additional game states ("bust" and "stand"). The actions correspond to attempting to throw at a specific target or to "stand". The rewards are fixed according to the rules of the game and the transition probabilities are estimated based on the learned behavior to parameter mapping.

B. Evaluation in Simulation

We first evaluated our approach using a MATLAB based simulation. The throw is modeled as a two dimensional ballistic flight of a point mass. The targets correspond to segments of the ground line. The parameters are the initial horizontal and vertical velocities of the ball. The parameters used to initialize the learning make the ball drop in front of the first target. The cost function for the behavior level is the sum of the squared initial velocities and the distance between the desired and the achieved outcome. Figure 6 illustrates how the player learns to throw more accurately while playing. Figure 7 illustrates how learning to perform the lower level actions more reliably enables the player to perform better in the game.

C. Evaluation on a Real BioRob

We employ a BioRob to throw balls in a catapult like fashion. The arm is approximately $0.75m$ long, and it can reach $1.55m$ above the ground. The targets are located at a distance of $2.5m$ from the robot at a height of $0.9m$, $1.2m$, and $1.5m$ respectively. The ball is placed in a funnel-shaped receptacle. The parameters are duration and amount of acceleration for two joints that are in the throwing plane. The robot starts in a fixed initial position, accelerates the two joints according to the parameter indicating the magnitude,

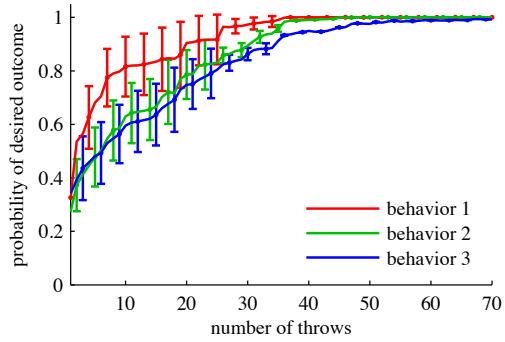


Fig. 6. This figure illustrates the transition probabilities of the three behaviors to their associated outcome in simulation. For example, the red line indicates the probability of gaining one point when throwing at target 1. After approximately 50 throws the player has improved his accuracy level such that he always hits the desired target. The plots are averaged over 10 runs with the error-bars indicating standard deviations.

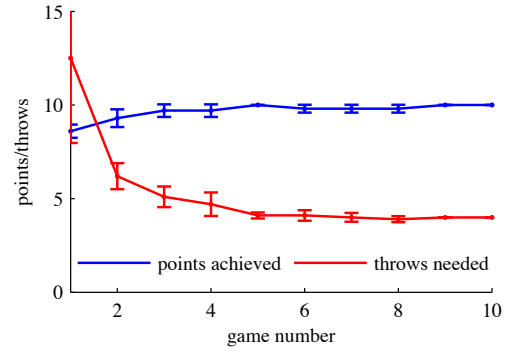


Fig. 7. This figure illustrates the improvement of the player over the number of games in simulation. Due to the large penalty for busting the framework always uses a safe strategy. Already after five completed games the player reaches almost always the maximum possible score of 10. As the number of throws is not punished there are initially many throws that miss the target. After 7 games the number of throws has converged to 4, which is the minimum required number. The plots are averaged over 10 runs with the error-bars indicating standard deviations.

and accelerates in the opposite direction after the time determined by the other parameter in order to break. Finally the robot returns to the initial position. See Figure 9 and the attached video for an illustration of one throwing motion.

Executing the throw with identical parameters will only land at the same target in approximately 60% of the throws, due to the high velocities involved and small differences in putting the ball in the holder. Thus, the algorithm has to deal with large uncertainties. The cost function for the lower actions is the sum of the squared acceleration magnitude, the squared acceleration duration, and the distance between the desired and the achieved outcome. The three components are normalized to lie in the same range. The setup makes it intentionally hard to hit target 3. The target can only be hit with a very restricted set of parameters. For targets 1 and 2 increasing the amount of acceleration or the duration will result in a higher hit. Target 3 is at the limit where higher accelerations or longer durations will lead to a throw in a downward direction with a high velocity. In combination

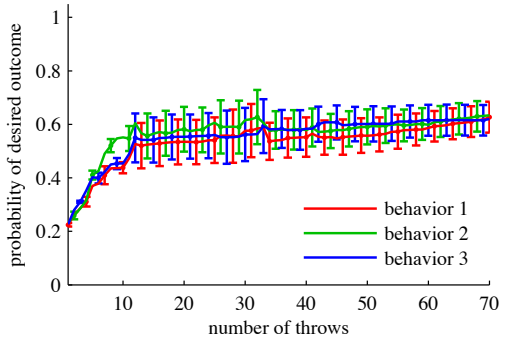


Fig. 8. This figure illustrates the transition probabilities of the three behaviors to their associated outcome on the BioRob similar to Figure 6. The skill improves a lot in the first 15 throws after that the improvement levels off. Initially action 2, associated with target 2 (which lies in the center) is most likely to succeed. The success rate of 60% corresponds to the level of reproducibility of our setup. The framework manages to handle this large uncertainty by choosing to stand early on. The plots are averaged over 4 runs with the error-bars indicating standard deviations.

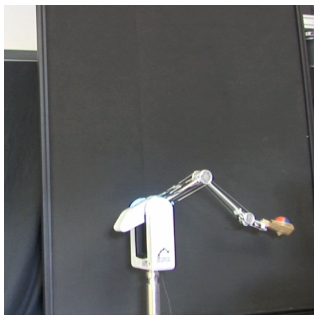
with the behavior level cost function, which favors parameters that hit target 1 or 2, it is not surprising that at the end of the learning process target 3 is not hit frequently. The robot successfully learns to consistently score 9 or 10 points with five throws. Figure 8 illustrates how the robot learns to throw more accurately within the physical limits of the system.

The typical behavior of one complete experiment is as follows: At the beginning the robot explores in a fairly large area and stands as soon as it reaches a score of 8, 9, or 10. Due to the large punishment it is not willing to attempt to throw at 1 or 2 while having a large uncertainty, and, thus, a high chance of busting. Later on, it has learned that attempting to throw at 2 has a very low chance of ending up in 3 and hence will attempt to throw 2 points if the current score is 8. We setup the policy iteration to favor behaviors with a higher number, if the values of the behaviors are identical. The first throws of a round will often be aimed at 3, even if the probability of hitting target 2 using this action is actually higher than hitting the associated target 3. Until 8 or more points have been accumulated, action 3 is safe (i.e., cannot lead to busting), does not entrain a punishment if missing or hitting a lower target, and has a large learning potential.

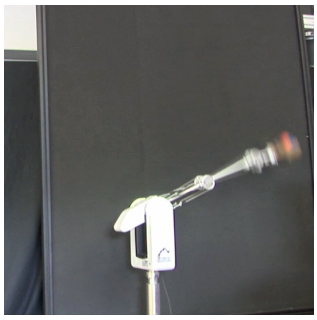
IV. Conclusion

In this paper, we have introduced a framework that allows a player to work on his skill regarding the individual components of game-play while maintaining a satisfactory level of success at the general game. This setup is especially applicable for casual games where the players learn during playing and do not resort to fine-tuning there performance by practicing skills separately.

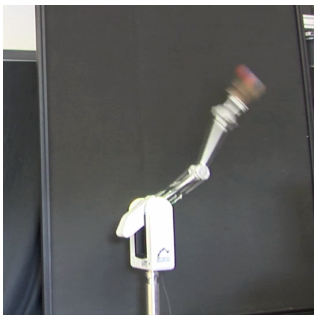
The framework is composed of two levels: the strategy level and the behavior level. The strategy is learned using Policy Iteration [2]. The behaviors are learned using Cost-regularized Kernel Regression [15]. The two levels are coupled by the transition probabilities of the strategy level,



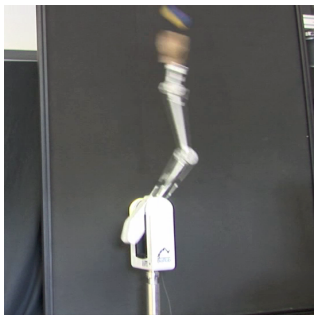
(a) The initial position.



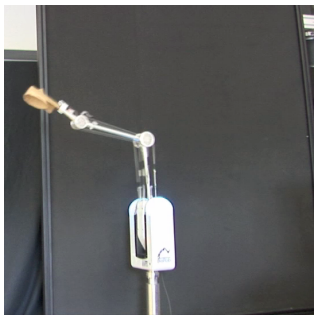
(b) The robot accelerates in the shoulder and elbow joints.



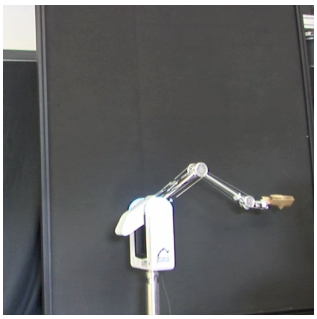
(c) End of acceleration. The breaking starts.



(d) The robot slows down while the ball flies towards the target.



(e) The robot has stopped and returns to the initial position.



(f) The robot is again in the initial position, ready for the next throw.

Fig. 9. These frames of the accompanying video illustrate one throwing motion with the BioRob.

which can be estimated from the learned parameters of the behavior level.

We have evaluated the setup on a side-stall-style game both in simulation and with a real BioRob. The real robot experiment had to deal with high uncertainties due to the limited reproducibility of the throws. The system manages to handle this issue well by not attempting high risk actions. An interesting alternative for future research would be to try to learn equifinal paths. These are paths the lead to the same outcome even if there is some variation in the execution. This strategy has been observed for human dart players [20].

REFERENCES

[1] D. Kohler, "Optimal strategies for the game of darts," *The Journal of the Operational Research Society*, vol. 33, no. 10, pp. 871–884, 1982.

[2] R. Sutton and A. Barto, *Reinforcement Learning*. MIT Press, 1998.

[3] C. Versino and L. M. Gambardella, "Learning fine motion in robotics: Experiments with the hierarchical extended kohonen map," in *Proceedings of the International Conference on Neural Information Processing (ICONIP)*, 1996.

[4] R. P. N. Rao and O. Fuentes, "Hierarchical learning of navigational behaviors in an autonomous robot using a predictive sparse distributed memory," *Auton. Robots*, vol. 5, no. 3-4, pp. 297–316, 1998.

[5] C. Urdiales, A. Bandera, E. Pérez, A. Poncela, and F. Sandoval, *Autonomous robotic systems*. Physica-Verlag GmbH, 2003, ch. Hierarchical planning in a mobile robot for map learning and navigation, pp. 165–188.

[6] C. Weber, M. Elshaw, A. Zochios, and S. Wermter, "A multimodal hierarchical approach to robot learning by imitation," in *Proceedings of the Int. Workshop on Epigenetic Robotics: Modeling Cognitive Development in Robotic Systems*, 2004.

[7] Y. Demiris and A. Dearden, "From motor babbling to hierarchical learning by imitation: a robot developmental pathway," in *Proceedings of the Fifth International Workshop on Epigenetic Robotics: Modeling Cognitive Development in Robotic Systems*, 2005.

[8] J. Z. Kolter, P. Abbeel, and A. Y. Ng, "Hierarchical apprenticeship learning with application to quadruped locomotion," in *Proceedings of the Twenty-First Annual Conference on Neural Information Processing Systems (NIPS)*, 2007.

[9] Y. Wu and Y. Demiris, "Hierarchical learning approach for one-shot action imitation in humanoid robots," in *Proceedings of the 11th International Conference on Control, Automation, Robotics and Vision (ICARCV)*, 2010.

[10] O. Fuentes, R. P. N. Rao, and M. Van Wie, "Hierarchical learning of reactive behaviors in an autonomous mobile robot," in *Proceedings of the IEEE Int. Conference on Systems, Man and Cybernetics*, 1995.

[11] M. Kaiser and R. Dillmann, "Hierarchical learning of efficient skill application for autonomous robots," in *Proceedings of the International Symposium on Intelligent Robotic Systems (SIRS)*, 1995.

[12] C. K. Tham, "Reinforcement learning of multiple tasks using a hierarchical CMAC architecture," *Robotics and Autonomous Systems*, vol. 15, no. 4, pp. 247–274, 1995.

[13] J. Diard and O. Lebeltel, "Bayesian programming and hierarchical learning in robotics," in *Proceedings of th International Conference on Simulation of Adaptive Behavior (SAB)*, 2000.

[14] V. Soni and S. Singh., "Reinforcement learning of hierarchical skills on the sony aibo robot," in *Proceedings of the 5th International Conference on Development and Learning (ICDL)*, 2006.

[15] J. Kober, E. Oztop, and J. Peters, "Reinforcement learning to adjust robot movements to new situations," in *Proceedings of the Robotics: Science and Systems Conference (R:SS)*, 2010.

[16] J. Peters, K. Mülling, and Y. Altün, "Relative entropy policy search," in *Proceedings of the Twenty-fourth National Conference on Articial Intelligence (AAAI), Physically Grounded AI Track*, 2010.

[17] Masters Games Ltd. (2011, March) The rules of darts. [Online]. Available: http://www.mastersgames.com/rules/darts-rules.htm

[18] JARM Group. (2011, March) Fairground stall hire for indoor and outdoor events. [Online]. Available: http://www.sidestalls.net/

[19] Wikipedia. (2011, March) Coconut shy. [Online]. Available: http://en.wikipedia.org/wiki/Coconut_shy

[20] H. Müller and E. Loosch, "Functional variability and an equifinal path of movement during targeted throwing," *Journal of Human Movement Studies*, vol. 36, no. 3, pp. 103–126, 1999.