Guiding Exploration by Combining Individual Learning and Imitation in Societies of Autonomous Robots

Willi Richert, Oliver Niehörster, Florian Klompmaker

Abstract Robots have a powerful means to drastically cut down the exploration space with imitation. However, as existing imitation approaches usually require repetitive demonstrations of the skill to learn in order to be useful, those are typically not applicable in groups of robots. In these settings usually each robot has its own task to accomplish and should not be disturbed by teaching others. As a result an imitating robot most of the time has only one observation of a specific skill from which it can learn.

We present an approach that allows an individually learning robot to make use of such cases of sporadic imitation which is the normal case in groups of robots. Thereby, a robot can use imitation in order to guide its exploration efforts towards more rewarding areas in the exploration space. This is inspired by imitation often found in nature where animals or humans try to map observations into their own capability space. We show the feasibility by realistic simulation of Pioneer robots.

1 Introduction

With the benefits of drastically cutting down the exploration space imitation is one of the most powerful learning techniques one can find in nature [5, 6, 8]. This has been acknowledged also by robotics researchers when they embraced different methods to apply imitation to learn tennis swings or drumming movements [10] or e.g. to forage [9]. However, except for the work on imitating skill sequences in all these experiments the demonstrator is always determined (often the human) and the time frame where the imitation has to pay attention is provided beforehand. The task to be learned by imitation is then repeated several times and the robot afterwards has to derive a generalized representation of the imitated task and be able to replay it. Up to now no research has been carried out regarding sporadic imitation, which is apparently very important when robots in groups should benefit from each

Intelligent Mobile Systems, University of Paderborn / C-LAB, Germany, `richert@c-lab.de`

Please use the following format when citing this chapter:

Richert, W., Niehörster, O. and Klompmaker, F., 2008, in IFIP International Federation for Information Processing, Volume 268; *Biologically-Inspired Collaborative Computing*; Mike Hinchey, Anastasia Pagnoni, Franz J. Rammig, Hartmut Schmeck; (Boston: Springer), pp. 233–244.

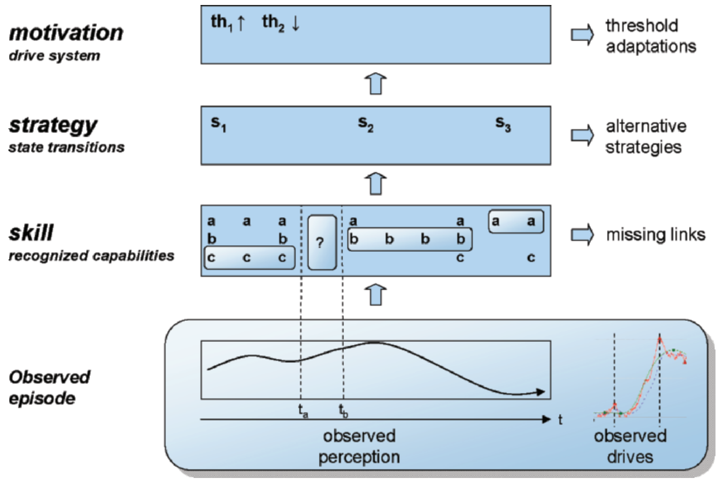


Fig. 1 Procedure of interpreting another robot's performance in order to imitate it.

others learning efforts. Typically, the imitation process should not interrupt the observed robot, so that the imitating robot often has only one example of the same type of interesting behavior to learn from. As this usually does not provide enough information for learning a generalized version of the observed action, it can help the observer to narrow the learning exploration space. This is the aim of our paper.

With the presented approach comprising the strategy and low-level skill layers an observing robot can benefit from the imitation process

1. by observing new state sequences for which it could spent more exploration efforts,
2. by observing new behaviors for already known state transitions, and
3. by incorporating other robot's transition data condensed into its own strategy.

In Fig. 1 an example is shown in which the robot (imitator) tries to *understand* the observed behavior episode of another robot (demonstrator). The observed episode consists of the recorded perception and the demonstrator's visible "well-being", a kind of emotional state that comprises its overall state in form of a set of drives. Therefore the imitator first translates the observations into its own perception to see what it would perceive if itself would have been in the demonstrator's situation. It then scans the subjective perception and allows its low-level skill to give so-called *votes* about how well each skill could have achieved the perception changes. Using an algorithm inspired by Viterbi those votes are then used together with the likeliness of the demonstrator's state space to find the most likely path corresponding to the observations. In this paper we will focus at the skill and strategy layer, as they are most important to the understanding of the observed behavior.

2 Related Work

Most approaches regarding imitation of robotic behavior are based on Hidden Markov Models (HMM) and use the Viterbi algorithm to synthesize behavior thereof. Billard et al. [4] use e.g. the Viterbi algorithm to let the upper part of a robot replay a limited set of arm movements that move colored objects. In their work the demonstrator-imitator roles are known and fixed. Also the start and end points of the behavior to imitate is known to the robot. They split the imitation task into the observation and imitation processes, having the goal to minimize the discrepancy between the demonstrated and imitated data sets. In their approach the robot is only able to learn low-level behavior and this can only be done from scratch. In contrast to Billard we do not aim to imitate for the sake of copying another robot's low-level behavior, but to gather new inspiration for the imitating robot to drive its learning efforts to. This will have to include all levels of abstraction, not only low-level behavior.

Closest to our approach come Inamura et al. [11, 12] with their *Mimesis Loop* approach. Thereby they are able to symbolize observed low-level behavior traces. This is used as top-down teaching from the user's side in combination with the bottom-up learning from the robot's side. As this is useful to decrease the programming effort it is an exclusive solution, not allowing to be used with other learning techniques like e.g. Reinforcement Learning. Also their approach is not able to use already existing abstract states of the imitator in the recognition process. Once a robot has extracted enough information to construct a HMM based on the recognized low-level behaviors it is fixed to that HMM – no exploratory actions on the abstract states are possible any more. Furthermore, the segmentation process that splits the continuous movement trajectories into basic movements uses a fixed scheme. With that it is not possible to allow for ambiguities at the recognition phase.

In our approach we assume that the robot has already decent self-learning capabilities. Imitation is used to guide the robot to the "salient" points in exploration space. With more experience the robot will have collected better skills and a more realistic strategy representation. This in turn will enable it to extract more knowledge from its observation efforts.

3 Imitation Supporting Architecture

The desired outcome of the observation and recognition phase in an imitation process is a state-action-trace that results in a performance similar to the observations. For this the robot has to find abstract states in its own strategy that should play a role when replaying the imitated behavior. Furthermore, it should only regard states that can be connected via actions the imitating robot is capable of. This leads to a tight coupling of the strategy and skill component in the system architecture, accomplishing the recognition of other robots in terms of its own strategy and skill capabilities. Therefore, the strategy and skill layers will be described before. The strategy is modelled with a Semi-Markov Decision Process (SMDP) that has a dynamically adjustable state space (Sec. 3.1) and uses self-developed skills as actions, which are triggered in terms of goal functions on the perception (Sec. 3.2).

3.1 Learning strategies

The strategy layer is inspired by the AMPS approach [13]. It uses a domain-dependent abstraction method to generalize actual state realizations into abstract regions (in our work we use nearest neighbor [7]). The Reinforcement Learning algorithm is then applied onto these regions which simplifies and speeds up the whole process significantly. As the regions can be merged and split at run-time we use Value Iteration [15] to determine the best policy. AMPS, however, applies the splitting and merging also to the action space, which works fine in artificial domains but will not cope with the domain dependency one is typically faced with in real environments. Here, we use as the strategy's actions goal functions which have to be realized by a separate skill learning layer.

In contrast to the pure AMPS method, which by the nature of Reinforcement Learning always learns one strategy to reach one goal, self-adapting systems often have to fulfill several goals – sometimes contradictory ones. Take for example a system that has to fulfill a task while paying attention to its diminishing resources. If it accomplishes the task the resources might get exhausted. On the other hand, if it always stays near the fuel station, the task won't be accomplished. As already described we use abstract drives which the designer has to specify. These drives may also contain competing goals. The big advantage of our approach is that the robot can learn a separate strategy for each drive. Depending on how big the actual motivation is for every drive it has now a means to choose the right strategy for the actual perception and drive state.

3.2 Learning low-level skills

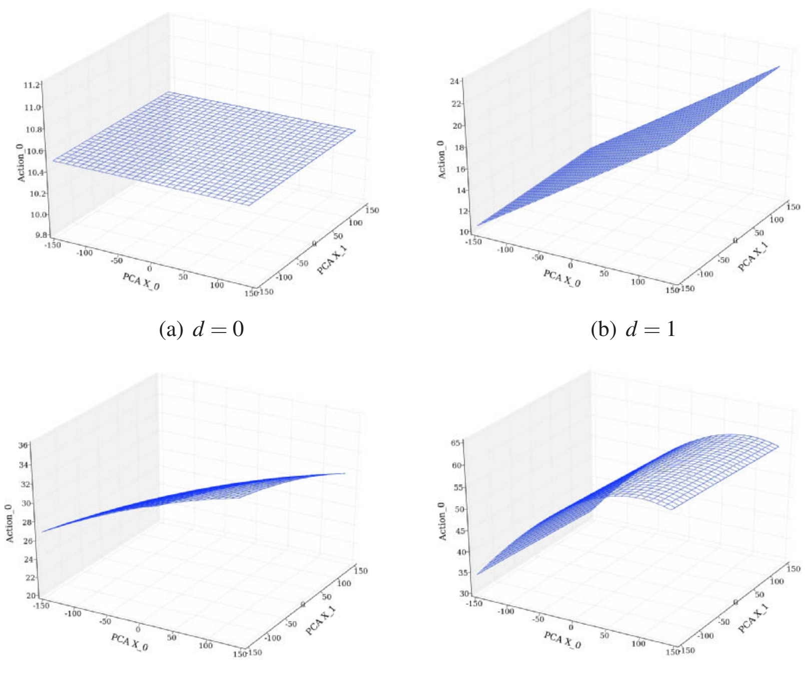
The input of the skill-learning algorithm is given by the strategy layer (Sec. 3.1) in terms of an error function e. Fig. 3 shows a camera image that has been taken from the robot used in the experiments (cf. Sec. 5). The ball that is recognized by a vision algorithm has the properties width and the 2D coordinates of the image. Let d be the euclidean distance of the ball to the image center and Δr the difference between the maximum size of the vision image and the size of the ball in it. The error function that formulates the goal to maximize the ball in the middle of the camera image e.g. would be:

$$e(d,\Delta r) = \sqrt{d^2 + \Delta r^2} \tag{1}$$

The first step is then to get a set of training examples that will later be generalized. During this initial exploration the algorithms gathers information about the relationship between the actuators and the effects. The changing of the actuators is called an *action* A. An *effect* is the perceived result of an action. The actions are generated randomly and are applied for some time. In this phase we call the actuator values the input I and the perceived effect the output O, as they are seen from the skill learning algorithm's perspective. This information are the components of a trace T with the length t: $T = (A, \{(I_0, O_0), \dots, (I_{t-1}, O_{t-1})\})$. Several traces are recorded. Now the error function e is used to extract the good traces forming the training set. To get as many traces as possible, every trace is cut at the position i $(0 \leq i \leq t-1)$ of the lowest error $e(O_i)$. Then every trace not leading to an improvement in terms of e is discarded.

Previous to the generalization, the number of traces and the dimensions have to be refined to reduce the generalization complexity. The most important attributes of a trace are A, I_0, O_0 and O_{t-1}. If the actuator configuration I_0 and the sensor vector O_0, which describe the current situation, are given, A has to be used to reach the effect O_{t-1}. To reduce the number of traces, we do an agglomerative hierarchical clustering. Only the mentioned attributes of a trace are used. The distance measure between two traces is the euclidean distance of the attribute values. The distance between two clusters is defined by average-linkage. The dimensions of I_0 and O_0 depend on the number of actuators and effect properties. Actuators that don't influence the effect can be ignored in the generalization step. Another side-effect of the dimension reduction is the noise reduction, because also the data dimensions with no significant effect to the action-effect can be ignored. We use PCA [1] for this and specify the number of principal components to be kept by the fraction of variance to be explained. In our experiments we were able to reduce the dimensions from six to two while maintaining 95 percent of the data's accuracy.

The last two steps reduce the trace data to the basic properties. In the PCA step a mapping from the data into a new artificial space is done. To generalize the data, a mapping from the principal components. $x_0, \dots, x_n$ to the individual actor elements $a_i \in A$ is calculated. We use a polynomial regression for every a_i. To get the simplest polynomial of $\sum_{i=0}^{d} \prod_{j=0}^{n} p_{ij} {x_j}^i$ that fits the data sufficiently the algorithm starts with $d = 0$ and increments it until the prediction error drops below a predefined error threshold. This process can be seen in Fig. 2. There is also a threshold for the complexity's degree. To avoid over-fitting a maximal possible degree can be specified. Finally, a function $f_i(x_0, \dots, x_n) = a_i$ is calculated for each a_i. When applying the learned skill I_0 and O_0 are known as the current parameter values of the actuators and sensors. A mapping to the PCA space has then to be done before using the calculated f_is to build the next action A. With this approach the robot can reach maximal adaptivity and robustness with regard to sudden breaks or graceful degradation [14].



(a) $d = 0$ (b) $d = 1$

(c) $d = 2$ (d) $d = 3$

Fig. 2 Finding the simplest reasonable hypothesis for the first actor element in PCA space. The graphs show the fitted function for one actor dimension dependent on the two calculated PCA dimensions. The degree of the polynomial is incremented from $d = 0$ up to $d = 3$. In Fig. 2(d) the final function can be seen. The increase of d has been stopped because the fitting error falls below a defined threshold.

4 Sporadic Imitation

With the described means for strategy and skill learning we can now adapt the Viterbi algorithm which is often used to imitate using HMMs. Before we explain the core of our imitation algorithm, we will therefor give a short overview of the Viterbi algorithm following the notation of Bengio [3].

4.1 Viterbi

The Viterbi algorithm [16] tries to find the most likely hidden state sequence s_1^T (*Viterbi path*) that explains the observation sequence o_1^T. This can be done by maximizing the following constraints:

$$s_1^{T*} = \arg\max_{s_1^T} P(s_1^T \mid o_1^T) = \arg\max_{s_1^T} P(s_1^T, o_1^T) \tag{2}$$

Using Bellman's dynamic programming algorithm [2] the Viterbi algorithm determines the maximum efficiently in time $O(Tn)$ where n is the number of non-zero transition probabilities. It recursively calculates the probability

$$V(s,t) = \max_{s_1^{t-1}} P(o_1^t, s_1^{t-1}, s_t = s) \tag{3}$$

that s is the hidden state at time t given the observations o_1^t for all $s \in S$:

$$V(s,t) = P(o_t \mid s_t = s) \max_{s'} P(s_t = s \mid s_{t-1} = s') V(s', t-1) \tag{4}$$

V is initialised with $V(s,1) = \max_{s_1} P(o_1 \mid s_1 = s) P(s_1 = s)\ \forall\, s \in S$. The most likely path can now be extracted using

$$\varphi(s,t) = \arg\max_{s'} P(s_t = s \mid s_{t-1} = s') V(s', t-1)\ , \tag{5}$$

which determines the best predecessor of state s at time t.

4.2 Understanding observed behavior

The imitation approaches usually found in literature calculate the Viterbi path to find the state sequence the imitator should realize in order to exactly copy the observed behavior. This is done using the state space (assumed to be fix) of the inferred HMM, which is assumed to reflect the demonstrator's state space. In contrast to those methods it is important to see that we use a method similar to the calculated Viterbi path to explain the observations recorded from the demonstrator with the imitator's already existing state and action space. Thereby, the imitator tries to *understand* the demonstrator with the knowledge it already has in terms of its own state space (cf. Sec. 3.1) and behavior repertoire (cf. Sec. 3.2).

If the observations provide enough information to infer the corresponding state, $P(o_t \mid s_t)$ could be straightforwardly calculated out of the state representation chosen for the specific domain. If e.g. a nearest neighbor approach is chosen to map state observations to abstract states used in the SMDP, $P(o_t \mid s_t)$ could e.g. chosen to be inversely dependent on the distance to the labeled observation instances in the kNN-representation. However, this is seldom the case in realistic applications so that in order to be able to use Viterbi for inference on the imitator's self-learned knowledge, the robot has to 1) infer the probable state transitions, and 2) guess which of its behaviors could have realized those observed state transition.

The calculation of $P(s_t = s \mid s_{t-1} = s')$ in Eq. 4 is more involved. If one would just take the transition probability of its greedy action in s_{t-1} the robot would not get new insight about other and maybe better state transition behaviors in that specific state. Instead, it should guess from the observations which of the behavior in its own behavior repertoire would best match the recorded observations.

Let us now consider state transition $\langle s_{t_a}, s_{t_b} \rangle$, where $s_{t_a} \neq s_{t_b}$. Firstly, for every recorded observation step $\langle o_{t-1}, o_t \rangle$ ($t \in [t_a, t_b]$) all the behaviors are asked to give a vote $P_b(o_t \,|\, o_{t-1})$ representing the ability of behavior b to be able to realize that step[1]. These are determined by means of the corresponding error function with which the behaviors were learnt. These votes are then divided by the time span of the full state transition:

$$P_b(s_{t_b} \,|\, s_{t_a}) = \frac{\sum_{t=t_a}^{t_b} P_b(o_t \,|\, o_{t-1}, s_{t_a})}{t_b - t_a} \tag{6}$$

At every state transition, one can now determine the most likely transition action $b_{ml} = \arg\max_b P_b(s_{t_b} \,|\, s_{t_a})$. It can be used to retrieve the transition probability in the observer's SMDP that would most probably correspond to the observation of the demonstrator: $P(s_{t_b} \,|\, s_{t_a}) = P(s_{t_b} \,|\, s_{t_a}, b_{ml})$. Thereby, we get the recursive solution

$$V(s,t) = \max_b P_b(o_t \,|\, s_t = s, o_{t-1}) \max_{s'} P(s_t = s \,|\, s_{t-1} = s', a = b_{t-1}) V(s', t-1)\ , \tag{7}$$

in which $P(s_t = s \,|\, s_{t-1} = s', a = b_{t-1}) = T(s', a, s)$ are the transition probabilities learnt in the strategy layer. $\varphi(s,t)$ is determined accordingly. For full reference, the whole algorithm is depicted in Alg. 1. It has the same time complexity as the Viterbi algorithm.

With this information the observing robot can now either remember the $\langle s_i, a, s_{i+1} \rangle$-traces for later replay or spend direct reward along that trace in its strategy layer. If $P(s_{t_b} \,|\, s_{t_a}, a_{greedy})$ is below a predefined threshold (θ in Alg. 1) it is assumed that the robot has no behavior that could probably generate the observed movement from time t_a to t_b, marking where it could most efficiently spend its valuable exploration time. Of course, in this case it is wise not to incorporate the understood sub-sequences of the observed trace, but to wait until behavior for the missing link has been learnt so that the full trace is understood.

5 Evaluation

To evaluate the approach robots were put into an environment with soccer balls that had to be transported onto an elevated platform (Fig. 4). To achieve that they can simply push the ball or use their grippers to pick the ball and release it on the target area. The robots have a defined field of view (fov) of 60°. The field size is $100m^2$. They are able to perceive via their vision capabilities the distance and bearing of the nearest soccer ball if it is within their fov. The platform onto which the ball has to be put is given as absolute coordinates to the robot, which also knows its own position. Overall the robot can perceive the following attributes:

[1] Note the different time scales at the observation and state recordings notations.

Algorithm 1 RECOGNIZE: Recognize familiar behavior and save unrecognizable behavior for later exploration

```
Input: $O_1^T$: observation $\langle (o_1, e_1), \dots, (o_T, e_T) \rangle$ as an (observation, evaluation)-episode stream
    where $e_1 < e_T$; $S$ and $T(s', a, s)$: state space and transition probabilities of the SMDP
Output: Recognized most likely state transitions and missing links
 1: Transform $O_1^T$ into subjective observations $\rightarrow o_1^T$
 2: $\Gamma \leftarrow \emptyset$      // collects understood $\langle s', a, s \rangle$ triples
 3: $\Psi \leftarrow \emptyset$      // collects missing links $\langle s', s \rangle$ that must be explored later on
 4: $V(s, 1) \leftarrow \max_{s_1} P(o_1 \mid s_1 = s) P(s_1 = s) \; \forall \, s \in S$
 5: $t_{last} \leftarrow 1$
 6: $t \leftarrow 2$
 7: while $t < |T|$ do
 8:   for $s \in S$ do
 9:     $b_{t-1} \leftarrow \arg\max_b P_b(o_t \mid s_{t-1} = s, o_{t-1})$
10:     $V(s,t) \leftarrow \max_b P_b(o_t \mid s_{t-1} = s, o_{t-1}) \max_{s'} T(s', b_{t-1}, s) V(s', t-1)$
11:     $\varphi(s,t) \leftarrow \arg\max_{s'} T(s', b_{t-1}, s) V(s', t-1)$
12:     if $\varphi(s,t) \neq \varphi(s, t_{last})$ then
13:       $s'_{last} \leftarrow \varphi(s, t_{last})$
14:       $b_{ml} \leftarrow \arg\max_b P_b(s \mid s_{t_{last}})$
15:       $s_{last} \leftarrow \varphi(s,t)$
16:       $\Gamma \leftarrow \Gamma \cup \langle s'_{last}, b_{ml}, s_{last} \rangle$
17:       $t_{last} \leftarrow t$
18:       break
19:     end if
20:   end for
21:   if $\max_b P_b(o_t \mid s_{t-1}, o_{t-1}) < \theta$ then
22:     while $\max_b P_b(o_t \mid o_{t-1}) < \theta$ and $t < |T| - 1$ do
23:       $t \leftarrow t + 1$
24:     end while
25:     $\Psi \leftarrow \Psi \cup \langle t_{last}, t \rangle$
26:     $V(s,t) \leftarrow \max_{s_t} P(o_t \mid s_t) P(s_t) \; \forall \, s \in S$
27:   end if
28:   $t \leftarrow t + 1$
29: end while
30: return $\Gamma$, $\Psi$
```

- the relative coordinates and width of the ball in the camera image (Fig. 3): (x_b, y_b, w_b)
- distance of the ball from the ground the robot is standing on to detect gripper activities: h_b
- distance and bearing of the target zone: (d_z, θ_z)

The skill layer has as the capabilities to rotate and to translate the robot and to manipulate the gripper. If it is not using its grippers it is nevertheless able to move the balls around in the field by simply pushing them. In the imitation process only the positions of the ball and the robots can be observed. The robot's perception, called *observation* in the algorithm, is thus:

$$o = (x_b, y_b, w_b, h_b, d_z, \theta_z)$$

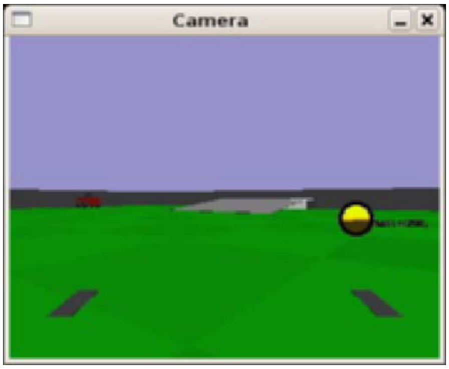


Fig. 3 The perception of the robot that we use in our experiments. It shows one camera image with the ball that has been recognized by our vision algorithm.

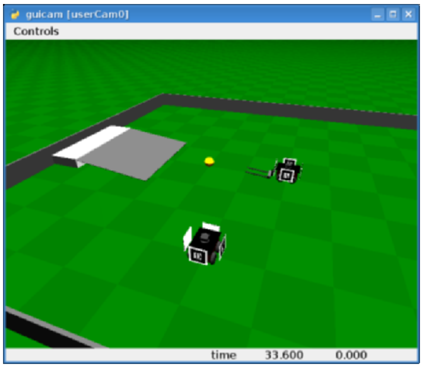


Fig. 4 Experimental scenario: The ball has to be put onto the elevated platform.

Changes in this data are used by the already learned skills in the recognition process to check whether the they could have accomplished those changes.

The experiment goes as follows: Two robots, called demonstrator and imitator in the following, are equipped with appropriate strategies and skills in order to move the ball around: The demonstrator is setup with manually handcrafted code, comprising three skills: approaching ball, lifting the ball, and approaching the goal. The imitator is missing the behavior to lift the ball. It has instead individually learned the skill to approach the ball (cf. Sec. 3.2) and is able to approach the goal, and the corresponding strategy using those skills (cf. Sec. 3.1). In the experiment, the imitator is allowed to observe the demonstrator. The new exploration hints as received from the presented algorithm it has obtained via the observation process are then analyzed. The actual exploration in thereby collected narrowed exploration space is not focused in this paper.

As can be seen in Fig. 5 the imitator has successfully recognized episodes in the demonstrator's movements that coincide with the imitator's own behavior knowledge (dark areas). The *B* denotes the time span in which the demonstrator recognized a behavior resembling its own *approach ball* behavior, and *G* resembling its *approach goal* behavior. It is interesting that the imitator even was able to detect *not understandable* behavior as such (light areas) and bootstrap the recognition process as soon as it has reasonable explanations for the observed behavior data. This *missing link* can now be used in the subsequent exploration processes to direct the exploration towards it, while the understandable regions can be used, e.g., to adapt the strategy towards more aggressively using them.

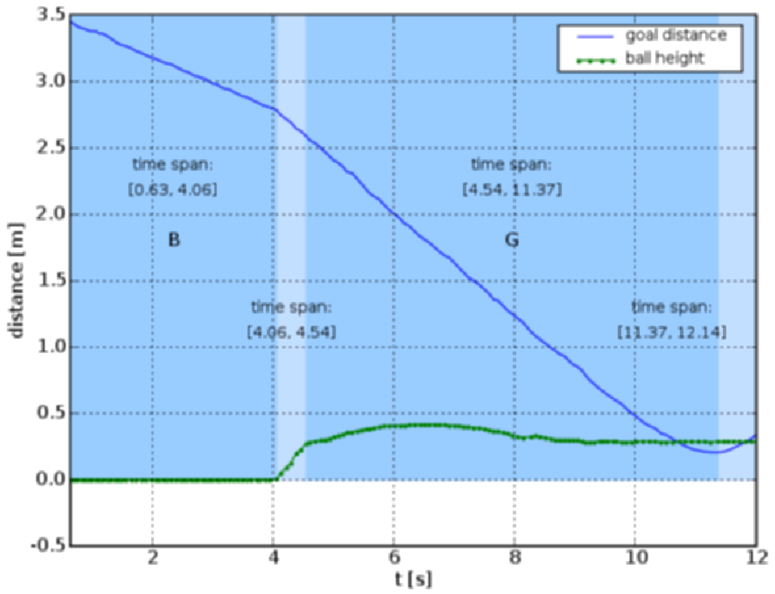


Fig. 5 Recognition results during the imitation process: *B* and *G* (dark area) denote the behavior in time that the demonstrator has understood as equivalent to its *approaching ball* and *approaching goal* behavior. The behavior between them (light area), lifting the ball, is recognized as a missing link.

6 Conclusion

We have shown how sporadic imitation can be accomplished to guide the exploration efforts towards more interesting spaces. For the first time it was shown how inspired by the Viterbi algorithm the maximum likely path of states can be found corresponding to the observation with full reference to the observers own already learned low-level skill capabilities. With it, the observer could reliably explain the demonstrator's performance in terms of its own capabilities if it had skills that could describe the observations or recognize intervals in the observation that could not be understood and should be explored in more detail later on.

Future research should concentrate on more fine-grained dissemination of the *unknown* regions. Using $P_b(s_{t_b} \,|\, s_{t_a})$ (Eq. 6) the robot is not able e.g. to detect that more than one action is necessary to be explored in order to accomplish the state transition $\langle s_{t_a}, s_{t_b} \rangle$. Here it would be helpful to look for consecutive ε-homogeneous action sequences. Such a sequence would then contain actions of the same type with probability $1-\varepsilon$.

References

1. E. Alpaydin. *Introduction To Machine Learning*. MIT Press, 2004.
2. R. Bellman. *Dynamic Programming*. Courier Dover Publications, 2003.
3. Y. Bengio. Markovian models for sequential data. *Neural Computing Surveys*, 2:129–162, 1999.
4. A. Billard, Y. Epars, S. Calinon, G. Cheng, and S. Schaal. Discovering optimal imitation strategies. *Robotics and Autonomous Systems*, 47(2-3):69–77, 2004.
5. Billard, A. Learning motor skills by imitation: a biologically inspired robotic model, 2000.
6. Borenstein, E. and Ruppin, E. Enhancing autonomous agents evolution with learning by imitation. In *Second International Symposium on Imitation in Animals and Artifacts*, 2003.
7. T. M. Cover and P. E. Hart. Nearest neighbor pattern classification. *IEEE Transactions on Information Theory*, 13:21–27, 1967.
8. Demiris, J. and Hayes, G. Imitation as a dual-route process featuring predictive and learning components: a biologically-plausible computational model. In K. Dautenhahn and C. Nehaniv, editors, *Imitation in animals and artifacts*, pages 327–361, Cambridge, MA, USA, 2002. MIT Press.
9. Y. Gatsoulis, G. Maistros, Y. Marom, and G. Hayes. Learning to forage through imitation. In *Proceedings of the Second IASTED International Conference on Artificial Intelligence and Applications (AIA2002)*, pages 485–491, Sept. 2002.
10. A. Ijspeert, J. Nakanishi, and S. Schaal. Movement imitation with nonlinear dynamical systems in humanoid robots. In *International Conference on Robotics and Automation (ICRA2002)*, 2002.
11. T. Inamura, Y. Nakamura, H. Ezaki, and I. Toshima. Imitation and primitive symbol acquisition of humanoids by the integrated mimesis loop. *Robotics and Automation, 2001. Proceedings 2001 ICRA. IEEE International Conference on*, 4, 2001.
12. T. Inamura, I. Toshima, Y. Nakamura, and J. Saitama. Acquiring Motion Elements for Bidirectional Computation of Motion Recognition and Generation. *Experimental Robotics VIII*, 2003.
13. M. J. Kochenderfer. *Adaptive Modelling and Planning for Learning Intelligent Behaviour*. PhD thesis, School of Informatics, University of Edinburgh, 2006.
14. W. Richert, O. Lüke, B. Nordmeyer, and B. Kleinjohann. Increasing the autonomy of mobile robots by on-line learning simultaneously at different levels of abstraction. In *IEEE International Conference on Autonomic and Autonomous Systems (ICAS'08)*, March 2008.
15. R. S. Sutton and A. G. Barto. *Reinforcement Learning: An Introduction.* MIT Press, Cambridge, 1998.
16. A. Viterbi. Error bounds for convolutional codes and an asymptotically optimum decoding algorithm. *Information Theory, IEEE Transactions on*, 13(2):260–269, 1967.