Autonomous Skill Acquisition on a Mobile Manipulator

George Konidaris[1,3] **Scott Kuindersma**[2,3] **Roderic Grupen**[2] **Andrew Barto**[3]
MIT CSAIL[1]
gdk@csail.mit.edu
Laboratory for Perceptual Robotics[2]
University of Massachusetts Amherst
{*scottk,grupen*}*@cs.umass.edu*
Autonomous Learning Laboratory[3]
University of Massachusetts Amherst
barto@cs.umass.edu

Abstract

We describe a robot system that autonomously acquires skills through interaction with its environment. The robot learns to sequence the execution of a set of innate controllers to solve a task, extracts and retains components of that solution as portable skills, and then transfers those skills to reduce the time required to learn to solve a second task.

Introduction

Hierarchical reinforcement learning (Barto and Mahadevan 2003) offers a family of methods for learning and planning using high-level macro-actions (or *skills*) rather than (or in addition to) low-level primitive actions. These approaches are appealing to robotics researchers who hope to design robots that can learn and plan at a high-level while ultimately having to perform control using low-level actuators.

A core research goal in hierarchical RL is the development of skill discovery methods whereby agents can acquire their own high-level skills through interaction with the environment in the context of solving larger problems. Although most skill acquisition research has focused on small discrete problems, some recent work has aimed at making these methods feasible in high-dimensional, continuous domains (Mugan and Kuipers 2009; Konidaris and Barto 2009a; 2009b). In particular, an algorithm called CST (Konidaris et al. 2010) has recently been shown to be capable of acquiring skills from demonstration trajectories on a mobile robot. CST segments trajectories into chains of skills, allocating each its own abstraction (out of a library of available abstractions), and merges chains from multiple trajectories into a skill tree.

We describe a robot system that learns to sequence the execution of a set of innate controllers to solve a task and then uses the resulting solution trajectories as input to CST. The system thereby autonomously acquires new skills through interaction with its environment. We show that the robot is able to reduce the time required to solve a second task by transferring the acquired skills.

Background

Hierarchical RL and the Options Framework

The options framework (Sutton, Precup, and Singh 1999) adds methods for hierarchical planning and learning using temporally-extended actions to the standard RL framework. Rather than restricting the agent to selecting single time-step actions, it models higher-level decision-making using *options*: actions that have their own policies and which may require multiple time steps to complete. An option, o, consists of three components: an *option policy*, π_o, giving the probability of executing each action in each state in which the option is defined; an *initiation set* indicator function, I_o, which is 1 for states where the option can be executed and 0 elsewhere; and a *termination condition*, β_o, giving the probability of option execution terminating in states where the option is defined. Options can be added to an agent's action repertoire alongside its primitive actions, and the agent chooses when to execute them in the same way it chooses when to execute primitive actions. An option can be considered a particular model of the general notion of a skill, and from here we use the terms interchangeably.

We may wish to define an option's policy in a smaller state space than the full task state space. This is known as an *abstraction*; here we define an abstraction M to be a pair of functions (σ_M, τ_M), where $\sigma_M : S \rightarrow S_M$ is a *state abstraction* mapping the overall state space S to a smaller state space S_M, and $\tau_M : A \rightarrow A_M$ is a *motor abstraction* mapping the full action space A to a smaller action space A_M. When using an abstraction, the agent's sensor input is filtered through σ_M and its policy π maps from S_M to A_M.

Skill acquisition thus involves creating an option, defining its termination condition and initiation set, optionally determining the appropriate abstraction, and learning its policy. Creation and termination are typically performed by the identification of a terminating goal. The initiation set is then the set of states from which the goal can be reached. Option abstractions are typically either learned from data (Jonsson and Barto 2001), selected from a library (Konidaris and Barto 2009a), or inferred from the structure of a factored MDP (Mugan and Kuipers 2009; Vigorito and Barto 2010). Finally, given an *option reward function*, policy learning becomes just another RL problem and can be achieved using standard RL methods.

CST

CST (Konidaris et al. 2010) is a recently developed skill discovery method capable of acquiring skills from human-provided demonstration trajectories. It segments each trajectory into a chain of skills—allocating each skill its own abstraction—and merges chains from multiple trajectories into a single skill tree; this is accomplished incrementally and online. CST uses a library of abstractions, and segments each trajectory by automatically detecting when either the most relevant abstraction changes, or when a segment becomes too complex to represent using a single linear value function approximator. This is depicted in Figure 1.

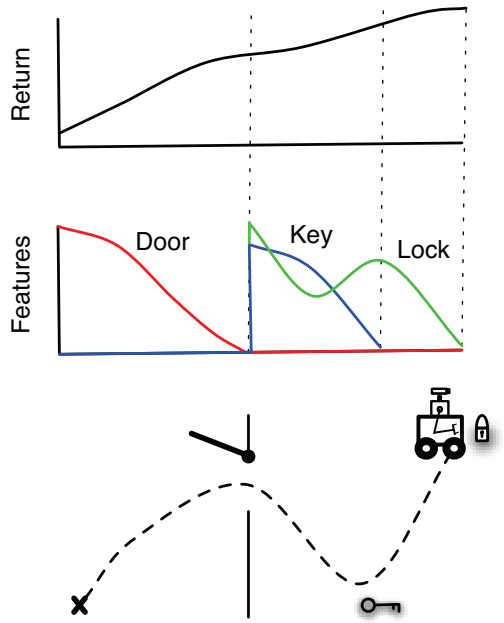

Figure 1: An illustration of a trajectory segmented into skills by CST. A robot executes a trajectory where it goes through a door, approaches and picks up a key, and then takes it to a lock (bottom). The robot is equipped with three possible abstractions: features describing its distance to the doorway, the key, and the lock. The value of these features change during trajectory execution (middle) as the distance to each object changes while it is in the robot's field of view. The robot also obtains an estimate of return for each point along the trajectory by summing the (discounted) rewards obtained from that point on (top). CST splits the trajectory into segments by finding an MAP segmentation such that the return estimate is best represented by a piecewise linear value function where each segment uses a single abstraction. Segment boundaries are shown with dashed vertical lines.

Each option's initiation set is obtained using a classifier: states in its segment are positive examples and all other states are negative examples. Each option's termination condition is the initiation set of the skill that succeeds it (or the target of the trajectory, in the case of the final skill), resulting in a chain of options that can be executed sequentially to take the robot from its starting position to the goal. Given multiple trajectories leading to the same goal, CST merges them by considering the likelihood of each segment being represented by the same value function.

CST is suitable for skill acquisition in mobile robots because it is online, and given an abstraction library it segments demonstration trajectories into sequences of skills that are each represented using a small state space. This use of skill-specific abstractions is a key advantage of hierarchical RL because it allows problems that are high-dimensional when considered monolithically to be adaptively broken into subtasks that may themselves be low-dimensional (Konidaris and Barto 2009a). Additionally, a change in abstraction is a useful measure of subtask boundaries, and the use of agent-centric abstractions facilitates skill transfer (Konidaris and Barto 2007). For more details on CST, see Konidaris et al. (2010).

Autonomous Skill Acquisition on a Mobile Manipulator

Previous work with CST has acquired skills from demonstration trajectories obtained from a human. This section describes a robot that learns to solve a task itself, and thereby generates its own sample trajectories. We used a pair of tasks to demonstrate the feasibility of autonomous robot skill acquisition and the effect of acquired skills. In the first, the robot learned to sequence the execution of a set of innate controllers to solve a mobile manipulation task and then extracted skills from the resulting solution. We compared the performance of the robot with and without the acquired skills in a second, similar, task.[1]

The uBot-5

The uBot-5, shown in Figure 2, is a dynamically balancing, 13 degree of freedom mobile manipulator (Deegan, Thibodeau, and Grupen 2006). Balancing is achieved using an LQR that keeps the robot upright and compensates for forces exerted upon it during navigation and manipulation. The uBot-5 has two arms, each terminated by a small ball that can be used for basic manipulation tasks.[2]

The uBot's control system was implemented primarily in Microsoft Robotics Developer Studio (Johns and Taylor 2008) and allowed differential control of its wheels and position control of each of its hands (though we only used the right hand in this work). The uBot was equipped with two cameras mounted on a pan/tilt unit. Images were extracted and transmitted wirelessly to an off-board computer that processed them using the ARToolkit system (Kato and Billinghurst 1999) to identify augmented reality tags (ARTags) present in the robot's visual field. The uBot was able to identify the location and orientation of visible tags with an update rate of approximately 8Hz. Multiple ARTags were placed in known configurations around important objects, allowing the robot to localize each object even when only a single ARTag was visible.

[1] Videos of the robot solving each task are available at: `http://people.csail.mit.edu/gdk/arsa.html`

[2] A grasping hand prototype is expected to be working shortly.

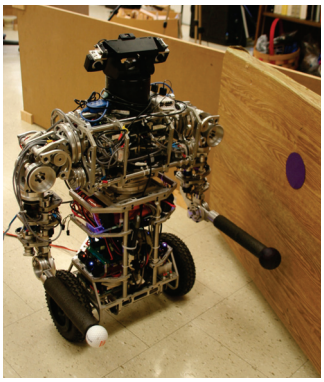

Figure 2: The uBot-5.

The robot had access to innate navigation and manipulation controllers. Given a target object, the navigation controller first aligned the robot with the wall normal at the object's location, then turned the robot to face the object and approach it. This guaranteed that the robot reliably arrived in a good position to manipulate the target object. The uBot also had controllers that moved its hand to one of seven positions: withdrawn, extended, and then extended and moved to the left, right, upwards, downwards, or outwards. Each of these controlled the position of the hand relative to the centroid of the target object.

Before starting a navigation, the robot performed a visual scan of the room to determine the locations of visible objects. It then executed an orientation controller which turned it away from the location it was facing and toward its target. This controller was crucial in safely avoiding walls and was therefore not learned or included in segmentation. Throughout the experiment, the robot actively tracked the ARTags surrounding target objects using its pan/tilt head.

The uBot was given a library of abstractions. Each abstraction paired one of the uBot's motor modalities (body or hand) with a task object. Abstractions pertaining to the robot's hand contained state variables expressing the difference between its position and centroid of an object. Abstractions using the robot's body and a target object contained state variables expressing the distance from the body to the object, the distance from the body to the nearest point on the wall upon which the object was attached, and the angle between the body and the wall normal vector.

To solve each task, the uBot learned a discrete model of the task as an MDP. This model allowed the uBot to plan online using dynamic programming with learning rate $\alpha = 0.1$. The value function was initialized optimistically to zero for unknown state-action pairs, except for when the robot used acquired skills, when the state-action pairs corresponding to basic manipulation actions were initialized to a time cost of three hours. This resulted in a robot that always preferred to use higher-level acquired skills when possible, but could also make use of lower-level innate controllers when all other options had been exhausted. The robot received a reward of -1 for each second that passed.

A new closed-loop controller was synthesized for each acquired skill by fitting a spline to the solution trajectory to identify a sequence of relative waypoints. This allowed robust replay while retaining the ability to learn to improve each controller using a policy search algorithm if necessary.

The Red Room Tasks

The Red Room Domain consisted of two tasks. We used the first task as a training task: the uBot learned to solve it by sequencing its innate controllers and then extracted skills from the resulting solution trajectories. We then compared the time the uBot took to initially solve the second task using its innate controllers against the time required when using acquired skills.

The First Task The first task consisted of a small room containing a button and a handle. When the handle was pulled after the button had been pressed a door in the side of the room opened, allowing the uBot access to a compartment containing a switch. The goal of the task was to press the switch. Sensing and control for the objects in the room was performed using touch sensors, with state tracked and communicated to the uBot via an MIT Handy Board (Martin 1998). Figure 3 shows a schematic drawing and photographs of the first task.

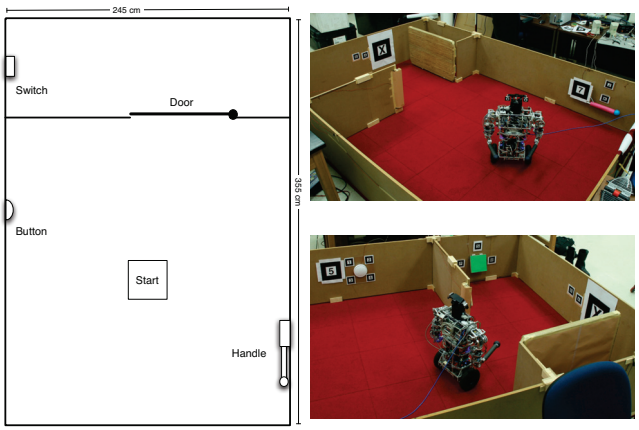

Figure 3: The first task in the Red Room Domain.

The state of the first task at time i was described as a tuple $s_i = (r_i, p_i, h_i)$, where r_i was the state of the room, p_i was the position of the robot, and h_i was the position of its hand. The state of the room at time i consists of four state bits, indicating the state of the button (pressing the button flipped this bit), the state of the handle (this bit was only flipped once per episode, and only when the button bit was set), whether or not the door was open, and whether or not the switch had been pressed (this bit was also only flipped once per episode since the episode ended when it was set). The uBot could find itself at one of five positions: its start position, in front of the button, in front of the handle, through the door, and in front of the switch. Each of these positions was marked in the room using ARTags—a combination of small and large tags were used to ensure that each position was visible in the robot's cameras from all of the relevant

locations in the room. The robot had a navigate action available to move it to any position visible when it performed a visual sweep with its head. Thus, the robot could always move between the button and the switch, but could only move through the door entrance once the door was open; only then could it see the switch and move towards it. Finally, the robot's hand could be in one of seven positions: withdrawn (allowing it to execute a navigation action), extended, and then extended and moved to the left, right, upwards, downwards, or outwards. The robot had to be facing an object to interact with it. In order to actuate the button and the switch the robot had to extend its hand and then move it outwards; in order to actuate the handle it had to extend its hand and then move it downwards.

The Second Task The second Red Room task was similar to the first: the robot was placed in a room with a group of manipulable objects and a door. In this case, the robot had to first push the switch, and then push the button to open the door. Opening the door hid a button in the second part of the room. The robot had to navigate to the second part of the room and pull a lever to close the door again. This revealed the second button, which it had to press to complete the task.

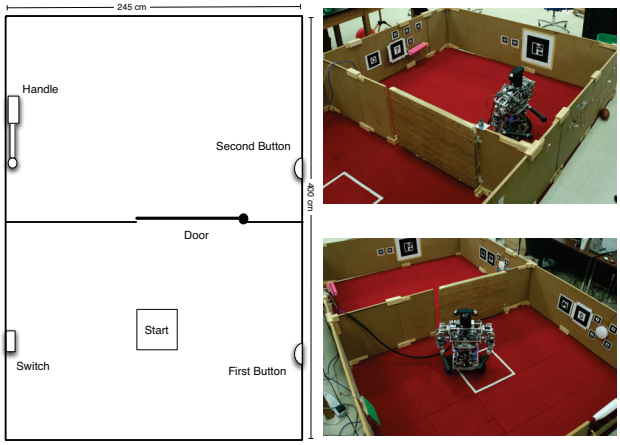

Figure 4: The second task in the Red Room Domain.

The state of the second task at time i similarly consisted of five state bits, indicating the state of the switch, the first button, the handle, the second button and the door. Here, the uBot may find itself in one of six locations: its start position, in front of the switch, in front of the first button, through the door, in front of the handle, and in front of the second button. Note that this room contained the same object types as the first task, and so the robot was able to apply its acquired skills to manipulate them. In general object classification would require visual pattern matching, but for simplicity we provided object labels.

Results

The uBot's first task was to sequence its innate controllers to solve the first Red Room. Since the robot started with no knowledge of the underlying MDP, it had to learn both how to interact with each object and in which order interaction should take place. Figure 5 shows the uBot's learning curve. The robot was able to find the optimal controller sequence after 5 episodes, reducing the time taken to solve the task from approximately 13 minutes to around 3. This did not include the time required by hardwired controllers not subject to learning (e.g., the controller that safely oriented the uBot from one object to another).

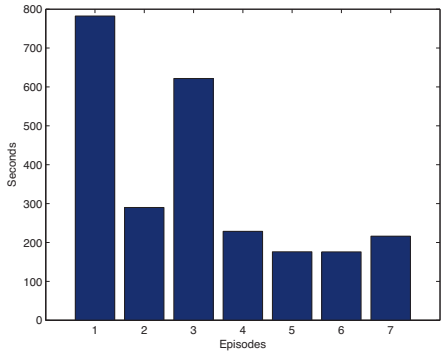

Figure 5: The uBot's learning curve in the first task.

The resulting optimal sequence of controllers were then used to generate 5 demonstration trajectories for use in CST (using a 1st order Fourier basis, and CST parameters expected skill length $k = 150$, min and max changepoint particles $M = 60$ and $N = 120$, and expected noise variance parameters $\sigma_v = 60^2$ and $\beta_v = 10^{-6}$). The algorithm segmented those trajectories into the same sequence of 10 skills, and merged all chains into a single chain (using a 5th order Fourier Basis, $\sigma_v = 500^2$). An example segmentation is shown in Figure 6; a description of each skill along with its relevant abstraction is given in Figure 7.

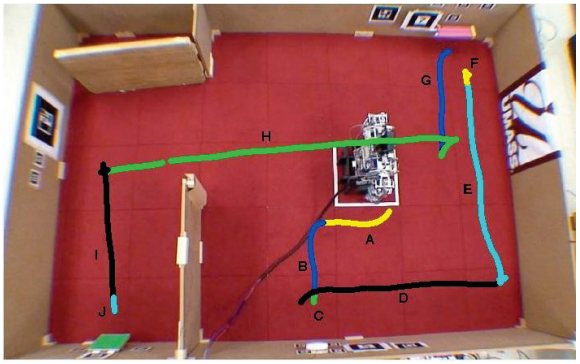

Figure 6: A trajectory from the learned solution to the first task, segmented into skills.

CST extracted skills that corresponded to manipulating objects in the environment, and navigating towards them. In the navigation case, each controller execution was split into two separate skills. These skills corresponded exactly to the two phases of the navigation controller: first, aligning the robot with the normal of a feature, and second, moving

#	Abstraction	Description
A	body-button	Align with the button.
B	body-button	Turn and approach the button.
C	hand-button	Push the button.
D	body-handle	Align with the handle.
E	body-handle	Turn and approach the handle.
F	hand-handle	Pull the handle.
G	body-entrance	Align with the entrance.
H	body-entrance	Turn and drive through the entrance.
I	body-switch	Approach the switch.
J	hand-switch	Press the switch.

Figure 7: A brief description of each of the skills extracted from the trajectory in Figure 6, with selected abstractions.

the robot toward that feature. We do not consider the resulting navigation skills further since they are room-specific and cannot be used in the second task. In the object-manipulation case, sequences of two controllers were collapsed into a single skill: for example, extending the hand and then reducing the distance between its hand and the button to zero was collapsed into a single skill which we might label *push the button*. We fitted the resulting policies for replay using a single demonstrated trajectory, and obtained reliable replay for all manipulation skills.

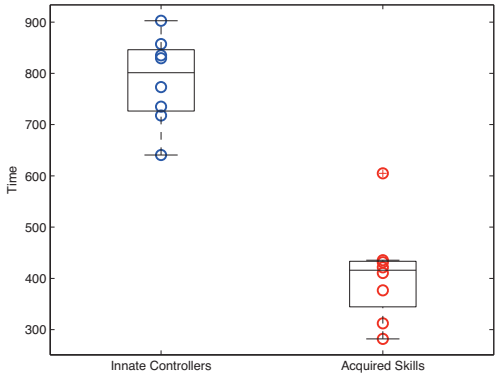

Figure 8: The time required for the uBot-5 to first complete the second task, given innate controllers or acquired skills.

Figure 8 shows the time required for the uBot's first episode in the second task, given either its original innate controllers or, additionally, the manipulation skills acquired in the first Red Room task (again, this does not include controllers not subject to learning). We performed 8 runs of each condition. The presence of acquired skills nearly halved the mean time to completion (from 786.39 seconds to 409.32 seconds), and this difference is significant (Welch two-sample t-test, $t(13.785) = 8.22, p < 0.001$); moreover, the sampled times for the two conditions do not overlap.[3]

[3]Note that one of the runs using skill acquisition is marked as an outlier (with a cross) in Figure 8. During this run, the robot explored virtually all transitions available in the MDP before finally finding the solution. This corresponds to near worst-case behavior using acquired skills; it still requires less time (by about 30 seconds) than the fastest observed run using only innate controllers.

Related and Future Work

The use of hierarchical learning and control in robots predates such work in the reinforcement learning community (Maes and Brooks 1990), and includes research on approaches to building hierarchies via scheduled learning of individual skills (Huber, MacDonald, and Grupen 1996; Huber 2000). However, existing research where the agents autonomously identify skills to acquire in the process of solving a larger task has primarily been performed in small discrete domains. Some research within the field of learning from demonstration focuses on extracting skills from demonstration trajectories —for example, Jenkins and Matarić (2004) segment demonstrated data into *motion primitives* and thereby build a motion primitive library. For a thorough review of the field see Argall et al. (2009).

A promising recent direction involves acquiring skills outside of the context of any particular task, by developing agents that are *intrinsically motivated* to acquire skills that may become useful in the future, as opposed to extrinsically motivated by a particular task (Singh, Barto, and Chentanez 2005; Oudeyer and Kaplan 2007; Hart and Grupen 2011). This area has focused mainly on the development of heuristics that are able to identify potentially useful skills in advance, and the design of artificial reward functions that encourage the agent to explore efficiently (Singh, Barto, and Chentanez 2005) or to choose to practice new skills (Stout and Barto 2010). Though most of the skill acquisition methods in this area have only been demonstrated on small artificial domains, some research has focused on skill acquisition in robots. Soni and Singh (2006) focused on learning and using macro-actions with pre-designated salient events as subgoals on a Sony Aibo robot. More recently, Hart and Grupen (2011) described a humanoid robot progressing through a developmental schedule whereby it learned to sequence and retain a set of controllers to achieve goals designated by an intrinsic reward mechanism.

An important assumption made here is the availability of a pre-existing abstraction library. Future work may develop techniques for feasibly building skill-specific abstractions from scratch in real-time; alternatively, it may aim to build an abstraction library from sensorimotor data over a much longer timescale than that afforded to a single skill.

It is widely recognized that too many skills may make a problem harder rather than easier. Therefore, another important direction for further research is *skill management*, which could develop methods for building compact libraries of acquired skills, or for using the contexts in which a skill has been previously used successfully to restrict the contexts in which it is likely to be considered in the future.

Finally, future research may address methods for using acquired skill hierarchies to facilitate planning. The use of high-level skills is particularly promising here because it may avoid the need for a low-level model of the robot.

Summary and Conclusions

We have described a mobile robot system that can acquire skills by learning to solve one problem, and then apply the resulting skills to improve performance in another. It is worth considering the implications of these results. Although the uBot started off with the capacity to *learn* to, for example, push the button, this was accomplished through a laborious process of trial-and-error exploration through many combinations of manipulation actions within a particular task. However, since this sequence of manipulation actions happened to be useful in *solving a problem*, it was extracted as a single action that can be deployed as a unit—requiring only a single action selection decision—when the robot encounters a new problem. Had the uBot attempted transfer its *entire* policy from the first task to the second, it would have performed very poorly. Instead, transfer was affected via the isolation and retention of skills—effectively *policy components*—that are suitable for reuse in later tasks.

While some elements of intelligence in robots can be directly designed, other aspects involve knowledge that can only be gained by the agent itself or that may change in unpredictable ways over its lifetime. Although the results presented in this paper are necessarily limited, they represent a step toward creating robot systems that display a hallmark of human intelligence: the ability to use their own experience to autonomously and incrementally build knowledge structures that improve their problem-solving abilities over time.

Acknowledgments

We would like to thank the members of the LPR for their technical assistance. AGB and GDK were supported in part by the AFOSR under grant FA9550-08-1-0418. GDK was also supported in part by the AFOSR under grant AOARD-104135 and the Singapore Ministry of Education under a grant to the Singapore-MIT International Design Center. SRK is supported by a NASA GSRP fellowship from Johnson Space Center. RAG was supported by the Office of Naval Research under MURI award N00014-07-1-0749.

References

Argall, B.; Chernova, S.; Veloso, M.; and Browning, B. 2009. A survey of robot learning from demonstration. *Robotics and Autonomous Systems* 57:469–483.

Barto, A., and Mahadevan, S. 2003. Recent advances in hierarchical reinforcement learning. *Discrete Event Dynamic Systems* 13:41–77.

Deegan, P.; Thibodeau, B.; and Grupen, R. 2006. Designing a self-stabilizing robot for dynamic mobile manipulation. In *Proceedings of the Robotics: Science and Systems Workshop on Manipulation for Human Environments*.

Hart, S., and Grupen, R. 2011. Learning generalizable control programs. *IEEE Transactions on Autonomous Mental Development* 3(1). In press.

Huber, M.; MacDonald, W.; and Grupen, R. 1996. A control basis for multilegged walking. In *Proceedings of the 1996 IEEE Conference on Robotics and Automation*, 2988–2993.

Huber, M. 2000. *A Hybrid Architecture for Adaptive Robot Control*. Ph.D. Dissertation, Department of Computer Science, University of Massachusetts Amherst.

Jenkins, O., and Matarić, M. 2004. Performance-derived behavior vocabularies: data-driven acquisition of skills from motion. *International Journal of Humanoid Robotics* 1(2):237–288.

Johns, K., and Taylor, T. 2008. *Professional Microsoft Robotics Developer Studio*. Hoboken, New Jersey: Wrox Press.

Jonsson, A., and Barto, A. 2001. Automated state abstraction for options using the U-Tree algorithm. In *Advances in Neural Information Processing Systems 13*, 1054–1060.

Kato, H., and Billinghurst, M. 1999. Marker tracking and HMD calibration for a video-based augmented reality conferencing system. In *Proceedings of the 2nd IEEE and ACM International Workshop on Augmented Reality*.

Konidaris, G., and Barto, A. 2007. Building portable options: Skill transfer in reinforcement learning. In Veloso, M., ed., *Proceedings of the Twentieth International Joint Conference on Artificial Intelligence*, 895–900.

Konidaris, G., and Barto, A. 2009a. Efficient skill learning using abstraction selection. In Boutilier, C., ed., *Proceedings of the Twenty First International Joint Conference on Artificial Intelligence*, 1107–1112.

Konidaris, G., and Barto, A. 2009b. Skill discovery in continuous reinforcement learning domains using skill chaining. In Bengio, Y.; Schuurmans, D.; Lafferty, J.; Williams, C.; and Culotta, A., eds., *Advances in Neural Information Processing Systems 22*, 1015–1023.

Konidaris, G.; Kuindersma, S.; Barto, A.; and Grupen, R. 2010. Constructing skill trees for reinforcement learning agents from demonstration trajectories. In Lafferty, J.; Williams, C.; Shawe-Taylor, J.; Zemel, R.; and Culotta, A., eds., *Advances in Neural Information Processing Systems 23*, 1162–1170.

Maes, P., and Brooks, R. 1990. Learning to coordinate behaviors. In *Proceedings of the American Association of Artificial Intelligence*, 796–802.

Martin, F. 1998. *The Handy Board Technical Reference*. Cambridge MA: MIT Media Lab.

Mugan, J., and Kuipers, B. 2009. Autonomously learning an action hierarchy using a learned qualitative state representation. In *Proceedings of the Twenty First International Joint Conference on Artificial Intelligence*.

Oudeyer, P.-Y., and Kaplan, F. 2007. What is intrinsic motivation? a typology of computational approaches. *Frontiers in Neurobotics* 1(6):1–14.

Singh, S.; Barto, A.; and Chentanez, N. 2005. Intrinsically motivated reinforcement learning. In Saul, L.; Weiss, Y.; and Bottou, L., eds., *Advances in Neural Information Processing Systems 17*, 1281–1288.

Soni, V., and Singh, S. 2006. Reinforcement learning of hierarchical skills on the Sony Aibo robot. In *Proceedings of the Fifth International Conference on Development and Learning*.

Stout, A., and Barto, A. 2010. Competence progress intrinsic motivation. In *Proceedings of the Ninth IEEE International Conference on Development and Learning*, 257–262.

Sutton, R.; Precup, D.; and Singh, S. 1999. Between MDPs and semi-MDPs: A framework for temporal abstraction in reinforcement learning. *Artificial Intelligence* 112(1-2):181–211.

Vigorito, C., and Barto, A. 2010. Intrinsically motivated hierarchical skill learning in structured environments. *IEEE Transactions on Autonomous Mental Development* 2(2).