Autonomous Learning of High-Level States and Actions in Continuous Environments

Jonathan Mugan and Benjamin Kuipers, *Fellow, IEEE*

***Abstract*—**

How can an agent bootstrap up from a pixel-level representation to autonomously learn high-level states and actions using only domain-general knowledge? In this paper we assume that the learning agent has a set of continuous variables describing the environment. There exist methods for learning models of the environment, and there also exist methods for planning. However, for autonomous learning, these methods have been used almost exclusively in discrete environments.

We propose attacking the problem of learning high-level states and actions in continuous environments by using a qualitative representation to bridge the gap between continuous and discrete variable representations. In this approach, the agent begins with a broad discretization and initially can only tell if the value of each variable is increasing, decreasing, or remaining steady. The agent then simultaneously learns a qualitative representation (discretization) and a set of predictive models of the environment. These models are converted into plans to perform actions. The agent then uses those learned actions to explore the environment.

The method is evaluated using a simulated robot with realistic physics. The robot is sitting at a table that contains a block and other distractor objects that are out of reach. The agent autonomously explores the environment without being given a task. After learning, the agent is given various tasks to determine if it learned the necessary states and actions to complete them. The results show that the agent was able to use this method to autonomously learn to perform the tasks.

***Index Terms*—unsupervised learning, reinforcement learning, qualitative reasoning, intrinsic motivation, active learning.**

I. Introduction

WE would like to build intelligent agents that can autonomously learn to predict and control the environment using only domain-general knowledge. Such agents could simply be placed in an environment, and they would learn it. After they had learned the environment, the agents could be directed to achieve specified goals. The intelligence of the agents would free engineers from having to design new agents for each environment. These agents would be flexible and robust because they would be able to adapt to unanticipated aspects of the environment.

Designing such agents is a difficult problem because the environment can be almost infinitely complex. This complexity means that an agent with limited resources cannot represent and reason about the environment without describing it in a simpler form. And possibly more importantly, the complexity of the environment means that it is a challenge to generalize from experience since each experience will be in some respect different. A solution to the difficulty of learning in a complex environment is for the agent to autonomously learn useful and appropriate abstractions.

There are approaches that do impressive learning in specific scenarios [1]–[3]. But most autonomous learning methods require a discrete representation and a given set of discrete actions [4]–[8]. Our goal is to enable autonomous learning to be done in a continuous environment and to enable an agent to learn its first actions.

The Problem: The world is continuous and infinitely complex (or effectively so). The dynamics of the world are also continuous, and add further complexity. An agent acts in that world by sending continuous low-level motor signals. Our goal is to show how such an agent can learn a hierarchy of actions for acting effectively in such a world. We simplify the problem of perception for our agent. Instead of dealing with continuous high-bandwidth pixel-level perception, we assume that the agent has trackers for a small number of moving objects (including its own body parts) within an otherwise static environment. These trackers provide the agent wih a perceptual stream consisting a relatively small number of continuous variables. Learning such trackers and the models that provide dynamically updated model parameters is done by methods outside the scope of this project, such as the Object Semantic Hierarchy [9]. Our solution to the problem of learning a hierarchy of actions in continuous environments is the Qualitative Learner of Action and Perception, QLAP.

A. The Qualitative Learner of Action and Perception, QLAP

QLAP has two major processes:

(1) Modeling Events: An *event* is a qualitative change in the value of some state variable (Section II). QLAP starts by identifying *contingencies* between events: situations where the observation of one event E_1 means that another event E_2 is more likely to occur soon after. QLAP searches for improved descriptions of the conditions for contingencies, trying to find sufficiently deterministic simple models that predict the results of actions (Section III).

This is done by introducing new distinctions into the qualitative abstractions of the domains of particular variables, and by identifying additional dependencies on context variables, making predictions more reliable. These improved models of the forward dynamics of the environment are represented as dynamic Bayesian networks (DBNs). This process is depicted in Figure 1.

(2) Modeling Actions: Define an *action* to be the occurrence of a particular event: a particular qualitative change to some

J. Mugan is with the School of Computer Science, Carnegie Mellon University, Pittsburgh, PA 15213 USA (e-mail: jmugan@cs.cmu.edu).

B. Kuipers is with the Department of Electrical Engineering and Computer Science, University of Michigan, Ann Arbor, MI 48109 USA (e-mail: kuipers@umich.edu).

variable. A reliable DBN model that leads to that event can be transformed (by familiar RL methods) into a *plan* for a accomplishing that consequent event by means of achieving antecedent events, and thus for carrying out that action. Since the plan thus consists of embedded actions, we get a natural hierarchical structure on actions, via the plans available to carry them out (Section IV).

A hierarchy of actions and plans must ground out in motor actions: qualitative changes to the values of motor variables that the agent can carry out simply by willing them. QLAP ensures that its higher-level actions and plans have this grounding in motor actions because it learns everything autonomously, starting from random exploration of the consequences of setting its motor variables (i.e. "motor babbling"). The process of modeling actions is depicted in Figure 2.

To perform exploration and learning, the two processes of Modeling Events and Modeling Actions run continuously as the agent acts in the world (Section V). Those actions can be driven initially by random motor babbling. However, they can be driven by autonomous exploration through various intrinsic motivation drives. They can also be driven by explicit coaching, or by "play" — posing various goals and making plans to achieve them. (See [10] for a video of QLAP.[1])

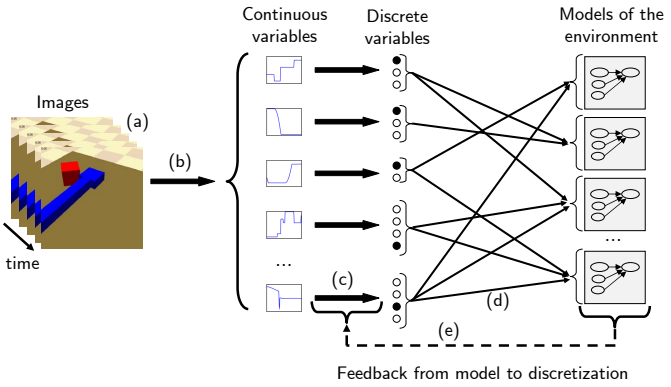


Fig. 1: Perception in QLAP.

B. Contributions

To the best of our knowledge, QLAP is the only algorithm that learns states and hierarchical actions through autonomous exploration in continuous, dynamic environments with continuous motor commands. For the field of Autonomous Mental Development, QLAP provides a method for a developing agent to learn its first temporally-extended actions and to learn more complex actions on top of previously-learned actions.

The related field of reinforcement learning is about enabling an agent to learn from experience to maximize a reward signal [11]. QLAP addresses three challenges in reinforcement learning: (1) continuous states and actions, (2) automatic hierarchy construction, and (3) automatic generation of reinforcement learning problems. Continuous states and actions are a challenge because it is hard to know how to generalize from experience since no two states are exactly

[1]http://videolectures.net/aaai2010_mugan_qlap

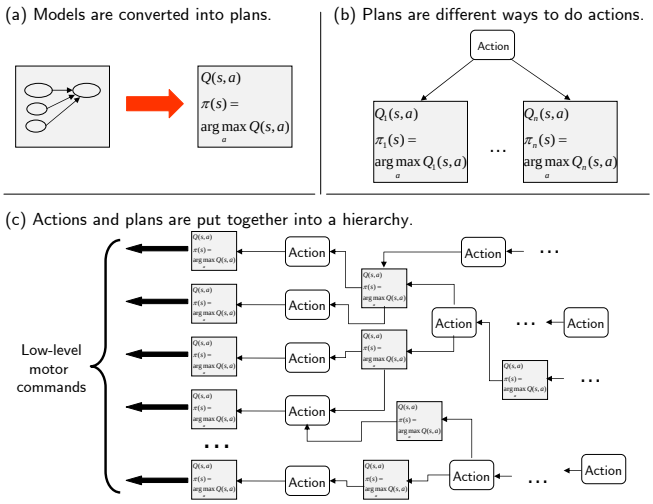


Fig. 2: Actions in QLAP.

alike. There exist many function approximation methods, and other methods that use real values [3], [12]–[16], but QLAP provides a method for discretizing the state and action space so that the discretization corresponds to the "natural joints" in the environment. Finding these natural joints allows QLAP to use simple learning algorithms while still representing the complexity of the environment.

Learning of hierarchies can enable an agent to explore the space more effectively because it can aggregate smaller actions into larger ones. QLAP creates a hierarchical set of actions from continuous motor variables. Currently, most reinforcement learning problems must be designed by the human experimenter. QLAP autonomously creates reinforcement learning problems as part of its developmental progression.

Section II discusses the qualitative representation used to discretize the continuous data. Section III explains how QLAP learns models of events, and Section IV describes how models of events are converted into actions. Section V discusses how the QLAP agent explores and learns about the world. Experimental results are presented in Section VI, and the paper concludes with a discussion (Section VII), an overview of related work (Section VIII), and a summary (Section IX).

II. Qualitative Representation

A qualitative representation allows an agent to bridge the gap between continuous and discrete values. It does this by encoding the values of continuous variables relative to known landmarks [17]. The value of a continuous variable can be described qualitatively. Its qualitative magnitude is described as equal to a landmark value or as in the open interval between two adjacent landmark values. Its qualitative direction of change can be increasing, steady, or decreasing. Because a landmark is intended to represent an important value of a variable where the system behavior may change qualitatively, a qualitative representation allows the agent to generalize and to focus on important events.

A. Landmarks

A *landmark* is a symbolic name for an point on a number line. Using landmarks, QLAP can convert a continuous variable $\tilde{v}$ with an infinite number of values into a qualitative variable v with a finite set of qualitative values $\mathcal{Q}(v)$ called a *quantity space* [17]. A quantity space $\mathcal{Q}(v) = L(v) \cup I(v)$, where $L(v) = \{v_1^*, \cdots, v_n^*\}$ is a totally ordered set of landmark values, and $I(v) = \{(-\infty, v_1^*), (v_1^*, v_2^*), \cdots, (v_n^*, +\infty)\}$ is the set of mutually disjoint open intervals that $L(v)$ defines in the real number line. A quantity space with two landmarks might be described by (v_1^*, v_2^*), which implies five distinct qualitative values, $\mathcal{Q}(v) = \{(-\infty, v_1^*), v_1^*, (v_1^*, v_2^*), v_2^*, (v_2^*, +\infty)\}$. This is shown in Figure 3.

Fig. 3: Landmarks divide the number line into a discrete set of qualitative values.

QLAP perceives the world through a set of continuous input variables and affects the world through a set of continuous motor variables.[2] For each continuous input variable $\tilde{v}$, two qualitative variables are created: a discrete variable $v(t)$ that represents the qualitative magnitude of $\tilde{v}(t)$, and a discrete variable $\dot{v}(t)$ that represents the qualitative direction of change of $\tilde{v}(t)$. Also, a qualitative variable $u(t)$ is created for each continuous motor variable $\tilde{u}$.[3] The result of these transformations is three types of qualitative variables that the agent can use to affect and reason about the world: *motor variables*, *magnitude variables*, and *direction of change variables*. The properties of these variables are shown in Table I.

TABLE I: Types of Qualitative Variables

Type of Variable	Initial Landmarks	Learn Landmarks?
motor	{0}	yes
magnitude	{}	yes
direction of change	{0}	no

Each direction of change variable $\dot{v}$ has a single intrinsic landmark at 0, so its quantity space is $\mathcal{Q}(\dot{v}) = \{(-\infty, 0), 0, (0, +\infty)\}$, which can be abbreviated as $\mathcal{Q}(\dot{v}) = \{[-], [0], [+]\}$. Motor variables are also given an initial landmark at 0. Magnitude variables initially have no landmarks, treating zero as just another point on the number line. Initially, when the agent knows of no meaningful qualitative distinctions among values for $\tilde{v}(t)$, we describe the quantity space with the empty list of landmarks, $\{\}$, as $\mathcal{Q}(\dot{v}) = \{(-\infty, +\infty)\}$. However, the agent can learn new landmarks for magnitude and motor variables. Each additional landmark allows the agent to perceive or affect the world at a finer granularity.

[2]QLAP can handle discrete (nominal) input variables. See [18] for details.

[3]When the distinction between motor variables and non-motor variables is unimportant, we will refer to the variable as v.

B. Events

If a is a qualitative value of a qualitative variable A, meaning $a \in \mathcal{Q}(A)$, then the *event* $A_t \rightarrow a$ is defined by $A(t-1) \neq a$ and $A(t) = a$. That is, an event takes place when a discrete variable A changes to value a at time t, from some other value. We will often drop the t and describe this simply as $A{\rightarrow}a$. We will also refer to an event as E when the variable and value involved are not important, and we use the notation $E(t)$ to indicate that event E occurs at time t.

For magnitude variables, $A_t \rightarrow a$ is really two possible events, depending on the direction that the value is coming from. If at time $t-1$, $A(t) < a$, then we describe this event as $\uparrow A_t {\rightarrow} a$. Likewise, if at time $t-1$, $A(t) > a$, then we describe this event as $\downarrow A_t {\rightarrow} a$. However, for ease of notation, we generally refer to the event as $A_t {\rightarrow} a$. We also say that event $A_t {\rightarrow} a$ is *satisfied* if $A_t = a$.

III. Modeling Events

There are many methods for learning predictive models in continuous environments. Such models have been learned, for example, using regression [3], [12]–[14], neural networks [15], and Gaussian processes [16]. But as described in the introduction, we want to break up the environment and represent it using a qualitative representation.

In a discretized environment, dynamic Bayesian networks (DBNs) are a convenient way to encode predictive models. Most work on learning DBNs learn a network to predict each variable at the next timestep for each action, e.g. [19]–[21]. However, QLAP learns the set of actions, and QLAP works in environments where events may take more than one timestep.

QLAP learns two different types of DBN models. The first type of DBN models are those that predict events on change variables (change DBNs). The second type of DBN models are those for reaching magnitude values (magnitude DBNs); Section III-E. To learn change DBNs, QLAP uses a novel DBN learning algorithm. Given the current discretization, QLAP tracks statistics on all pairs of events to search for contingencies (Section III-A) where an antecedent event leads to a consequent event. When such a contingency is found, QLAP converts it to a DBN with the antecedent event as the parent variable and the consequent event as the child variable. QLAP adds context variables to the DBN one at a time as they make the DBN more reliable. For each DBN, QLAP also searches for a new discretization that will make the DBN more reliable. This new discretization then creates new possible events and allows new DBNs to be learned. This method is outlined in Figure 4.

A. Searching for Contingencies

The search for change DBNs begins with a search for contingencies. A contingency represents the knowledge that if the antecedent event occurs, then the consequent event will soon occur. An example would be if you flip a light switch, then the light will go off. QLAP searches for contingencies by tracking statistics on pairs of events E_1, E_2 and extracting those pairs into a contingency where the occurrence of event E_1 indicates that event E_2 is more likely to soon occur than it would otherwise.

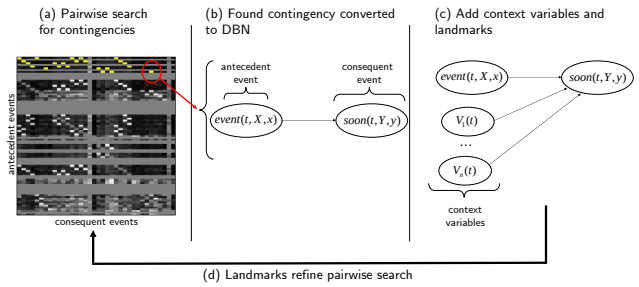


Fig. 4: (a) Do a pairwise search for contingencies that use one event to predict another. The antecedent events are along the y-axis, and the consequent events are along the x-axis, The color indicates the probability that the consequent event will soon follow the antecedent event (lighter corresponds to higher probability). When the probability of the consequent event is sufficiently high, it is converted into a contingency (yellow). (b) When a contingency is found, it is used to create a DBN. (c) Once a DBN is created, context variables are added to make it more reliable. (d) The DBN creates a self-supervised learning problem to predict when the consequent event will follow the antecedent event. This allows new landmarks to be found. Those landmarks create new events for the pairwise search. (Best viewed in color.)

1) Definition: To define contingencies in a continuous environment, we have to discretize both variable values and time. To discretize variable values, we create a special Boolean variable $event(t, X{\to}x)$ that is true if event $X_t{\to}x$ occurs

$$event(t, X{\to}x) \equiv X_t{\to}x \tag{1}$$

To discretize time, we use a time window. We define the Boolean variable $soon(t, Y{\to}y)$ that is true if event $Y_t{\to}y$ occurs within a time window of length k

$$soon(t, Y{\to}y) \equiv \exists t' \; [t \leq t' < t+k \; \wedge \; event(t', Y{\to}y)] \tag{2}$$

(The length of the time window k is learned by noting how long it takes for motor commands to be observed as changes in the world, see [18].) With these variables, we define a contingency as

$$event(t, X{\to}x) \Rightarrow soon(t, Y{\to}y) \tag{3}$$

which represents the proposition that if the *antecedent event* $X_t{\to}x$ occurs, then the *consequent event* $Y{\to}y$ will occur within k timesteps.

2) The Pairwise Search: QLAP looks for contingencies using a pairwise search by tracking statistics on pairs of events $X{\to}x$ and $Y{\to}y$ to determine if the pair is a contingency. QLAP learns a contingency $E_1 \Rightarrow E_2$ if when the event E_1 occurs, then the event E_2 is more likely to soon occur than it would have been otherwise

$$Pr(soon(t, E_2)|E_1(t)) > Pr(soon(t, E_2)) \tag{4}$$

where $Pr(soon(t, E_2))$ is the probability of event E_2 occurring within a random window of k timesteps. Specifically, the contingency is learned when

$$Pr(soon(t, E_2)|E_1(t)) - Pr(soon(t, E_2)) > \theta_{pen} = 0.05 \tag{5}$$

QLAP performs this search considering all pairs of events, excluding those where

1) The consequent event is a magnitude variable (since these are handled by the models on magnitude variables as discussed in Section III-E).
2) The consequent event is on a direction of change variable to the landmark value $[0]$ (since we want to predict changes that result in moving towards or away from landmarks).
3) The magnitude variable corresponding to the direction of change variable on the consequent event matches the magnitude variable on the antecedent event (since we want to learn how the values of variables are affected by other variables).

B. Converting Contingencies to DBNs

In this section we describe how QLAP converts a contingency of the form $event(t, X{\to}x) \Rightarrow soon(t, Y{\to}y)$ into a dynamic Bayesian network. A dynamic Bayesian network (DBN) is a compact way to describe a probability distribution over time-series data. Dynamic Bayesian networks allow QLAP to identify situations when the contingency will be reliable.

1) Adding a Context: The consequent event may only follow the antecedent event in certain contexts, so we also want to learn a set of qualitative context variables $\mathcal{C}$ that predict when event $Y{\to}y$ will soon follow $X{\to}x$. This can be represented as a DBN r of the form

$$r = \langle \mathcal{C} : event(t, X{\to}x) \Rightarrow soon(t, Y{\to}y) \rangle \tag{6}$$

which we abbreviate to

$$r = \langle \mathcal{C} : X{\to}x \Rightarrow Y{\to}y \rangle \tag{7}$$

In this notation, event $E_1 = X{\to}x$ is the antecedent event, and event $E_2 = Y{\to}y$ is the consequent event. We can further abbreviate this QLAP DBN r as

$$r = \langle \mathcal{C} : E_1 \Rightarrow E_2 \rangle \tag{8}$$

Figure 5 shows the correspondence between this notation and standard DBN notation. Because we consider only cases where event E_1 occurs, we can treat the conditional probability table (CPT) of DBN r as defined over the probability that event $Y{\to}y$ will soon follow event $X{\to}x$ for each qualitative value in context $\mathcal{C}$. If the antecedent event does not occur, then the CPT does not define the probability for the consequent event occurring. If the antecedent event occurs, and the consequent event does follow soon after, we say that the DBN *succeeds*. Likewise, if the antecedent event occurs, and the consequent event does not follow soon after, we say that the DBN *fails*. These models are referred to as dynamic Bayesian networks and not simply Bayesian networks because we are using them to model a dynamic system. An example of a DBN learned by QLAP is shown in Figure 6.

The set $\mathcal{C} = \{v_1, \ldots, v_n\}$ consists of the variables in the conditional probability table (CPT) of the DBN $r = \langle \mathcal{C} : E_1 \Rightarrow E_2 \rangle$. The CPT is defined over the product space

$$\mathcal{Q}(\mathcal{C}) = \mathcal{Q}(v_1) \times \mathcal{Q}(v_2) \times \cdots \times \mathcal{Q}(v_n) \tag{9}$$

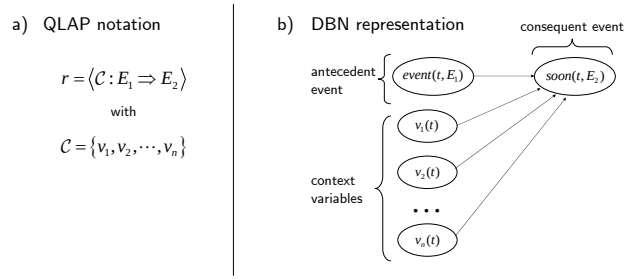


Fig. 5: Correspondence between QLAP DBN notation and traditional graphical DBN notation. (a) QLAP notation of a DBN. Context $\mathcal{C}$ consists of a set of qualitative variables. Event E_1 is an *antecedent event* and event E_2 is a *consequent event*. (b) Traditional graphical notation. Boolean parent variable $event(t, E_1)$ is true if event E_1 occurs at time t. Boolean child variable $soon(t, E_2)$ is true if event E_2 occurs within k timesteps of t. The other parent variables are the context variables in $\mathcal{C}$. The conditional probability table (CPT) gives the probability of $soon(t, E_2)$ for each value of its parents. For all elements of the CPT where $event(t, E_1)$ is false, the probability is undefined. The remaining probabilities are learned through experience.

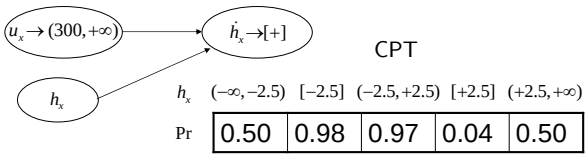


Fig. 6: An example DBN. This DBN says that if the motor value of u_x becomes greater than 300, and the location of the hand, h_x, is in the range $-2.5 \leq h_x < 2.5$, then the variable $\dot{h}_x$ will most likely soon become $[+]$ (the hand will move to the right). (The limits of movement of h_x are -2.5 and $+2.5$, and so the prior of 0.5 dominates outside of that range.)

Since $\mathcal{C}$ is a subset of the variables available to the agent, $\mathcal{Q}(\mathcal{C})$ is an abstraction of the overall state space $\mathcal{S}$

$$\mathcal{Q}(\mathcal{C}) \subseteq \mathcal{Q}(v_1) \times \mathcal{Q}(v_2) \times \cdots \times \mathcal{Q}(v_m) = \mathcal{S} \qquad (10)$$

where $m \geq n$ is the number of variables available to the agent.[4]

2) Notation of DBNs: We define the reliability for $q \in \mathcal{Q}(\mathcal{C})$ for DBN r as

$$rel(r, q) = Pr(soon(t, E_2) | E_1(t), q) \qquad (11)$$

which is the probability of success for the DBN for the value $q \in \mathcal{Q}(\mathcal{C})$. These probabilities come from the CPT and are calculated using observed counts.

The *best reliability* of a DBN gives the highest probability of success in any context state. We define the best reliability $brel(r)$ of a DBN r as

$$brel(r) = \max_{q \in \mathcal{Q}(\mathcal{C})} rel(r, q) \qquad (12)$$

(We require 5 actual successes for $q \in \mathcal{Q}(\mathcal{C})$ before it can be considered for best reliability.) By increasing the best reliability $brel(r)$ we increase the reliability of DBN r. And we say that a DBN r is *sufficiently reliable* if at any time $brel(r) > \theta_{SR} = 0.75$.

The entropy of a DBN $r = \langle \mathcal{C} : E_1 \Rightarrow E_2 \rangle$ is a measure of how well the context $\mathcal{C}$ predicts that event E_2 will soon follow event E_1. Since we only consider the timesteps where event E_1 occurs, we define the entropy $H(r)$ of a DBN r as

$$H(r) = \sum_{q \in \mathcal{Q}(\mathcal{C})} H(soon(t, E_2) | q, E_1(t)) Pr(q | E_1(t)) \qquad (13)$$

By decreasing the entropy $H(r)$ of DBN r, we increase the determinism of DBN r.

C. Adding Context Variables

QLAP iteratively adds context variables to DBNs to make them more reliable and deterministic. This hillclimbing process of adding one context variable at a time is inspired by the marginal attribution process in Drescher's [5] schema mechanism. By only considering the next variable to improve the DBN, marginal attribution decreases the search space. Drescher spun off an entirely new model each time a context variable was added, and this resulted in a proliferation of models. To eliminate this proliferation of models, we instead modify the model by changing the context.

QLAP initially hillclimbs on best reliability $brel(r)$ because the DBN models will eventually be used for planning. In our experiments, we found this to be necessary to make good plans because we want to find some context state in which the model is reliable. This allows the agent to set a subgoal getting to that reliable context state. However, we also want the model to be deterministic, so after a model is sufficiently reliable, QLAP hillclimbs on reduction of entropy $H(r)$.

D. Learning New Landmarks

Learning new landmarks allows the agent to perceive and represent the world at a higher resolution. This increase in resolution allows existing models to be made more reliable and allows new models to be learned. QLAP has two mechanisms for learning landmarks. The first is to learn a new landmark to make an existing DBN more reliable. The second is to learn a new landmark that predicts the occurrence of an event.

1) New Landmarks on Existing DBNs: QLAP learns new landmarks based on previously-learned models (DBNs). For any particular DBN, predicting when the consequent event will follow the antecedent event is a supervised learning problem. This is because once the antecedent event occurs, the environment will determine if the consequent event will occur. QLAP takes advantage of this supervisory signal to learn new landmarks that improve the predictive ability of DBNs.

For each DBN $r = \langle \mathcal{C} : E_1 \Rightarrow E_2 \rangle$, QLAP searches for a landmark on each magnitude and motor variable v in each open interval $q \in \mathcal{Q}(v)$. Finding this landmark is done using the information theoretic method of Fayyad and Irani [22]. The best candidate is then adopted if it has sufficient information gain, and the product of the information gain and the probability of being in that interval $Pr(v = q)$ is sufficient, and the adopted landmark would improve DBN r.

[4] In our experiments, we limit n to be 2.

2) New Landmarks to Predict Events: QLAP also learns new landmarks to predict events. QLAP needs this second landmark learning process because some events may not be preceded by another known event. An example of this is when an object moves because it is hit by another object. In this case, it needs to learn that a distance of 0 between objects is significant, because it causes one of the objects to move.

QLAP learns landmarks to predict events by storing histograms of observed real values of variables. If a landmark v^* is found that co-occurs with some event E, then the agent can predict the occurrence of event E by learning a DBN of the form $\langle \mathcal{C} : v \rightarrow v^* \Rightarrow E \rangle$. Using these histograms, QLAP searches for such a landmark preceding event E by looking for a variable $\tilde{v}$ such that the distribution of $\tilde{v}$ is significantly different just before the event E than otherwise.

E. Magnitude DBN Models

A magnitude value can be less than, greater than, or equal to a qualitative value. We want to have models for a variable $\tilde{v}$ reaching a qualitative value q. Intuitively, if we want $v = q$ and currently $v(t) < q$, then we need to set $\dot{v} = [+]$. This section describes how this process is modeled.

For each magnitude variable v and each qualitative value $q \in \mathcal{Q}(v)$, QLAP creates two models, one that corresponds to approaching the value $v = q$ from below on the number line, and another that corresponds to approaching $v = q$ from above. For each magnitude variable Y and each value $y \in \mathcal{Q}(Y)$, these models can be written as

$$r^+ = \langle \mathcal{C} : \dot{Y} \rightarrow [+] \Rightarrow Y \rightarrow y \rangle \quad (14)$$

$$r^- = \langle \mathcal{C} : \dot{Y} \rightarrow [-] \Rightarrow Y \rightarrow y \rangle \quad (15)$$

DBN r^+ means that if $Y_t < y$ and $\dot{Y} = [+]$, then eventually event $Y \rightarrow y$ will occur (DBN r^- is analogous in this discussion). As the notation suggests, we can treat $\dot{Y} \rightarrow [+] \Rightarrow Y \rightarrow y$ similarly to how we treat a contingency, and we can learn context variables for when this model will be reliable. These models are based on the test-operate-test-exit (TOTE) models of Miller *et al.* [23].

Magnitude DBNs do not use the "soon" predicate because how long it takes to reach a qualitative value is determined by how far away the variable is from that value. Instead, statistics are gathered on magnitude DBNs when the agent sets $\dot{Y} = [+]$ to bring about $Y \rightarrow y$. DBN r^+ is successful if $Y \rightarrow y$ occurs while $\dot{Y} = [+]$, and it fails if the agent is unable to maintain $\dot{Y} = [+]$ long enough to bring about event $Y \rightarrow y$. Like change DBNs, magnitude DBNs will be used in planning as described in Section IV.[5]

IV. Modeling Actions

QLAP uses the learned DBN models to create plans for performing actions. There are two broad planning frameworks within AI: STRIPS-based goal regression [24], and Markov Decision Process (MDP) planning [25]. Goal regression has the advantage of working well when only some of the variables are relevant, and MDP planning has the advantage of providing a principled framework for probabilistic actions [26]. Planning in QLAP was designed to exploit the best of both frameworks. A broad principle of QLAP is that the agent should learn a factored model of the environment to make learning and planning more tractable. QLAP uses MDP planning to plan within each learned factor and uses goal regression to stitch the factors together.

QLAP defines an *action* to achieve each qualitative value of each variable. QLAP creates *plans* from the learned models, where each plan is a different way to perform an action. An action can have zero, one, or many plans. If an action has no plans it cannot be performed. If an action has multiple plans, then the action can be performed in more than one way. The actions that can be called by each plan are QLAP actions to bring about qualitative events. This stitches the plans together and leads to an action hierarchy because the plans to perform one QLAP action call other QLAP actions as if they were primitive actions. This hierarchical action network encodes all of the learned skills of the agent. See Figure 7.

Each plan is policy of an MDP created from a DBN. The policy for each plan is learned using a combination of model-based and model-free methods. QLAP uses MDP planning instead of only goal regression because the transitions are probabilistic. Since the variables in the state space of the plan only come from the DBN model, we minimize the problem of state explosion common to MDP planning. Additionally, we use MDP planning instead of a more specialized planning algorithm such as RRT [27] because RRT and similar algorithms are designed specifically for moving in space, while the actions taken by the QLAP plan may be arbitrary, such as "hit the block off the table."

In this section, we define actions and plans in QLAP. We discuss how change and magnitude DBNs are converted into plans. We then discuss how QLAP can learn when variables need to be added to the state space of a plan, and we conclude with a description of how actions are performed in QLAP.

A. Actions and Plans in QLAP

Actions are how the QLAP agent brings about changes in the world. An action $a(v, q)$ is created for each combination of qualitative variable v and qualitative value $q \in \mathcal{Q}(v)$. An action $a(v, q)$ is *called* by the agent and is said to be *successful* if $v = q$ when it terminates. Action $a(v, q)$ *fails* if it terminates with $v \neq q$. Statistics are tracked on the reliability of actions. The reliability of an action a is denoted by $rel(a)$, which gives the probability of succeeding if it is called.

When an action is called, the action chooses a plan to carry it out. Each plan implements only one action, and an action can have multiple different plans where each plan is a different way to perform the action. This gives QLAP the advantage of being able to use different plans in different situations instead of having one big plan that must cover all situations. As with actions, we say that a plan associated with action $a(v, q)$ is successful if it terminates with $v = q$ and fails if it terminates with $v \neq q$.

[5]While context variables are learned on magnitude DBNs, experiments showed that landmarks learned from these models were not useful to the agent, so these models are not used for learning landmarks in QLAP.

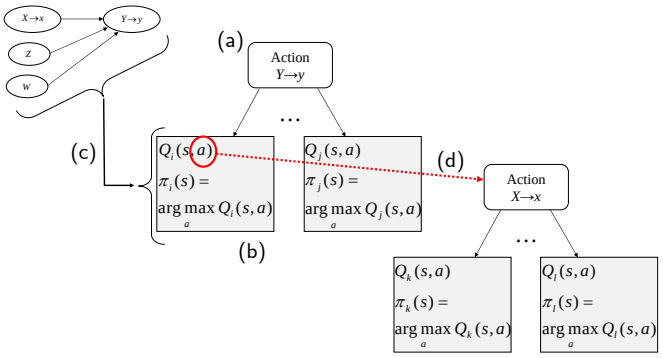


Fig. 7: Planning in QLAP. (a) QLAP defines an action for each qualitative value of each variable. This action is to bring variable Y to value y. (b) Each action can have multiple plans, which are different ways to perform the action. Each plan comes from an MDP. The value $Q_i(s,a)$ is the cumulative, expected, discounted reward of choosing action a in state s in MDP $\mathcal{M}_i$. Given the function Q_i, the policy π_i can be computed. (c) Plans are created from models. The state space for an MDP is the cross product of the values of X, Y, Z, and W from the model (although more can be added if needed). (d) The actions for each plan are QLAP actions to move to different locations in the state space of the MDP. This is reminiscent of goal-regression. Above, we see that one of the actions for $\mathcal{M}_i$ is to call the QLAP action to bring about $X{\rightarrow}x$. This link results from event $X{\rightarrow}x$ being the antecedent event of the DBN model to bring about event $Y{\rightarrow}y$.

Models learned by QLAP can result in plans. Each plan is represented as a policy π_i over an MDP $\mathcal{M}_i = \langle \mathcal{S}_i, \mathcal{A}_i, T_i, R_i \rangle$. A Markov Decision Process (MDP) is a framework for temporal decision making [25]. Since QLAP learns multiple models of the environment, QLAP learns multiple MDPs. And as with models, each MDP represents a small part of the environment. The actions used within each MDP are QLAP actions. And since these actions, in turn, use plans that call other QLAP actions, the actions and plans of QLAP are tied together, and planning takes the flavor of goal regression.

We can think of this policy π_i as being part of an option $o_i = \langle \mathcal{I}_i, \pi_i, \beta_i \rangle$ where $\mathcal{I}_i$ is a set of initiation states, π_i is the policy, and β_i is a set of termination states or a termination function [28] . An option is like a subroutine that can be called to perform a task. Options in QLAP follow this pattern except that π_i is a policy over QLAP actions instead of being over primitive actions or other options.

We use the terminology of a plan being an option because options are common in the literature, and because QLAP takes advantage of the non-Markov termination function β_i that can terminate after a fixed number of timesteps. However, plans in QLAP differ from options philosophically because options are usually used with the assumption that there is some underlying large MDP. QLAP assumes no large, underlying MDP, but rather creates many little, independent MDPs that are connected by actions. Each small MDP $\mathcal{M}_i$ created by QLAP has one policy π_i.

B. Converting Change DBNs to Plans

When a DBN of the form $r_i = \langle \mathcal{C} : X \rightarrow x \Rightarrow Y \rightarrow y \rangle$ becomes sufficiently reliable it is converted into a plan to bring about $Y \rightarrow y$.[6] This plan can then be called by the action $a(Y, y)$.

This plan is based on an MDP $\mathcal{M}_i$. In this section, we will first describe how QLAP creates MDP $\mathcal{M}_i$ from a DBN r_i. We will then describe how QLAP learns a policy for this MDP. And finally, we will describe how this policy is mapped to an option.

1) Creating the MDP from the DBN: QLAP converts DBNs of the form $r_i = \langle \mathcal{C} : X{\rightarrow}x \Rightarrow Y{\rightarrow}y \rangle$ to an MDP of the form $\mathcal{M}_i = \langle \mathcal{S}_i, \mathcal{A}_i, T_i, R_i \rangle$. The state space $\mathcal{S}_i$ consists of the Cartesian product of the values of the variables in DBN r_i. The actions in $\mathcal{A}_i$ are the QLAP actions to bring the agent to the different states of $\mathcal{S}_i$. The transition function T_i comes from the CPT of r_i and the reliability $rel(a)$ of different actions $a \in \mathcal{A}_i$. The reward function simply penalizes each action with a cost of 2 and gives a reward of 10 for reaching the goal of $Y = y$.

2) Learning a Policy for the MDP: QLAP uses three different methods to learn the policy π_i for each MDP $\mathcal{M}_i$. (1) QLAP uses the transition function T_i and the reward function R_i to learn the Q-table Q_i using dynamic programming with value iteration [29]. The Q-table value $Q_i(s,a)$ represents the cumulative, expected discounted reward of taking action $a \in \mathcal{A}_i$ in state $s \in \mathcal{S}_i$. The policy π_i then follows directly from Q_i because for each state s, the agent can choose action a that maximizes $Q_i(s,a)$ (or it can choose some other action to explore the world). (2) As the agent further experiences the world, policy π_i is updated using the temporal difference learning method Sarsa(λ) [29]. (3) And as the agent gathers more statistics, its transition model may be improved. QLAP also updates the model by occasionally running one loop of the dynamic programming.

3) Mapping the Policy to an Option: An option has the form $o_i = \langle \mathcal{I}_i, \pi_i, \beta_i \rangle$. We have described how the policy π_i is learned. When an option o_i is created for a DBN $r_i = \langle \mathcal{C} : X{\rightarrow}x \Rightarrow Y{\rightarrow}y \rangle$, the set of initiation states $\mathcal{I}_i$ is the set of all states.

The termination function β_i terminates option o_i when it *succeeds* (the consequent event occurs) or when it exceeds resource constraints (300 timesteps, or 5 action calls) or when the agent gets stuck. The agent is considered stuck if none of the self variables (see Section V-C for a discussion of how the agent learns which variables are part of "self") or variables in $\mathcal{S}_i$ change in 10 timesteps.

C. Converting Magnitude DBNs into Plans

As discussed in Section III, each qualitative value $y \in \mathcal{Q}(Y)$ on each magnitude variable Y has two DBNs

$$r^+ = \langle \mathcal{C} : \dot{Y}{\rightarrow}[+] \Rightarrow Y{\rightarrow}y \rangle \quad (16)$$

$$r^- = \langle \mathcal{C} : \dot{Y}{\rightarrow}[-] \Rightarrow Y{\rightarrow}y \rangle \quad (17)$$

[6]There are some restrictions on this to limit resource usage. For example, actions that can be reliably performed successfully do not get more plans, the agent must be able to bring about the antecedent event $X{\rightarrow}x$ with sufficient reliability, and each action may have at most three plans. See [18].

that correspond to achieving the event $Y{\rightarrow}y$ from below and above the value $Y = y$, respectively. Both of these DBN models are converted into plans to achieve $Y{\rightarrow}y$. And like with change DBNs, each magnitude DBN r_i is converted into a plan based on an MDP $\mathcal{M}_i$. For MDP $\mathcal{M}_i$, the state space $\mathcal{S}_i$, the set of available actions $\mathcal{A}_i$, the transition function T_i, and the reward function R_i are computed similarly as they are for change plans.

The result of this is that each action $a(v, q)$ on a magnitude variable has two plans. There is one plan to perform the action when $v < q$, and another plan to perform the action when $v > q$. See [18] for further details.

D. Improving the State Space of Plans

The state space of a plan consists of the Cartesian product of the quantity spaces $\mathcal{Q}(v)$ of the variables in the model from which it was created. But what if there are variables that were not part of the model, but that are nonetheless necessary to successfully carry out the plan? To learn when new variables should be added to plans, QLAP keeps statistics on the reliability of each plan and uses those statistics to determine when a variable should be added.

1) Tracking Statistics on Plans: QLAP tracks statistics on plans the same way it does when learning models. For change DBN models, QLAP tracks statistics on the reliability of the contingency. For magnitude models, QLAP tracks statistics on the ability of a variable to reach a qualitative value if moving in that direction. For plans, QLAP tracks statistics on the agent's ability to successfully complete the plan when called.

To track these statistics on the probability of a plan o being successful, QLAP creates a *second-order* model

$$r_o^2 = \langle \mathcal{C}_o : call(t, o) \Rightarrow succeeds(t, o) \rangle \tag{18}$$

The child variable of second-order DBN r_o^2 is $succeeds(t, o)$, which is true if option o succeeds after being called at time t and is false otherwise. The parent variables of r_o^2 are $call(t, o)$ and the context variables in $\mathcal{C}_o$. The Boolean variable $call(t, o)$ is true when the option is called at time t and is false otherwise. When created, model r_o^2 initially has an empty context, and context variables are added in as they are for magnitude and change models. The notation for these models is the same as for magnitude and change models: QLAP computes $rel(o)$, $rel(o, s)$ and $brel(o)$. Therefore a plan can also be sufficiently reliable if at any time $brel(o) > \theta_{SR} = 0.75$.

2) Adding New Variables to the State Space: Second-order models allow the agent to identify other variables necessary for the success of an option o because those variables will be added to its context. Each variable that is added to r_o^2 is also added to the state space $\mathcal{S}_i$ of its associated MDP $\mathcal{M}_i$. For example, for a plan created from model $r_i = \langle \mathcal{C} : X{\rightarrow}x \Rightarrow Y{\rightarrow}y \rangle$, the state space $\mathcal{S}_i$ is updated so that

$$\mathcal{S}_i = \mathcal{Q}(\mathcal{C}_o) \times \mathcal{Q}(\mathcal{C}) \times \mathcal{Q}(X) \times \mathcal{Q}(Y) \tag{19}$$

(variables in more than one of $\mathcal{C}_o$, $\mathcal{C}$, $\{X\}$, or $\{Y\}$ are only represented once in $\mathcal{S}_i$). For both magnitude and change options, an action $a(v, q)$ where $v \in \mathcal{Q}(\mathcal{C}_o)$ is treated the same way as $v \in \mathcal{Q}(\mathcal{C})$.

E. Performing Actions

QLAP actions are performed using plans, and these plans call other QLAP actions. This leads to a hierarchy of plans and actions.

1) Calling and Processing Actions: When an action is *called*, it chooses a plan and then starts executing the policy of that chosen plan. Executing that policy results in more QLAP actions being called, and this process continues until a motor action is reached. When an action $a(u, q)$ is called on a motor variable u, then QLAP sends a random motor value within the range covered by the qualitative value $u = q$ to the body.

This hierarchical structure of actions and plans means that multiple actions will be performed simultaneously. Each plan only keeps track of what action it is currently performing. And when that action terminates, the next action is called based on according to the policy. So as the initial action called by the agent is being processed, the path between that initial action and a motor actions continually changes.

2) Terminating Actions: An action $a(v, q)$ terminates if $v = q$, in which case it *succeeds*. It also terminates if it *fails*. An action fails if

(a) it has no plans, or
(b) for every plan for this action, the action to bring about the antecedent event of the plan is already in the call list, or
(c) its chosen plan fails.

Similar to an action, a plan to bring about $v = q$ terminates if $v = q$, in which case it *succeeds*. It also terminates if it *fails*. A plan to bring about $v = q$ fails if

(a) the termination function β is triggered by resource constraints, or
(b) there is no applicable action in the current state, or
(c) the action chosen by the policy is already in the call list, or
(d) the action chosen by the policy immediately fails when it is called.

V. Exploration and Learning

The QLAP agent explores and learns autonomously without being given a task. This autonomous exploration and learning raises many issues. For example, how can the agent decide what is worth exploring? As the agent explores, it learns new representations. How can it keep from learning unnecessary representations and getting bogged down? Should the agent use the same criteria for learning all representations? Or should it treat some representations as especially important? And finally, can the agent learn that some parts of the environment can be controlled with high reliability and low latency so that they can be considered part of "self"?

Previous sections have explained how QLAP learns representations that take the form of landmarks, DBNs, plans, and actions. This section explains how learning in QLAP unfolds over time. We first discuss how the agent explores the environment. We then discuss developmental restrictions that determine what representations the agent learns and the order in which it learns them. We then discuss how QLAP pays special attention to goals that are hard to achieve. And finally, we discuss how the agent learns what is part of "self."

A. Exploration

The QLAP agent explores the environment autonomously without being given a task. Instead of trying to learn to do a particular task, the agent tries to learn to predict and control all of the variables in its environment. However, this raises difficulties because there might be many variables in the environment, and some may be difficult or impossible to predict or control. This section explains how the agent determines what should be explored and the best way to go about that exploration.

Initially, the agent motor babbles for 20,000 timesteps by repeatedly choosing random motor values and maintaining those values for a random number of time steps. After that point, QLAP begins to practice its learned actions. An outline of the execution of QLAP is shown in Algorithm 1.

The agent continually makes three types of choices during its exploration. These choices vary in time scale from coarse to fine. The agent chooses: a learned action $a(v, q)$ to practice, the best plan o_i for performing the action $a(v, q)$, and the action based on policy π_i for plan o_i.

1) Choosing a Learned Action to Practice: One method for choosing where to explore is to measure prediction error and than to motivate the agent to explore parts of the space for which it currently does not have a good model. This form of intrinsic motivation is used in [30], [31]. However, focusing attention on states where the model has poor prediction ability can cause the agent to explore spaces where learning is too difficult.

Schmidhuber [32] proposed a method whereby an agent learns to predict the decrease in the error of the model that results from taking each action. The agent can then choose the action that will cause the biggest decrease in prediction error. Oudeyer, Kaplan, and Hafner [33] apply this approach with a developing agent and have the agent explore regions of the sensory-motor space that are expected to produce the largest decrease in predictive error. Their method is called Intelligent Adaptive Curiosity (IAC).

QLAP uses IAC to determine which action to practice. After the motor babbling period of 20,000 timesteps, QLAP chooses a motor babbling action with probability 0.1, otherwise it uses IAC to choose a learned action to practice. Choosing a learned action to practice consists of two steps: (1) determine the set of applicable actions that could be practiced in the current state s, and (2) choose an action from that set.

The set of applicable actions to practice consists of the set of actions that are not currently accomplished, but could be performed. For a change action, this means that the action must have at least one plan. For a magnitude action $a(v, q)$, this means that if $v_t < q$ then $a(\dot{v}, [+])$ must have at least one plan (and similarly for $v_t > q$).

QLAP chooses an action to practice by assigning a weight w_a to each action a in the set of applicable actions. The action is then chosen randomly based on this weight w_a. The weights are assigned using a version of Intelligent Adaptive Curiosity (IAC) [33] that measures the change in the agent's ability to perform the action over time and then chooses actions where that ability is increasing.

2) Choosing the Best Plan to Perform an Action: When an action is called, it chooses a plan to perform the action. QLAP seeks to choose the plan that is most likely to be successful in the current state. To compare plans, QLAP computes a weight w_o^s for each plan o in state s. To compute the weight w_o^s for plan o in state s, QLAP computes the product of the reliability of the DBN r that led to the plan $rel(r, s)$ and the reliability of the second-order DBN $rel(o, s)$ so that

$$w_o^s = rel(r, s) \cdot rel(o, s) \tag{20}$$

To choose the plan to perform the action, QLAP uses ϵ-greedy action plan selection ($\epsilon = 0.05$). With probability $1-\epsilon$, QLAP chooses the plan with the highest weight. And with probability ϵ it chooses a plan randomly. To prevent loops in the calling list, a plan whose DBN has its antecedent event already in the call list is not applicable and cannot be chosen.

3) Choosing an Action within a Plan: Recall from Section IV that QLAP learns a Q-table for each plan that gives a value for taking each action a in state s. Here again, QLAP uses ϵ-greedy selection. With probability $1 - \epsilon$, in state s, QLAP chooses action a that maximizes $Q_i(s, a)$, and with probability ϵ, QLAP chooses a random action. This action selection method balances exploration with exploitation [29].

Algorithm 1 The Qualitative Learner of Action and Perception (QLAP)

```
for t = 0 : ∞ do
    sense environment
    convert input to qualitative values using current landmarks
    update statistics for learning new contingencies
    update statistics for each DBN
    if mod(t, 2000) == 0 then
        learn new DBNs
        update contexts on existing DBNs
        delete unneeded DBNs and plans
        if mod(t, 4000) == 0 then
            learn new landmarks on events
        else
            learn new landmarks on DBNs
        end if
        convert DBNs to plans
    end if
    if current exploration action is completed then
        choose new exploration action and action plan
    end if
    get low-level motor command based on current qualitative state and plan of current exploration action
    pass motor command to robot
end for
```

B. Targeted Learning

Since QLAP creates an action for each variable and qualitative value combination, a QLAP agent is faced with many potential actions that could be learned. QLAP can choose different actions to practice based on the learning gradient,

but what about the thresholds to learn predictive DBN models and plans? Some actions might be more difficult to learn than others, so it seems reasonable that the requirements for learning representations that lead to learning such actions should be loosened.

QLAP does targeted learning for difficult actions. To learn a plan for an action chosen for targeted learning, QLAP

1) **Lowers the threshold needed to learn a contingency.** Recall from Section III-A2, that a contingency is learned when

$$Pr(soon(t, E_2)|E_1(t)) - Pr(soon(t, E_2)) > \theta_{pen} = 0.05 \quad (21)$$

If event E_2 is chosen for targeted learning, QLAP makes it more likely that a contingency will by learned by setting $\theta_{pen} = 0.02$.

2) **Lowers the threshold needed to learn a plan.** Recall from Section IV-B that one of the requirements to convert a change DBN r into a plan is that $brel(r) > \theta_{SR} = 0.75$. If event E_2 is chosen for targeted learning, QLAP makes it more likely that a DBN will be converted to a plan by setting $\theta_{SR} = 0.25$.

This leaves the question of when to use targeted learning of actions. An event is chosen as a goal for targeted learning if the probability of being in a state where the event is satisfied is less than 0.05; we call such an event *sufficiently rare*. This is reminiscent of Bonarini *et al.* [34]. They consider desirable states to be those that are rarely reached or are easily left once reached.

C. Learning What Is Part of "Self"

One step towards tool use is making objects in the environment part of "self" so that they can be used to perform useful tasks. The representation of self is straightforward in QLAP. A change variable is part of self if it can be quickly and reliably manipulated. QLAP learns what is part of self by looking for variables that it can reliably control with low latency.

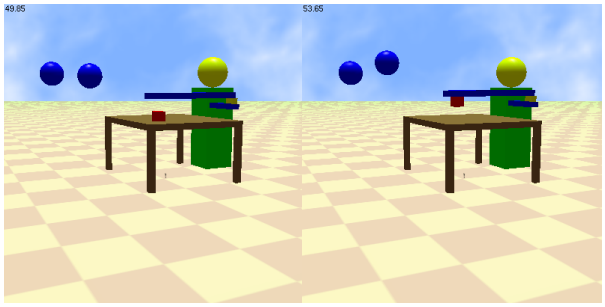
Marjanovic [35] enabled a robot to identify what was part of self by having the robot wave its arm and having the robot assume that the only thing moving in the scene was itself. The work of Metta and Fitzpatrick [36] is similar but more sophisticated because it looks for optical flow that correlates with motor commands of the arm. Gold and Scassellati [37] use the time between giving a motor command and seeing movement to determine what is part of self. Our method for learning self is similar to that of Gold and Scassellati, but we learn what is part of self while learning actions.

A direction of change variable $\dot{v}$ is part of self if:

1) the average time it takes for the action to set $\dot{v} = [+]$ and $\dot{v} = [-]$ is less than k, and
2) the actions for both $\dot{v} = [+]$ and $\dot{v} = [-]$ are sufficiently reliable. where k is how long it takes for motor commands to be observed as changes in the world, see [18].

VI. Evaluation

Evaluating autonomous learning is difficult because there is no pre-set task on which to evaluate performance. The approach we take is to first have the agent learn autonomously in an environment; we then evaluate if the agent is able to perform a set of tasks. It is important to note that during learning the agent does not know on which tasks it will be evaluated.



(a) Not grasping (b) Grasping

Fig. 8: The evaluation environment (shown here with floating objects)

A. Evaluation Environment

The evaluation environment is implemented in Breve [38] and has realistic physics. Breve simulates physics using the Open Dynamics Engine (ODE) [39]. The simulation consists of a robot at a table with a block and floating objects. The robot has an orthogonal arm that can move in the x, y, and z directions. The environment is shown in Figure 8, and the variables perceived by the agent for the core environment are shown in Table II. The block has a width that varies between 1 and 3 units. The block is replaced when it is out of reach and not moving, or when it hits the floor. Each timestep in the simulator corresponds to 0.05 seconds. I.e., 20 timesteps/second is equal to 1200 timesteps/minute is equal to $72,000$ timesteps per hour. See [18] for further details.

The robot can grasp the block in a way that is reminiscent of both the palmer reflex [40] and having a sticky mitten [41]. The palmer reflex is a reflex that is present from birth until the age 4-6 months in human babies. The reflex causes the baby to close its hand when something touches the palm. In the sticky mittens experiments, three-month-old infants wore mittens covered with Velcro that allowed them to more easily grasp objects.

Grasping is implemented on the robot to allow it to grasp only when over the block. Specifically, the block is grasped if the hand and block are colliding, and the Euclidean 2D distance from the center of the block in the x and y directions is less than half the width of the palm, $3/2 = 1.5$ units.

In addition to the core environment, QLAP is also evaluated with distractor objects. This is done using the *floating extension environment*, which adds two floating objects that the agent can observe but cannot interact with. The purpose of this environment is to evaluate QLAP's ability to focus on learnable relationships in the presence of unlearnable ones. The objects float around in an invisible box. The variables added to the core environment to make the floating extension environment are shown in Table III.

B. Experimental Conditions

We compare the performance of QLAP with the performance of a supervised learner on a set of tasks. The supervised learner is trained only on the evaluation tasks. This puts QLAP at a disadvantage on the evaluation tasks because QLAP is not informed of the evaluation tasks and QLAP learns more than the evaluation tasks. We hope QLAP can demonstrate developmental learning by getting better at the tasks over time, and that QLAP can do as well as the supervised learner.

Each agent is evaluated on three tasks. These are referred to as the *core tasks*.

1) **move the block** The evaluator picks a goal to move the block *left* ($\dot{T}_L = [+]$), *right* ($\dot{T}_R = [-]$), or *forward* ($\dot{T}_T = [+]$). The goal is chosen randomly based on the relative position of the hand and the block. A trial is terminated early if the agent hits the block in the wrong direction.
2) **hit the block to the floor** The goal is to make $bang =$ true.
3) **pick up the block** The goal is to get the hand in just the right place so the robot can grasp the block and make $T =$ true. A trial is terminated early if the agent hits the block out of reach.

The supervised learner is trained using linear, gradient-descent Sarsa(λ) with binary features [29] where the binary features come from tile coding. Tile coding is a way to discretize continuous input for reinforcement learning. Both the QLAP and the supervised learning agents are evaluated on the core tasks in both the core environment and the floating extension environment under three experimental conditions.

1) **QLAP** The QLAP algorithm.
2) **SupLrn-1** Supervised learning, choosing an action every timestep.
3) **SupLrn-10** Supervised learning, choosing an action every 10 timesteps.

SupLrn-1 and SupLrn-10 are both used because SupLrn-1 has difficulty learning the core tasks due to high task diameter.

QLAP learns autonomously for 250,000 timesteps (corresponding to about 3.5 hours of physical experience) as described in Section V. The supervised learning agents repeatedly perform trials of a particular core task for 250,000 timesteps. At the beginning of each trial, the core task that the supervised learning agent will practice is chosen randomly. The state of the agent is saved every 10,000 timesteps (about every 8 minutes of physical experience). The agent is then evaluated on how well it can do the specified task using the representations from each stored state.

At the beginning of each trial, a block is placed in a random location within reach of the agent and the hand is moved to a random location. Then, the goal is given to the agent. The agent makes and executes plans to achieve the goal. If the QLAP agent cannot make a plan to achieve the goal, it moves randomly. The trial is terminated after 300 timesteps or when the goal is achieved. The agent receives a penalty of -0.01 for each timestep it does not achieve the goal and a reward of 9.99 on the timestep it achieves the goal. (SupLrn-10 gets a penalty of -0.1 every 10th timestep it does not reach the goal and a reward of 9.99 on the timestep it reaches the goal.)

Each evaluation consists of 100 trials. The rewards over the 100 trials are averaged, and the average reward is taken as a measure of ability. For each experiment, 20 QLAP agents and 20 supervised learning agents are trained.

TABLE II: Variables of the core environment

Variable	Meaning
u_x, u_y, u_z	force in x, y, and z directions
u_{UG}	ungrasp the block
h_x, h_y, h_z	global location of hand in x, y, and z directions
$\dot{h}_x$, $\dot{h}_y$, $\dot{h}_z$	derivative of h_x, h_y, h_z
y_{TB}, $\dot{y}_{TB}$	top of hand in frame of reference of bottom of block (y direction)
y_{BT}, $\dot{y}_{BT}$	bottom of hand in frame of reference of top of block (y direction)
x_{RL}, $\dot{x}_{RL}$	right side of hand in frame of reference of left side of block (x direction)
x_{LR}, $\dot{x}_{LR}$	left side of hand in frame of reference of right side of block (x direction)
z_{BT}, $\dot{z}_{BT}$	bottom side of hand in frame of reference of top of block (z direction)
z_F, $\dot{z}_F$	distance to the floor
T_L, $\dot{T}_L$	location of nearest edge of block in x direction in coordinate frame defined by left edge of table
T_R, $\dot{T}_R$	location of nearest edge of block in x direction in coordinate frame defined by right edge of table
T_T, $\dot{T}_T$	location of nearest edge of block in y direction in coordinate frame defined by top edge of table
c_x, $\dot{c}_x$	location of hand in x direction relative to center of block
c_y, $\dot{c}_y$	location of hand in y direction relative to center of block
T	block is grasped, true or false. Becomes true when the hand is touching the block and the 2D distance between the center of the hand and the center of the block is less than 1.5.
$bang$	true when block hits the floor

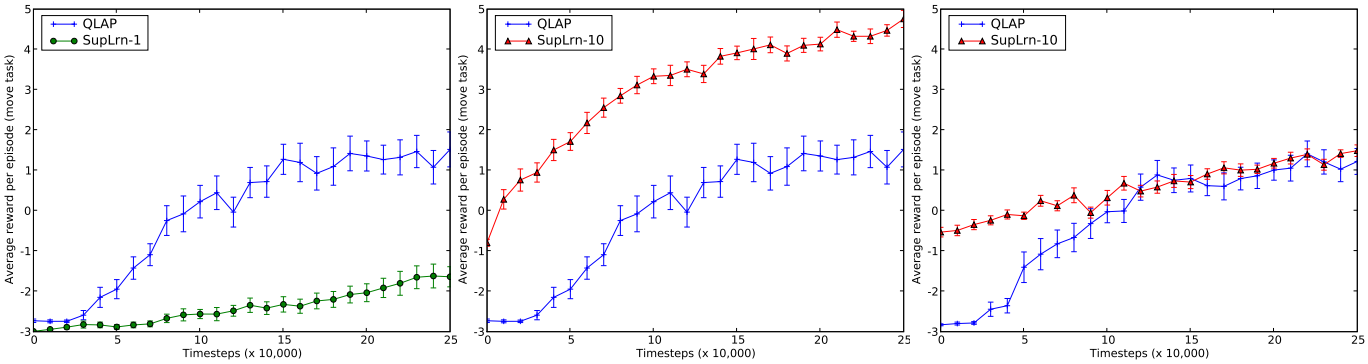
TABLE III: Variables added to the core environment to make up the floating extension environment

Variable	Meaning
f_x^1, f_y^1, f_z^1	location of first floating object in x, y, and z directions
$\dot{f}_x^1$, $\dot{f}_y^1$, $\dot{f}_z^1$	derivative of f_x^1, f_y^1, f_z^1
f_x^2, f_y^2, f_z^2	location of second floating object in x, y, and z directions
$\dot{f}_x^2$, $\dot{f}_y^2$, $\dot{f}_z^2$	derivative of f_x^2, f_y^2, f_z^2

C. Results

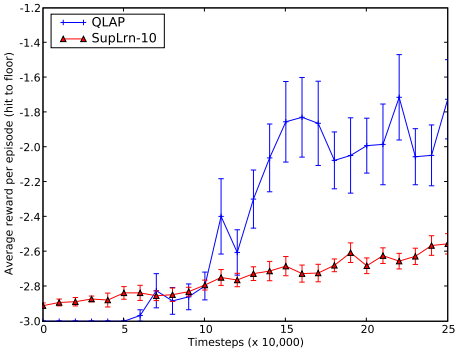
The results are shown in Figures 9 and 10. Figure 9 compares QLAP and supervised learning on the task of moving the block in the specified direction. As can be seen in Figure 9(a), SupLrn-1 was not able to do the task well compared to QLAP due to the high task diameter (the number of timesteps needed to complete the task). Having the supervised learning agents choose an action every 10 timesteps improved their performance, as can be seen in Figure 9(b). But as can be seen by visually inspecting Figure 9(c), the performance of supervised learning degrades much more than the performance of QLAP degrades when the distractor objects are added.

This same pattern of QLAP outperforming SupLrn-1, SupLrn-10 doing as well or better than QLAP in the environment without the distractor objects, but then QLAP not

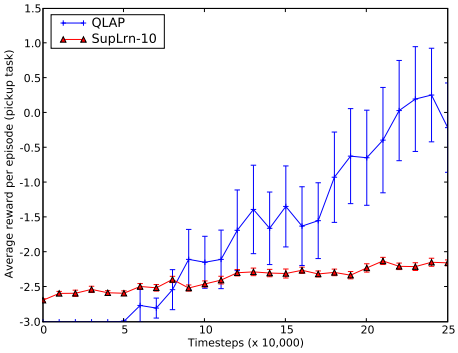


(a) QLAP and SupLrn-1 (b) QLAP and SupLrn-10 (c) Float: QLAP and SupLrn-10

Fig. 9: Moving the block. (a) QLAP does better than SupLrn-1 because of the high task diameter. (b) SupLrn-10 does better than QLAP. (c) When the floating objects are added, the performance of SupLrn-10 degrades much more than the performance of QLAP degrades.



(a) Float: QLAP and SupLrn-10



(b) Float: QLAP and SupLrn-10

Fig. 10: QLAP outperforms supervised reinforcement learning using tile coding on the more difficult tasks in the floating extension environment. (a) Knock the block off the table. (b) Pick up the block.

degrading with the distractor objects was also observed for the tasks of hitting the block off the table and picking up the block. Figure 10 shows the performance of QLAP and supervised learning on the tasks of hitting the block off the table and picking up the block. For brevity, this figure only contains the final comparison of QLAP and SupLrn-10 with floating objects on those tasks. We see that QLAP does better than SupLrn-10 on these more difficult tasks in the environment with floating objects. For all three tasks, QLAP displays developmental learning and gets better over time.

These results suggest the following conclusions: (1) QLAP autonomously selects an appropriate coarse temporal coding for actions, out-performing the fine-grained actions used in SupLrn-1, due to the resulting large task diameter; (2) QLAP is more robust to distractor events than SupLrn-10.

D. Additional evaluation: QLAP learns landmarks that are generally useful

We want to show that the learned landmarks really do represent the "natural joints" in the environment. Since we have no ground truth for what the natural joints of the environment are, we will compare the results of tabular Q-learning using landmarks learned with QLAP with the results of tabular Q-learning using randomly generated landmarks. If the landmarks do represent the joints in the environment, then the tabular Q-learner using learned landmarks should do better than the one using random landmarks.

1) Experimental Environment: Tabular Q-learning does not generalize well. During exploratory experiments, the state space of the core environment was so large that tabular Q-learning rarely visited the same state more than once. We therefore evaluate this claim using a smaller environment and a simple task. We use the *2D core environment* where the hand only moves in two dimensions. It removes the variables of the z direction from the core environment. It subtracts u_z, u_{UG}, h_z, $\dot{h}_z$, z_{BT}, $\dot{z}_{BT}$, c_x, $\dot{c}_x$, c_y, $\dot{c}_y$, T, and $bang$.

2) Experimental Conditions:

(a) **QLAP landmarks** Tabular Q-learning using landmarks learned using QLAP after a previous run of 100,000 timesteps on the 2D core environment.

(b) **random landmarks** Tabular Q-learning using randomly generated landmarks.

To generate the random landmarks, for each magnitude or motor variable v, a random number of landmarks between 0 and 5 is chosen. Each landmark is then placed in a randomly chosen location within the minimum and maximum range observed for v during a typical run of QLAP. Note that motor

variables already have a landmark at 0, so each motor variable had between 1 and 6 landmarks.

3) Results: The results are shown in Figure 11. Tabular Q-learning works much better using the learned landmarks than using the random ones.

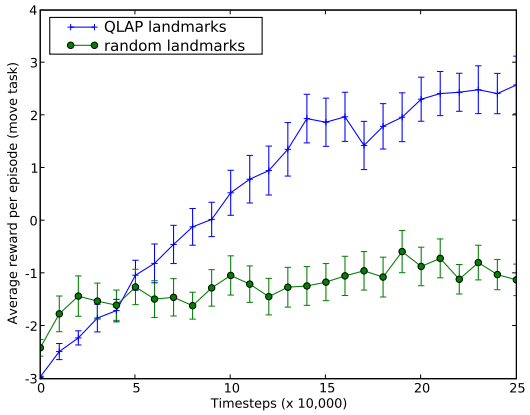


Fig. 11: QLAP landmarks enable the agent to learn the task better than do random landmarks.

VII. Discussion

A. Description of Learned Representations

One of the first contingencies (models) that QLAP learns is that if it gives a positive force to the hand, then the hand will move to the right. But this contingency is not very reliable because in the simulator it takes a force of at least 300 units to move the hand. The agent learns a landmark at 300 on that motor variable, and modifies the model to state that if the force is at least 300, then the hand will move to the right. But this model still is not completely reliable, because if the hand is already all the way to the right, then it can't move any farther. But from this model, the agent can note the location of its hand each time it applies a force of 300, and it can learn a landmark to indicate when the hand is all the way to the right. The completed model states that if the hand is not all the way to the right and a force of at least 300 is given, then the hand will usually move to the right. Even this model is not completely reliable, because there are unusual situations where, for example, the hand is stuck on the block. But the model is probabilistic, so it can handle nondeterminism.

The agent also learns that the location of the right side of the hand in the frame of reference of the left side of the block has a special value at 0. It learns this because it notices that the block begins to move to the right when that value is achieved. It then creates a landmark to indicate that value and an action to reach that value. Based on this landmark, QLAP can learn a contingency (model) that says if the value goes to 0, then the block will move to the right. It can then learn other landmarks that indicate in which situations this will be successful. In a similar way, it learns to pick up the block and knock the block off the table.

B. Theoretical Bounds

QLAP is reasonably efficient in time and space. Let V be the number of variables. QLAP searches for pairs of events to form a contingency, so this process is $O(V^2)$ in both time and space. QLAP searches for context variables and landmarks for each learned contingency. It does this by considering one variable at a time for each DBN, thus these processes are $O(V^3)$ in time and space. This of course means that if you had a very large number of variables, such as 1000, that the variables would need to be categorized so that only a subset of possible contingencies were considered.

MDP planning is known to be computationally expensive because the state space grows exponentially with the number of variables. However this explosion is limited in QLAP because QLAP builds many MDPs, each consisting of only a few variables. Essentially, QLAP searches for the simplest MDP models that give reasonable descriptions of the observed dynamics of the environment. The search is a greedy breadth-first search, incrementally increasing the number of variables in the MDP (via the DBN). Note that any given MDP model describes the world in terms of a certain set of explicit variables. The rest of the dynamics of the world are encoded in the probabilities in the CPT tables. Presumably, there is a general trade-off between the number of variables in the model and the determinism of the CPT. However, QLAP is based on the assumption that that general trade-off is dominated by the choice of the *right* variables and the *right* landmarks for those variables. That is, the right small models may well be quite good, and the hill-climbing methods of QLAP can often find them.

C. Assumptions of QLAP

QLAP assumes that any goal that an outside observer would want the agent to accomplish is represented with an input variable. QLAP also assumes that meaningful landmarks can be found on single variables. In some cases when these assumptions are violated, QLAP can do a search on combinations of variables (see [18]).

QLAP also assumes a set of continuous motor primitives that correspond to orthogonal directions of movement. QLAP builds on the work of Pierce and Kuipers [42]. In their work, an agent was able to use principal components analysis (PCA) [43] to learn a set of motor primitives corresponding to turn and travel for a robot that had motors to turn each of two wheels independently.

VIII. Related Work

QLAP learns states and hierarchical actions in continuous, dynamic environments with continuous motors through autonomous exploration. The closest direct competitor to QLAP is the work of Barto, Jonsson, and Vigorito. Given a DBN model of the environment, the VISA algorithm [44] creates a causal graph which it uses to identify state variables for options. Like QLAP, the VISA algorithm performs state abstraction by finding the relevant variables for each option. Jonsson and Barto [20] learn DBNs through an agent's interaction with a discrete environment by maximizing the posterior of the DBN given the data by building a tree to represent the conditional probability. Vigorito and Barto [45] extends [20],

[44] by proposing an algorithm for learning options when there is no specific task.

This work differs from QLAP in that learning takes place in discrete environments with events that are assumed to occur over one-timestep intervals. The work also assumes that the agent begins with a set of discrete actions. Because QLAP is designed for continuous environments with dynamics, QLAP uses a qualitative representation. This qualitative representation leads to a novel DBN learning algorithm for learning predictive models, and a novel method for converting those models into a set of hierarchical actions.

Shen's LIVE algorithm [46] learns a set of rules in first-order logic and then uses goal regression to perform actions. The algorithm assumes that the agent already has basic actions, and the experiments presented are in environments without dynamics such as the Tower of Hanoi. Another method for learning planning rules in first-order logic is [7], [8]. The rules they learn are probabilistic, given a context and an action their learned rules provide a distribution over results. This algorithm assumes a discrete state space and that the agent already has basic actions such as pick up.

QLAP's structure of actions and plans is reminiscent of the MAXQ value function decomposition [47]. QLAP defines its own actions and plans as it learns, and as the agent learns a more refined discretization the hierarchy changes. There has also been much work on learning hierarchy. Like QLAP, Digney [48] creates a task to achieve each discrete value of each variable. However, QLAP learns the discretization. Work has been done on learning a hierarchical decomposition of a factored Markov decision process by identifying exits. Exits are combinations of variable values and actions that cause some state variable to change its value [44]. Exits roughly correspond to the DBNs found by QLAP except that there is no explicit action needed for QLAP DBNs. Hengst [49] determined an order on the input variables based on how often they changed value. Using this ordering, he identified exits to change the next variable in the order and created an option for each exit.

There has been other work on structure learning in sequential decision processes where the environment can be modeled as a factored MDP. Degris *et al.* [19] proposed a method called SDYNA that learns a structured representa- tion in the form of a decision tree and then uses that structure to compute a value function. Strehl *et al.* [21] learn a DBN to predict each component of a factored state MDP. Hester and Stone [50] learn decision trees to predict both the reward and the change in the next state. All of these methods are evaluated in discrete environments where transitions occur over one-timestep intervals.

IX. Summary and Conclusion

The Qualitative Learner of Action and Perception (QLAP) is an unsupervised learning algorithm that allows an agent to autonomously learn states and actions in continuous environments. Learning actions from a learned representation is significant because it moves the state of the art of autonomous learning from grid worlds to continuous environments. Another contribution of QLAP is providing a method for factoring the environment into small pieces. Instead of learning one large predictive model, QLAP learns many small models. And instead of learning one large plan to perform an action, QLAP learns many small plans that are useful in different situations.

QLAP starts with a bottom-up process that detects contingencies and builds DBN models to identify the conditions (i.e., values of context variables) under which there are near-deterministic relations among events. Meanwhile, an action is defined for each event, where the action has the intended effect of making that event occur. The desirable situation is for an action to have one or more sufficiently reliable plans for implementing the action. If there are no plans for an action, or if the plans are not sufficiently reliable, the action is still defined, but it is not useful, so it will not be used as a step in a higher-level plan. Each plan comes from a sufficiently reliable DBN, where the overall structure is that DBNs lead to MDPs, MDPs are converted into policies, and policies are plans.

QLAP explores autonomously and tries to learn to achieve each qualitative value of each variable. To explore, the agent continually chooses an action to practice. To choose which action to practice, QLAP uses Intelligent Adaptive Curiosity (IAC). IAC motivates the agent to practice actions that it is getting better at, and IAC motivates the agent to stop practicing actions that are too hard or too easy.

QLAP was evaluated in environments with simulated physics. The evaluation was performed by having QLAP explore autonomously and then measuring how well it could perform a set of tasks. The agent learned to hit a block in a specified direction and to pick up the block as well or better than a supervised learner trained only on the task. The evaluation also showed that the landmarks learned by QLAP were broadly useful. Future work will consist of incorporating continuous learning methods within the discretized representation learned by QLAP. This should enable QLAP to leverage both best of discrete learning and the best of continuous learning.

Acknowledgment

This work has taken place in the Intelligent Robotics Lab at the Artificial Intelligence Laboratory, The University of Texas at Austin. Research of the Intelligent Robotics lab is supported in part by grants from the Texas Advanced Research Program (3658-0170-2007), and from the National Science Foundation (IIS-0413257, IIS-0713150, and IIS-0750011).

References

[1] A. Saxena, J. Driemeyer, J. Kearns, and A. Ng, "Robotic grasping of novel objects," *Advances in neural information processing systems*, vol. 19, p. 1209, 2007.

[2] P. Fitzpatrick, G. Metta, L. Natale, S. Rao, and G. Sandini, "Learning about objects through action-initial steps towards artificial cognition," in *IEEE International Conference on Robotics and Automation, 2003. Proceedings. ICRA'03*, vol. 3, 2003.

[3] S. Vijayakumar, A. D'souza, and S. Schaal, "Incremental online learning in high dimensions," *Neural Computation*, vol. 17, no. 12, pp. 2602–2634, 2005.

[4] C. M. Vigorito and A. G. Barto, "Intrinsically motivated hierarchical skill learning in structured environments," *IEEE Transactions on Autonomous Mental Development (TAMD)*, vol. 2, no. 2, 2010.

[5] G. L. Drescher, *Made-Up Minds: A Constructivist Approach to Artificial Intelligence*. Cambridge, MA: MIT Press, 1991.

[6] P. R. Cohen, M. S. Atkin, T. Oates, and C. R. Beal, "Neo: Learning conceptual knowledge by sensorimotor interaction with an environment," in *Agents '97*. Marina del Rey, CA: ACM, 1997.

[7] L. S. Zettlemoyer, H. Pasula, and L. P. Kaelbling, "Learning planning rules in noisy stochastic worlds." in *Proc. 20nd Conf. on Artificial Intelligence (AAAI-2005)*, 2005, pp. 911–918.

[8] H. Pasula, L. Zettlemoyer, and L. Kaelbling, "Learning symbolic models of stochastic domains," *Journal of Artificial Intelligence Research*, vol. 29, pp. 309–352, 2007.

[9] C. Xu and B. Kuipers, "Towards the Object Semantic Hierarchy," in *Proc. of the Int. Conf. on Development and Learning (ICDL 2010)*, 2010.

[10] J. Mugan and B. Kuipers, "The qualitative learner of action and perception, QLAP," in *AAAI Video Competition (AIVC 2010)*, 2010, http://videolectures.net/aaai2010_mugan_qlap.

[11] R. S. Sutton and A. G. Barto, *Reinforcement Learning*. Cambridge MA: MIT Press, 1998.

[12] C. G. Atkeson, A. W. Moore, and S. Schaal, "Locally weighted learning," *Artificial Intelligence Review*, vol. 11, no. 1/5, pp. 11–73, 1997.

[13] ——, "Locally weighted learning for control," *Artificial Intelligence Review*, vol. 11, no. 1/5, pp. 75–113, 1997.

[14] S. Vijayakumar and S. Schaal, "Locally weighted projection regression: An O(n) algorithm for incremental real time learning in high dimensional space," in *Proceedings of the Seventeenth International Conference on Machine Learning (ICML 2000)*, vol. 1, 2000, pp. 288–293.

[15] M. Jordan and D. Rumelhart, "Forward models: Supervised learning with a distal teacher," *Cognitive Science*, vol. 16, pp. 307–354, 1992.

[16] C. Rasmussen, "Gaussian processes in machine learning," *Advanced Lectures on Machine Learning*, pp. 63–71, 2006.

[17] B. Kuipers, *Qualitative Reasoning*. Cambridge, Massachusetts: The MIT Press, 1994.

[18] J. Mugan, "Autonomous Qualitative Learning of Distinctions and Actions in a Developing Agent," Ph.D. dissertation, University of Texas at Austin, 2010.

[19] T. Degris, O. Sigaud, and P. Wuillemin, "Learning the structure of factored Markov decision processes in reinforcement learning problems," in *ICML*, 2006, pp. 257–264.

[20] A. Jonsson and A. Barto, "Active learning of dynamic bayesian networks in markov decision processes," *Lecture Notes in Artificial Intelligence: Abstraction, Reformulation, and Approximation - SARA*, pp. 273–284, 2007.

[21] A. Strehl, C. Diuk, and M. Littman, "Efficient structure learning in factored-state MDPs," in *AAAI*, vol. 22, no. 1, 2007, p. 645.

[22] U. Fayyad and K. Irani, "On the handling of continuous-valued attributes in decision tree generation," *Machine Learning*, vol. 8, no. 1, pp. 87–102, 1992.

[23] G. A. Miller, E. Galanter, and K. H. Pribram, *Plans and the Structure of Behavior*. Holt, Rinehart and Winston, 1960.

[24] N. J. Nilsson, *Principles of Artificial Intelligence*. Tioga Publishing Company, 1980.

[25] M. Puterman, *Markov Decision Problems*. New York: Wiley, 1994.

[26] C. Boutilier, T. Dean, and S. Hanks, "Decision theoretic planning: Structural assumptions and computational leverage," *Journal of Artificial Intelligence Research*, vol. 11, no. 1, p. 94, 1999.

[27] J. Kuffner Jr and S. Lavalle, "RRT-connect: An efficient approach to single-query path planning," in *Proc. IEEE Int. Conf. Robot. Autom.(ICRA*, 2000, pp. 995–1001.

[28] R. S. Sutton, D. Precup, and S. Singh, "Between MDPs and semi-MDPs: A framework for temporal abstraction in reinforcement learning," *Artificial Intelligence*, vol. 112, no. 1-2, pp. 181–211, 1999.

[29] R. S. Sutton and A. G. Barto, *Reinforcement Learning*. Cambridge MA: MIT Press, 1998.

[30] X. Huang and J. Weng, "Novelty and Reinforcement Learning in the Value System of Developmental Robots," *Proc. 2nd Inter. Workshop on Epigenetic Robotics*, 2002.

[31] J. Marshall, D. Blank, and L. Meeden, "An emergent framework for self-motivation in developmental robotics," *Proc. of the 3rd Int. Conf. on Development and Learning (ICDL 2004)*, 2004.

[32] J. Schmidhuber, "Curious model-building control systems," in *Proc. Int. Joint Conf. on Neural Networks*, vol. 2, 1991, pp. 1458–1463.

[33] P. Oudeyer, F. Kaplan, and V. Hafner, "Intrinsic Motivation Systems for Autonomous Mental Development," *Evolutionary Computation, IEEE Transactions on*, vol. 11, no. 2, pp. 265–286, 2007.

[34] A. Bonarini, A. Lazaric, and M. Restelli, "Incremental Skill Acquisition for Self-Motivated Learning Animats," in *From Animals to Animats 9: 9th International Conference on Simulation of Adaptive Behavior, SAB*. Springer, 2006, pp. 357–368.

[35] M. Marjanovic, B. Scassellati, and M. Williamson, "Self-taught visually guided pointing for a humanoid robot," in *From Animals to Animats 4: Proc. Fourth Int l Conf. Simulation of Adaptive Behavior*, 1996, pp. 35–44.

[36] G. Metta and P. Fitzpatrick, "Early integration of vision and manipulation," *Adaptive Behavior*, vol. 11, no. 2, pp. 109–128, 2003.

[37] K. Gold and B. Scassellati, "Learning acceptable windows of contingency," *Connection Science*, vol. 18, no. 2, pp. 217–228, 2006.

[38] J. Klein, "Breve: a 3d environment for the simulation of decentralized systems and artificial life," in *Proc. of the Int. Conf. on Artificial Life*, 2003.

[39] R. Smith, *Open dynamics engine v 0.5 user guide*, http://ode.org/ode-latest-userguide.pdf., 2004.

[40] V. G. Payne and L. D. Isaacs, *Human Motor Development: A Lifespan Approach*. McGraw-Hill Humanities/Social Sciences/Languages, 2007.

[41] A. Needham, T. Barrett, and K. Peterman, "A pick-me-up for infants' exploratory skills: Early simulated experiences reaching for objects using 'sticky mittens' enhances young infants' object exploration skills," *Infant Behavior and Development*, vol. 25, no. 3, pp. 279–295, 2002.

[42] D. M. Pierce and B. J. Kuipers, "Map learning with uninterpreted sensors and effectors." *Artificial Intelligence*, vol. 92, pp. 169–227, 1997.

[43] R. O. Duda, P. E. Hart, and D. G. Stork, *Pattern Classification*. Wiley-Interscience Publication, 2000.

[44] A. Jonsson and A. Barto, "Causal graph based decomposition of factored MDPs," *The Journal of Machine Learning Research*, vol. 7, pp. 2259–2301, 2006.

[45] C. M. Vigorito and A. G. Barto, "Autonomous hierarchical skill acquisition in factored mdps," in *Yale Workshop on Adaptive and Learning Systems*, New Haven, Connecticut, 2008.

[46] W.-M. Shen, *Autonomous Learning from the Environment*. W. H. Freeman and Company, 1994.

[47] T. Dietterich, "The MAXQ method for hierarchical reinforcement learning," *ICML*, 1998.

[48] B. Digney, "Emergent hierarchical control structures: Learning reactive/hierarchical relationships in reinforcement environments," in *From animals to animats 4: proceedings of the Fourth International Conference on Simulation of Adaptive Behavior*. The MIT Press, 1996, p. 363.

[49] B. Hengst, "Discovering hierarchy in reinforcement learning with HEXQ," in *Proceedings of the Nineteenth International Conference on Machine Learning*, 2002, pp. 243–250.

[50] T. Hester and P. Stone, "Generalized model learning for reinforcement learning in factored domains," in *Proceedings of The 8th International Conference on Autonomous Agents and Multiagent Systems-Volume 2*. International Foundation for Autonomous Agents and Multiagent Systems, 2009, pp. 717–724.

Jonathan Mugan is a Post-Doctoral Fellow at Carnegie Mellon University. He received his Ph.D. in computer science from the University of Texas at Austin. He received his M.S. from the University of Texas at Dallas, and he received his M.B.A. and B.A. from Texas A&M University.

Benjamin Kuipers joined the University of Michigan in January 2009 as Professor of Computer Science and Engineering. Prior to that, he held an endowed Professorship in Computer Sciences at the University of Texas at Austin. He received his B.A. from Swarthmore College, and his Ph.D. from MIT.