Advances in Cognitive Systems 4 (2016) 93-111 Submitted 08/2015; published 06/2016

A Planning and Learning Hierarchy using Qualitative Reasoning for the On-Line Acquisition of Robotic Behaviors

Timothy Wiley TIMOTHYW@CSE.UNSW.EDU.AU
Claude Sammut CLAUDE@CSE.UNSW.EDU.AU
Bernhard Hengst BERNHARDH@CSE.UNSW.EDU.AU
School of Computer Science and Engineering, University of New South Wales, Australia

Ivan Bratko BRATKO@FRI.UNI-LJ.SI
Faculty of Computer and Information Science, University of Ljubljana, Slovenia

Abstract

Trial-and-error learning is often needed to acquire a new skill. Humans can use domain knowledge to minimize the number of trials required. However, existing reinforcement learning systems are either incapable of reasoning about domain knowledge or use hard-coded domain knowledge. Thus, these systems are insufficient for the online learning of robotic skills. We present a hierarchical architecture that learns the domain knowledge of a robotic system in the form of a qualitative model. The model is used by a symbolic planner that reduces the search space for trial-and-error learning. We evaluate the architecture on a real robot that learns to climb over obstacles.

1. Introduction

When humans learn a new skill they often employ a combination of reasoning using background knowledge and trial-and-error learning. For example, a tennis player learning how to serve may be told by the coach that the service action is similar to throwing a ball, that is, it is like throwing the racquet at the ball. This advice helps the player visualize the action, but practice is required to achieve an operational level of skill because there are many parameters to be tuned, such as the height of the ball toss and the speed of swinging the racket. However, without the coach's advice, the player may take much longer to find an acceptable action. Similarly, a robot acquiring a new behavior can benefit from high-level reasoning and background knowledge to reduce the search space of parameter settings required to execute the low-level actuator actions of the behavior.

Learning the parameter settings for a behavior is a complex task for robots because they often operate in large continuous domains, have noisy sensors, and imprecise actuators. Learning behaviors is commonly achieved by some form of reinforcement learning (see Section 9). However, most reinforcement learning systems assume that a simulation is available, allowing thousands of trials to be performed before an acceptable behavior is achieved. If we wish to acquire a skill online, as the robot operates, rather than learning in simulation, trial-and-error learning must be restricted to as few trials as possible because each trial is time consuming and could damage the robot. In this paper, we present a Planning and Learning Hierarchy (P/LH), inspired by human learning that is

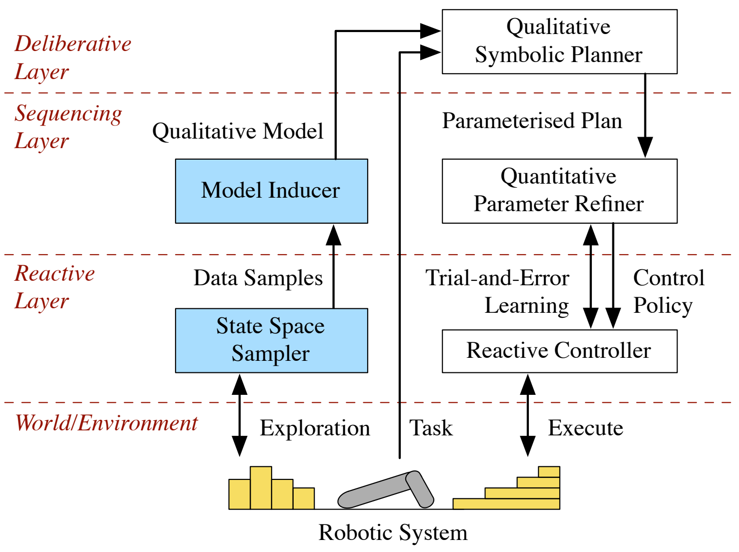


Figure 1. Overview of the Planning and Learning Hierarchy in relation to the three-layered architecture for robotic software. The left side learns the domain knowledge of a robotic system, while the right side acquires new robotic skills from domain knowledge.

intended to reduce the number of trials required to learn a skill. Figure 1 shows an overview of the P/LH, which extends earlier work in hybrid learning systems (Brown & Sammut, 2011; Sammut & Yik, 2010) that use high-level symbolic planning to reason about domain knowledge to constrain the search through the space of action parameters.

A planning hierarchy (right side of Figure 1) is provided with domain knowledge for the robot and its environment, along with a task that the robot must complete, for which a behavior is to be acquired. Domain knowledge is expressed as a qualitative model (de Kleer & Brown, 1984; Forbus, 1984) that describes the dynamics and possible actions of the robot within the environment. A *symbolic planner* uses the qualitative model to produce an rough plan to complete the task, but the parameters for each action are under-constrained. A *parameter refiner* narrows these constraints by online trial-and-error learning to produce a control policy for the acquired behavior. At the lowest level of the hierarchy, a *reactive controller* is required to execute both the refinement trials and the control policy on the robot.

The domain knowledge for P/LH may be programmed or learned by the robot. In the coaching example, above, the domain knowledge already exists and is passed on from teacher to student. However, it is often difficult to accurately model a robot and its interaction with its environment because motors and sensors may not perform to specification and parts of the environment may be unknown. Obtaining this knowledge usually involves a considerable amount of human learning and programming. A better solution is to give the robot the ability to acquire its own domain knowledge. Learning is performed by two additional components in the architecture (left side of Figure 1). The *state space sampler* makes the robot "play" by performing random actions and recording the actions and the variable values that describe the resulting states. From these samples,

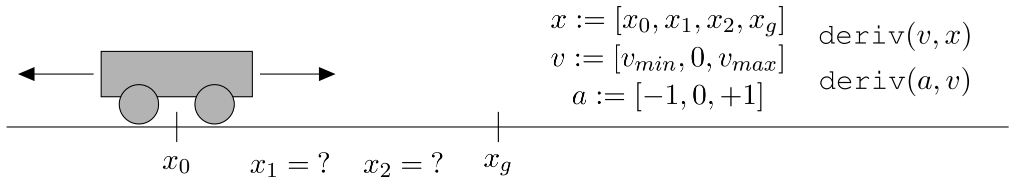


Figure 2. Domain knowledge for a 1D Cart, where the task is to move from x_0 to x_g by changing its acceleration (a). The value of intermediate points x_1 and x_2, where the cart's acceleration is changed, is unknown.

a model inducer constructs domain knowledge in the form of a qualitative model, which can be used to acquire behaviors for multiple tasks. Learning the domain knowledge is done offline, analogous to a human exploring a domain before attempting to perform any real tasks. Thus, model building is not part of the performance element of the system, but parameter tuning is embedded in it. To adjust the parameter values, the robot needs feedback that can only be obtained by attempting to execute a plan and changing the control policy if it fails. The robot may stop learning once a working policy found or it can continue to tune parameters to improve its performance.

The P/LH reduces the number of trials required by the parameter refinement stage to achieve a working policy. To demonstrate its efficiency, the P/LH is applied to a complex locomotion task for the Negotiator multi-tracked robot (Figure 4a), intended for urban search and rescue. All of the experiments described below were conducted on a real robot.

2. The Planning and Learning Hierarchy

A common architecture for robot software, such as 3T (Bonasso et al., 1997) or ATLANTIS (Gat, 1998), consists of three levels: (1) the upper deliberative layer is responsible for long-term planning; (2) the intermediate sequencing layer selects parameters required by the motor commands that execute the plan; and (3) the lowest reactive layer implements direct control the robot's actuators and sensors. The intention of the P/LH (Figure 1) is to improve the efficiency of acquiring complex behaviors by lifting much of the learning from the reactive layer to the deliberative layer.

2.1 Planning and Parameter Refinement

The symbolic planner combines concepts from classical planning and qualitative reasoning, specifically the QSIM system (Kuipers, 1986). An action model includes preconditions and postconditions similar to those in STRIPS (Fikes & Nilsson, 1971), but is extended by the addition of parameters that set numerical values for the operation of actuators (Sammut & Yik, 2010). Unlike STRIPS, the post-condition is computed by QSIM, when it predicts the qualitative outcome of operating an actuator. These postconditions impose constraints on the action's parameters. Like a classical planner, action models are used to find a sequence of actions that achieve a specified goal. However, the actions are qualitative and thus do not specify precise values for actuators, like desired joint angles or speeds. Thus, each action is only an approximation of the precise movements the robot must per-

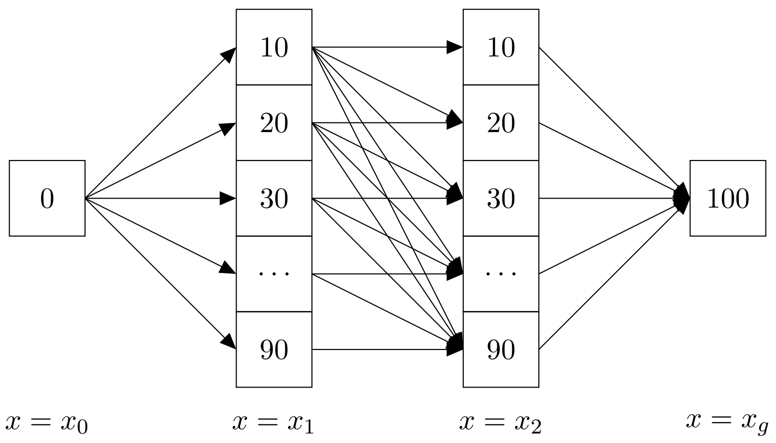


Figure 3. Representation of the SMDP for the 1D cart task. Not all combinations of options may lead to a successful completion of the task. Each arrow represents a single option that targets a possible parameter value for one action of the plan in subsection 2.1.

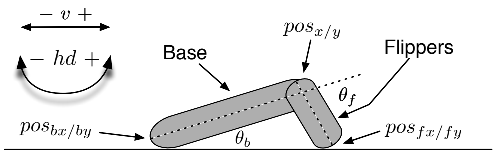
form. The constraints produced by QSIM provide bounds on the possible values of the parameters, but these bounds may be too broad to guarantee the correct execution of the action.

As an example, consider the 1D cart in Figure 2, whose state is represented by three variables: the position of the cart, x, its velocity, v, and its acceleration, a. The task is to move the cart from its initial position, x_0, to a goal position, x_g. The domain knowledge specifies that the cart's velocity is bounded by minimal and maximal values, and that the cart is controlled by setting a value for acceleration: *decelerating* ($a = -1$), *coasting* ($a = 0$), or *accelerating* ($a = +1$). The qualitative dynamics of the cart state that acceleration is the derivative of velocity, and that the velocity is the derivative of the x-position, both with respect to time. The planner may devise a three-action plan: (1) accelerate from $x = x_0$ to a point $x = x_1$; (2) coast to a second point $x = x_2$; (3) decelerate and stop at the goal $x = x_g$. Each action changes the cart's acceleration and is durative. For example, action (1) starts in position x_0 and terminates when position x_1 is reached. Thus, the actions are parameterized by the x position at which they terminate. However, qualitatively it is only known that $x_1 > x_2$ and that both x_1 and x_2 are in the range $x_0 \ldots x_g$ between the initial and goal positions. Obviously, there are combinations of values for x_1 and x_2 that make the cart to stop at the goal, but there are many combinations that will fail.

The *parameter refiner* narrows the parameter constraints to a *satisficing*[1] region that will always lead to successful completion of the task. The constraints may be further refined if an optimal solution is required. Refinement of the parameter constraints is formalized as a semi-Markov decision problem (SMDP) using options (Sutton et al., 1999). Options perform the sequencing layer function of mapping the coarse qualitative constraints into specific quantitative values required for a motor command. For each action generated by the planner, there is a set of options, where each option represents a single point drawn from the quantitative parameter space of that action. That is, one option corresponds to one possible implementation of an action. The job of the refiner is to search

1. *Satisficing*, coined by Simon (1956), is defined as searching until an acceptable solution is found.

(a)



(b)

Figure 4. The Negotiator robot (a) attempting a USAR task and (b) the qualitative variables required used to model the robot.

for satisficing (or optimal) combinations of options that result in successful completion of the task. Returning to the 1D cart example, recall that the action, setting the acceleration of the cart, is parameterized by the x position at which the action terminates. The domain knowledge assigns qualitative labels to the x landmark values, but in the "real world", these labels correspond to specific quantitative values, such as $x_0 = 0$ and $x_g = 100$. Figure 3 illustrates the SMDP and options for the three-action plan for the cart, where, for this example, we assume that the numerical values for each parameter have been discretized. Selecting $x_1 = 10$ and $x_2 = 90$ as the combination of options results is a successful completion of the plan, while selecting $x_1 = 20$ and $x_2 = 20$ will fail, and selecting $x_1 = 50$ and $x_2 = 50$ gives optimal execution. By trial and error, combinations of options are evaluated against a reward. For a satisficing solution, options received a positive reward if they lead to successful achievement of the goal, or a negative reward otherwise. For an optimal solution, an immediate reward which represents properties such as execution time is assigned to each option, and standard SMDP mechanics calculate a globally optimal combination of options.

Once an action has been mapped to a set of motor commands, these are executed by the *reactive controller*. In our experiments, the reactive controller is implemented is a collection of PID control loops to achieve the target values given by the numerical action parameters.

2.2 Learning the Domain Knowledge

In the P/LH, learning occurs in two different modules using two different techniques. In the previous section, we gave an overview of learning constraints on the parameters values to map a qualitative action into a set of numerical motor commands. The second learning module constructs the qualitative background knowledge by observing the interaction of the robot with its environment.

During *state-space sampling*, the robot's state is recorded as actuators are randomly operated in order to explore the robot's dynamics and interaction with the environment. The Padé algorithm (Žabkar et al., 2011) applies a localized regression, called *tubed regression*, to label each sample in the collected data set with the local qualitative dynamics of the robot. For example, each sample may be labelled by the local qualitative derivative: increasing, decreasing, or steady. The labelled data set is segmented into piecewise monotonic subsets that are input to a general-purpose

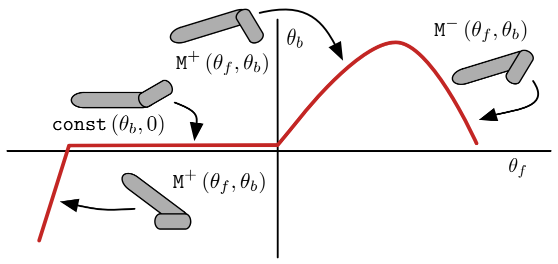

θ_b
θ_f
$\mathtt{M}^{-}(\theta_f, \theta_b)$
$\mathtt{M}^{+}(\theta_f, \theta_b)$
$\mathtt{const}(\theta_b, 0)$
$\mathtt{M}^{+}(\theta_f, \theta_b)$

Figure 5. For the Negotiator, the relationship between the angle of the base (θ_b) and the angle of the flippers (θ_f). The relationship is defined by four distinct regions, using four $Rules$.

learning system that induces piecewise qualitative models of the robot's dynamics. The piecewise models are combined to generate the full qualitative model of the domain knowledge of the robot.

The P/LH is applied to a locomotion task for the iRobot Negotiator, shown in Figure 4. This robot is typical of those used in urban search and rescue. The Negotiator has two main tracks plus a pair of articulated subtracks, or "flippers", that let the robot change its geometry and maneuver over obstacles. Our aim is to efficiently learn control policies, on board the robot, for locomotion tasks often encountered in urban rescue, such as climbing steps or staircases and traversing loose rubble.

3. Domain Knowledge as a Qualitative Model

The robot's domain knowledge, including its interaction with the environment is described by a *qualitative model*. Our representation (Wiley et al., 2013a) extends the Kuipers (1984) framework for causal reasoning, which qualitatively describes the state and dynamics of a system. We divide QSIM variables into two categories. A *control variable* is associated with a motor command and a *change* in a control variable corresponds to an action. For example, changing the acceleration in the 1D cart is an action. A *state variable* is any other variable required to uniquely described the state of the system. On the Negotiator, the control variables include the robot's heading, hd, the velocity, v, and the position of the flippers, θ_f. The state variables include the position of the robot, pos_x/pos_y, and the angle of its base relative to the ground, θ_b.

The qualitative value of a variable is described by a magnitude and rate of change over time. The magnitude of a variable is given relative to symbolic *landmark* values in the variable's domain, and may be either at a landmark or in the interval between two consecutive landmarks. The rate of change of a variable over time is either increasing (`inc`), steady (`std`), or decreasing (`dec`). For example, if the Negotiator is moving towards a step, the x position of the robot (pos_x) may be described as increasing between the robot's starting position and the position of the step. Thus, the value of a variable is given by a pair, which can take one of three forms:

$$l_i \ldots l_{i+1}/\mathtt{dec} \qquad l_i/\mathtt{std} \qquad l_i \ldots l_{i+1}/\mathtt{inc}$$

A *qualitative state* is an assignment of a qualitative value to each variable of the system and the *qualitative state space* is the set of all such assignments.

Table 1. Common QDE constraints.

QDE	Description
$\mathtt{M}^+(x,y)$	Monotonicity, i.e., the directions of change x and y are the same (both positive or both negative).
$\mathtt{M}^-(x,y)$	Inverse monotonicity, i.e., x and y change in opposite directions.
$\mathtt{sum}(x,y,z)$	$z = x + y$
$\mathtt{deriv}(x,y)$	y is the derivative of x with respect to time.
$\mathtt{const}(x,k)$	$x = k/std$, i.e., the value of x is k and remains steady
`qnull`	The qualitative state is invalid.

The dynamics of the robot are described by a set of rules that determine if a qualitative state is "legal", that is, whether the robot can physically achieve the state. A rule consists of a guard and a qualitative constraint:

$$Rule := Guard \Rightarrow Constraint$$

If a qualitative state satisfies the guard then the constraint of the rule is applied. The guard is a predicate that describes an *operating region* (Williams, 1984). For example, the relationship between the angle of the Negotiator's base, θ_b, and the angle of its flippers, θ_f, varies through four discontinuous regions, as shown in Figure 5, requiring piecewise models of the system's dynamics (Nishida & Doshita, 1987). Thus, a qualitative state is legal if, and only if, that state is contained within an operating region that has a rule whose constraints are satisfied.

Constraints are specified as *qualitative differential equations* (QDEs). Table 1 lists the common QDEs, including `const` and `qnull`, introduced by Wiley et al. (2013b). QDEs place restriction on the qualitative values of pairs or triplets of variables. For example, $\mathtt{M}^+(x,y)$ states that the rate of change of variables x and y, must be the same, that is, if x is increasing then y must also be increasing. The planning module of the P/LH uses the qualitative domain knowledge to find a sequence of actions to achieve a specified goal.

4. The Qualitative Planner

A plan is a sequence of actions, $a_i^{\mathbb{Q}}$, that transforms the initial state, $s_0^{\mathbb{Q}}$, into successive states until the goal state, $s_g^{\mathbb{Q}}$, is reached. The superscript, $\mathbb{Q}$, is used to remind the reader that these states are qualitative, i.e., they are represented by a set of qualitative values.

$$s_0^{\mathbb{Q}} \xrightarrow{a_0^{\mathbb{Q}}} s_1^{\mathbb{Q}} \xrightarrow{a_1^{\mathbb{Q}}} s_2^{\mathbb{Q}} \rightarrow \ldots \xrightarrow{a_n^{\mathbb{Q}}} s_g^{\mathbb{Q}}$$

An *action model*, shown in Figure 2, consists of an identifier, a precondition, a postcondition (or effect), an implementation, and a set of parameter variables. The precondition specifies the qualitative states in which the action may be selected. The postcondition is a set of qualitative states, any of which may result from the action. The implementation details how the action is to be executed by setting a qualitative value for every control variable. Actions may set several control variables simultaneously, causing multiple actuators to operate concurrently. The parameter list of an action is the variables whose values govern how the action is executed on the robot.

Table 2. A qualitative action used for symbolic planning, where each $s_i^{\mathbb{Q}}$ is a "legal" qualitative state, each $cvar_i^{\mathbb{Q}}$ is a control variable with a qualitative value defined by a magnitude (`mag`) and rate of change (`roc`), and each var_i is a variable of the system.

Identifier:	`name`
Pre-condition:	$\left\{s_i^{\mathbb{Q}}, s_j^{\mathbb{Q}}, \ldots\right\}$
Post-condition:	$\left\{s_k^{\mathbb{Q}}, s_l^{\mathbb{Q}}, \ldots\right\}$
Implementation:	$cvar_i^{\mathbb{Q}} = \mathtt{mag}/\mathtt{roc}, \ldots$
Parameters:	$var_i, var_j \ldots$

Action models are nondeterministic, since they allow a transition into one of several possible new states. The nondeterminism occurs because, unlike classical planning, the set of possible post-conditions resulting from the execution of an action is computed by QSIM from the precondition and implementation. For example, with the 1D-cart from Figure 2 an action's precondition may require that the cart is coasting between the initial and goal positions ($x = x_0...x_g/\mathtt{inc}, v = 0..v_{max}/\mathtt{std}, a = 0/\mathtt{std}$), with "decelerate" ($a = -1/\mathtt{std}$) listed as the action's implementation. From this QSIM predicts two possible successor states: either the cart stops at the goal ($x = x_g/\mathtt{std}$) or before it ($x = x_0...x_g/\mathtt{std}$), but it cannot decide which will occur. The use of parameterized actions addresses this nondeterminism. We make the assumption that, with the correct choice of parameter values for each action, any of the states QSIM predicts can be successfully targeted. In the 1D cart, by switching from coasting to decelerating at the appropriate x position, either of the above outcomes can be targeted deterministically.

The planner uses forward chaining to search for a sequence of actions that will achieve the goal. However, for a complex system, the number of actions and the size of the qualitative state space renders a naive search strategy intractable. We have proposed two solutions in previous work. In Wiley et al. (2013a), we introduced a heuristic termed the *qualitative magnitude distance*. The heuristic estimates, from any qualitative state, the minimum number of actions required to reach the goal using the difference in magnitudes of the qualitative values of variables in the respective qualitative states. This planner, implemented in Prolog, combines this heuristic with an A* search and assumes that the best plan has the smallest number of actions. The original formulation of QSIM does not have the concept of an action model. Instead, the QSIM constraint solver generates a sequence of qualitative states. For planning, action models are generated on-the-fly, during the search, derived from the qualitative state transitions. In Wiley et al. (2014b), we observed that QSIM's calculation of the postconditions of an action can be formulated as a constraint satisfaction problem, which can be solved efficiently using answer set programming (ASP; Gebser et al. 2013). In this case, the planner is directed to find the shortest sequence of actions. Planning with both the Prolog implementation and ASP is tractable. However, the ASP planner is considerably faster, provided that continuous variables can be discretized into a small number of bins. If this is not possible, the Prolog planner, which is more general, must be used at the cost of speed. Earlier publications (Wiley et al. 2014a; 2014b) have compared and evaluated these two alternatives.

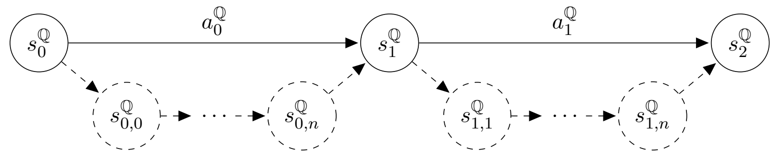


Figure 6. Qualitative planning with funnel states. The top row, with solid arrows, is a plan between funnel states. Each action generalizes across a sequence of qualitative states.

To further minimize the impact of QSIM's nondeterminism, the planners incorporate *funnel states* (Brown & Sammut, 2011), which are critical regions in the robot's state space that it must pass through to complete a task, as illustrated by Figure 6. From the initial state, $s_0^{\mathbb{Q}}$, the system must pass through funnel states $s_1^{\mathbb{Q}}$ and $s_2^{\mathbb{Q}}$ before reaching the goal $s_g^{\mathbb{Q}}$. However, we noted earlier that actions are durative. Thus, the system may pass through several intermediate states, and there may be more than one path from one funnel state to the next. The exact path taken by the robot to reach the funnel state is unimportant, provided the it is reached. Funnel states buttress our assumption that a desired qualitative state may always be reached, with the correct parameter setting, as multiple ways of reaching a funnel state gives greater flexibility.

The output of the planner is a list of actions that should achieve the agent's goal, but this list can be interpreted as a collection of sequential constraints on the control policy, which is found by the parameter refiner described in the following section.

5. Parameter Refinement

The parameter refiner narrows the constraints on the action parameters of a plan by online trial-and-error learning. Refinement is treated as a semi-Markov decision problem (SMDP) over options (Sutton et al., 1999), mapping the qualitative state space of the deliberative layer of the P/LH into the *quantitative state space* of the reactive layer. An option represents a specific way of executing a single action of a plan on the robot. As we saw in Section 2 with the 1D cart, the qualitative range to which QSIM restricts an action's parameters is converted to a quantitative range using the known values of each variable's landmarks, from which the range is discretized. One option is constructed for each parameter value in the discretized range. Options are chained to construct an SMDP, leading from a quantitative initial state to a quantitative goal state. Executing an option on the robot involves starting in a specific quantitative state and attempting to reach a goal state, defined by the parameters of the related action. However, it may not be possible for the robot to reach the goal, nor may the option represent the optimal approach to completing the task. Therefore, refinement of a task is reduced to finding "good" sequences of options, where "good" may be defined as any option that reaches the goal (*satisficing*) or as the best sequence.

Standard SMDP mechanisms measure the quality of each option for completing the task. An immediate reward is received for each option, from which standard MDP backpropagated value iteration (Sutton & Barto, 1998) calculates an expected delayed reward. For a satisficing refinement, options are assigned an immediate reward of success $(+1)$ or failure (-1), depending on whether the robot can successfully execute the option, and the expected reward copies the maximum of successor options. Thus, the satisficing options are ones that, at the completion of learning, have success $(+1)$ for their delayed reward. For optimality, the immediate reward is some measure according to the optimality criterion. For example, to optimize for speed, the immediate reward might be the length of time the robot takes to execute the option, and the expected rewards accumulate times from successive options.

Immediate rewards are acquired by online experimentation. To conduct a trial, the robot chooses a sequence of options from the initial state to the goal and then attempts to execute them. As each option is completed, immediate rewards are assigned according to the type of refinement (satisficing or optimal). The trial concludes when the robot either reaches the goal or fails to execute an option. The sequence of options chosen for each trial depends on the type of refinement. For satisficing refinement, the robot selects a trial that maximizes the number of options that have not been tested, and trials continue testing all options that may lead to a successful completion of the task. For optimal refinement, we employ the MCMC hill-climbing approach of Sammut and Yik (2010), which carries out random trials until it finds a successful sequence of options. For subsequent trials, the robot chooses a sequence of options that is spatially similar to the current optimal solution, conducting trials until no untested spatially similar options exist.

6. Learning a Qualitative Model

The domain knowledge used by the P/LH may be programmed or learned. The first stage in learning is to collect training data by sampling the *quantitative* state of the system as the robot operates its actuators and interacts with the environment. The second stage involves inducing a qualitative model from the training data.

The *model inducer* is built on Padé (Žabkar et al., 2011), a tool for learning qualitative models from numeric data. Padé induces a qualitative function, $y = f(\mathbf{x})$, of a single value, y, with respect to the argument, $\mathbf{x}$. First, each sample of the training data is labelled with the local qualitative behavior (increasing, decreasing, or steady) of the target function, $f(\mathbf{x})$. Next, a general-purpose machine learning algorithm induces a classifier from these training data, creating the target function. For the P/LH, we want to learn the effect of changing a control variable on the state of the robot. That is, we want to learn

$$\mathbf{svars} = f(\mathbf{cvars}),$$

where $\mathbf{svars}$ is the set of state variables and $\mathbf{cvars}$ is the set of control variables. Padé can only induce a function of one variable. To learn a multi-variable function it must be decomposed into several functions of <state variable / control variable> pairs. Padé induces functions for each pair, which are combined to form the complete qualitative model.

We have extended Padé to improve its ability to induce models of robot systems. Training data sampled from a robot often suffer from severe sampling bias, which is problematic for the labelling stage. To reduce this bias, the data set is discretized and each bin is resampled so that every bin contains an equal number of training instances. Another problem is that the training data only contain positive samples, that is, samples from regions of the robot's state space that were visited during sampling. Padé will overgeneralize across the unvisited regions. However, these regions may indicate parts of the state space that the robot cannot reach. To correct for this, random `qnull`

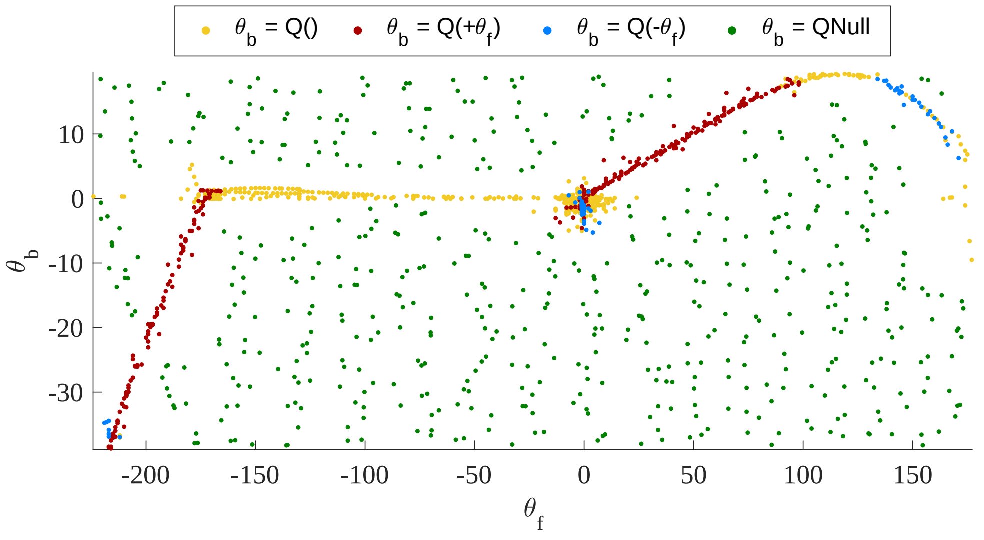


Figure 7. Labelled data set produced by Padé for the relation between the angle of the flipper (θ_f) and the angle of the base (θ_b) on Negotiator. Padé's labelling mirrors the theoretical relationship shown in Figure 5. The figure also shows the seeded `qnull` samples.

samples from the unvisited regions (see Table 1) are added to the data set as negative examples, to indicate that regions not represented in the original sample are invalid. The revised training data are then used to induce a pairwise model. The assumption that unvisited regions are invalid may result in generalisations that are too conservative, but otherwise learning from positive only data would be difficult. Consider, for example, inducing a pairwise model for the relationship between the angle of the flipper (θ_f) and the angle of the base (θ_b) on the Negotiator. Figure 5 describes the theoretical relationship between the two angles and Figure 7 shows labeled training data. The raw training data were collected from observations of the flipper-base relation. These were then evenly redistributed and passed to Padé, which labelled the samples $Q(+\theta_f)$, $Q(-\theta_f)$, or $Q()$. These state that, for the specific sample of the training data, the value of θ_f is increasing in relation to θ_b, decreasing in relation to θ_b, or unrelated to θ_b, respectively. Additional `qnull` instances were randomly added as artificial negative examples.

Now that the robot has a labelled training set, any symbolic induction algorithm may be used to construct a mapping between a qualitative state and a qualitative action. As suggested by Žabkar et al. (2011), the system uses C4.5 (Quinlan, 1993) to construct a decision tree. Figure 8 shows the decision tree that is induced for the Negotiator's flipper-base relationship using the labelled data shown in Figure 7. A decision tree can be trivially converted into rules, forming a qualitative model of the relationship. The guard for a rule is derived by combining the conditions leading to a leaf node and the qualitative behaviors of the leaf nodes become monotonic QDEs using the forms in Table 1. For example, the qualitative behavior $\theta_b = Q(+\theta_f)$ equates to $\mathtt{M}^+(\theta_b, \theta_f)$, and $\theta_b = Q(\mathtt{qnull})$ reduces to the `qnull`. The main limitation of using Padé is that this conversion cannot generate

(a)

```
theta_b<=-5.072
|    theta_f<=-178.415: Q(+theta_f)
|    theta_f>-178.415: Q(null)
theta_b>-5.072
|    theta_b<=1.628
|    |    theta_f>17.279: Q(null)
|    |    theta_f<=17.279
|    |    |    theta_b<=-2.772: Q(+theta_f)
|    |    |    theta_b>-2.772: Q()
|    theta_b>1.628
|    |    theta_f<=133.524
|    |    |    theta_b>18.704: Q()
|    |    |    theta_b<=18.704
|    |    |    |    theta_f<=-10.273: Q(null)
|    |    |    |    theta_f>-10.273
|    |    |    |    |    theta_f<=98.500: Q(+theta_f)
|    |    |    |    |    theta_f>98.500: Q(null)
|    |    theta_f>133.524
|    |    |    theta_f<=154.558: Q(-theta_f)
|    |    |    theta_f>154.558: Q()
```

(b)

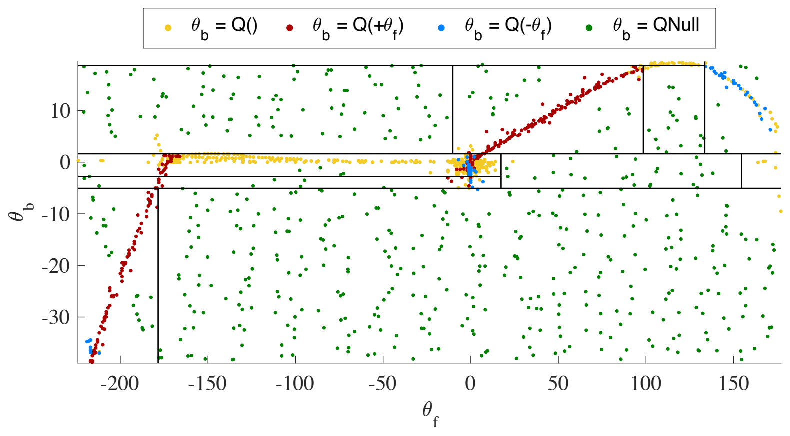


Figure 8. A text description of the decision tree is given in (b) which is induced from training examples in Figure 7, which is then overlaid in (b) on the plot of the training examples.

rules that involve derivatives or arithmetic. Finally, the complete qualitative model is the union of all pairwise state-control variables models.

7. The Planning and Learning Hierarchy (P/LH) on a Multi-tracked Robot

We have evaluated the P/LH using the multi-tracked Negotiator robot (Figure 4) to complete two terrain traversal tasks: climbing a single large obstacle, such as a step, and climbing a 45-degree staircase. The Negotiator may climb a step using one of two methods (Figure 9). The obvious approach is to raise the flippers to approximately 45-degrees before the robot drives forward over

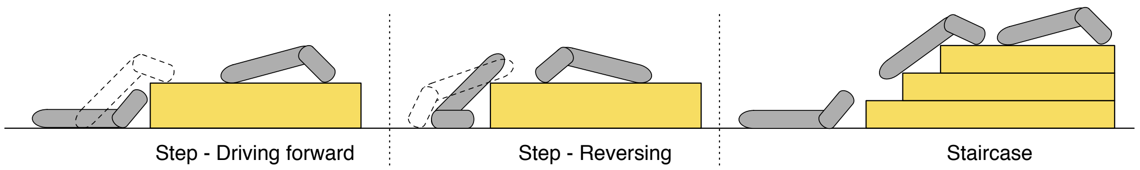


Figure 9. Depiction of Negotiator performing the two terrain traversal tasks: climbing a step and climbing a staircase. The step-climbing task may be solved by one of the two methods: driving forward and reversing.

the step. A more complex method is needed if the step is too high for the flippers to get traction. The robot must execute a 180-degree turn and reverse up to the step. Then by supporting its weight on the flippers, the robot raises the base and places it on the step. The flippers are rotated to provide enough leverage so that the robot may continue reversing onto the step. This approach may seem counterintuitive, but getting the longer base onto the step lets the robot climb onto higher obstacles. Climbing a staircase (Figure 9) shares similarities with the step task. The robot first raises its flippers to drive up and onto the stairs. However, the flippers must be lowered to gain enough traction to drive up the staircase. This additional traction becomes more important as the pitch of the staircase increases. The staircase also introduces a safety concern. As the robot reaches the top, it may fall onto the landing at the top of the staircase, potentially damaging computers, sensors, or motors. The fall can be prevented by lowering the flippers at the top of the staircase.

To apply the P/LH to solve both terrain traversal tasks, a model of the staircase and the step may be learned or programmed. The experiments presented here used handcrafted qualitative models for both tasks. The planner was provided with the qualitative models and tasked with finding three plans: one with the qualitative model configured for a low step, one for a high step, and one for climbing a staircase. Previous papers (Wiley et al., 2013a; 2014a; 2014b) have described theoretical and experimental results with the planner.

Table 3. Performance of the planner for terrain traversal tasks.

Task	No. Actions	No. Options
Low step	4	442
High step	10	2904
Staircase	9	799

Table 3 lists the number of actions in the plan and the number of options in the SMDP used for refinement. The low step requires the shortest plan, which consists of only four actions and 15% of the number of options compared to the substantially more complex plan for the high step. Much of the complexity in the high step plan is due to the coordination of the movements of Negotiator's flippers and the location of the robot relative to the step. In this task, the robot is restricted to operating either the flippers or the drive train, which lengthens the plan and increases the number of options. The flippers and drive train can be operated in parallel for the other two tasks. The staircase task has more actions than simply climbing one step because the robot must rotate the flippers to gain traction and to prevent it from being damaged.

Table 4. Number of trials required to find a desired set of parameters for each terrain traversal task.

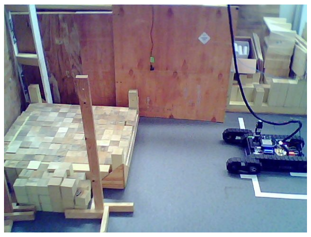
Task	Satisficing			Optimisation		
				Time	Safety	Both
	average	minimum	maximum	average	average	average
Low step	3	1	11	30	N/A	N/A
High step	17	1	45	51	N/A	N/A
Staircase	14	7	36	37	53	60

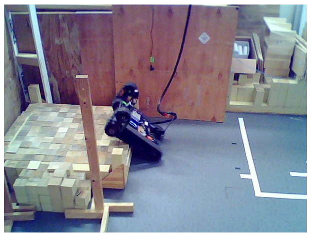
Table 4 summarizes the number of online trials that the parameter refiner requires to find a satisficing control policy for each task and to find control policies for various optimization tasks. Some snapshots from these trials are shown in Figure 10. Trials for a satisficing solution were constructed by randomly choosing options that had not previously been tested, until a satisficing set of options was found. Optimization trials were constructed using a hill-climbing method, with trials being conducted until the optimal set of options did not change between consecutive trials.

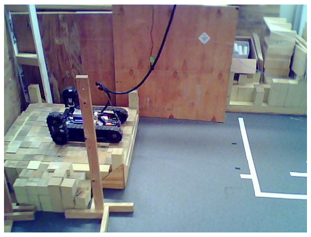
Although all three tasks may appear simple, each plan has critical points where incorrect parameters cause it to fail. For the low step, if the flipper position is too low or too high (Figure 10j), the robot cannot to get onto the leading edge of the step. This is also the case for the start of the staircase plan (Figure 10k). Choosing the wrong flipper values at the top of the staircase may cause the robot to topple backwards down the stairs or to fall heavily onto the landing (Figure 10l). The high step has many failure scenarios if the combination of parameters for operating the drive train and configuring the flippers is chosen poorly.

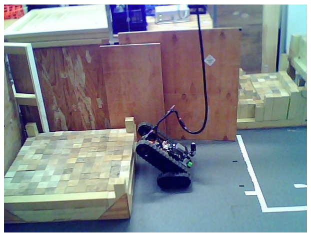
For all three tasks, we were primarily concerned with finding a single, reliable *satisficing* solution. That is, we desired a set of parameter values that guarantees the trained reactive controller will let the robot successfully complete the task every time it is attempted. Thus, for the satisficing refinement, trials were run until the system found a sequence of options that always completed the task. We are also interested in finding *optimal* solutions for each task, where optimality may be defined in various ways. One obvious measure for optimality is *time*, that is, the fastest execution of the plan by the robot. Risk of damage to the robot, *safety* is another measure, so we want a plan that minimizes that risk. Options that find a balance between time and safety may also be criteria for optimization. For step climbing, only the length of time to execute a plan is optimized. This is because these tasks have a low chance of the robot falling from a great height. However, as there is a high risk of falling from staircase, the robot's reward weights safety more heavily than time.

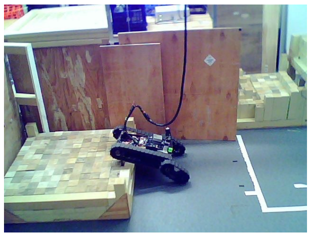
A significant claim of our work is that the use of the P/LH enables reactive controllers to be feasibly trained online, that is, on board the robot as it is executing. These results demonstrate that feasibility, especially where only a satisficing solution is required. The real time required to find an optimized control policy for the high step and staircase scenarios is between three to four hours. This includes time for resetting the robot between each trial and for minor running repairs. Where an only satisficing solution is required, no more than 30 minutes is required to find a control policy for the the physical robot.

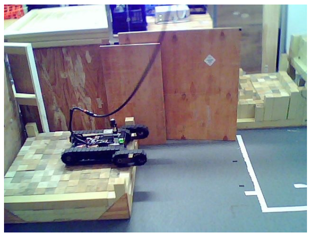

(a) Step - Approach 1

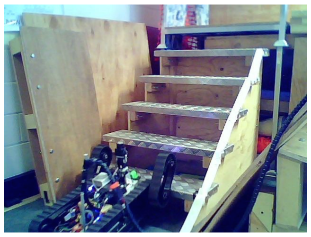

(b)

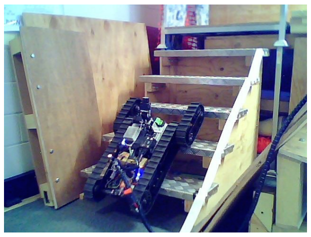

(c)

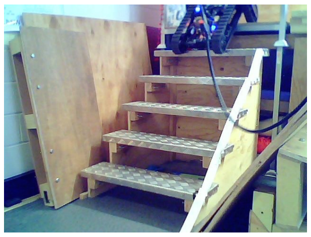

(d) Step - Approach 2

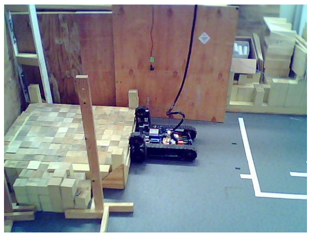

(e)

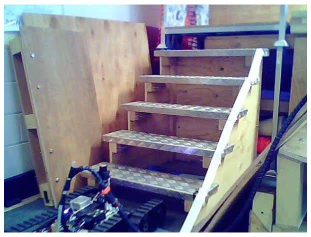

(f)


(g) Staircase


(h)


(i)


(j) Failing on Step


(k) Failing on Staircase

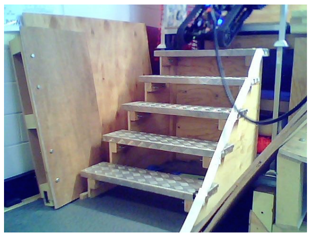

(l) Dangerous fall on Staircase

Figure 10. Various stages of trials for the three terrain traversal tasks. Figures (a) – (c) show the step climbing task when driving forward, Figures (d) – (f) the step climbing task when reversing, and Figures (g) – (i) the staircase climbing task. Figures (j) – (l) show trials which may fail or be unsafe.

8. Related Work

Several cognitive architectures incorporate high-level planning with reactive control layers. For example, RCS (Albus & Barbera, 2005) provides a detailed framework for constructing complex autonomous systems. Laird et al. (2012) describe extensions to the SOAR architecture that let it invoke SLAM and navigation systems. Ferrein & Lakemeyer (2008) introduce an architecture that includes a deliberative layer, programmed in a logical action language, on top of a reactive control layer. The main distinction of our work is that it extends qualitative simulation so that it can be used in planning. The planner incorporates a constraint solver that combines semi-numerical reasoning with logical inference.

Other architectures combine task planning with motion planning or trial-and-error learning. For example, Tenorth et al. (2014) use a PDDL planner for cooking a pancake and Lagriffoul et al. (2014) use a similar technique for pick-and-place tasks. Each action has a parameterized motion description that a motion planner uses to generate the actuator movements executed on the robot. Dynamic motion primitives (DMPs; Schaal et al. 2005) are parameterized mathematical models for complex motor actions. Common or repeated actions are stored in libraries and can be combined to construct more complex behaviors. The parameters of the DMPs can be refined by reinforcement learning to train robot arms to play table tennis and shoot an ice hockey puck (Neumann et al., 2014), or to throw darts and flip pancakes (Kober et al., 2011). However, these methods rely on time-consuming simulations to construct the Dynamic motion primitives and they cannot reason using domain knowledge. In contrast, our P/LH is well suited to online learning on noisy mobile robots that are difficult to simulate.

Reinforcement learning is often used to acquire new behaviors, such as visual UAV navigation (El-Fakdi & Carreras, 2013) and teaching a dog-like robot to jump (Theodorou et al., 2010). These also use handcrafted simulators for initial learning, since the number of trials required increases rapidly as the complexity of the domain increases. The learned policies are refined in a second stage of online learning. Imitation Learning, or behavioral cloning, has been used to learn control policies using training data obtained by observing the actions of a human expert (Michie et al., 1990), with applications to tasks such as plotting a simulated aircraft (Sammut et al., 1992; Šuc et al., 2004; Ng et al., 2006) and controlling container crane (Šuc & Bratko, 1999). Behavioral cloning is complementary to the methods presented here, as the performance traces provided by the human experts can be used as training data for learning at both the deliberative and control layers.

9. Conclusions and Future Work

The normal engineering approach to building robot behaviors is to carefully model the robot and its environment, and to hand craft control functions to achieve specified goals. However, it becomes increasingly difficult to construct these models and behaviors as we build more complex robots and expect them to perform more difficult tasks. Rather than handcrafting them, it is less laborious for the robot to learn a model of itself and how its actions affect the environment. The Planning and Learning Hierarchy (P/LH), described here, acquires its models from examples of the robot's interaction with objects. The examples can be provided as traces from a human operator controlling the robot or from the robot's own experimentation. To reduce complexity, the robot learns in two stages. First, the robot builds a qualitative model that lets it reason about actions and create a high level plan to achieve a goal. Initially, the plan only specify qualitative actions, lacking the precise numerical values required for motor commands. However, the actions provide constraints so that the second stage of learning can find operational parameter values efficiently. We formulate the second stage of learning as a semi-Markov decision problem, since the robot must perform some trial and error to obtain enough information to find those parameters.

All of the components of P/LH are implemented and have been tested individually. The experimental results presented in this paper are from those isolated tests. We are currently conducting experiments to demonstrate the complete learning cycle, starting with building qualitative models from the robot "playing" in its environment through planning to achieve a specified goal and refining the action models through online learning during the performance of the task. One motivation for this work is to be able learn in a small number of trials on a real robot. We wanted this because, as stated above, it is difficult to build accurate simulations. Thus, the experiments used a real tracked rescue robot with both learned and handcrafted qualitative models.

A major advantage of symbolic representations over numerical ones is that the former are easier to reason about and to generalize. It should be possible to generalize the qualitative models that the robot builds for one problem and transfer them to another. A state space may contain many regions that share similarities, as do the resulting qualitative models for those regions. For example, obstacles often have regions of flat ground before and after, so whether the obstacle is a staircase or a single block, the robot should be able to execute similar actions leading to and away from the obstacle. Automatically identifying and reusing common regions should reduce the need to induce new qualitative models and let existing plans be reused. In ongoing work, we are adapting methods from inductive logic programming (Muggleton et al., 2014) to generalize qualitative models to be stored in libraries, along with associated control policies. If successful, this will take robots some way toward the ability of reasoning in a new task its way to a workable solution, as do humans.

Acknowledgements

We thank Jure Žabkar from the University of Ljubljana for his help in using the Padé software. We thank Torsten Schaub and Max Ostrowski from the University of Potsdam for their assistance with the Clingo-4 ASP solver, which are used in our symbolic planner.

References

Albus, J. S., & Barbera, A. J. (2005). RCS: A cognitive architecture for intelligent multi-agent systems. *Annual Reviews in Control*, *29*, 87–99.

Bonasso, R. P., Firby, R. J., Gat, E., Kortenkamp, D., Miller, P. D., & Slack, M. G. (1997). Experiences with an architecture for intelligent, reactive agents. *Journal of Experimental & Theoretical Artificial Intelligence*, *9*, 237–256.

Brown, S., & Sammut, C. (2011). Learning tool use in robots. *Advances in Cognitive Systems: Papers from the AAAI Fall Symposium* (pp. 58–65). Menlo Park, CA: AAAI Press.

El-Fakdi, A., & Carreras, M. (2013). Two-step gradient-based reinforcement learning for underwater robotics behavior learning. *Robotics and Autonomous Systems*, *61*, 271–282.

Ferrein, A., & Lakemeyer, G. (2008). Logic-based robot control in highly dynamic domains. *Robotics and Autonomous Systems*, *56*, 980–991.

Fikes, R. E., & Nilsson, N. J. (1971). STRIPS: A new approach to the application of theorem proving to problem solving. *Artificial Intelligence*, *2*, 189–208.

Forbus, K. D. (1984). Qualitative process theory. *Artificial Intelligence*, *24*, 85–168.

Gat, E. (1998). On three-layer architectures. *Artificial Intelligence and Mobile Robotics*, *195*, 195–210.

Gebser, M., Kaminski, R., Kaufmann, B., & Schaub, T. (2013). *Answer set solving in practice*. Synthesis Lectures of Artificial Intelligence and Machine Learning. Morgan & Claypool.

de Kleer, J., & Brown, J. S. (1984). A qualitative physics based on confluences. *Artificial Intelligence*, *24*, 7–83.

Kober, J., Oztop, E., & Peters, J. (2011). Reinforcement learning to adjust robot movements to new situations. *Proceedings of the Twenty-Second International Joint Conference on Artificial Intelligence* (pp. 2650–2658). Barcelona, Spain: AAAI Press.

Kuipers, B. J. (1984). Commonsense reasoning about causality: Deriving behavior from structure. *Artificial Intelligence*, *24*, 169–203.

Kuipers, B. J. (1986). Qualitative simulation. *Artificial Intelligence*, *29*, 289–338.

Lagriffoul, F., Dimitrov, D., Bidot, J., Saffiotti, A., & Karlsson, L. (2014). Efficiently combining task and motion planning using geometric constraints. *The International Journal of Robotics Research*, *33*, 1726–1747.

Laird, J. E., Kinkade, K. R., Mohan, S., & Xu, J. Z. (2012). Cognitive robotics using the Soar cognitive architecture. *Proceedings of the AAAI 2012 Workshop on Cognitive Robotics* (pp. 46–54). Orlando, FL: AAAI Press.

Michie, D., Bain, M., & Hayes-Michie, J. (1990). Cognitive models from subcognitive skills. *Knowledge-base Systems in Industrial Control*, *44*, 71–99.

Muggleton, S. H., Lin, D., Pahlavi, N., & Tamaddoni-Nezhad, A. (2014). Meta-interpretive learning: Application to grammatical inference. *Machine Learning*, *94*, 25–49.

Neumann, G., Daniel, C., Paraschos, A., Kupcsik, A., & Peters, J. (2014). Learning modular policies for robotics. *Frontiers in Computational Neuroscience*, *8*, 1–13.

Ng, A. Y., Coates, A., Diel, M., Ganapathi, V., Schulte, J., Tse, B., Berger, E., & Liang, E. (2006). Autonomous inverted helicopter flight via reinforcement learning. In *Experimental Robotics IX*, 363–372. Springer.

Nishida, T., & Doshita, S. (1987). Reasoning about discontinuous change. *Proceedings of the Sixth National Conference on Artificial Intelligence* (pp. 643–648). Seattle, WA: AAAI Press.

Quinlan, J. R. (1993). *C4.5: Programs for machine learning*. San Manteo, CA: Morgan Kaufmann.

Sammut, C., Hurst, S., Kedzier, D., & Michie, D. (1992). Learning to fly. *Proceedings of the Ninth International Workshop on Machine Learning* (pp. 385–393). Aberdeen, Scotland: Morgan Kaufmann.

Sammut, C., & Yik, T. F. (2010). Multistrategy learning for robot behaviours. In J. Koronacki, Z. Raś, S. Wierzchon, & J. Kacprzyk (Eds.), *Advances in machine learning I*, 457–476. Springer.

Schaal, S., Peters, J., Nakanishi, J., & Ijspeert, A. (2005). Learning movement from primitives. *Proceedings of the Eleventh International Symposium on Robotics Research* (pp. 561–572). Zurich, Switzerland: Springer.

Simon, H. A. (1956). Rational choice and the structure of the environment. *Psychological Review*, *63*, 129–138.

Šuc, D., & Bratko, I. (1999). Symbolic and qualitative reconstruction of control skill. *Electronic Transactions on Artificial Intelligence*, *3*, 1–22.

Šuc, D., Bratko, I., & Sammut, C. (2004). Learning to fly simple and robust. *Proceedings of the Fifteenth European Conference on Machine Learning* (pp. 407–418). Pisa, Italy: Springer.

Sutton, R. S., & Barto, A. G. (1998). *Reinforcement learning: An introduction (1st ed.)*. Cambridge, MA: MIT Press.

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

Tenorth, M., Bartels, G., & Beetz, M. (2014). Knowledge-based specification of robot motions. *Proceedings of the Twenty-First European Conference on Artificial Intelligence* (pp. 873–878). Prague, Czech Republic: IOS Press.

Theodorou, E. A., Buchli, J., & Schaal, S. (2010). A generalized path integral control approach to reinforcement learning. *Journal of Machine Learning Research*, *11*, 3137–3181.

Wiley, T., Sammut, C., & Bratko, I. (2013a). Planning with qualitative models for robotic domains. *Poster Collection in the Second Annual Conference on Advances in Cognitive Systems* (pp. 251–266). Baltimore, MD.

Wiley, T., Sammut, C., & Bratko, I. (2013b). Using planning with qualitative simulation for multistrategy learning of robotic behaviours. *Proceedings of the Twenty-Seventh International Workshop on Qualitative Reasoning* (pp. 24–31). Bremen, Germany.

Wiley, T., Sammut, C., & Bratko, I. (2014a). Qualitative planning with quantitative constraints for online learning of robotic behaviours. *Proceedings of the Twenty-Eighth AAAI Conference on Artificial Intelligence* (pp. 2578–2584). Quebec City, Canada: AAAI Press.

Wiley, T., Sammut, C., & Bratko, I. (2014b). Qualitative simulation with answer set programming. *Proceedings of the Twenty-First European Conference on Artificial Intelligence* (pp. 915–920). Prague, Czech Republic: IOS Press.

Williams, B. C. (1984). Qualitative analysis of MOS circuits. *Artificial Intelligence*, *24*, 281–346.

Žabkar, J., Mozina, M., Bratko, I., & Demsar, J. (2011). Learning qualitative models from numerical data. *Artificial Intelligence*, *175*, 1604–1619.