Learning Guided Planning for Robust Task Execution in Cognitive Robotics

Sertac Karapinar and **Sanem Sariel-Talay** and **Petek Yildiz** and **Mustafa Ersen**
Artificial Intelligence and Robotics Laboratory
Computer Engineering Department
Istanbul Technical University, Istanbul, Turkey
{karapinarse,sariel,petekyildiz,ersenm}@itu.edu.tr

Abstract

A cognitive robot may face failures during the execution of its actions in the physical world. In this paper, we investigate how robots can ensure robustness by gaining experience on action executions, and we propose a lifelong experimental learning method. We use Inductive Logic Programming (ILP) as the learning method to frame new hypotheses. ILP provides first-order logic representations of the derived hypotheses that are useful for reasoning and planning processes. Furthermore, it can use background knowledge to represent more advanced rules. Partially specified world states can also be easily represented in these rules. All these advantages of ILP make this approach superior to attribute-based learning approaches. Experience gained through incremental learning is used as a guide to future decisions of the robot for robust execution. The results on our Pioneer 3DX robot reveal that the hypotheses framed for failure cases are sound and ensure safety in future tasks of the robot.

Introduction

A cognitive robot possesses abilities to plan to attain its goals, to execute its plan and to reason about dynamic cases. Plan execution may fail in the physical world due to unexpected outcomes (Pettersson 2005; Karapinar, Altan, and Sariel-Talay 2012). Robustness is crucial for success, and the robot should gain experience in the physical world and use this experience in its future tasks. We investigate incremental learning methods for cognitive robots. Our approach is based on applying an experimental learning process to learn from action execution failures, and then applying an adaptive planning strategy to use experience gained through learning. This work builds on our earlier work (Karapinar, Altan, and Sariel-Talay 2012; Yildiz, Karapinar, and Sariel-Talay 2013). Learning is from experimentation and failures that are experienced in the physical world. We use an Inductive Logic Programming (ILP) approach.

As a motivating example to illustrate the learning problem, consider a cognitive robot equipped with an arm and a two-finger gripper. During the long-term operation of the robot, it may need to take the same action several times in different contexts. Considering a chess playing scenario, the robot interacts with chess pieces by *pick up*, *put down*, *push* and *move* actions. To ensure robustness, the robot needs to monitor its execution. Let's assume that the robot always fails in *picking up* the *knight piece*. This failure might occur because of several reasons such as the shape of the piece, its weight, a vision problem or improper grasp position. In this case, the robot needs to find another way to complete a move with another action. Action *push* may be an alternative to *pick up* if there is a free way to the intended position. Therefore, the action selection mechanism of the robot should take the gained experience into account after action execution failures. An efficient learning mechanism is needed to match the execution context and the failure situation in such cases and to update the knowledge base so that the recovery precautions are included in the plan.

In this work, we discuss how the experience gained from failures can be used to guide planning and how hypotheses can be generalized by using ILP. There exist methods that learn stochastic models of actions (Pasula, Zettlemoyer, and Kaelbling 2007), effects of actions (Hermans, Rehg, and Bobick 2011; Usug and Sariel-Talay 2011; Usug, Altan, and Sariel-Talay 2012) and control rules for planning (Borrajo and Veloso 1994; Estlin and Mooney 1997; Aler, Borrajo, and Isasi 2002; Fernandez, Aler, and Borrajo 2004; Duran 2006). Our approach differs from earlier work in the way learning outcomes are applied. In our approach, action execution experience gained in the real world is used to provide feedback to the robot to improve its performance on its future tasks. Context situations (the actions, the objects in interest and their relations) are considered for derivation of hypotheses which are expressed in first-order logic sentences. Derived hypotheses are then used to devise heuristics for planning. We propose a hybrid heuristic guidance method for failure prevention.

In this paper, we first review the literature on applying adaptive planning strategies. Then, we describe the details of our learning-guided adaptive planning method. We then present the empirical results and a discussion on the generalization of hypotheses followed by the conclusions.

Background

In this section, we formulate planning, learning and learning guided planning problems for robust task execution by cognitive robots, and in the following section, we present our solution to this problem.

Planning

Cognitive robots need automated onboard action planning for online generation of action sequences against exogenous events. A planning task Π can be described on a state space S containing a finite and discrete set of states including an initial state s_0 and a goal state s_G, and a state transition function $s_{t+1} = f(o_t, s_t)$ where $o_t \in O(s_t)$ is an operator applied in state s_t. Planning operators $o \in O$ are defined as symbolically abstracted representations of actions $a \in A$ and can be formalized as a tuple $o = \{pre(o), add(o), del(o), cost(o)\}$ where $pre(o)$ defines the preconditions, $add(o)$ and $del(o)$ define the effects of the operator and $cost(o)$ represents the cost of the corresponding action. $o_t \in O(s_t)$ is defined as the set of applicable operators in each state $s_t \in S$ determined by checking preconditions of the operators to satisfy $pre(o_t) \subseteq s_t$. By applying o_t at state s_t, a new state $s_{t+1} = add(o_t) \cup (s_t \setminus del(o_t))$ is observed. Planning task is achieved by a planner to reach s_G from s_0 by selecting a sequence of operators from $O(s_t)$ at successive states s_t and executing the corresponding actions $a_t \in A$ in the given order. Classical planners accept this model as a compact representation of the planning domain (Δ). Some planners use control formulas if they are defined in Δ while selecting o_t. After searching the whole space of operators, the planner constructs a valid plan $P = o_{0:G}$ by considering an optimization criteria (e.g., makespan) and the duration/cost of each operator.

In our work, we use TLPlan (Bacchus and Ady 2001), a forward-chaining temporal planner. This planner uses search control formulas that are expressed in terms of Linear Temporal Logic (LTL) formulas. These rules enable the planner to reduce its search space by pruning unpromising branches that lead to dead-ends or suboptimal plans in the search tree. These rules can be from either select rules to decide on which action to select or reject rules for rejection. TLPlan uses $\cup$ (until), $\Box$ (always), $\Diamond$ (eventually), $\bigcirc$ (next) from LTL temporal modalities and $GOAL$ as an additional one. $\cup$ states that a formula is true until another becomes true. $\Box$ means a formula is true in all world states. $\bigcirc$ specifies that a formula is true in the next world state. $\Diamond$ means a formula is true now or becomes true in some future state. $GOAL$ is used to specify whether a formula appears in s_G.

Having generated a valid plan P, the robot can execute each corresponding action $o_t \rightarrow a_t \in A$ in sequence in the physical world. If all goes well with execution, the robot successfully attains s_G. However, due to non-deterministic actions and different sources of uncertainty in physical environments, several failures (Karapinar, Altan, and Sariel-Talay 2012) may be encountered. Our primary focus is action execution failures. Some of these failures are encountered due to misbeliefs or lack of knowledge about some environmental features.

Learning

The robot should maintain background knowledge KB = (F, H) where F is the set of facts and H is the set of hypotheses. Whenever needed, the robot infers new hypotheses (H) based on its observations such that $KB \models H$, and updates its KB according to this inference result. H is said to improve the robot's performance on its future tasks. The problem that we investigate asks for developing a learning method to map from action execution contexts to failure cases. This is needed to either handle or prevent from failures for robustness. Since the robot can observe its execution and environment during its whole lifetime, an incremental and continual approach is needed. Furthermore, the learning algorithm should be able to represent hypotheses by logic-based sentences since the knowledge base of a cognitive robot is represented symbolically to reason and plan for achieving its goals. Since the robot has partial observability, the facts that it can extract from the world do not always include the values of all of the variables that describe the world. An Inductive Logic Programming approach meets all these requirements. For this reason, it is the approach that we use in our solution.

Integrating Planning and Learning

The main object of this study is developing methods to use experience on the future tasks of a robot. Alternative solutions exist for using the gained experience. Several learning systems have been developed to acquire search control rules automatically such as HAMLET (Borrajo and Veloso 1994) using lazy learning, EVOCK (Aler, Borrajo, and Isasi 2002; Fernandez, Aler, and Borrajo 2004) using genetic programming, and SCOPE (Estlin and Mooney 1997) and Grasshopper (Leckie and Zukerman 1998) using inductive learning. SCOPE uses Explanation-Based Learning (EBL) on UCPOP planner. Grasshopper (Leckie and Zukerman 1998) learns control rules to guide PRODIGY planner for improving planning speed by using inductive learning.

In some of these works, previously generated plans with hand-coded rules are used as training examples for learning. Two types of rules are learned: select rules from positive examples and reject rules from negative examples. Learned rules may be incorrect, incomplete or too many for effective search. In such cases, they can be corrected by both generalization and specialization (Borrajo and Veloso 1996), or improvements through optimization (Aler, Borrajo, and Isasi 2002) and removing non-useful ones (Cohen 1990) according to a utility function to generate high-utility rule set. In most of the existing approaches, control-rules are learned from planning failures to speed-up planning (Katukam and Kambhampati 1994). They learn reject rules to prevent the planner from dead-ends or depth-limit failures by using EBL. In another work (Duran 2006), both macro-operators and control rules are learned to combine their benefits. The learned control rules are for deciding when to use macro-operators. A similar study (Rintanen 2000) presents use of plan operators with control rules in domain-independent planning. Apart from learning search control rules, there exist methods to learn action representations in a planning framework. One of the most important works in the area addresses learning stochastic models of action outcomes (Pasula, Zettlemoyer, and Kaelbling 2007). In that work, probabilistic STRIPS rules for operators are learned from training data.

One of the most relevant studies (Haigh and Veloso 1999)

integrates planning, monitoring and learning processes. The system estimates action costs and probabilities from real-world executions to avoid failures by creating control rules. Learner uses regression trees to learn correlations between features of the environment, action costs and situations, then generates situation-dependent rules. Control rules are used to decide which goals and which actions to be selected. Similarly, ROBEL (Morisset and Ghallab 2008) learns robust ways of performing tasks. The main problem that we would like to tackle differs from that of these systems in the way learning is accomplished and its results are used. First, the derived hypotheses are needed to be integrated with reasoning. Second, the features that are learned correspond to context situations which may involve features of objects and their relations. These facts necessitate knowledge-based learning methods to be integrated with planning systems.

Learning Guided Planning

Our work builds on our earlier work on an experimental learning approach for framing hypotheses on failure cases (Karapinar, Altan, and Sariel-Talay 2012). We extend this work by an adaptive planning strategy that uses these hypotheses (Yildiz, Karapinar, and Sariel-Talay 2013). In this section, we first present an overview of our previous learning method, and then the details of our new method to devise heuristics for planning.

Lifelong Experience-based Learning

We use *Inductive Logic Programming (ILP)* as a part of our continual planning, execution and learning framework for cognitive robots (Karapinar, Altan, and Sariel-Talay 2012). The main purpose in learning is finding a set of hypotheses that are consistent with the observed data during the execution of a constructed plan. Each hypothesis is expressed by a set of first-order logic sentences. ILP helps to build, update or abandon hypotheses as the robot acquires new observations. ILP based-learning is provided by the FOIL algorithm (Quinlan 1990). It helps robots build their experience through observing different states of execution during their lifetime. Framed hypotheses involve either known or observed features of objects, their relations and the observable features of world states as contexts of failures. Each hypothesis is associated with a weight (w) based on its confidence on correctly modeling observations. Note that only the relevant facts from the world state and the outcome of an observed action are taken into account.

Mapping from Hypotheses to Heuristics

Lifelong learning procedure continually frames new hypotheses during execution. These hypotheses are to be used to improve the performance of the robot on its future tasks. We analyze three ways of using hypotheses: (i) deriving new control formulas (ii) updating the models of operators corresponding to the failed actions (iii) setting an adaptive cost computation method for the operators. In the first approach, the selection of a failed operator is completely abandoned to prevent its selection on specific contexts. In the second approach, the preconditions of the failed operators are updated to prevent their selection in specific branches during search. In the third approach, the cost values of failed operators are updated to set preference models.

We investigate these three approaches in a scenario with several objects to be picked up and moved to a destination. The objects have several observable features some of which are predefined. Assume that the following is a hypothesis framed after observations taken from this environment.

$$category(box) \wedge color(red) \Rightarrow pickupFail$$

Based on this hypothesis, a search control formula can be constructed in LTL according to the context of the given hypothesis as following:

$$\Box(\forall[obj : object(obj)](box(category, obj)) \\ \wedge(red(color, obj)) \wedge \forall[robot : agent(robot)] \\ \Longrightarrow \bigcirc(\neg holding(robot, obj)))$$

This formula specifies that if the given context is satisfied by the world state (including the features of the objects), the robot believes that it will fail in the execution of action *pick up*, and the effects of the operator will not appear in the next world state. The formula represents a reject rule for a failed operator by inserting the distinctive effects of the operator (e.g., $holding$ for *pick up* operator) in its consequent part.

The precondition update for the given hypothesis can be done in the following way:

```
(def-defined-predicate (pickupFailContext ?obj)
  (and
    (= (category ?obj) box)
    (= (color ?obj) red)
   )
)
(def-adl-operator (pickup ?robot ?obj)
  (pre
    (and
      (not (pickupFailContext ?obj))
      (handempty ?robot)
      (....)
    )
  )
  (del (...))
  (add (...))
)
```

This precondition update prevents the selection of operator *pick up* for the context defined in the hypothesis representing a failure case.

The cost update method also considers the context of the failure case and increases the cost of the operator by a factor to prevent its selection in a future plan according to the following equation.

$$cost' = cost + w * k \quad (1)$$

where the update factor is proportional to the weight of the corresponding hypothesis. The gain value, k, is set to a number to guarantee that the cost penalty is greater than the maximum cost value of any other operator. We analyzed the effects of these three methods in our previous work (Yildiz, Karapinar, and Sariel-Talay 2013). According to this analysis, we adopted a hybrid approach which

integrates precondition update and cost update methods to prevent the selection of a failed operator, but also to ensure its selection when there is no alternative way. Based on the derived hypotheses, our hybrid method proposes the necessary updates to guide the future planning tasks. We consider three states after executing an action, namely `success`, `fail-safe` and `fail-unsafe` of which we repeat definitions here for convenience.

Definition 1 (success state) If all the desired effects of the action occurs in the environment, the situation is specified as `success`.

Definition 2 (fail-safe state) If the state of an execution is not `success` but the state does not change, the situation is specified as `fail-safe`. For example, the robot fails in *picking up* an object but the state of the object is not changed.

Definition 3 (fail-unsafe state) If the execution of an action fails and there is any damage and/or dangerous situation (e.g., an undesirable state is observed) or the robot cannot judge whether there is any harmful situation, the situation is specified as `fail-unsafe`. For example, the robot fails in *picking up* an object and the state of the object is changed. It may be broken into pieces.

If the state corresponds to a `success`, Δ is not updated. In a `fail-safe` state, the robot is allowed to select the same action but with a price. In this case, the cost of the operator is increased by a factor proportional to the weight of the hypothesis. In a `fail-unsafe` state, the precondition of the failed operator is updated. In both of these cases, the antecedent part (context) of the hypothesis is used to either encode new preconditions or update the cost function.

Experimental Results

We have implemented our service robot scenario in the simulation environment. The robot is tasked to move five different objects to its destination. These objects and their attributes are: obj_1 is a tall, plastic bottle, obj_2 is a small, blue box, obj_3 is a small, green box, obj_4 is a small, yellow ball and obj_5 is a tray. Before learning, the plan is constructed to move all these objects on the tray. The constructed plan is illustrated in Figure 1. During the execution of its plan, the robot discovers that it cannot place the bottle on top of the tray, and cannot pick up the ball. After observing these situations, it derives new hypotheses and use these hypotheses to guide its future planning tasks. A new plan for the same scenario is constructed based on the gained experience. The new plan, illustrated on the right hand side of the figure, excludes all the failed operators. As expected, its plan duration is longer than that of the initial case although the number of operators is less. However, it is safer than the previous plan.

Robot Experiments

Our real-world experiments are set in our laboratory environment with our Pioneer 3-DX robot equipped with a standard gripper in front of it, a sonar ring and an RGB-D sensor.

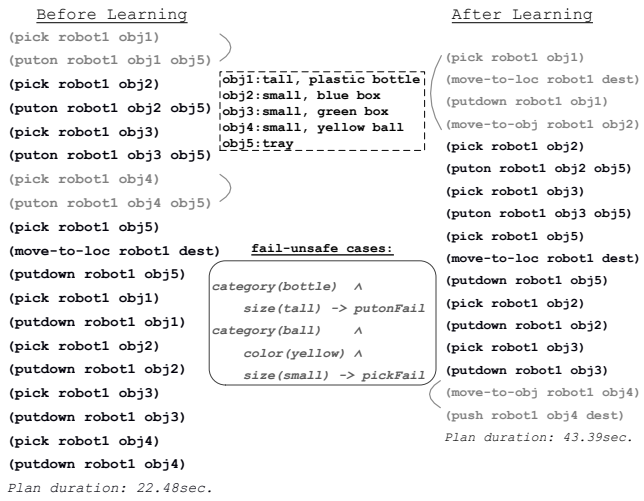

Figure 1: In an initial plan to move the given objects to a destination, all objects are moved together on top of the tray. However, in the execution, the bottle cannot be placed on top of the tray, and the ball cannot be picked up. After learning, the plan is updated to select alternative actions for these two objects. The changed plan steps are given in gray.

The objects used in the experiment are from four main categories, namely, rectangular prism box, cylindrical box, plastic bowling pin and ball. Our robot grasping one of the objects used in the experiment is illustrated in Figure 2. We use LINE-MOD (Hinterstoisser et al. 2012) to recognize these objects in the scene for interpretation of the world state. This method is originally proposed for textureless objects where local invariant descriptors cannot be used for matching between the model and the scene. To handle this situation, a multi-modal template matching approach is used in LINE-MOD. For recognizing an object, LINE-MOD takes into account both surface normals to model the shape of the object and color gradients to distinguish objects in different colors. As well as different shaped objects from different categories, we have used different colored objects of the same category to see the effects of the vision algorithm.

Figure 2: Our Pioneer 3-DX robot grasping one of the objects used in the experiments.

In our first experiment, we measure the success of the action *pick up* on all the objects five times for each. The distance of an object from the robot is computed by taking the center of the recognized and aligned object template as a ref-

Table 1: *Pick up* success for the given objects

Category	Shape	Color	Material	Size	Success
box	*prism*	*green*	*paper*	*small*	*true*
box	*prism*	*black*	*paper*	*large*	*true*
box	*cylinder*	*green*	*plastic*	*large*	*true*
box	*cylinder*	*red*	*plastic*	*large*	*false*
pin		*orange*	*plastic*	*small*	*false*
pin		*green*	*plastic*	*small*	*true*
ball	*sphere*	*purple*	*plastic*	*small*	*true*
ball	*sphere*	*white*	*foam*	*small*	*true*

erence. The robot can detect a failure by using its RGB-D sensor when it is far from the object, the sonar sensor ring when it gets closer, and the gripper position sensor during grasping.

Table 1 shows the results of the first experiment. As these results illustrate, the robot fails in executing action *pick up* for one of the boxes and one of the bowling pins. The reason behind the former failure is due to the physical deformation of the object that causes the sonar sensors fail in determining whether the object is in front. But this is not an observable attribute by the robot. Instead, the observable physical attributes of the given objects are considered for deriving hypotheses. In the latter case, the alignment of the registered vision template is the main problem. The robot sometimes confuses the orange pin's template with that of the green one (Figure 3), which causes a difference in the distance calculation, and results in a failure, most of the time, while approaching the object. The sonar sensor does not give accurate values either since the top part of the plastic pin is narrow. All the observations taken by the robot are fed into the ILP learner to frame hypotheses on failure cases for action *pick up* on different objects. The hypothesis space generated after the observations are given below.

$color(green) \Rightarrow pickupSuccess$

$category(ball) \wedge shape(sphere) \Rightarrow pickupSuccess$

$category(box) \wedge shape(prism) \wedge color(black) \wedge material(paper) \Rightarrow pickupSuccess$

$category(box) \wedge color(red) \Rightarrow pickupFailSafe$

$category(pin) \wedge color(orange) \Rightarrow pickupFailUnsafe$

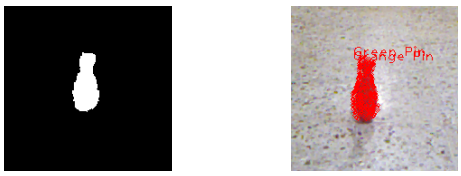

Figure 3: In some cases, the robot confuses the orange plastic bowling pin with the green one. (left) The mask of the orange pin from which the template is extracted. (right) The case where the object cannot be recognized well.

This hypothesis space suggests that operator *pick up* should be penalized (i.e., its cost function and preconditions are updated) when executing it in two different contexts: a red colored box and an orange colored bowling pin. These hypotheses provide a preference model for the robot. By adjusting the cost of action *pick up* based on these hypotheses, the robot is guided to select another action (e.g., *push*) to move these objects (if it does not fail either) or to *pick up* alternative objects (for which *pick up* is known to succeed) depending on the context of the task. Let's assume, based on a new observation, the following hypothesis is added into the hypothesis space:

$category(pin) \wedge color(orange) \Rightarrow pushFailSafe$

In this case, action *push* fails in a fail-safe mode but action *pick up* should be avoided for the given context. When there is no alternative to *push*, given a new task on *an orange pin*, the planner is expected to come up with a plan that includes operator *push*. As expected, our hybrid heuristic method can guide the planner in this way. However, both precondition update and control formula update methods fail in finding a valid plan. When planning efficiency is analyzed, it has been observed that as the number of objects that fit in the context cases increase, the planning time decreases as the experience is used to guide planning in the way encountered failures are taken into account (Yildiz, Karapinar, and Sariel-Talay 2013).

Discussion

The main superiority of the *ILP learner* to conventional classifiers is its knowledge-based representation. It can represent hypotheses in first-order logic and incorporate background knowledge. To illustrate why it is better in our system, let's consider the *Blocks World* planning domain with *pick up* and *stack* actions. We analyze the hypotheses derived after the failure of action *stack* in different contexts.

After detecting the failure, all the relevant facts from the world state is included in the observation. We use an intuitive approach for getting the relevant facts from the state: using spatial locality of the objects. In the $StackFailure$ case, the facts related to the objects (the block in hand and the one on the top of the stack to put on) involved in the parameter list of the action and the objects that are in close proximity to these objects (the stacked objects all the way to the table) are considered.

Let's assume that we want to stack three blocks on top of each other. The first *stack* action ($stack(b, c)$) succeeds but the second *stack* action ($stack(a, b)$) fails. Given the observations, the following hypotheses are derived:

$holding(b) \wedge clear(c) \wedge ontable(c) \Rightarrow StackSuccess$

$holding(a) \wedge clear(b) \wedge ontable(c) \wedge on(b, c) \Rightarrow StackFailure$

Since there is only a single observation of a failed *stack*, the derived hypothesis includes all the observed state information in its antecedent part. As one of the strengths of ILP, this hypothesis can be generalized to abstractions if the casual model of the world exists. For example, if the following rules exist in KB:

$\forall x\ ontable(x) \wedge clear(x) \Rightarrow Tower(1)$

$\forall x, n\ (n > 1) \wedge Tower(n-1) \wedge on(x, Tower(n-1)) \wedge clear(x) \Rightarrow Tower(n)$

By using this background knowledge, the hypotheses are generalized as follows:

$Tower(1) \Rightarrow StackSuccess$

$Tower(2) \Rightarrow StackFailure$

If the robot detects a failure after the execution of the third *stack* in a four-block scenario, the derived hypothesis space is given below:

$Tower(n) \wedge (1 \leq n \leq 2) \Rightarrow StackSuccess$

$Tower(3) \Rightarrow StackFailure$

Obviously, an attribute-based learner cannot determine such relations easily. It needs to encode all pairwise relations instead. For this specific example, the real cause of the failure may be related to a vision problem, the instability of the robot's arm after a certain height or displacement of the blocks in the horizontal line leading to imbalance. If there is no quantitative or qualitative measurement way for these issues, it is difficult to isolate the failure. However, even when the actual underlying reason cannot be identified, the robot believes that it will fail in executing action *stack* to put on a block on top of a tower with three or more blocks, and then plans accordingly in its future tasks.

Conclusions

Our approach for robust task execution includes an experience-based learning method, ILP, to learn from action execution failures. The incremental learning process is used to frame hypotheses for relating different contexts to failure situations. In the derived hypotheses, the observable attributes of and the relations among the objects and the relevant facts of the world are specified. The results of the learning process are then used to guide the future decisions on planning for robust execution. ILP also enables using background knowledge to interpret the observations. Background knowledge can be used to generalize hypotheses. On the other hand, attribute-based learners do not incorporate background knowledge. Since ILP can use partial specifications of the world states, it can deal with the missing data problem. All these advantages of ILP make it more efficient for the experimental learning process.

Acknowledgements

This research is funded by a grant from the Scientific and Technological Research Council of Turkey (TUBITAK), Grant No. 111E-286. TUBITAK's support is gratefully acknowledged.

References

Aler, R.; Borrajo, D.; and Isasi, P. 2002. Using genetic programming to learn and improve control knowledge. *Artificial Intelligence* 141:141–1.

Bacchus, F., and Ady, M. 2001. Planning with resources and concurrency a forward chaining approach. In *Proceedings of the 17th International Joint Conference on Artificial Intelligence - Vol. 1*, 417–424.

Borrajo, D., and Veloso, M. 1994. Incremental learning of control knowledge for improvement of planning efficiency. In *In AAAI-94 Fall Symposium on Planning and Learning*, 5–9.

Borrajo, D., and Veloso, M. 1996. Lazy incremental learning of control knowledge for efficiently obtaining quality plans. *AI Review Journal. Special Issue on Lazy Learning* 11:371–405.

Cohen, W. W. 1990. Learning approximate control rules of high utility. In *In Proceedings of the Seventh International Conference on Machine Learning*, 268–276. Morgan Kaufmann.

Duran, G. 2006. Integrating macro-operators and control-rules learning. In *The International Conference on Automated Planning and Scheduling*.

Estlin, T. A., and Mooney, R. J. 1997. Learning to improve both efficiency and quality of planning. In *In Proceedings of the Fifteenth International Joint Conference on Artificial Intelligence*, 1227–1232. Morgan Kaufmann.

Fernandez, S.; Aler, R.; and Borrajo, D. 2004. Using previous experience for learning planning control knowledge. In *Proceedings of the Seventeen International Florida Artificial Intelligence Symposium (FLAIRS04)*. AAAI Press.

Haigh, K. Z., and Veloso, M. M. 1999. Learning situation-dependent costs: Improving planning from probabilistic robot execution. *Robotics and Autonomous Systems* 29:145–174.

Hermans, T.; Rehg, J. M.; and Bobick, A. 2011. Affordance prediction via learned object attributes. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA): Workshop on Semantic Perception, Mapping, and Exploration*.

Hinterstoisser, S.; Cagniart, C.; Ilic, S.; Sturm, P.; Navab, N.; Fua, P.; and Lepetit, V. 2012. Gradient response maps for real-time detection of textureless objects. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 34(5):876–888.

Karapinar, S.; Altan, D.; and Sariel-Talay, S. 2012. A robust planning framework for cognitive robots. In *Proceedings of the AAAI-12 Workshop on Cognitive Robotics (CogRob)*.

Katukam, S., and Kambhampati, S. 1994. Learning explanation-based search control rules for partial order planning. In *AAAI*, 582–587.

Leckie, C., and Zukerman, I. 1998. Inductive learning of search control rules for planning. *Artificial Intelligence* 101(1-2):63–98.

Morisset, B., and Ghallab, M. 2008. Learning how to combine sensory-motor functions into a robust behavior. *Artificial Intelligence* 172(4-5):392 – 412.

Pasula, H.; Zettlemoyer, L. S.; and Kaelbling, L. P. 2007. Learning symbolic models of stochastic domains. *Journal of Artificial Intelligence Research (JAIR)* 29.

Pettersson, O. 2005. Execution monitoring in robotics: A survey. *Robotics and Autonomous Systems* 53:73–88.

Quinlan, J. R. 1990. Learning logical definitions from relations. *Machine Learning* 5:239–266.

Rintanen, J. 2000. Incorporation of temporal logic control into plan operators. In *In ECAI*, 526–530. IOS Press.

Usug, U. C.; Altan, D.; and Sariel-Talay, S. 2012. Robots that create alternative plans against failures. In *10th IFAC Symposium on Robot Control*.

Usug, U. C., and Sariel-Talay, S. 2011. Dynamic temporal planning for multirobot systems. In *Proceedings of the AAAI-11 Workshop on Automated Action Planning for Autonomous Mobile Robots (PAMR)*.

Yildiz, P.; Karapinar, S.; and Sariel-Talay, S. 2013. Learning guided symbolic planning for cognitive robots. In *The IEEE International Conference on Robotics and Automation (ICRA), Autonomous Learning Workshop*.