Advances in Cognitive Systems: Papers from the 2011 AAAI Fall Symposium (FS-11-01)

Tool Use Learning in Robots

Solly Brown and Claude Sammut
School of Computer Science and Engineering, The University of New South Wales,
Sydney, Australia 2052

Abstract

Learning to use an object as a tool requires understanding what goals it helps to achieve, the properties of the tool that make it useful and how the tool must be manipulated to achieve the goal. We present a method that allows a robot to learn about objects in this way and thereby employ them as tools. An initial hypothesis for an action model of tool use is created by observing another agent accomplishing a task using a tool. The robot then refines its hypothesis by active learning, generating new experiments and observing the outcomes. Hypotheses are updated using Inductive Logic Programming. One of the novel aspects of this work is the method used to select experiments so that the search through the hypothesis space is minimised.

Introduction

Contrary to popular belief, tool use is not a uniquely human trait. Tool use has been observed in a wide range of both captive and wild animals: chimpanzees use sticks to extract termites from termite mounds (van Lawick-Goodall 1970); Egyptian vultures select appropriately sized stones to crack open stolen eggs (Goodall and van Lawick 1966); burrowing owls use mammalian dung to 'fish' for dung beetles (Levey, Duncan, and Levins 2004); and some bottle-nosed dolphins use sponges on their noses to protect themselves from stings while foraging on the sea floor (Krützen et al. 2005). Some creatures such as the New Caledonian crow are even adept at tool-making and solving novel problems using tools (Hunt 1996; Kenward et al. 2005).

Animals that make use of tools do so in order to enable or improve their ability to carry out important tasks. The New Caledonian crow, for example, uses twig and leaf-based hook tools to extract grubs from crevices, a feat that would be more difficult or impossible without these tools. By exploiting objects in the environment, they are able to overcome the limitations of their effectors and expand the range of goals they can achieve.

Humans are, of course, more proficient at using tools for problem solving than other animals. No other creature is able to exploit objects in its environment more effectively, or shows as wide-ranging an ability to use tools, as humans. Tools are so ubiquitous in our everyday lives that there is scarcely an activity we perform which does not involve using one or more objects to make the task possible, or to enable it to be carried out in a simpler or more efficient manner.

Too use is a kind of behaviour that is exhibited by intelligent organisms and therefore is worthy of study from an AI perceptive. It is also interesting for machine learning because of the added complexity of learning when tools are introduced. This complexity arises from the fact that when several objects may potentially be used as tools, the robot must learn to select the appropriate object, taking into consideration the spatial constraints of the environment and the structural constraints of the tool. Thus, the hypothesis language of the learner must be rich enough to express spatial and structural relations. However, a richer hypothesis language entails greater search complexity. As we shall seem some knowledge of the requirements of tool use can constrain this search space, to some extent. In this work, we are interested in giving a robot the capability of learning to use tools but we do not attempt to explicitly model human or animal learning and their cognitive constraints.

We define a tool as an object that is deliberately employed by an agent to help it achieve a goal that would otherwise be more difficult or impossible to accomplish. We would like a robot to be able to learn to use a new tool after observing it being used correctly by another agent. This observation results in an initial hypothesis about the properties and use of the tool. The hypothesis is refined by active experimentation, selecting different types of objects as the tool and manipulating them in different ways. By analysing the teacher's demonstration and then experimenting, the aim is to learn a new tool action that will allow the agent to perform variations of the task.

More precisely, a *tool action* is an action that involves the manipulation of an object (the tool), with the effect of changing the properties of one or more other objects (the targets) or the agent itself, in a way that helps enable the preconditions of one or more other actions in the agent's plan library. The changed properties of the target objects that result from the tool action are called the sub-goals of the tool action.

There are four things that the robot must learn so that it consistently solves a tool-use task:

1. what useful effect the tool action achieves,
2. how to identify a good tool (i.e. what object properties are

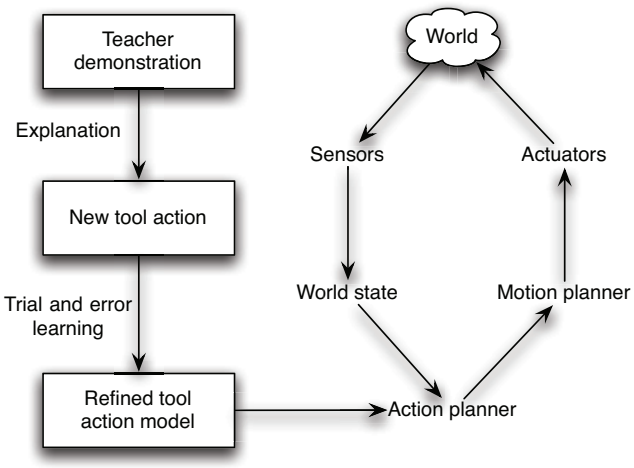

Figure 1: Learning architecture.

necessary),

3. the correct position in which the tool should be placed,
4. how the tool should be manipulated after being placed.

We focus on learning the first three of these. In a robot, learning is necessarily one component of a larger cognitive system. The robot must perceive its environment through sensors and construct a representation of the world. It must have, or learn, a model of its own actions so that it can plan a sequence of actions to achieve a goal. Instructions must be sent to actuators to execute the actions and the execution must be monitored to ensure that the action is completed, as planned. The architecture of the system is shown in Figure 1. It involves a passive learning phase, in which the robot observes another agent, and an active learning phase, in which the robot designs, performs and critiques its own experiments.

Our action models are expressed in first-order logic, therefore the learning algorithm uses Inductive Logic Programming (Muggleton 1991) to acquire these models. They are tested by experimentation, which requires transforming a plan into an sequence of operational instructions to the robot. One of the novel features of our work is the combination constraint solving and planning to turn a qualitative action sequence into a sequence of instructions with quantitative parameters. In the following section, we describe previous work on tool use learning. We then give an example of tool use that is used to illustrate the methods described in the following sections. We describe our experimental method, followed by our conclusions and suggestions for future work.

Related Work

Tool use has received relatively little attention in Artificial Intelligence and Robotics research. Bicici *et al* (2003) survey early AI research related to the reasoning and functionality of objects and tools. Perhaps the first attempt to build a robot agent specifically tailored towards learning and tool-use was report by Wood *et al.*, (2005). In this work, an artificial neural network was used to learn appropriate postures for using reaching and grasping tools, on board the Sony Aibo robot platform. Stoytchev (2005) has implemented an approach to learning tool-using behaviours with a robot arm. The robot investigates the effects of performing its innate primitive behaviours (including pushing, pulling, or sideways arm movements) whilst grasping different reaching tools provided by the user. The results of this exploratory learning are used to solve the task of using the tool to move an object into a desired location. Later work by Sinapov and Stotchev (2008) follows up on this approach to learning tool affordances. Our work is complementary to this in that we learn tool properties in a form that can be used in by a high-level planner.

Although there is little other research that directly addresses tool use learning, work in the area of learning models of agent actions is relevant to our approach. Gil (1994) used learning by experimentation to acquire planning knowledge. Martin and Geffner (2000) and Schmid and Kitzelmann (2011) also describe a system for learning plans. The PRODIGY architecture (Veloso et al. 1995) integrates planning and learning. Benson (1996) used Inductive Logic Programming to learn action models of primitive actions and then combined them into useful behaviours. In our tool learning, we generate explanations of problem solving traces. Langley and Choi (2006) describe an explanation-based method for control programs. Shavlik (1990) was also able to learn recursive and iterative concepts with explanation-based learning. Other work has since focused on learning action models for planning in more complex environments, allowing for stochastic action (Pasula, Zettlemoyer, and Kaelbling 2007) or partial observability (Amir 2005). Levine and DeJong (2006) also used an explanation-based approach to acquiring planning operators. In the present work, we do not address the problem of uncertainty in sensing and action (Toussaint et al. 2010) but we discuss this further in the conclusions and future work. As in our work, Toussaint also combines relational planning and trajectory planning.

An Example of Learning to Use a Tool

A robot is set the goal of obtaining an object, in this case a small box, that is placed in a tube lying on the ground. The tube is open at one end and closed at the other, as shown in Figure 2. The robot is unable to reach directly into the tube to pick up the box because the tube is too narrow. The robot must use a hooked stick tool to pull the box out of the tube before it can be picked up.

As shown in Figure 2, the robot is provided with a selection of different objects that can potentially be used as a tool for accomplishing the task. Some of these objects are clearly inappropriate since they cannot be inserted into the tube, lack a suitable "hook" affordance or are not long enough. However, the robot does not know ahead of time which objects make good tools. The only information the robot has is provided by its sensors, i.e. the object's dimensions and the relative orientation of its surfaces.

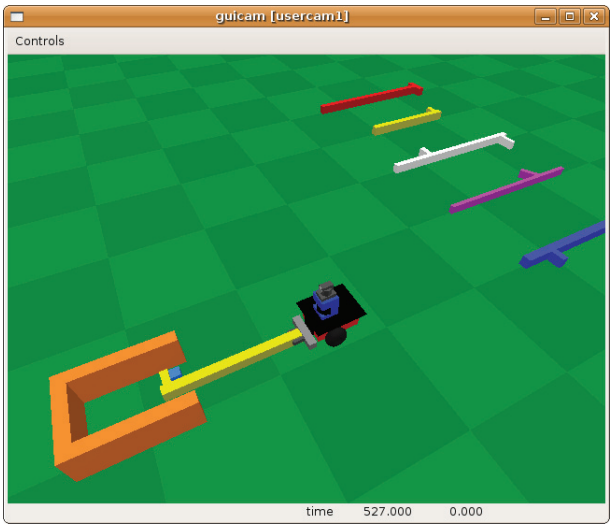

Figure 2: Using a tool to reach an object in a closed "tube".

The robot is also provided with a set of behaviours that it can use to affect changes in the world. In this example, the robot is given **goto**, **pickup-object** and **drop-object** behaviours. We assume that the agent does not already possess any sort of **pull-with-tool** behaviour. Learning this behaviour and using it to obtain the box is the objective of the problem.

The behaviours are represented by action models that describe how executing each behaviour affects the world. The action models are written as STRIPS-like operators (Fikes and Nilsson 1971). These are used by the robot's planner to create sequences of behaviours that achieve the robot's goals. A robot already competent in solving the tube problem might construct the following plan:

goto(stick-tool), pickup(stick-tool), goto(tube),
pull-with-tool(stick-tool, box),
drop(stick-tool), pickup(box).

The difficulty for our robot is that not only does it lack the appropriate tool-using behaviour, **pull-with-tool**, but it also lacks the action model of this behaviour. This means that the agent is initially unable to form a plan of how to achieve the goal of obtaining the box. It must learn this missing behaviour. We simplify learning by providing the agent with an observation trace of a "teacher" performing the same task. In the case of the tube, our teacher demonstrates by simply picking up an appropriate stick-hook and using it to pull the box from the tube. The robot uses this example to create an initial hypothesis for the tool action, which is the starting point for the robot's experimentation.

In the tube problem the necessary properties of the tool include having a right-angled hook at the end of the handle and that the hook is on the correct side of the handle (if the box is sitting in the left side of the tube, the agent needs a left-sided hook stick). Learning the type of tool and its pose requires experimenting with a variety of tools and poses. We will describe this process after giving details about the action representation.

Action Representation

Like STRIPS, the robot's action model includes the preconditions and effects of the action. A **pickup** action is specified as:

pickup(Obj)

PRE	forall(Tube:tube, ¬in(Obj,Tube)),
	empty-gripper,
	forall(x:obj, ¬obstructing(x,Obj))
ADD	gripping(Obj)
DEL	empty-gripper,
PRIMTV	fwd, back, rotleft, rotright
MOVING	robot

The precondition states that the object must not be in any tube, the robot's gripper must by empty and there must not be any other object obstructing the target object. The additional lists of primitive motor actions and objects moved by this action are needed to turn the action into motor commands to the robot.

Action models are abstract and do not specify behaviours in sufficient detail that they can be executed by the robot. In the example above, the action model says nothing about the precise position that the robot must be in after the action is executed, nor does it specify the path that the robot must take to get itself into position. However, the goal provides constraints for a motion planner. Actions can be made operational by using a constraint solver to create a range of acceptable solutions (e.g. any position such that the gripper surrounds the target object) then a motion planner can output a set of primitive motor actions to the robot (in this example, manoeuvring the robot to a position within the bounds set by the constraint solver). This process is illustrated in Figure 3.

We use the FF planner (Hoffmann and Nebel 2001) to generate the plans and the constraint logic programming language, *ECLiPSe* (Apt and Wallace 2007) to generate the constraints. A Rapid Random Tree (RRT) path planner (Kuffner and LaValle 2000) produces the primitive actions for the robot. RRT requires a set of motion primitives, a list of the objects that are manipulated by the planner, and a specification of their relative spatial pose during the manipulation. This is the reason why we extend the STRIPS representation to include a list of the objects involved in the action and a set of robot motion primitives. The motion planner outputs a path for the objects that leads to the desired goal state. A simple controller outputs the motor commands required to follow the path. Brown (Brown 2009) gives a detailed discussion of the implementation.

The advantage of using a motion planner to generate behaviours is that it allows the robot to quickly find solutions to novel subgoals. The disadvantage is a loss of full generality and the ability to solve difficult manipulation problems involving fully closed-loop control. In the present work, we are interested in learning how and why tools are useful at an abstract level. Other research has investigated the combination of planning and reinforcement learning (Ryan 2002) to also acquire low-level control behaviours. Our architecture

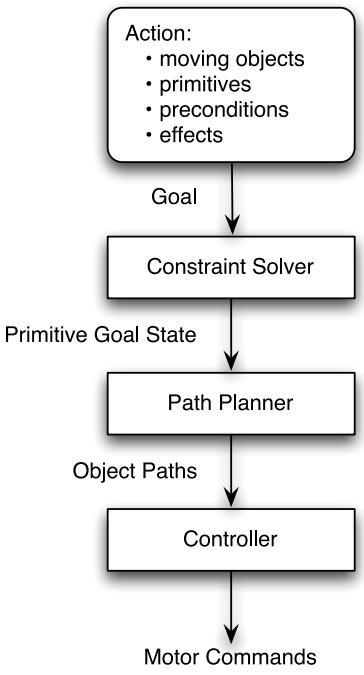

Figure 3: Behaviour generation.

can be extended to incorporate these kinds of closed-loop control learning.

Learning by Explanation

The aim of learning by explanation is to identify novel tool actions in the teacher's demonstration and to construct an action model that describes it. This includes identifying the sub-goal that the tool achieves and some of the necessary preconditions for using the tool. It involves the following steps: (1) watch a teacher using a tool to complete a task; (2) identify actions in the teacher's demonstration by matching them against actions stored in the robots plan library; (3) any sections of the demonstration that cannot be matched to known actions are labelled as components of a novel action; (4) a STRIPS model for the novel action is constructed by identifying the subset of literals in the action's start and end states that are relevant to explaining how the teacher achieved the goal.

Identifying novel tool actions in the teacher's demonstration

To recognise novel behaviours, the robot begins by trying to explain what the teacher is doing during its demonstration. The robot constructs an explanation by trying to match its own set of action models to the execution trace of the teacher's demonstration. Gaps in the explanation, where the agent is unable to match an existing behaviour to what the teacher is doing, are designated novel behaviours, which the robot can try to learn.

The first difficulty faced by the learner is in labelling the parts of the demonstration that it recognises. The demonstration trace is provided to the agent as a discrete time series $w_1, w_2, \ldots, w_n$ of snapshots of the low-level world state taken every tenth of a second. This world state is comprised of the positions and orientations of each object in the world at a particular point in time. Note that we do not provide the learner with a demonstration trace that is nicely segmented at an abstract level. Rather, the trace is a sampling of a continuous time series of object poses and the learner must arrive at a segmentation of the trace by itself.

The trace is segmented into discrete actions by applying an heuristic, that action boundaries occur at points at which objects start or stop moving. Thus, when the agent grips a stick-tool and starts moving it towards the tube, an action boundary is recognised. When the tool makes contact with the box and causes it to start moving also, another segment boundary is recognised. In this way, the robot constructs a very general movement-based description of the actions of the teacher. In the case of the tube problem, the segments correspond to the robot moving to pick up the stick, placing it against the box, pulling the box from the tube, placing the stick down and finally, moving to pick up and carry away the box.

Once the learner has segmented the teacher's trace, it attempts to match segments to its existing set of action models. Each of the robot's action models incorporates a STRIPS model along with a list of the objects that are moved during the action. A model is matched to a segment by checking that three conditions are met. The moving objects in the demonstration match the model. The preconditions of the model are true at the start of the action segment. The effects of the model have been achieved by the end of the action segment. In the unusual case where more than one abstract action matches a segment, the segment is labelled with the action that has the most specific set of preconditions. Segments that cannot be matched to any existing action model are labelled as components of a novel action. In the tube example, this produces the following labelling of the teacher's demonstration:

> goto(stick), pickup(stick),
> **novel-action(stick,box)**,
> drop(stick), goto(box), pickup(box).

where **novel-action(stick,box)** is actually a compound action involving two unrecognised actions. In the first, the stick is moved by the teacher (positioning the tool) and in the second, the stick and box are moved together (the actual tool action that pulls the box from the tube). The learner must now try to explain how this compound tool action was used to achieve the goal of acquiring the box. It does so by constructing two action models, for positioning and tool use, that are consistent with the other actions in its explanation of the demonstration.

Constructing novel action models from explanations

We use an heuristic that actions occurring before a novel action should enable the novel action's preconditions. Similarly, the effects of the novel action should help enable the

preconditions of actions occurring later in the demonstration. This heuristic is based upon the assumption that the teacher is acting rationally and optimally, so that each action in the sequence is executed in order to achieve a necessary sub-goal. The program constructs a STRIPS model by examining the start and end states of the novel action and identifies relevant literals in the actions executed prior to and after the novel action. The effects of the novel action are defined as any state changes that occur during the action that support an action precondition later in the demonstration.

In the case of the tube problem, ¬**in(box,tube)** becomes true during the novel action segment. This effect enables the preconditions of the **pickup(box)** action that occurs later in the demonstration. In general, there may be many irrelevant effects. However, this explanation-based reasoning allows us to eliminate all but the effects that were important for achieving the goal.

The preconditions of the novel tool action are constructed by a similar argument. The learner examines the world state at the start of the novel action and identifies the subset of state literals that were produced by the effects of earlier actions. The effect **gripping(stick)** occurs before the novel action and is still true at the start of the action. Since it is a known effect of the **pickup(stick)** action, it is considered a relevant literal to include in the novel action preconditions.

A common pattern for many tool-use problems is that there is a positioning step, followed by the application of the tool. We use this as a template for constructing two action models for the tube problem. The preconditions and effects identified in the demonstration are converted to literals with parameters by simply substituting each instance of an object with a unique variable.

position-tool(Tool,Box)
PRE in-gripper(Tool), gripping,
ADD **tool-pose(Tool,Box)**, obstructing(Tool,Box)
DEL –

The first action represents the robot getting itself into the correct position so that the tool can be applied. Note that we have not yet determined what that position is. The predicate **tool-pose(Tool,Box)**, which expresses this position, will be learned in the experimentation stage. A side-effect of this action is that the tool is obstructing the object. Later, when the robot tries to pick up the object, this side-effect will have to be undone. The tool action is:

pull-from-tube(Tool,Box,Tube)
PRE **tool-pose(Tool,Box)**,
gripping(Tool), in-tube(Box,Tube)
ADD: –
DEL: in-tube(Box,Tube)

The **tool-pose(Tool,Box)** effect of the position-tool action becomes a precondition of the tool action. There is only one subgoal of the tool action in this case, corresponding to an object being removed from the tube. This model becomes the starting point for learning the **tool-pose(Tool,Box)** predicate, which we describe next.

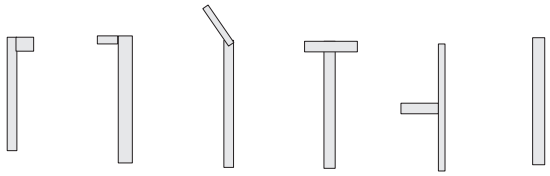

Figure 4: Examples of tools for pull-from-tube task.

Learning by Experimentation

The robot must learn the *tool pose state*, i.e. the state in which the correct object has been selected as the tool and the pose in which it can be applied successfully. It does so by testing a variety of tools and tool poses in a series of learning tasks. Thus, trial-and-error learning has two components: generating new learning tasks and updating the robot's hypothesis for the description of the **tool-pose** concept depending on the outcome of the experiment. The process for learning the concept describing the tool pose state is as follows:

1. Select a tool that satisfies the current hypothesis for the type and pose of the tool and place it in a pose defined by the hypothesis. Generate a motion plan that solves the task from this state and execute it, observing whether the action sub-goal is achieved.
2. If the action achieves the desired sub-goal, label the initial state as a positive example. If the action fails to achieve the desired sub-goal, label it as a negative example.
3. If the example is positive, generalise the hypothesis. If the example is negative specialise the hypothesis.
4. If the current task has been successfully solved, a new learning task is generated and presented to the robot. If the robot has failed, the current task is reset.
5. The robot continues its experimentation until the agent is able to solve a pre-defined number of consecutive tasks without failure.

The robot's world contains a selection of tool objects available for solving the problem. The dimensions and shapes of the tools are chosen randomly according to a type definition specified by the user. Figure 4 shows some tools that are available for the tube task. Only the two hook tools, on the left of the figure, are suitable for solving the task.

Incremental learning begins with an initial hypothesis for the concept being learned and successively generalises or specialises that hypothesis depending on whether a new training example is positive or negative. When the algorithm encounters a positive example that is not covered by its current hypothesis, the hypothesis must be generalised to cover the new example. If the example is negative and is covered by the current hypothesis, it must be specialised to exclude the negative example.

There are different ways in which we can represent the current hypothesis. One way is to store a clause in first-order logic and generalise or specialise it, as required. An alternative method, first introduced by Mitchell (Mitchell 1982), is to maintain a *version space*, which is the set of all possible hypotheses that are consistent with the training examples

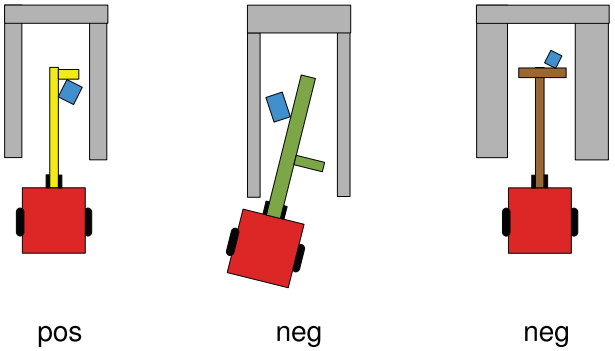

Figure 5: Positive and negative examples of tool use.

seen so far. We do not have to explicitly store all hypotheses. Instead, we only need to keep the most general boundary and the most specific boundary of this set. A generality ordering on expressions in the representation language then allows us to construct all the hypotheses, if needed. The most-specific and most-general hypotheses are represented as clauses of a logic program:

$$h_S \leftarrow \text{saturation}(e_0).$$
$$h_G \leftarrow \text{true}.$$

e_0 is the initial example derived from the teacher's demonstration. The saturation of e_0 is the set of all literals in e_0 plus all literals that can be derived from e_0 and the robot's background knowledge. The most-general clause, initially, covers ever possible instance of the hypothesis space. Now, when a new positive example is found not to be covered by h_S, h_S is generalised by finding the least general generalisation of h_S and the new example (Plotkin 1970). When a new negative example is encountered that is covered by h_G, h_G is specialised by borrowing literals from h_S in such way as to create the minimal specialisation that will exclude the example.

The training examples are generated by the robot itself when it conducts an experiment to test a new hypothesis. If it creates a training example that is not too different from previous positive examples, it is more likely to also be positive. So a conservative method of learning is to test examples that are close to the most-specific boundary. This process is divided into two. The tool pose hypothesis incorporates both structural literals (describing the physical composition, structure, and shape of the tool) and spatial literals (describing how the tool should be placed relative to other objects). We first select a tool that best matches h_S and then try to select a tool pose that matches as closely as possible that specified in h_S. To select a tool, each available tool object is scored for suitability according to the number of literals in h_S it satisfies. It must also satisfy all of the structural constraints in h_G. In a similar way, pose attempts to maximise the number of satisfied literals in h_S. In this case, however we are interested in satisfying the greatest number of spatial literals. This will allow the agent to place its selected tool in a pose that is as similar as possible to the previous positive examples it has observed.

Once the tool and pose have been selected, a plan is generated and executed by the robot. Figure 5 shows three examples generated for the tube problem. If the plan succeeds, we have a new positive example, which is used to generalise h_S. If it fails, h_G must be specialised to exclude the new negative example.

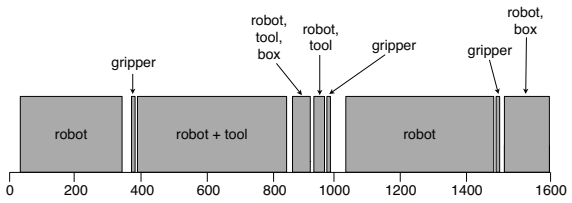

Figure 6: Robot and object motion during the teacher's demonstration. Units are 10ths of seconds.

The most-specific clause is built using only positive examples. We use a constrained form of least general generalisation (Plotkin 1970) to find the minimal generalisation that covers the positive examples. The most-general clause is built by generalising the most-specific clause. Negative-based reduction is used to simplify the most-specific clause without covering additional negative examples. This is a process of successively eliminating literals that do not change the coverage of the clause. We use negative-based reduction to recreate h_G when a new negative example arrives.

The algorithm borrows heavily from GOLEM (Muggleton and Feng 1992). The main difference is that GOLEM's bias is to learn the shortest, most-general hypothesis possible. In contrast, we maintain both a most-specific and most-general boundary on the hypothesis space. The algorithm is rerun after each example is received and the current hypothesis is rebuilt.

Evaluation

This system has been implemented and tested on several learning problems using the Gazebo robot simulator (Koenig and Howard 2004). The code is mostly written in Prolog with low-level control written in C++. The robot is a Pioneer 2, equipped with a simple gripper. Although this robot is quite limited in its ability to manipulate objects, it clearly demonstrates the idea that tools can enhance the range of problems that a robot is able to solve. Indeed, robots with limited effectors often have a lot to gain by using tools to overcome these limitations. In this section, we trace the robot's learning on the pull-from-tube-task.

After the robot observes the trainer's demonstration, the first step is to segment the trace of the trainer's behaviour into discrete actions. Figure 6 shows the results of clustering object motions, as described earlier.

We describe the *primitive state* of the world in terms of positions and orientations of all the objects. From this representation, the robot builds an *abstract state* description that represent the properties and relationships that exist between objects and agents in the world. The abstract state description is expressed in first-order logic and is generated from a set of definitions provided to the agent as back-

Table 1: Explanations of teacher's demonstration.

Seg	*Moving objects*	*Explanation*
1	robot	*put_in_gripper(hookstick)*
2	gripper	*grip(hookstick)*
3	robot, hookstick	*recognise_carry_obj(hookstick)*
4	robot, hookstick, box	*??*
5	robot, hookstick	*move_obstacle(hookstick,box)*
6	gripper	*ungrip(hookstick)*
7	robot	*remove_from_gripper(hookstick), put_in_gripper(box)*
8	gripper	*grip(box)*
9	robot, box	*recognise_carry_obj(box)*

ground knowledge. Note that these definitions could also be acquired through learning but we do not attempt that here.

The robot can construct an explanation of the teacher's activities (Table 1) by matching abstract action models to the motion segments it has identified. An abstract model matches a segment if the action preconditions are true at the beginning of the segment and the effects have been achieved by the end of the segment. The moving objects in the segment must also match the moving objects named in the corresponding action model.

At this point, the explanations are turned into the incomplete action models **position-tool** and **pull-from-tool**. The robot now enters its experimentation phase in which it constructs h_G and h_S for the definition of the **tool-pose** predicate. Each new example causes the learner to update the version space. The sequence of experiments is show in figure 7. After twelve experiments, the h_G is:

tool-pose$_G$(Tool,Box,State) ←
in-tube-side(Box,Tube,Side,State),
attached-side(Tool,Hook,Side),
touching(Hook,Box,back,State),
attached-angle(Tool,Hook,rightangle),
attached-end(Tool,Hook,back).

This states that the tool must have a hook attached at a right angle and be on the same side as the target object in the tube. The hook must be touching the back of the object.

We evaluate the robot's performance by the number of experiments required before it can consistently solve a new task. Twelve experiments were required to learn **pull-from-tube**. A similar task, learning to extract an object from an open tube by pushing it through the tube, requires between 10 and 15 experiments, depending on random variations in the tools made available to the robot for its experiments. A third experiment was the classic Shakey problem of pushing a ramp up to a platform so that the robot can push a block off the platform. Between 9 and 13 experiments were required to learn that the ramp can be used as a tool to mount the platform.

Conclusions and Future Work

This research contains several novel features. We integrate tool-use learning and problem solving in a robot that learns

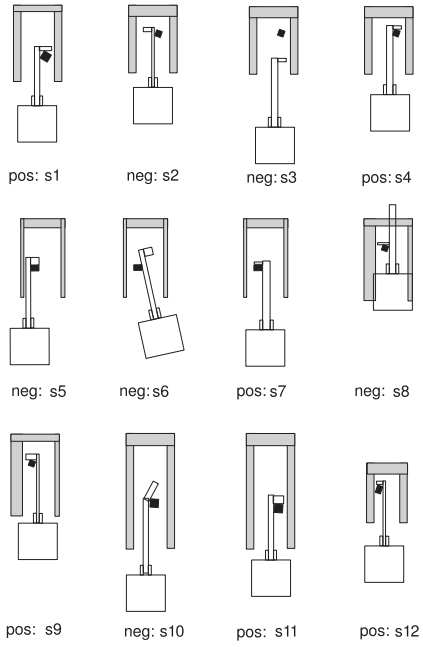

Figure 7: Examples generated during learning by experimentation.

new tool actions to help it solve a planning problem. We have devised a novel action representation that integrates symbolic planning, constraint solving and motion planning. The system incorporates a novel method for incremental learning that uses relative least general generalisation but adopts a version-space representation. This is the basis for a new method for generating examples in an incremental setting where cautious generalisation is desirable. Our approach is based upon sampling near the most-specific hypothesis boundary in the version space. In addition, our robot is able to infer the approximate form of a new action model by observing a teacher and examining the context of the plan in which it is executed. The robot also uses trial-and-error to refine this approximate model.

There are several important issues that we do not address here. Some tool-use behaviours require complex non-linear control of the tool or target object. Learning these types of behaviours is a field of research in itself. We have avoided them by choosing tasks that require simple manipulation that can be achieved with a generic controller. The learning tasks we attempt are restricted to those that can be easily implemented in a rigid-body simulation. Problems involving liquids or deformable bodies are not considered. Nevertheless, the general learning system is intended to be applicable to a wide range of scenarios. We do not try to handle learning by analogy. For example, using a ladder to reach a light bulb and using a stick to reach an object on a shelf are conceptually similar problems.

Another issue that we have avoided is uncertainty in sensors and actuators. Real sensors always have measurement errors and real actuators are never perfectly accurate. We can handle actuation errors to some extent by using reactive

controllers to implement the planners actions. However, a more robust solution may require systems capable of learning probabilistic representations, as in statistical relational learning or incorporating aspects of relational reinforcement learning.

References

Amir, E. 2005. Learning partially observable deterministic action models. In *International Joint Conference on Artificial Intelligence (IJCAI 2005)*, 1433–1439.

Apt, K. R., and Wallace, M. G. 2007. *Constraint Logic Programming using ECLiPSe*. Cambridge University Press.

Benson, S. 1996. *Learning action models for reactive autonomous agents*. Ph.D. Dissertation, Department of Computer Science, Stanford University.

Bicici, E., and St Amant, R. 2003. Reasoning about the functionality of tools and physical artifacts. Technical Report 22, NC State University.

Brown, S. 2009. *A relational approach to tool-use learning in robots*. Ph.D. Dissertation, School of Computer Science and Engineering, The University of New South Wales.

Fikes, R., and Nilsson, N. 1971. STRIPS: A new approach to the application of theorem proving to problem solving. *Artificial Intelligence* 2(3-4):189–208.

Gil, Y. 1994. Learning by experimentation: Incremental refinement of incomplete planning domains. In *International Conference on Machine Learning*.

Goodall, J., and van Lawick, H. 1966. Use of tools by egyptian vultures. *Nature* 212:1468–1469.

Hoffmann, J., and Nebel, B. 2001. The FF planning system: Fast plan generation through heuristic search. *Journal of Artificial Intelligence Research* 14:2001.

Hunt, G. 1996. Manufacture and use of hook-tools by New Caledonian crows. *Nature* 379:249–251.

Kenward, B.; Weir, A.; Rutz, C.; and Kacelnik, A. 2005. Tool manufacture by naive juvenile crows. *Nature* 433(121).

Koenig, N., and Howard, A. 2004. Design and use paradigms for Gazebo, an open-source multi-robot simulator. In *International Conference on Intelligent Robots and Systems*, volume 3, 2149–2154.

Krützen, M.; Mann, J.; Heithaus, M. R.; Connor, R. C.; Bejder, L.; and Sherwin, W. B. 2005. Cultural transmission of tool use in bottlenose dolphins. *Proceedings of the National Academy of Sciences* 102(25):8939–8943.

Kuffner, J., and LaValle, S. 2000. RRT-Connect: An efficient approach to single-query path planning. In *International Conference on Robotics and Automation*.

Langley, P., and Choi, D. 2006. Learning recursive control programs from problem solving. *Journal of Machine Learning Research* 7:493–518.

Levey, D.; Duncan, R.; and Levins, C. 2004. Use of dung as a tool by burrowing owls. *Nature* 431:39.

Levine, G., and DeJong, G. 2006. Explanation-based acquisition of planning operators. In *ICAPS*, 152–161.

Martin, M., and Geffner, H. 2000. Learning generalized policies in planning using concept languages. In *KR'00*, 667–677.

Mitchell, T. 1982. Generalization as search. *Artificial Intelligence* 18:203–266.

Muggleton, S., and Feng, C. 1992. Efficient induction of logic programs. In Muggleton, S., ed., *Inductive Logic Programming*. Academic Press. 281–298.

Muggleton, S. 1991. Inductive logic programming. *New Generation Computing* 8:295–318.

Pasula, H. M.; Zettlemoyer, L. S.; and Kaelbling, L. P. 2007. Learning symbolic models of stochastic domains. *Journal of Artificial Intelligence Research* 29:309–352.

Plotkin, G. 1970. A note on inductive generalization. In Meltzer, B., and Mitchie, D., eds., *Machine Intelligence*. Edinburgh University Press.

Ryan, M. R. K. 2002. Using abstract models of behaviours to automatically generate reinforcement learning hierarchies. In Sammut, C., and Hoffmann, A., eds., *International Conference on Machine Learning*, 522–529. Sydney: Morgan Kaufmann Publishers Inc.

Schmid, U., and Kitzelmann, E. 2011. Inductive rule learning on the knowledge level. *Cognitive Systems Research* 12(3-4):237 – 248. Special Issue on Complex Cognition.

Shavlik, J. W. 1990. Acquiring recursive and iterative concepts with explanation-based learning. *Machine Learning* 5:39–70.

Sinapov, J., and Stoytchev, A. 2008. Toward autonomous learning of an ontology of tool affordances by a robot. In *National Conference on Artificial Intelligence*.

Stoytchev, A. 2005. Behaviour-grounded representation of tool affordances. In *International Conference on Robotics and Automation*.

Toussaint, M.; Plath, N.; Lang, T.; and Jetchev, N. 2010. Integrated motor control, planning, grasping and high-level reasoning in a blocks world using probabilistic inference. In *ICRA*, 385–391. IEEE.

van Lawick-Goodall, J. 1970. Tool-using in primates and other vertebrates. In Lehrman, D.; Hinde, R.; and Shaw, E., eds., *Advances in the Study of Behaviour*, volume 3. London: Academic Press. 195–249.

Veloso, M.; Carbonell, J.; Pérez, A.; Borrajo, D.; and Blythe, J. 1995. Integrating planning and learning: The PRODIGY architecture. *Journal of Experimental and Theoretical Artificial Intelligence* 7(1):81–120.

Wood, A.; Horton, T.; and St Amant, R. 2005. Effective tool use in a habile agent. In *Systems and Information Engineering Design Symposium*. IEEE. 75–81.