Relational Tool Use Learning by a Robot in a Real and Simulated World

Handy Wicaksono[1,2] and **Claude Sammut**[1]
[1] School of Computer Science and Engineering, University of New South Wales, Australia
[2]Electrical Engineering Department, Petra Christian University, Indonesia
{handyw,claude}@cse.unsw.edu.au

Abstract

A robot's ability to perform complex manipulation tasks can be aided by being able to learn to use tools. This work aims to develop a complete robot system that can learn to use a simple tool from an observation of another agent employing a similar tool. Once the concept of the tool has been learned, the tool description is stored so that it can be used in future task planning. A relational learning system builds the tool description, which is generalised so that it can be applied under different conditions. Learning and planning are assisted by a physics simulator. Experiments are performed on a Baxter robot.

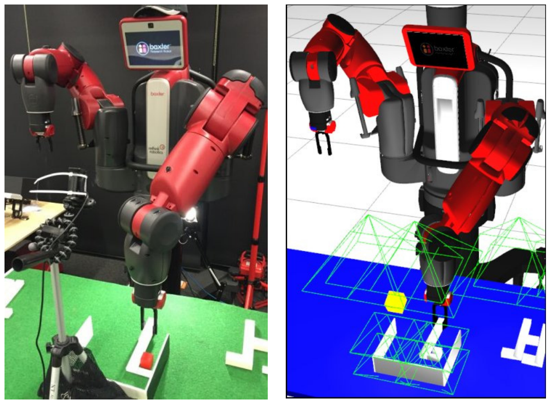

Figure 1: A physical and simulated Baxter robot learn to use a hook-like tool to pull a cube

1 Introduction

Humans use tools to help them complete every day tasks. In fact, the ability to make tools is a defining characteristic of human beings [Johnson-Frey, 2003]. When given a new task, we are capable of visualising the task in our minds and apply some form of reasoning so to plan how to achieve a goal [Hesslow, 2002]. A robot can also perform tasks that might not be feasible without the use of a tool. Like Brown & Sammut [2012] we construct a system that learns how objects may be used as tools by observing another agent performing the task. Whereas this previous work was only demonstrated in simulation, we have extended this approach to work on a real robot, a Rethink Robotics Baxter. However, simulation is still used to help the learning and planning systems test hypotheses before attempting any actions in the real world. Physical and simulated Baxter are shown on Fig. 1.

We define a tool as an object that is deliberately employed by an agent to help it achieve a goal that would otherwise be difficult or impossible to achieve. We develop a system that learns a tool action model which explains changes in the properties of one or more objects affected by the tool, given that certain preconditions are met. Starting from the initial observed example of tool use, the learner attempts to construct a general action model, testing hypotheses by trial and error. Inductive Logic Programming (ILP) [Muggleton, 1991] is used to construct and refine the action model.

A major difference between the previous simulation studies [Brown & Sammut, 2012], [Haber, 2015] and this work is that the simulators provided complete and exact information about the world. The simulations did not attempt to model sensor or actuator noise. We extend this approach to work on a real robot, dealing with all the uncertainties inherent in vision, gripping and motor actions. However, our learning system still uses a simulator to minimise experimentation in the real world [Sushkov & Sammut, 2012]. This work develops a framework and conducts experiments to investigate this issue. Our simulator has models of sensors and actuators that include noise. A useful analogy for simulation here is the way a human may try to visualise an action as a form of planning before attempting the action.

In the following sections we discuss relevant previous work, then we describe our representation of states and actions. The learning mechanism and experimental setup are explained next. Finally, we evaluate our approach with experiments on the real robot and in simulation.

2 Related Work

In previous work on learning to use tools, Wood et al. [2005] used artificial neural networks to learn the posture of a Sony Aibo robot so that it could reach an object by using a tool placed on its back. Stoytchev's [2005] system learned to select the correct tool by its affordances which were grounded in the robot's behaviours. However, the result could not be generalised to handle other new tools. Sinapov & Stoytchev [2007] overcame this weakness by learning a predictive model using k-NN and decision trees. Nevertheless, the generalisation was limited to different size of the same shape tool.

More recent work by Tikhanov et al. [2013] combined exploratory behaviours and geometrical feature extraction to learn affordances and tool use. The strength of their approach was in building a complete system that integrated perception and learning in the real world. Mar et al. [2015] extended their work by learning the grasp configurations that influenced the outcome of a tool use action.

Previous work used feature vector representations that describe an object's primitive attributes but are limited in their ability to describe relations between components of the tool and relations to other objects. Learning tasks have mostly been limited to tool selection. Only recently, there have been attempts to learn how to grasp the tool [Mar *et al.*, 2015]. Relational learning methods overcome some limitations of previous approaches [Brown & Sammut, 2012] and may even be extended to be able to design new tools [Wicaksono & Sammut, 2015].

As mentioned above, a simulator can be used to minimise learning in the real world that may be costly or dangerous. The system by Barrett et al. [Barrett *et al.*, 2010] performed reinforcement learning in a simulator and transferred its results to a humanoid robot that learned to hit a ball. Farchy et al. [2013] proposed a Grounded Simulation Learning framework where the simulator's parameters may be changed over time to match the real robot's behaviour. Sushkov & Sammut [2012] developed a system in which a robot performs experiments in simulation to determine which ones would yield the highest information gain if performed in the real world.

Almost all learning systems to date have been "one-shot" learners, that is, they only learn a single concept or behaviour. Our learning system is persistent, accumulating new concepts and using them to aid in further learning.

3 States and Actions Representation

We maintain a primitive and an abstract representation of states and actions. The primitive state representation includes the position of all objects in the world, which are extracted from camera images. To simplify perception, objects are predefined and can be detected by their two dimensional appearance only.

To be classified as a tool, an object must possess particular structural (e.g. side where a hook is attached on a handle) and spatial properties (e.g. a hook is touching back side of a cube). These are collected to describe the desired *tool pose*, expressed as a Prolog clause. For example:

```
tool_pose(Handle,Hook,Box,State):-
   attached_end(Handle,Hook,back),
   attached_side(Handle,Hook,Side),
   in_tube_side(Box,Tube,Side,State),
   in_tube_side(Hook,Tube,Side,State),
   touching(Hook,Box,back,State).
```

This described a hook that is positioned inside a tube and behind an object that is to be pulled out of the tube.

We use the STRIPS [Fikes & Nilsson, 1972] action model to describe tool actions:

PRE : condition that must hold so that the action can be performed

EFFECTS: conditions that become true or false as a result of performing the action

A simplified example of the `position_tool` action description is shown below.

```
position_tool(Tool,Box)
    PRE     : in_gripper(Tool), gripping
    EFFECTS : tool_pose(Tool,Box)
```

The abstract action does not provide the numeric goal. We acquire it by solving spatial literals in an action model's effect by using a constraint solver.

4 Tool Use Learning

In this section we describe tool use learning in the real and simulated world.

4.1 Learning Mechanism

In a cumulative learning framework [Sammut, 2013] an agent stores learned concepts and uses them as domain knowledge for planning and for future learning. Inductive Logic Programming is a good setting for this kind of learning as concepts are represented as Prolog programs, which can be stored in Prolog's database to be used as background knowledge. That fact that the representation is based on first order logic is not an impediment to its use in robotics as Prolog programs can call primitives that handle sensors and actuators. Most modern Prolog systems also incorporate constraint solvers that are extremely useful in mapping logical constraints into numerical values for motor actions.

The tool pose is critical to successfully applying a tool, so we will focus our discussion on learning that. The learner must first observe another agent performing a task using the tool. The system uses this example to form its initial hypothesis for the tool pose. Hypothesis formation adapts Mitchell's [1978] version space approach in that the initial example represents the systems most specific hypothesis (h_s) for the tool pose and the most general hypothesis (h_g), at first, covers everything. The system refines its hypothesis by conducting experiments. It may generalise the most specific hypothesis, then construct an example that is consistent with its generalisation.

In our case, the example is a tool action. When this is executed, if the result is what that action model predicted, then the generalisation is retained. If the result is not as predicted, the generalisation is rejected. Similarly, the most general hypothesis can be specialised. Variations on the experiments may be repeated to increase confidence in the results. The learning algorithm is a modification of Haber's [2015] and is shown in Algorithm 1.

Algorithm 1 Tool use learning in the simulated and real world

```
 1: input: new action model $M$,$N$ trials,
        $K$ consecutive success, $h_g$ = true,
        $h_s$ = preconditions of $M$,
        $medium$ = experimental medium
 2: for each $medium$ (simulated & real world) do
 3:     while $success < K$ or $index < N$ do
 4:         $e = generate\_experiment(h_s, h_g)$
 5:         $tool = select\_tool(h_s, h_g)$
 6:         for each $e_i$ in $e$ do
 7:             $pose = select\_pose(e_i)$
 8:             if $pose$ = null then
 9:                 break
10:         $success = execute\_exp(tool, pose)$
11:         if $success$ then
12:             label $pose$ positive
13:             increment $cons\_success$
14:             $h_s$ = generalise $h_s$
15:         else
16:             label $pose$ negative
17:             $cons\_success \leftarrow 0$
18:             $h_g$ = specialise $h_g$
19:         add $e_i$ to training data
20:         increment trial
```

4.2 Experimental hypotheses generation

Experimental hypotheses must be generated for learning. This requires the selection of a structure and pose for a tool. An object that matches the most structural properties in the tool pose is chosen as the tool. More specifically, we test whether an object satisfies all primary structural properties in h_g, and most of the secondary ones in h_s. The tool selection algorithm can be seen in Algorithm 2 [Haber, 2015].

Algorithm 2 Tool selection

```
1: input: boundary clauses $h_s$, $h_g$, world state $W$
2: for each $object$ in the world do
3:     $score \leftarrow$ structural similarity score($h_s, h_g, object$)
4: $bestTool \leftarrow$ object with highest score
5: return $bestTool$
```

Having a correct tool is useless if it is not positioned correctly. To select the correct pose, firstly, we collect the spatial literals in h_g then append to it the spatial predicates in h_s. Those predicates are then used to construct constraints for a constraint solver. The purpose here is to convert logical constraints, such as "behind" or "on the right" into numerical targets that can be given to a path planner as a target position. Thus, "behind" can be interpreted as a range between some object and the back of the scene. If there are multiple constraints, the constraint solver generates region of feasible target points. Since any point in the region is as good as any other, a random selection is made at this point. Algorithm 3 [Haber, 2015] shows the pseudocode for pose selection.

Algorithm 3 Pose selection

```
 1: let $A_1, A_2, \ldots, A_n$ denote spatial literals in $h_g$
 2: let $B_1, B_2, ..., B_n$ denote a shuffled list of spatial
        literals from $h_s$
 3: let $C$ represent the current set of spatial cons-
        traints to be applied
 4: add the most-general constraints to $C$
 5: solve $C$ and let $P$ represents tool pose solution
 6: for each successive literal $B_i$ in $B_1, ..., B_n$ do
 7:     if $B_i$ is already true in $P$ then
 8:         discard $B_i$
 9:     else
10:         append $B_i$ to the end of $C$
11:         find an updated $P$ to constraints in $C$
12:     if no solution $P$ exists then
13:         remove $B_i$ from the end of $C$
14: return the solution pose $P$
```

4.3 Inductive Logic Programming

The learning method is a form of Relative Least General Generalisation (RLGG) [Plotkin, 1971; Brown, 2009]. h_s is refined by finding the constrained Least General Generalisation (LGG) of a pair of positive examples, and h_g is refined by performing the negative-based reduction [Muggleton & Feng, 1990].

5 The Robot

In this section we describe the experimental platform, including its software architecture, object detection mechanism, and behaviour generation method.

5.1 Software Architecture

The software architecture is shown on Fig. 2 and consists of three layers [Bonasso *et al.*, 1997]. The ILP learning system, task planner and constraint solver comprise the top layer. The planner generates the sequence of abstract actions, however, it does not specify how they should be executed. The second maps the abstract actions into behaviours that are contained in third layer.

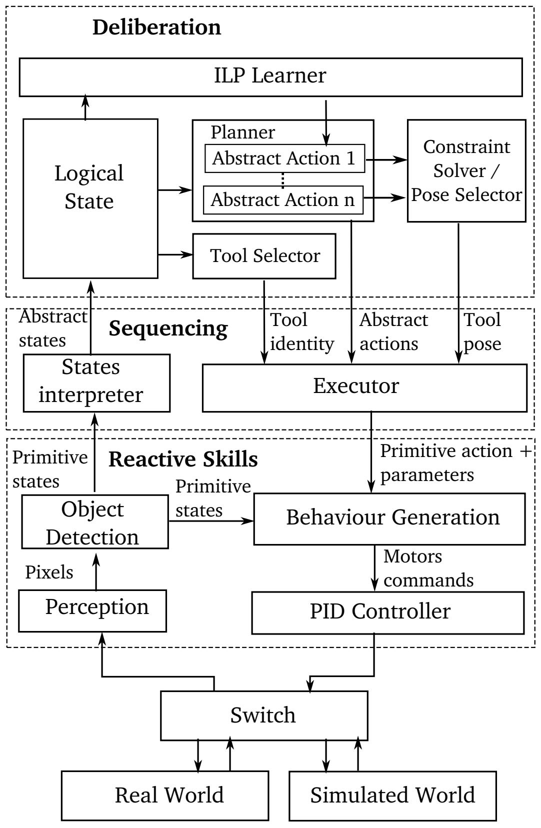

Figure 2: Robot software architecture consists of three layers: deliberation, sequencing, and reactive skills

The third layer also contains the perception modules, which relies on the OpenCV library, while the primitive behaviours use Robots Operating System (ROS) packages [Quigley *et al.*, 2009] for solving Inverse Kinematics (IK) and performing pre-defined joints movements.

We utilise a generic control program to perform the experiments either in the real or simulated world. Gazebo is chosen as the physics simulator because its tight integration with ROS. The simulated Baxter robot and the world's objects are developed in Unified Robot Description File (URDF) which are spawned into a Gazebo world.

5.2 Object Detection

To simplify object detection, the system uses a downward looking camera to match their 2D outlines. We use the Baxter in-hand camera for object recognition and visual servoing [Hutchinson, 1996]. An external camera provides a global view of the scene when Baxter camera is occluded. As both cameras have limited fields of view, their images are combined.

In pre-processing, the image is smoothed by Gaussian filter. A Canny edge detector is then applied, and their contours are acquired. Each contour is examined to determine its category (cube, tube, tools, or none of them). To distinguish them, we evaluate several properties of a contour: the number of sides, the area, the angle formed by two adjacent lines, the convexity, and the perimeter.

If a potential contour is detected, we acquire two corner points that have smallest and largest sums $((X_{min}, Y_{min}), (X_{max}, Y_{max}))$ as its representation. Special treatment is given to the tools, as they have different shapes. Each shape has one handle and, at least, one hook. The pseudocode for our object detection is shown in Algorithm 4.

Algorithm 4 Object detection

```
 1: Capture an image
 2: Perform Gaussian smoothing
 3: Detect the edges of objects using Canny detector
 4: Find the contours of the objects
 5: for each contour do
 6:     Classify the contour
 7:     if a contour is a tube then
 8:         find its min. & max. corner points
 9:     else if a contour is a cube then
10:         find its min. & max. corner points
11:     else if a contour is a tool then
12:         find its handle and hook(s)
13:         find their min. & max. corner points
14: return all min. & max. corner points
```

5.3 Behaviour Generation

An abstract action does not specify the numerical values required to drive motors. However, the can be obtained by using a constraint solver to satisfy the spatial literals in an action model's effect. In this project, we use the eCLiPSe [Apt & Wallace, 2006] constraint solver. Visual servoing is also required to reach the target of an action. The complete behaviour generation process can be seen on Fig. 3.

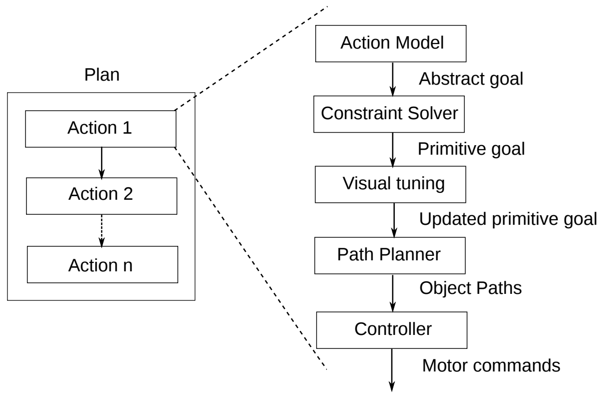

Figure 3: Behaviour generation process

6 Results and Discussion

Real and simulated Baxter robots are used in these experiments (see Fig. 1). The objects in the scene include a cube, a tube, and five tools with different shapes. We also have another set of tools whose hooks are narrower than their handles. The scene and the all the tools are shown in Fig. 4.

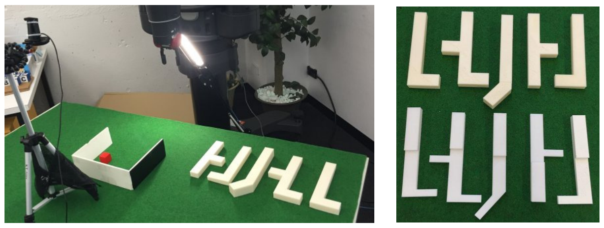

Figure 4: The scene and the complete tools

Two sets of experiments were conducted. Initially, we evaluated the performance of the object detection and action execution mechanisms. After that, we conducted tool use learning experiments in the real and simulated worlds. The task is for the robot to learn how to use the tool to pull a cube from inside a tube. Only the learning experiments are of interest here, as the preliminary experiments were only required to ensure the low-level routines were functioning properly.

For the learning experiments, we locate a cube in one of four different locations inside a tube: left-back, left-front, right-back, and right-front. In each trial, five objects with various width of hooks may be used as tools. We randomise the widths to test whether the learner is able to determine which tool properties are useful.

Initially, the real robot observes an example from a teacher, then experiments are performed in the simulation to create a hypothesis that explains the actions. The hypothesis is then tested and refined in the real world. We add Gaussian noise ($\mu = 0.0$, $\sigma = 0.21$) in our simulated camera image to represent a realistic physical one. To get accurate results and minimise learning errors in the real world, we repeat a trial in the same scenario five times in simulation, and choose the action with the highest probability of success.

Experiments proceed as follows:

1. Choose the experiment medium:
   (a) Real world
   (b) Simulation
2. Locate the cube inside a tube and select five tools with random various width of hooks
3. Perform a tool use experiment
4. Get the result and pass it to an ILP learner
5. If the experiment is success, locate the cube in new location, otherwise place it in the same location.

We start with the initial action model using the method of Brown & Sammut [2012] via Explanation Based Learning (EBL). We label an example as positive if the cube is located outside the tube at the end of the trial. Otherwise, it is negative. We conclude the experiment in each medium when the robot successfully accomplishes the task four times in a row. Figure 5 shows whole trials conducted with the real and simulated robot.

Sixteen trials are required to learn this tool use task. The first positive example is given by the teacher in the real world. Then, in simulation, the robot tries to find a similar tool and its location in second trial, but it fails as the cube location is changed. After several failures, in trial 6, the robot learns that the attachment side of the hook should be on the same side as the cube's location. The generalisation is shown below.

Trial 1:

```
in_tube_side1(Box,Tube,left,State)
in_tube_side2(Hook,Tube,left,State)
```

Trial 6:

```
in_tube_side1(Box,Tube,right,State)
in_tube_side2(Hook,Tube,right,State)
```

Generalisation result:

```
in_tube_side1(Box,Tube,Side,State)
in_tube_side2(Hook,Tube,Side,State)
```

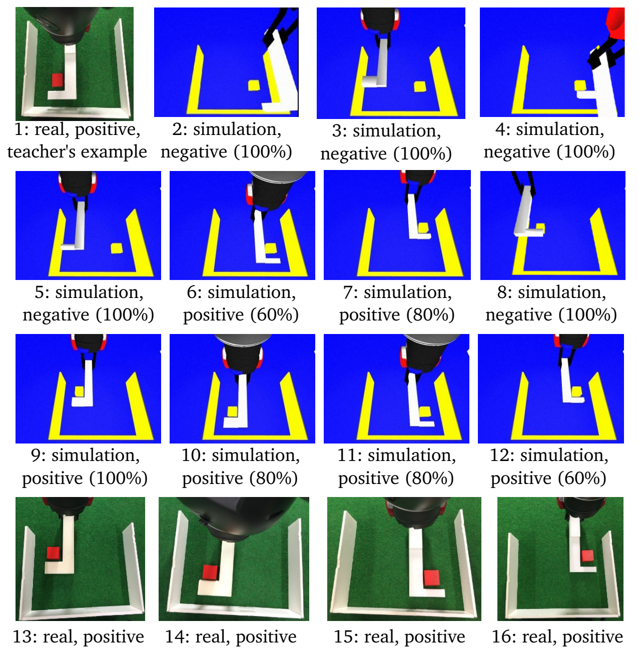
1: real, positive, teacher's example
2: simulation, negative (100%)
3: simulation, negative (100%)
4: simulation, negative (100%)
5: simulation, negative (100%)
6: simulation, positive (60%)
7: simulation, positive (80%)
8: simulation, negative (100%)
9: simulation, positive (100%)
10: simulation, positive (80%)
11: simulation, positive (80%)
12: simulation, positive (60%)
13: real, positive
14: real, positive
15: real, positive
16: real, positive

Figure 5: The complete learning trials in the real and simulated world. Each simulated trial is repeated five times and the highest probability result is chosen.

In trial 7, the robot learns that the hook must be located at the same end of tube as the cube. Another failure occurs in trial 8, as the robot tries to use a tool with a narrow hook. This is changed in trial 9, where a tool with a wide hook successfully pulls the object. The hypotheses remain the same after that, and the robot achieves four consecutive successes so the simulated experiments are stopped. Further experiments are conducted in the real world, and another four positive results are acquired. The descriptions of the initial and final hypotheses are shown below. It is obvious that the latter version is shorter and more specific than the former one.

Initial hypotheses:

```
tool_pose_hg(Handle,Box,State) :-
 true.

tool_pose_hs(Handle,Box,State) :-
 attached(Handle,Hook),
 attached_side(Handle,Hook,right),
 attached_end(Handle,Hook,back),
 attached_angle(Handle,Hook,right_angle),
 num_attachment(Handle,Hook,Hook2,1),
 narrower(Hook,Handle),
 in_tube1(Box,Tube,State),
 in_tube2(Hook,Tube,State),
 in_tube_side1(Box,Tube,right,State),
 in_tube_side2(Hook,Tube,right,State),
 in_tube_end1(Box,Tube,back,State),
 in_tube_end2(Hook,Tube,back,State),
 touching1(Hook,Box,back,State),
 touching2(Handle,Box,right,State).
```

Final hypotheses:

```
tool_pose_hg(Handle,Box,State) :-
 attached_side(Handle,Hook,Side).

tool_pose_hs(Handle,Box,State) :-
 attached(Handle,Hook),
 attached_end(Handle,Hook,back),
 attached_angle(Handle,Hook,right_angle),
 num_attachment(Handle,Hook,Hook2,1),
 in_tube1(Box,Tube,State),
 in_tube2(Hook,Tube,State),
 in_tube_side1(Box,Tube,Side,State),
 in_tube_side2(Hook,Tube,Side,State),
 in_tube_end1(Box,Tube,End,State),
 in_tube_end2(Hook,Tube,End,State),
 touching1(Hook,Box,back,State),
 touching2(Handle,Box,Side,State).
```

The results confirm that, as long as the simulation is reasonably accurate, we will be able to perform the early trials in the simulator (eleven trials, each is repeated five times), only requiring the real robot to do the final refinements (four trials). We also gain the advantage of having general knowledge representation in Horn clauses so it can be transferred to different experimental mediums easily.

Despite the successful results, several errors can occur in our experiments. The lighting and improper pose of objects can lead to a fault object detection, while failure of the inverse kinematics solver to produce a solution leads to invalid actions. One important issue that we do not address here is how to automatically create an accurate model of the world just from the robot's perception.

7 Conclusion and Future Work

A complete robot system has been developed to facilitate tool use learning in the real world. A physics simulator is used to assist the system so that trials in real world can be reduced. To learn in both worlds, we developed a generic control program, separated learning and control layers, and used a general knowledge representation system.

In future work, we intend to model the real world objects in the simulation automatically by utilising information from the vision system. We will also investigate creating new tools by 3D printing when a suitable tool is not available.

Acknowledgements

Handy Wicaksono is supported by The Directorate General of Resources for Science, Technology and Higher Education (DG-RSTHE), Ministry of Research, Technology, and Higher Education of the Republic of Indonesia.

References

[Johnson-Frey, 2003] S.H. Johnson-Frey. What's so special about human tool use? *Neuron*, 39(2):201–204, 2003.

[Hesslow, 2002] G. Hesslow. Conscious thought as simulation of behaviour and perception *Trends in cognitive sciences*, 6(6):242–247, 2002.

[Brown & Sammut, 2012] S. Brown & C. Sammut. A relational approach to tool-use learning in robots. In *Proceedings of Inductive Logic Programming*, pages 1-15, 2012. Springer Berlin Heidelberg.

[Muggleton, 1991] S. Muggleton. Inductive logic programming. *New Generation Computing*, 8(4):295-318, 1991.

[Haber, 2015] A. Haber. *A system architecture for learning robots*. Ph.D. Dissertation, School of Computer Science and Engineering, The University of New South Wales, 2015.

[Sushkov & Sammut, 2012] O. Sushkov & C. Sammut. Active Robot Learning of Object Properties In *Proceedings of IROS*, pages 2621–2628, 2012. IEEE.

[Wood *et al.*, 2005] A. B. Wood, T. E. Horton, & R. S. Amant. Effective tool use in a habile agent. In *Proceedings of In Systems and Information Engineering Design Symposium, 2005 IEEE*, pages 75–81, April 2005. IEEE.

[Stoytchev, 2005] A. Stoytchev. Behavior-Grounded Representation of Tool Affordances In *Proceedings of ICRA*, pages 3060–3065, April 2005, Barcelona, Spain. IEEE.

[Sinapov & Stoytchev, 2007] J. Sinapov & A. Stoytchev. Learning and generalization of behavior-grounded tool affordances in *Proceedings of 6th ICDL*, pages 19–24, July 2007. IEEE.

[Tikhanov *et al.*, 2013] V. Tikhanoff, U. Pattacini, L. Natale, L., & G. Metta. Exploring affordances and tool use on the iCub. in *Proceedings of IEEE-RAS International Conference on Humanoid Robots*, 2013.

[Mar *et al.*, 2015] T. Mar, V. Tikhanoff, G. Metta, & L. Natale. Self-supervised learning of grasp dependent tool affordances on the iCub Humanoid robot. in *Proceedings of ICRA*, 2015.

[Wicaksono & Sammut, 2015] H. Wicaksono, & C. Sammut. A learning framework for tool creation by a robot. in *Proceedings of ACRA*, 2015.

[Quigley *et al.*, 2009] M. Quigley, B. Gerkey, K. Conley, J. Faust, T. Foote, J. Leibs, E. Berger, R. Wheeler, & A. Ng. ROS: an open-source Robot Operating System. in *Proceedings of ICRA*, 2015.

[Barrett *et al.*, 2010] S. Barrett, M. E. Taylor, & P. Stone. Transfer learning for reinforcement learning on a physical robot. in *Proceedings of Ninth International Conference on 29 Agents and Multiagent Systems - Adaptive Learning Agents Workshop*, 2010.

[Farchy *et al.*, 2013] A. Farchy, S. Barrett, P. MacAlpine, & P. Stone. Humanoid Robots Learning to Walk Faster: From the Real World to Simulation and Back. in *Proceedings of of the 2013 International Conference on Autonomous Agents and Multi-agent Systems*, 2013.

[Fikes & Nilsson, 1972] R. E. Fikes & N. J. Nilsson. STRIPS: A new approach to the application of theorem proving to problem solving *Artificial intelligence*, 2(3):189-208, 1972.

[Sammut, 2013] C. Sammut The Child Machine vs The World Brain. *Informatica*, 37(1):3-8, 2013.

[Mitchell, 1978] T. M. Mitchell *Version spaces : an approach to concept learning.* Ph.D. Dissertation, Department of Electrical Engineering, Stanford University, 1978.

[Muggleton & Feng, 1990] S. Muggleton & C. Feng. Efficient induction in logic programs In *Proceedings of Inductive Logic Programming*, pages 368–381, 1990.

[Quinlan, 1990] J.R. Quinlan. Learning logical definitions from relations. *Machine Learning*, 5(3):239–266, 1990.

[Bonasso *et al.*, 1997] R.P. Bonasso, R.J. Firby, E. Gat, D. Kortenkamp, D.P. Miller, & M.G.Slack. Experiences with an architecture for intelligent, reactive agents. *Journal of Experimental & Theoretical Artificial Intelligence*, 9(2-3):237–256, 1997.

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

[Hutchinson, 1996] S. Hutchinson. A tutorial on visual servo control. *IEEE Trans. on Robotics and Automation*, 12(5):651–670, 1996.

[Plotkin, 1971] Plotkin, G. D. (1971). A further note on inductive generalization. In Meltzer, B. and Michie, D., editors, *Machine Intelligence 6*. Elsevier, New York.

[Apt & Wallace, 2006] K. R. Apt & M. Wallace. *Constraint logic programming using ECLiPSe.* Cambridge University Press, 2006.