Constructing Teleo-reactive Robot Programs

Krysia Broda[1] and **Christopher John Hogger**[2] and **Sam Watson**[3]

Abstract. This paper addresses the problem of synthesising programs for *teleo-reactive robots*. A robot of this kind possesses an internal ruleset, or program, which determines how it reacts to external stimuli. Although the robot possesses no inherent goal, its program is designed so as to predispose it towards achieving some overall desired effect, possibly in co-operation with others. It will usually have only a limited perception of the world in which it operates, and its reactive rules will not express any direct link between what it perceives and what its behaviour will achieve. Because of this, composing a suitable program by hand can require some ingenuity. This paper presents a formal framework within which such programs may be systematically constructed. It describes such a construction process and illustrates its application, including simulation in a Prolog environment.

1 Teleo-Reactive Robots

The behaviours of robotic agents can be contrasted in terms of how much reactivity they possess. Thus, a wholly reactive robot [3, 10] responds to a stimulus in a motiveless manner in simple accordance with its ruleset, whereas a proactive robot will plan comprehensively to achieve some goal. A range of interesting and practical behaviours lies between these extremes. In particular, teleo-reactive robots behave reactively but use rulesets devised with goal-oriented intent as in [9], which was the inspiration for this work. This paper presents a systematic procedure for the construction process. This section introduces the notation we use for our programs and illustrates its working. Section 2 describes the procedure, whilst Section 3 explains its various refinements using a *plan function*. Section 4 shows the procedure applied to an example and briefly describes a simulation environment. Finally, Section 5 describes current work and summarises our conclusions.

A teleo-reactive program consists of an ordered set of condition-action rules. Each one takes the form *conditions* $\rightarrow$ *action*, having a conjunction of conditions on the left and an action on the right. The rule is read as saying that the action can be taken provided the conditions are all satisfied. A robot controlled by the program commits to the action of whichever rule appears earliest in the program and has all its conditions satisfied. In order that the robot can remain always active the last rule in its program will always be, by convention, a *default* rule whose left-hand-side is *True*.

[1] Dept. of Computing, Imperial College, 180 Queen's Gate, London SW7 2BZ, UK. email: kb@doc.ic.ac.uk
[2] Dept. of Computing, Imperial College, 180 Queen's Gate, London SW7 2BZ, UK. email: cjh@doc.ic.ac.uk
[3] ING Barings Limited, 60 London Wall, London EC2M 5TQ, UK. email: Sam.Watson@ing-barings.com

The robot is assumed to possess sensors, which it uses to perceive the current state of both itself and its local environment; the sensory inputs are processed internally to determine the truth or falsity of the conditions in the rules. An action to which the robot commits will usually be such as to promote the environment to a new state. The robot's perceptions are presumed to be appropriate to the class of its supposed goals.

The particular program stored in the robot will depend wholly upon the nature of the application, but the prevailing assumption is that it will have been designed to enable the robot to take such actions as will assist progress towards the goal. This holds even when the environment is also being acted upon by other robots following their own agendas. A thorough overview of reactive robotic behaviour can be found in [1]. In this paper only single robot worlds are considered. Work is currently underway on the extension to multiple robot worlds.

Example In this example, posed in a conventional blocks-world, the goal is the building of a 3-tower (a tower comprised of 3 blocks) positioned arbitrarily upon the table. One or more robots may be acting to achieve this, but we will focus upon any one of them, named *Rob*. This is able to perceive whether or not it is holding a block (represented by *holding* and *empty*) and whether or not it is facing a tower of some size X ($size(X)$).

pick remove the top block from a tower and hold on to it;
place place a held block on the table or on top of a tower and release it;
wander wander around.

empty, size(1)	$\rightarrow$	pick
holding, size(2)	$\rightarrow$	place
holding, size(1)	$\rightarrow$	place
True	$\rightarrow$	wander

The effect of the default rule in this example is that before and after the constructive actions of picking and placing blocks there can be interludes of arbitrary duration when *Rob* simply wanders around. Equipped with the given program, the behaviour of *Rob* depends upon the initial state of the blocks on the table. Assuming that *Rob* is initially not holding anything, there are three cases to consider:

Initially all blocks lie separately upon the table In this case *Rob* will pick up the first block it finds, then find a second block on the table, then place the first block upon the second. It will then find and pick up the third block and place this on top of the existing 2-tower.

Initially all blocks are in a 3-tower In this case *Rob* can only wander around indefinitely.

Initially the blocks are in a 2-tower and a 1-tower
Rob will find and pick up the single block (the 1-tower) and place it upon the 2-tower.

After the goal state has been achieved *Rob* will wander indefinitely unless terminated by some extraneous control.

Suppose there are two robots at work, identically programmed as above. Now, either of them will build a 2-tower and then either of them will place the third block upon it. The goal will thus still be achieved. If there are three such robots, however, it is possible that all three will pick up distinct blocks, and thereafter be able only to wander around carrying them.

The difficulty of writing teleo-reactive programs sufficient to achieve particular goals can be gauged by trying to write one for the goal of building a 4-tower from 4 blocks using just one robot. It is not sufficient to extend the above program by adding an extra rule $holding, size(3) \rightarrow$ `place`. Whilst the goal might be achieved, it will not necessarily be – the robot could build two 2-towers and then be unable to progress further. One apparent remedy is to relax the conditions of a `pick` action so it may take a block from the top of any tower rather than take only a single block lying on the table. This, however, introduces the possibility of looping. Predicting the eventual achievement of the goal would then entail assumptions about the robot's propensity to wander around such as to escape, in the long run, from repetitive behaviour. If several robots were operating with this extended program they could undo each other's actions, and so engage in communal looping.

The difficulties arise not only because the robots are incognisant of the intended goals, but also because they are not, in general, equipped to sense the entire global state of the worlds in which they operate. Instead, they can operate only upon perceived states, that is, upon just those parts of the global state to which their sensors have immediate access. This is a consequence of the underlying presumption in the framework that the robots should be simple agents of limited resource, yet be carefully engineered to accomplish sophisticated tasks. For more discussion, see [11, 6].

Our work shares with [5] the notions of states, perceptions, actions and a program generation phase. However, their generation method relies on back-chaining from explicit goal descriptions at the object-level, whereas ours reasons with the whole evaluation space at the meta-level. Both of these are in contrast with [8], in which the programs react to goals by planning on the fly.

2 The Procedure

In this section we describe the program construction process with greater formality. There is an objective world – which includes the robot – capable of assuming various objective states, the totality of which is denoted by $\mathcal{O}$. For each o in $\mathcal{O}$ the robot can have various perceptions of o, according to where it is and what it is capable of sensing. The set of all possible logically-distinct perceptions of o is denoted by $per(o)$. For example, if o is the state in which there are two 1-towers, three 2-towers and the robot is empty, then two members of $per(o)$ might be ⟨size(1), empty⟩ and ⟨ size(2), empty ⟩.

Any perception should be satisfiable in its intended interpretation. This eliminates cases such as ⟨holding, empty⟩, which is unsatisfiable as the robot cannot be both holding and not holding. There may be many objective states of which the robot has a particular perception p. The set of all such states is $O(p) = \{o : o \in \mathcal{O}, p \in per(o)\}$. For example, if p is the perception ⟨size(1), empty⟩ then one member of $O(p)$ might be two 1-towers, three 2-towers and the robot empty, and another might be four 1-towers, one 4-tower and the robot empty. There must be an action which, when in some objective state, will allow a robot to shift its perception while remaining in the same objective state. We call this the `wander` action. In most of the cases considered in this paper the `wander` action is one that allows the robot to wander around its domain, although it may instead be a `wait` action. Often, `wander` is the action in the default rule, but it need not be.

Some assumptions are made to facilitate the program construction. These are:

1 The feasible actions that a robot may take at any instant are determined both by its inherent physical capabilities and by what (if anything) it is perceiving.
2 The result of every action is perceivable and it is what will be perceived immediately after the action. So, for example, after placing a block on a tower of $size(1)$ *Rob* will perceive a tower of $size(2)$ and after picking a block from a tower of $size(1)$ *Rob* will not perceive any tower.
3 Consequently, the feasible actions for a robot are independent of the objective state. Given its physical capabilities, the set of actions it may feasibly attempt is a function $A(p)$. In particular, the `wander` action w satisfies the property that $\forall p[w \in A(p)]$. The feasibility of any a in $A(p)$ is ensured by these stipulates: (i) whatever is perceived in p must be a feature of the objective state, and (ii) p is the logical precondition for a.

Principal Steps The program-building procedure employs further constructs termed the OP-graph and the plan function, both of which will be discussed presently. The procedure's overall scheme has the following steps:

1. Formulate the problem by establishing the contents of: the set $\mathcal{O}$ with the goal state(s) clearly identified, the set $per(o)$ for each o in $\mathcal{O}$ and the set $A(p)$ for each p in each $per(o)$.
2. Choose predicates to serve as descriptors of all these objective states, perceptions and actions.
3. Construct the OP-graph as detailed below.
4. Choose a suitable plan function $\mathcal{F}$ and apply it to the OP-graph. This step yields, for each possible perception p, exactly one rule $p \rightarrow a$ for some a in $A(p)$.
5. Simplify the rules as appropriate.

Constructing the OP-Graph The key construct used to guide the program development is a labelled directed graph that we call an OP-graph. Each of its nodes is a pair (o, p), where o is an objective state in $\mathcal{O}$ and p is a perception in $per(o)$. Any such node denotes a *situation*, combining the state of the world with what the robot perceives of it. A *goal node* is any node (o, p) for which o is a goal state.

The graph contains nodes for all situations admitted by the chosen problem formulation. From each node (o, p) there is a directed arc to every (o', p') for which an action a in $A(p)$ would take the objective world from state o to state o', and that arc is accordingly labelled by a. These arcs include those

labelled by the robot's `wander` action w, connecting (o, p) to every (o, p') for which p' is in $per(o)$; this signifies that the robot is at least capable (for instance, by random wandering) of making any of the perceptions in $per(o)$.

Applying the Plan Function The action the robot will take in any situation is determined by some chosen function $\mathcal{F}$ which we call the *plan function*. Given a situation (o, p) it returns some action in $A(p)$ and the robot commits to that action. $\mathcal{F}$ takes no account of o, being just a function from perceptions to actions. The action to which $\mathcal{F}$ commits may or may not be the "best" one the robot can take in any particular objective state. The choice of $\mathcal{F}$ powerfully influences the robot's efficacy, and various criteria for choosing it are discussed in Section 3. The point is that, for each distinct perception p the robot can make, $\mathcal{F}$ delivers exactly one rule $p \rightarrow a$ for some a in $A(p)$.

The application of $\mathcal{F}$ is equivalent to pruning edges of the OP-graph so as to leave a reduced OP-graph OPR, in which each node has only identically-labelled arcs emanating from it. Many nodes will be left with only one such arc, labelled by an action that changes the objective state. For nodes left with several emanating arcs, those arcs will typically all be labelled with the `wander` action, which enables the robot to alter its perception but not the objective state.

The rules obtained by the steps above may lend themselves to simplifications allowing a more compact program to be stored in the robot. The scope for simplification depends upon how the rules are first formulated and upon assumed constraints over perception descriptors.

3 Specifying a Plan Function $\mathcal{F}$

An OP-graph OPG is fixed by the given sets of Objective states, Perceptions and Actions. The term OPC refers to a *complete* OP-graph, in which all possible actions from a situation are indicated, whereas the term OPR refers to a *reduced* OP-graph which has been pruned by some plan function $\mathcal{F}_{\mathrm{OPG}}$. A plan function $\mathcal{F}$ is specified to satisfy the following:

a If $\mathcal{F}_{\mathrm{OPG}}(p) = a$ then $a \in A(p)$, which ensures that the action is feasible for the given perception.

b For every non-goal objective state o there is a perception $p \in per(o)$ such that $\mathcal{F}_{\mathrm{OPG}}(o, p) \neq$ `wander`, which ensures that the robot does not apply `wander` forever with no chance of changing the objective state. In the case of the goal state, it is likely that one does *not* want the robot to change the objective state, so action w is quite appropriate.

c If, for some perception p, each objective state o such that $p \in per(o)$ yields the same "best" action a, then $\mathcal{F}_{\mathrm{OPG}}(p) = a$. This "best action" property is easily justified. Suppose that $A(p)$ can be ordered by merit for an objective state. If a has the greatest merit in this order, whatever the objective state, then the robot should certainly take this action.

There are three further properties, the regression property, the limited troughs and the worst action properties, that are useful for determining the ability or otherwise of the robot to reach a goal state. These are best described with reference to the notion of *index* $I_{\mathrm{OPG}}(s)$ of a node s in an OP-graph OPG, which is the length of a shortest path from s to some goal situation unless no such path exists, when the index is ω.

The regression property of $\mathcal{F}_{\mathrm{OPG}}$ then states that, for each non goal situation s, there is a situation n that results from the action taken at s for the perception p of s (i.e. $\mathcal{F}_{\mathrm{OPG}}(p) = a$), such that $I_{\mathrm{OPG}}(n) < I_{\mathrm{OPG}}(s)$. For a goal situation g there is no situation n such that $I_{\mathrm{OPG}}(n) > I_{\mathrm{OPG}}(g)$. The regression property guarantees that each node is connected to a goal situation and also that once a goal situation is reached it is not acted upon to produce a non-goal situation.

A *trough* t is a connected subgraph of OPR comprising non-goal situations, having no paths emanating from t to any goal. A situation s belongs to a trough if its index $I_{OPR}(s) = \omega$. It is desirable to minimise the number of troughs, as their presence indicates a "dead-end" as far as achieving a goal state is concerned. In case the initial state is to be chosen at random, then it could be desirable also to minimise the number of different situations belonging to a trough, in order that the initial state shall not belong to a trough.

A further desirable property is concerned with situations in OPR derived from OPC that have an index $I_{OPR}(s)$ greater than the index $I_{\mathrm{OPC}}(s)$. Such arcs can arise when the "best" action for some situation is not selected by $\mathcal{F}_{\mathrm{OPG}}$. That is, when the "best" actions for a perception p differ for various objective states o where $p \in per(o)$. The difference between $I_{\mathrm{OPR}}(s)$ and $I_{\mathrm{OPG}}(s)$ should be minimised.

This feature, and the reduction and/or elimination of troughs, can often be encouraged by distinguishing explicitly between a perception p in two different objective states, which can sometimes be achieved by broadening perceptions to include memory (see Block Example). Instead of the same action for both states, different, more appropriate, actions can then be defined.

4 A Worked Example

An example of the construction procedure is given next, in some detail. The problem is to construct a tower of 4 blocks from an arbitrary collection of 4 blocks. Although simple, the example will illustrate most of the features discussed earlier. The robot in this case can carry out the same three actions

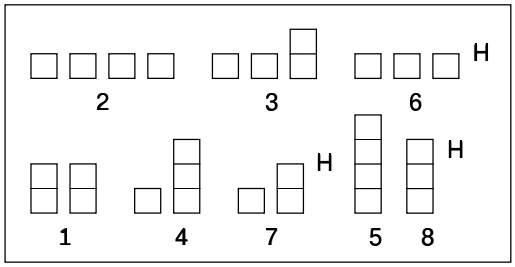

Figure 1. Objective States

of the introduction, namely to pick (from the table or from a tower), to place a block it is holding onto a tower (not the table) or to wander. It is assumed the robot can detect if it is holding something (*holding*) or not (*empty*) or facing a tower of size 1 ($size(1)$) and similarly for sizes 2 or 3. It can also detect if it is not facing a real tower (represented as facing a tower of $size(0)$).

The first step is to identify the perceptions $\mathcal{P}$ and the various objective states from the set $\mathcal{O}$ to which they belong. The objective states are in Figure 1 and the perceptions p and actions $\mathcal{A}(p)$ available to the robot at each perception p are in the table below. Because of the assumption made in Section 2, there is no need for the explicit perception $\{\neg holding, size(0)\}$ (represented as $\{\neg H, S(0)\}$), for the robot could perceive this only after wandering and before more wandering. However, there is a need for the perception $\{holding, size(0)\}$, for the robot would be in such a perception after picking up a block from a 1-tower. There are 8 objective states for this problem and 8 perceptions.

Name	Perception	Possible Obj. States	Possible Actions
a	$\{H, S(1)\}$	$\{6, 7\}$	`place`, `wander`
b	$\{H, S(2)\}$	$\{7\}$	`place`, `wander`
c	$\{H, S(3)\}$	$\{8\}$	`place`, `wander`
d	$\{\neg H, S(1)\}$	$\{2, 3, 4\}$	`pick`, `wander`
e	$\{\neg H, S(2)\}$	$\{1, 3\}$	`pick`, `wander`
f	$\{\neg H, S(3)\}$	$\{4\}$	`pick`, `wander`
g	$\{\neg H, S(4)\}$	$\{5\}$	`pick`, `wander`
h	$\{H, S(0)\}$	$\{6, 7, 8\}$	`wander`

The next step is to construct (explicitly or implicitly) the complete OP-graph. It is shown in Figure 2 and has 14 nodes, each consisting of an (objective state, perception) pair. The goal state is the pair $(5, g)$. The dotted edges are those that will be pruned by the selected plan function $\mathcal{F}$ as discussed next. Reflexive transitions involving `wander` that do not lead to new nodes are omitted for clarity, for wandering is significant only when it causes the robot to acquire a new perception.

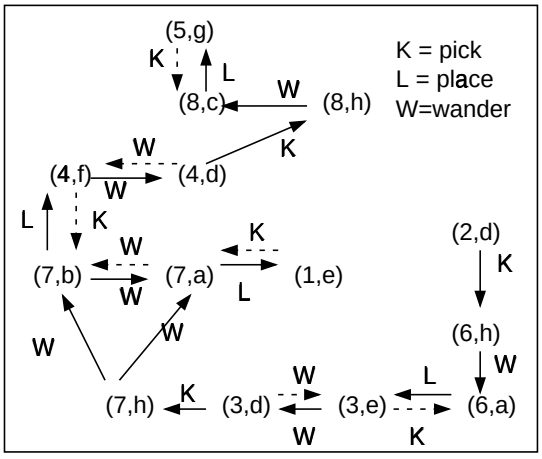

Figure 2. The Complete OP-graph OPC

From Figure 2 it can be seen that the goal state could be reached from any initial state as long as the robot selects a suitable action. The choices in the plan function come at the transitions from perceptions a and e. In all other cases, except for perceptions d and h, there is only one objective state and the best action can be easily selected. For d, the best action for each of the three objective states 2,3,4 is `pick`, so this is the one chosen. Similarly for h and the best action w. However, for a the best action depends on the objective state and is a different one for each of the two states, so there is a dilemma as to which action to choose. Similarly for perception e. Each of the four possible combinations yields a reduced OP-graph in which the goal is not reachable from all nodes . The 'best' case appears to be the one indicated by the solid edges in Figure 2, although it still possesses a trough, in that nodes $(7, a)$ and $(1, e)$ do not lead to the goal. But likewise for (3,e), (6,a), (6,h) and (2,d) in case the action of `pick` is selected in perception e. This particular trough could be eliminated only by distinguishing in some way between the perception e in objective states 1 and 3. In this case, reaching the goal state is not always assured as the robot is not equipped to distinguish between towers of the same height, so could wander in situation $(1, e)$ for ever. In some cases it may be possible for the robot to perceive it has reached a goal state, in which case a `null` or `stop` action could be included in $A(p)$ when $p \in per(g)$ for a goal state g. This would be most appropriate if the environment included just one robot. However, a robot may not always be able to perceive it has reached a goal state; for example, if the goal state had only towers of size (1) then, without memory, a robot could not know whether it had reached such a state. In that case the `wander` action would be the most appropriate.

A better program using local memory Typically, the problem of troughs can be partially overcome by enhancing the robot's perceptions with memories; in this example it is enough for the robot to be able to remember "having seen a tower of at least size 2", which we denote s_2. The action to update the robot's memory is intrinsic to the robot and need not be made explicit. The truth or falsity of the memory of s_2 will be treated as a constraint, used to choose the action in each perception. In the cases of d and g the same action is chosen whatever the memory value. For b, c, e and f, in case s_2 is initially false, then s_2 is implicitly reset to true, and the selected action is then performed. For a, the action for true s_2 is `wander` and for false s_2 it is `place`. Incorporating these changes into the initial OP-graph results in just two possible functions $\mathcal{F}$, depending on the action taken in e. The reduced OP-graph obtained using one of these functions results in a connected graph. The only situations in which the goal would still not be reached, assuming the robot starts out empty, are if the initial state were $(1, e)$, or $(3, d)$. The final program is as follows:

$\neg s2$, H, size(1)	$\rightarrow$	place
$s2$, H, size(1)	$\rightarrow$	wander
H, size(2)	$\rightarrow$	place
H, size(3)	$\rightarrow$	place
E, size(1)	$\rightarrow$	pick
$s2$, E, size(2)	$\rightarrow$	wander
T	$\rightarrow$	wander

which can be operationally simplified by removing rules 2 and 6 as they are covered by the last rule.

Simulated Implementation Robots programmed using this procedure have been simulated using our `Droneworld` system, which is a general framework able to simulate multiple logical agents. It is written in LPA MacProlog [7]. The agents possess normal-clause theories supplemented by constraints and type-checks. They possess their own theorem-provers, can support abduction and can import and export theorems proved by one another. The system provides a range of regimes for controlling how the agents operate and interact.

In particular, the regime we call Robotworld simulates state-space transitions by arranging that the agents' theorems are time-stamped and represent situations in the chosen domain (e.g. blockworld) at particular times.

Robotworld simulations have been undertaken for the examples in this paper and for others concerning multiple co-operating robots. Limitations of space here are such that we can give only a flavour of the robot programs as implemented. For instance, the first program for *Rob* given earlier is represented by these Prolog metarules:

```
rule(rob,[cond_action([e(rob),s(1)],pick)]).
rule(rob,[cond_action([h(rob),s(2)],place)]).
rule(rob,[cond_action([h(rob),s(1)],place)]).
rule(rob,[default_action(wander)]).
```

Here is a fragment of the output generated by Robotworld during the simulation of *Rob*'s attempt to build a 3-tower:

```
sit(3[[s(2),rob],e(rob),s(1)]) rob's action :wander
sit(4,[[s(1),rob],[s(2)],e(rob)]) rob's action :pick
sit(5,[[s(2),rob],h(rob)]) rob's action :place
```

The first line reports that after 3 time-steps the situation comprises a 2-tower and 1-tower, and that *Rob* is empty and is perceiving the 2-tower. It also reports that *Rob*'s action then is to wander, so yielding the next situation.

It is our intention in future work to automate the choice and application of the plan function as well as to define and evaluate suitable performance measures for the framework. In [4], artificial neural networks are used as a massively parallel logic programming system. Adapting these networks to provide a teleo-reactive system is also under consideration.

5 Some Final Comments

This paper has shown a systematic procedure for deriving a teleo-reactive program for a simple robot. By distinguishing between objective states and a robot's perceptions and formalising OP-graphs, various desired properties of the derived program were expressible, such as the regression property and the no-troughs property. The procedure was applied to the goal of building a tower of height 4. Furthermore, a simulation environment in Prolog has been built, which allows such programs to be run, for single or multiple robots.

Lest the reader think that the procedure is applicable only to robots building towers of bricks, it has been applied to controllers of processes, such as a multi-agent traffic light controller for a cross-roads and a temperature controller[2]. There is ongoing work in extending the framework to multi-robot situations. The particular features being considered are:

- Although a goal situation might be achievable by a single robot, several robots working together might achieve it more quickly. We call this *as if co-operation*, since the robots may appear to be co-operating even though they are not explicitly programmed to do so. By contrast, a goal situation might demand the explicit co-operation of and possible communication between several robots – perhaps having differing abilities. We call this *real co-operation*.
- If two robots have the same perception and can both execute an action, it may be the case that only one can perform the action at a time. Moreover, performing the action by one robot could invalidate the ability of the second robot to perform its previously applicable action due to a change of objective state and possibly of perception too. Synchronisation of the actions has to be addressed.
- Whereas one robot would never sensibly perform a wait action, unless there were external influences, within the multi-robot situation it is quite sensible to do so as the other robots may change the state.
- The procedure for generating programs could formalise perceptions from the point of view of each robot, or from a combined point of view, in which the perception of one robot included the perception of all other robots. If the latter approach were taken, the programs for individual robot could be formed by a projection of the combined program. On the other hand, if the former approach were taken, then the combined program could be synthesised from the individual programs. It is our intuition, based on our analysis so far, that consideration of two or a small number of robot cases is adequate for the analysis of larger numbers of robots. In view of this we do not believe that the scalability problem will be as severe as might otherwise be thought.

Other extensions to be investigated include the formalisation of hierarchical programs. In the robot programs in this paper, there are actions such as `pick`, or `wander`, that obviously themselves consist of several more basic actions. In the case of `wander`, the robot must presumably wander but avoid obstacles. In the case of `pick` it must first position itself correctly before grabbing the object. Such actions can themselves be written as teleo-reactive programs. But this introduces two more complications. First, should a robot complete an action before resuming its original program? This might not be the most productive way to operate the robot, if outside influences change the situation while it is carrying out some action, in which case resuming the original program could be the best course to pursue. Perhaps, even, a robot would not be able to finish its task due to some malfunction. Second, should a robot's program be divided into a hierarchy of programs or, once such a suite of programs has been defined, should they then be combined into a single program?

REFERENCES

[1] R. Arkin, *Behaviour -Based Robotics*, MIT Press, 1998.

[2] J. Atlee and J. Gannon, 'State-based model checking of event driven system requirements', in *Proceedings of the ACM SIGSOFT'91 Conf. on Software for Critical Systems*, (1991).

[3] R. Brooks, 'A robust layered control system for a mobile robot', *IEEE J. of Robotics and Automation*, **2**(1), (1986).

[4] A. d'Avila Garcez and G. Zaverucha, 'The connectionist inductive learning and logic programming system', *Applied Intelligence Journal*, **11**(1), (1999).

[5] L. P. Kaelbling and S. J. Rosenschein, *Behaviour -Based Robotics*, 35–48, MIT Press, 1991.

[6] T. Larsson, 'Robot wall following', Technical report, Dept. of Computing (DOC), Imperial College. MSc. Thesis, (1999).

[7] Logic Programming Associates Ltd. http://www.lpa.co.uk/.

[8] P. Maes, *Situated Agents can have Goals*, MIT Press, 1991.

[9] N. J. Nilsson, 'Teleo-reactive programs for agent control', *Journal of Artificial Intelligence Research*, **1**, (1994).

[10] M. J. Schoppers, 'Universal plans for reactive robots in unpredictable domains', in *IJCAI-87, San Francisco*, ed., Morgan Kaufmann, (1987).

[11] S. Watson, 'Reactive planning for multiple robots', Technical report, DOC, Imperial College. MSc. Thesis, (1999).