A Hybrid, Teleo-Reactive Architecture for Robot Control

Simon Coffey and Keith Clark

Imperial College London, SW7 2AZ, United Kingdom
{spc03,klc}@doc.ic.ac.uk

Abstract. In this paper we describe the structure of a proposed hybrid architecture for robot control. A BDI-style planning layer manipulates a plan library in which plans are comprised of hierarchical, suspendable and recoverable teleo-reactive programs. We also present preliminary simulation and implementation work.

Keywords: Hybrid robot architecture, BDI, teleo-reactive programming.

1 Introduction

Since their inception, behavioural robotics techniques have made great progress in many traditional areas of robotics research: localisation; navigation; search, etc. However, their application in the area of robot cooperation has been relatively limited in terms of producing truly versatile teams. This is not to say that behavioural approaches to multi-robot teams are uncommon; quite the opposite. However, such research tends to emphasise team-wide optimisation of a single task (e.g. [5, 12]). Genuine behavioural attempts at intra-team task distribution are rare [9], and those addressing a fully re-taskable team are even rarer.

It would be foolish at this point to try and reopen the behavioural vs. cognitive debate, and this is not our aim. However, it is notable that many robot teams to date have tended to be task-specific, and have solved their problems in a manner that could easily be viewed as a single, distributed robot rather than a society of autonomous, interacting robots [1]. By contrast, the world of multi-agent systems is heavily oriented towards the latter view, with human/computer interfaces working with versatile, taskable agents to address the user's needs. In many potential robotics scenarios – robotically assisted hospitals, for example – this approach makes more sense than a static, task-specific situated team.

The large array of interaction schemes used in the agent world have a great deal to offer robotics without compromising an individual robot's behavioural nature (or even team-wide behaviours, if such exist). We believe that by applying these techniques to the robotics world, it is possible to enable a richer variety of team-based applications, without compromising the essential advantages of a behavioural approach.

To address this goal, we present a hybrid robot control architecture based with varying fidelity on two previous agent control schemes: Beliefs-Desires-Intentions (BDI) and teleo-reactive (TR) programming. We replace the action-sequence style plans in traditional BDI with robust, recoverable TR programs, and augment both layers with a unified percept and world model store. In the following sections we review teleo-reactive programming and traditional BDI. We then examine some basic issues in the development of hybrid robot control architectures, then present our own proposed architecture. Finally we detail our early implementation work.

2 Teleo-reactive Programs and the Triple Tower

2.1 Teleo-reactive Programs

Nils J. Nilsson is responsible for proposing firstly a programming formalism for robust, goal-oriented robot programming (teleo-reactive programs [7]) and more recently an architecture for adapting this formalism for a more cognitive style of robot control in the form of his Triple Tower architecture [8].

Teleo-reactive programs provide the robot programmer with an intuitive and recoverable structure within which to write goal-directed programs. Similar in style to production rules (and to a lesser extent Brooks-style subsumption [3]), a teleo-reactive program consists of a series of prioritised condition/action rules. A teleo-reactive interpreter constantly re-evaluates the triggering conditions set for each rule, and executes the action corresponding to the highest-priority rule with a satisfied pre-condition. Generally, this is denoted

$$K_1 \rightarrow a_1$$
$$\ldots$$
$$K_m \rightarrow a_m$$

An action a_i may consist of a single, primitive action (e.g. `move_forward`), or may itself be a teleo-reactive subprocedure. A crucial point here is that unlike in conventional programming, where a launched subroutine assumes control of the execution of the program, a parent TR program continues to constantly evaluate its set of conditions *even when a subprocedure is invoked*. Thus if a different rule condition is triggered, the subprocedure is terminated by its parent. This mode of operation is highly amenable to multi-threaded design, as subprocedures can simply be launched as a new thread, and suspended or terminated by the parent as appropriate.

Nilsson defines two properties pertaining to teleo-reactive programs, the *completeness* property and the *regression property*. A teleo-reactive program is said to be complete if the conjunction of the preconditions for all of its rules, $K_1 \wedge \ldots \wedge K_m$, is a tautology; that is, that the set of the preconditions of all rules in the program represents the full set of possible percept states for the agent. In other words, there is no percept state for which the agent does not have a relevant response action.

A teleo-reactive program is said to satisfy the regression property if the precondition of each rule is expected to be satisfied by the action of a rule with a lower priority. For example, a rule with an action `find_door` would satisfy a precondition `at_door`; thus the program

$$\texttt{at\_door} \rightarrow \texttt{move\_through}$$
$$\top \rightarrow \texttt{find\_door}$$

would be considered to satisfy both the regression and completeness properties, since $\top \wedge$ `at_door` is a tautology, and the result of action `find_door` is to achieve the pre-condition `at_door`. A complete teleo-reactive program which satisfies the regression property is said to be *universal*.

Teleo-reactive programming appeals as the basis of a method for robot control due to their goal-directed nature, and recoverable properties. Rather than planning a discrete, explicitly sequenced series of actions (although simpler TR programs can appear like this), TR programs make action a direct mapping from sensor inputs, in the behavioural style of (cite some Brooks here). As a result, robots can opportunistically take advantage of external events which might satisfy some rule's precondition without the robot acting for itself, or recover from adverse external events without re-planning.

For example, a program intended to seek out trails, then follow them to a cache of some resource might be written:

$$\texttt{see\_resource} \rightarrow \texttt{collect\_resource}$$
$$\texttt{on\_trail} \rightarrow \texttt{follow\_trail}$$
$$\top \rightarrow \texttt{wander}$$

Should the robot be following a trail and for some reason lose it, or perceive that a cache it had found becomes exhausted, it will simply default to the lower priority rules as appropriate; no re-planning is needed. Conversely, should the robot be `wander`ing and fortuitously happen across a resource cache, it will simply skip the trail-following stage, and opportunistically execute its `collect_resource` action without having to do any intermediate re-planning to cope with this unexpected stroke of luck.

Appealing though they are for intuitively constructing simple (and even moderately complex) behaviours, teleo-reactive programs are limited in the extent to which they can be used as a robot control architecture. They contain no defined mechanism for communication; program state is (deliberately) minimal, and they are inherently directed at a single goal. All of these factors limit their (unmodified) application as a taskable, cooperative robot control system.

2.2 The Triple Tower

More recently [8], Nilsson described his Triple Tower architecture (Figure 1), building on the foundation of teleo-reactive programs to construct a more capable architecture aimed at a more cognitive style of robot control.

Acknowledging the need for control at different levels of abstraction, Nilsson proposes (orthogonally to the traditional three-layer architecture) three towers, respectively representing a robot's perceptual expertise, its model of the world at a given time, and its response functions. Each is intended to consist of an arbitrary number of levels, as appropriate to the specific application.

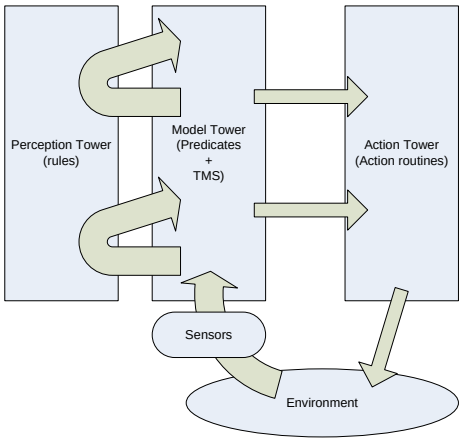

Fig. 1. Nilsson's Triple Tower

The Model Tower, as indicated in figure 1, contains the predicates which comprise the robot's world model. At the lowest level, these directly correspond to the data arriving from the robot's sensors. The robot then uses the inference rules contained in the Perception Tower to deduce higher-level information about its environment (e.g. navigational information, task progress, etc.). To borrow Nilsson's own example, a block-stacking robot might *directly* perceive only whether each block is on another block, or on the table. Deductive inference rules can then construct the agent's world model. For example, from this information the robot can infer whether a given block is clear to be picked up, whether any towers are formed, and what order they are formed in. This inferred information can then be used to create simpler TR programs than would otherwise be possible using the raw percepts.

The Action Tower, which consists of a number of teleo-reactive programs, accesses the Model Tower for use in its programs' conditional rules. Rather than

implement a subsumption-style suppression system, the higher-level TR programs invoke the lower-level programs as their actions, switching between lower-level programs as appropriate. Nilsson avoids explicitly enforcing layer separation, so that Action Tower programs at all levels can access all of the Model Tower if necessary. By design, however, higher levels are intended to access inferred beliefs, with the bottom levels of TR program accessing the raw percepts.

Despite its treatment of representation, and its facility for multi-level control, the Triple Tower architecture is still not ideally suited for cooperative robot control. At the highest level, there can still only be one TR program controlling the robot, so the robot can not be easily switched between tasks. Moreover, there is no explicit facility for communication (although this could be considered to simply be part of the robot's percepts), and the combination of this and the lack of taskability makes the architecture as described unsuitable for team-based robot control.

3 BDI

Perhaps the most well-known formal agent control architecture, BDI or "Beliefs, Desires and Intentions" was proposed by Rao and Georgeff [11] and extended further with the introduction of AgentSpeak(L) [10]. Following the work of Bratman *et al* [2], this seeks to model what Rao and Georgeff see as the three primary mental attitudes necessary for a rational agent. The three attitudes model the information, motivational and deliberative states of the agent, respectively.

Beliefs are kept deliberately conceptually distinct from knowledge, which is not represented in BDI. This is because a key assumption is that sensor information is incomplete, necessitating a store of information that represents the *likely* state of the environment. However, sensor inaccuracy and un-sensed environmental changes may at any point cause some or all of the stored information to be untrue. As a result, the information store must be non-monotonic. Since such a requirement does not correspond well with traditional theories of knowledge, the concept of belief is instead used. Similarly, desires are considered distinct from goals, as at any given moment there exists the possibility of mutually incompatible desires.

Intentions represent not what the agent is seeking to achieve, but the actions by which it (presently) intends to achieve its desires. In practice (e.g. in AgentSpeak(L)), this is implemented by giving an agent a library of uninstantiated plans, each describing a pre-defined sequence of actions designed to achieve a particular desire or goal. Augmented with preconditions, the BDI interpreter chooses a relevant plan, instantiates it and adds it to its execution stack. To return to the earlier trail-following robot example, a relevant plan to satisfy the goal `have(gold)` might be as follows:

```
+goal(have(R)) →
        wander(Time);
        ?believes(at(trail));
        follow(trail,Time);
        ?believes(see_resource(R));
        pickup(R).
```

Here, the `?believes(...)` represents a query against the agent's internal beliefs. The agent performs one atomic action, then checks to see if it has succeeded, only then proceeding to the next atomic action. The problem with respect to robot control is fairly clear. Should any of the actions fail, and the desired belief state fail to be reached, the plan has no fallback; it simply fails, leaving the planner to decide on another course of action. Worse still, if the robot happens across a resource by accident, the plan contains no contingency for this either, and will either continue

searching pointlessly for a trail before collecting anything, or will fail again and cause still more unnecessary re-planning. Contrast this with the much simpler TR program shown earlier, with its built-in facilities of recovery and opportunism, and the advantages that TR has to offer are clear.

It is this re-evaluation in the face of unexpected events that causes the most problems when attempting to apply action-sequence oriented BDI implementations directly to robotic applications. Designed for an agent world in which non-deterministic actions are relatively rare, and reliability is hoped to be the norm, not the exception, this re-evaluation represents a considerable cost in the unpredictable world of robotics. Actions can not be assumed to be completed, and changes in the world state can and will occur, frequently unnoticed. By contrast, TR programs with their durative actions and fall-back structure are ideally suited to a world in which not everything is in the robot's control.

4 Hybrid Architecture Issues

At their heart, all hybrid architectures seek to tackle the divide between the reactive, un-modelled control layers responsible for direct and reflexive control of a robot's basic functions, and the cognitive levels intended to allow logistical planning and re-evaluation of the robot's world model. Some systems take the approach of firmly delineating the two aspects of the robot's architecture [6], implementing a cognitive planner entirely separately from the behavioural layers of the robot. While this offers a certain conceptual simplicity, the problem of model consistency arises: does the symbolic state of the planner accurately represent the raw sensor data being used by the behavioural layer? Other architectures seek to incorporate behavioural concepts such as motivational signals into the cognitive layer [13].

Rather than treat cognition and reaction as distinct and immiscible, we seek to graduate the transition between the cognitive and the behavioural. Further, instead of maintaining a separation between the sensory data used by the behavioural layers, and the symbolic knowledge used by the cognitive layer, we instead maintain a distinct percept and data server that all layers of the control architecture can access. While it is to be expected that the cognitive layer will mostly access symbolic data and that the behavioural layer will mostly access raw sensor data, it seems counter-productive to enforce an artificial barrier between the two. In this sense, the percept system is similar in concept to Nilsson's Model Tower, differing in that the knowledge contained therein is explicitly available to all levels of the control architecture.

5 Our Architecture

We propose an architecture in which we bind a BDI-style cognitive layer with a graded behavioural layer composed of hierarchies of teleo-reactive programs. In our approach, we seek to smooth the transition between the cognitive layer of our architecture and the behavioural. In this, we follow the example of Nilsson's triple tower by using hierarchies of teleo-reactive programs, with each higher level of TR program operating using a information at a higher level of abstraction. Thus, the lowest level of program operates using the robot's raw sensor percepts as the basis for its rules' conditions, while the highest level can use the robot's belief state regarding the world and its team-mates if necessary.

In a BDI context, the set of highest-level TR programs corresponds to the BDI layer's plan library. Each is augmented with a `plan_for` attribute, which represents the event or goal that the plan is intended to address. Thus, an intention consists not of an instantiated series of pre-planned actions, but an invoked TR plan, to

which control of the robot's actuators is passed. In a similar manner to traditional BDI, these pseudo-intentions can be suspended and resumed as required in reaction to externally- or internally-generated events.

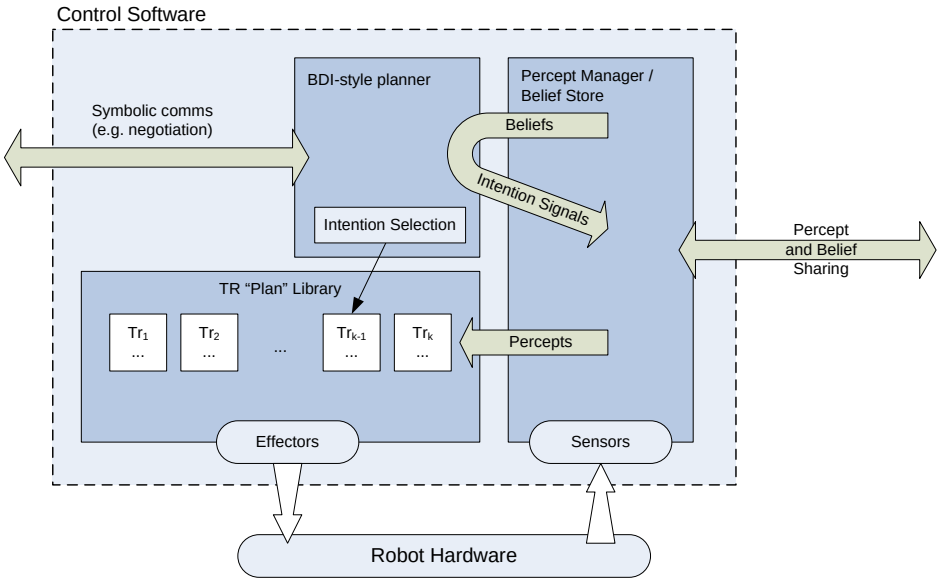

Fig. 2. Our hybrid architecture

In a significant simplification over traditional BDI, however, resumption of suspended intentions carries no risk of requiring re-planning. Since they are generally stateless, a suspended TR program can be resumed after an intervening program has run without the BDI layer needing to consider whether circumstances have changed in the meantime such as to invalidate the plan. Rather than leaving complex re-planning to the BDI layer, recovery is left to the TR program itself, which due to its inherently robust structure is able to accommodate any changes to the robot's state caused by the intervening program.

For example, the trail-following robot in Section 2 might find its foraging behaviour interrupted by an intervening request to deliver its current cargo to a depot, or to pass it to a team-mate. Being stateless, the foraging intention can be trivially suspended (or simply stopped) to allow the requested delivery to go ahead. On resumption of the suspended intention, the BDI layer need do no re-checking of plan guard conditions, etc.: it can simply restart the suspended program on the assumption that it will operate as designed, regardless of the changes in the robot's belief state caused by the delivery program.

In practice, the TR style makes intention suspension and resumption even easier through multi-threaded design. Each intention is given its own execution thread, running its own TR interpreter on such percepts and beliefs as it chooses to subscribe to from the percept module. Should the BDI module wish to switch to a new intention, it merely suspends the current top-level TR execution thread, leaving it dormant while the intervening intention executes. Again, the stateless nature of the TR programs allows the agent to either abandon the intention entirely or resume it at a later time by simply terminating or restarting the suspended thread.

The percept/belief subscription model is important to the efficient operation of the TR interpreter. By placing a subscription to the relevant belief within the belief store, the TR interpreter will receive updates only on changes to fluents that it has deemed relevant, and more importantly will only receive updates *when* they change. More efficient than busy querying of rule conditions on a constant cycle, this allows

the interpreter to suspend until relevant information makes re-assessment necessary. Programs which need to do so can still access streams of raw percept data, but this is intended to be limited to the lowest levels of program.

5.1 Percepts, Beliefs and Communication

The decision to use one unified module for percepts and beliefs allows the inclusion of several agent-style design techniques, such as multi-modal percept transmission using publish/subscribe, broadcast and point-to-point communication. It also facilitates intra-team percept sharing by allowing members to subscribe to percepts from another agent in the team transparently, effectively co-opting sensors throughout the team as appropriate.

This is not to say that the distinction between percepts and beliefs is lost, however. Access to them is mediated through the same software module to allow all levels to consult both beliefs and percepts if necessary. Discretion in this matter is left to the programmer; however it is to be expected that a coherent programming style will only attempt to use (for example) high level beliefs in a low-level TR program rarely.

The unified percept module also assists with the synthesis of percepts and communications to beliefs. Domain knowledge allows an agent or robot to contextually interpret its current percepts with respect to its belief state. For example, a robot whose belief state indicates it is in a corridor might interpret decreasing frontal sensor readings as indicating the end of the corridor, or a door opened in its path, depending on how far along the corridor it believes itself to be. By contrast to Nilsson's Model Tower, however, we use "live" deductive reasoning in a Prolog style, querying the database using inference rules on demand, instead of interpreting each incoming percept and adding the appropriate beliefs. This obviates the need for a Truth Maintenance System as with the Triple Tower, as beliefs are not permanently stored, and therefore need not be removed when obsolete.

5.2 Cooperative Task Allocation

Task distribution amongst collective members is a classic agent problem. Our architecture is agnostic with respect to distribution methods; robots might implement any one of a number of task distribution algorithms. What is required is that all agents are capable of advertising those tasks they are able (and willing) to perform, and of requesting tasks from appropriate partners. Our early simulation efforts (Section 6.1) demonstrate a basic selfless negotiation strategy being used to optimise foraging and collection tasks in a homogeneous team. More complex strategies for use in heterogeneous teams and more varied task environments can be implemented using traditional agent techniques. Robots acting as brokers might specialise in organising a particular type of task, or might use superior communications abilities to muster a team for a less capable robot in the field.

If a robot is to search for task partners, it must first be aware that it needs to do so. To this end, our TR plans must therefore be augmented with another attribute, `requires_coop([(N,T)|_])` [1], indicating that the task requires (or would like) N cooperative agents to perform task T in order to accomplish the goal. This attribute is used by the deliberative layer to locate and recruit available agents according to the task distribution scheme in use. If none can be found, an alternative TR plan is sought. Note that the attribute does not specify a manner in which the task is to be performed - in a heterogeneous team different robots may perform the

[1] Where `[(N,T)|_]` denotes an arbitrarily long list of tuples `(N,T)`, allowing an agent to specify several different cooperative tasks required to fulfil its goal

same task in different ways, and with differing degrees of efficiency. Here again, agent techniques such as Contract Net offer the ability to sensibly recruit the most appropriate agent/robot for the task, without having to bother about the details of how the task is achieved.

5.3 Cooperative Behaviours

One aspect of forming ad-hoc teams is the ability to not only agree to perform an agreed set of tasks together to achieve a common goal, but to cooperate *within* those tasks. On a behavioural level, this might mean maintaining a specific formation with other robots to achieve an efficient search pattern, or perhaps share range-finder data to cooperatively map an area. In each case, however, the individual robots have a specified set of data requirements from their team partners, each appropriate to the task being performed.

Again, this necessitates a further augmentation of each TR plan in the robot's library, `requires_data([(T,P)|_])`. This indicates that for each cooperating robot performing task T, the local robot requires the cooperating robot to forward its percepts matching P. This is again mediated by the deliberative layer. Here the agent techniques again come to the fore, as the network-transparent publish/subscribe features of the individual robots' percept modules allows the deliberative layer to simply place a subscription from the local robot to the cooperative team-member, whereupon the local robot's TR plans can simply treat the remote percepts like any other (allowing, of course, for issues of network latency and unreliability).

It should be noted at this point that all of the above presumes that any robots wishing to cooperate will have some shared ontology for both tasks and percept types. While our architecture is designed with heterogeneous teams in mind, some degree of commonality is clearly needed to allow even the most minimal of cooperation. Our initial implementation efforts presume a completely shared ontology; while teammate discovery and task allocation in an uncertain ontology environment would certainly represent an interesting research topic for the future, it is beyond the scope of our present work.

6 Early Implementation

Initial attempts at implementation have been made in both simulation and hardware. Simulation efforts have been limited purely to proof-of-concept levels, since it is our belief that to rely on simulation is to ignore the majority of the fundamental challenges of robotics.

6.1 Simulation

A basic foraging scenario formed the basis of initial simulation attempts [4], in which four independent agents operated in a grid world to collect a constantly replenished food resource. Combining a reactive search and navigation layer with more sophisticated task allocation and negotiation techniques, the agents' internal architecture represent a simplified prototype of the proposed architecture.

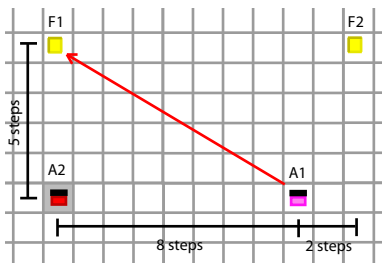

Fig. 3. Foraging agent scenario

While not implementing a full BDI-style upper layer, the agents demonstrate a useful convergence of deliberative and behavioural programming styles. Foraging is done reactively, using a basic TR program consulting only the agent's direct percepts. An

ant-style pheromone-laying algorithm directs the agent's search by guiding it away from previously explored areas in a pseudo-random walk. Independently, the deliberative layer monitors the food discovered by the agent and its team-mates, and attempts to optimise the agent's current task according to both its personal expediency and that of the team at large. It can do this independently of the search behaviour's implementation, and interacts with the behavioural layer solely by depositing intentions in the shared belief store.

The simulation further demonstrates the ease with which cooperative strategies can be used at different levels of an architecture without interfering; indeed, in a highly complementary manner. The pheromone-based search algorithm is reactive, unaware of the existence of other robots in any explicit sense, while the task distribution algorithms act to further enhance the search's efficiency, without needing any details of how the search itself is being performed.

6.2 Robotic Implementation

Implementation on robot hardware is still at a relatively early stage, but illustrates some incidental advantages of the architecture. Our research equipment is comprised of two Garcia robots from Acroname, each equipped with a BrainStem controller board (a controller package providing serial sensor interfaces and a minimal C-style programming environment), an Intel Stargate onboard host, and a CMUCam2 vision system. Sensors include 6 IR range sensors around the robot, and two downward-looking IR sensors for ledge avoidance.

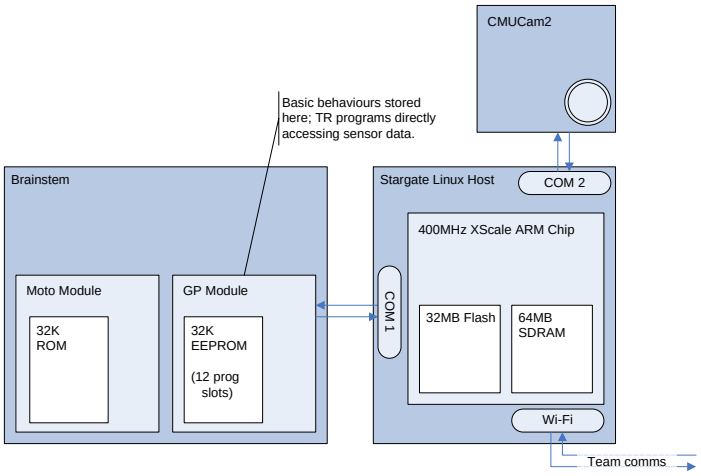

Fig. 4. Garcia hardware platform

As shown in figure 4, the execution platform is split between the Stargate host and the low-level BrainStem controller. Running on the StarGate are the BDI layer and the upper levels of TR program; running on the BrainStem are the lowest levels of TR program. In this sense the TR programs act in effect as a hardware abstraction layer; higher-level TR programs (and certainly the BDI layer) can be programmed without regard to the individual implementations of the lower-level programs. Indeed, the system could cope with more divisions of execution hardware or even the wholesale replacement of an entire hardware platform (e.g. swapping the BrainStorm for, say, a HandyBoard, a broadly comparable robotic controller board), since interaction between execution layers within the architecture is limited to invocation and termination calls.

The split nature of the hardware does force compromises regarding the handling of percepts, however. Because of the limited communication facilities between the BrainStem and Stargate, and the fact that the robot's sensors are not all attached to the same hardware, percept availability is not universal as desired in the original percept design. Primarily, the camera data is unavailable to any TR programs running on the BrainStem itself. This does not represent a particularly onerous limitation since the processing power of the BrainStem is limited, and is not ideally suited for vision algorithms.

7 Conclusions

Early simulation efforts have indicated the ease of programming offered by our proposed approach, as well as the validity and utility of allowing independent cooperation at multiple levels of the same robot's architecture. The intermediate staging of TR programs in place of plans allows both the behavioural and deliberative layers of the agent architecture to be programmed independently. This can be done without regard to the details of the other's implementation. Further, the nature of the architecture has proven itself extremely amenable to the multi-threaded programming style common in the world of agent research. Future work will concentrate solely on real-world robotics implementations, with a view to replicating the successes achieved thus far in simulation.

References

1. T. Balch and L. E. Parker. A taxonomy of multi-robot systems. In *Robot Teams: From Diversity to Polymorphism*, pages 3–22. A K Peters, Natick, Massachusetts, 2002.
2. M. E. Bratman, D. Israel, and M. Pollack. Plans and resource-bounded practical reasoning. In R. Cummins and J. L. Pollock, editors, *Philosophy and AI: Essays at the Interface*, pages 1–22. The MIT Press, Cambridge, Massachusetts, 1991.
3. R. A. Brooks. A robust layered control system for a mobile robot. *IEEE Journal of Robotics and Automation*, 2(1):14–23, March 1986.
4. S. Coffey and D. Gaertner. Implementing pheromone-based, negotiating forager agents. *Sixth International Workshop, CLIMA VI, City University London, UK, June 27-29, 2005. Revised Selected and Invited Papers*, pages 385–395, 2006. Lecture Notes In Artificial Intelligence, Vol. 3900.
5. R. Kurazume, S. Nagata, and S. Hirose. Cooperative positioning with multiple robots. In *Proceedings of the IEEE International Conference on Robotics and Automation*, volume 2, pages 1250–1257, 1994.
6. K. H. Low, W. K. Leow, and M. H. Ang, Jr. A hybrid mobile robot architecture with integrated planning and control. In *Proc. 1st International Joint Conference on Autonomous Agents and MultiAgent Systems (AAMAS-02)*, pages 219–226, 2002.
7. N. J. Nilsson. Teleo-reactive programs for agent control. *Journal of Artificial Intelligence Research*, 1:139–158, 1994.
8. N. J. Nilsson. Teleo-reactive programs and the triple-tower architecture. *Electronic Transactions on Artificial Intelligence*, 5:99–110, 2001.
9. L. E. Parker. ALLIANCE: An Architecture for Fault Tolerant Multi-Robot Cooperation. *IEEE Transactions on Robotics and Automation*, 14(2):220–240, April 1998.
10. A. S. Rao. AgentSpeak(L): BDI agents speak out in a logical computable language. In *Seventh European Workshop on Modelling Autonomous Agents in a Multi-Agent World*, pages 42–55, Eindhoven, The Netherlands, 1996.
11. A. S. Rao and M. P. Georgeff. BDI-agents: from theory to practice. In *Proceedings of the First Intl. Conference on Multiagent Systems*, San Francisco, 1995.
12. S. Roumeliotis and G. Bekey. Distributed Multi-Robot Localization. *IEEE Transactions on Robotics and Automation*, 18(5):781–795, October 2002.
13. A. Stoytchev and R. Arkin. Incorporating motivation in a hybrid robot architecture. *Journal of Advanced Computational Intelligence and Intelligent Informatics*, 8(3):269–274, May 2004.