A Modular Hierarchical Behavior-Based Architecture*

Scott Lenser, James Bruce, and Manuela Veloso

Computer Science Department
Carnegie Mellon University
5000 Forbes Ave
Pittsburgh, PA 15213
{slenser,jbruce,mmv}@cs.cmu.edu

Abstract. This paper describes a highly modular hierarchical behavior-based control system for robots. Key features of the architecture include: easy addition/removal of behaviors, easy addition of specialized behaviors, easy to program hierarchical structure, and ability to execute non-conflicting behaviors in parallel. The architecture uses a unique reward based combinator to arbitrate amongst competing behaviors such as to maximize reward. This behavior system was successfully used in our Sony Legged League entry in RoboCup 2000 where we came in third losing only a single game.

1 Introduction and Related Work

Briefly, our behavior system is a modular hierarchical behavior-based system with a unique behavior combinator. By modular, we mean that behaviors can be added, removed, or replaced without affecting other behaviors. By hierarchical, we mean that the system consists of many levels which operate at different levels of detail and/or time scales. Behaviors form the basic modular blocks upon which the architecture is built. Behavior modules can be swapped with similar behavior modules that accomplish similar goals (possibly by very different means). By behavior combinator, we mean a method for selecting which of a set of behaviors should be run together (or combined) to control the robot. We use a unique method of choosing which behaviors to run which allows behaviors to run in parallel while closely approximating the optimal policy (maximal reward) assuming the behavior reward values are calculable.

This behavior system was successfully used in the Sony Legged League of RoboCup 2000 [7] where we came in third losing only a single game. The quadruped robots for this league are generously provided by Sony [4]. The robots are fully autonomous, and have onboard cameras. See our paper in Agents [8] for a description of the overall team.

* This research was sponsored by Grants Nos. DABT63-99-1-0013, F30602-98-2-0135 and F30602-97-2-0250. The information in this publication does not necessarily reflect the position of the funding agencies and no official endorsement should be inferred.

Much work has been done on architectures for behaviors in general and behavior-based architectures in particular. Rodney Brooks has investigated behavior-based architectures in his subsumption architecture [3]. Ron Arkin has produced a thorough examination of behavior-based systems [1]. SRI has created a more deliberative form of behavior-based system in PRS [5]. We build upon the architecture used by the FU-Fighters small size RoboCup team [2]. This architecture is based upon the Dual Dynamics architecture [6].

We make several significant modifications to the architecture that together are the major contribution of this paper:

- We decouple the parts the system as much as possible while maintaining a hierarchical system. Our system can be viewed as a hierarchical analog of the subsumption architecture. We decouple behavior activations from the behavior level above them by introducing intermediate goals which allow the behaviors to calculate their own activation. The intermediate goals (or control sets) act as the only interface between behavior levels. We decouple the selection of behavior level from sensor level by allowing each behavior access to all of the sensors. This avoids unnecessarily restricting the information available to a behavior.
- We introduce a special combinator to select multiple non-conflicting behaviors to be run simultaneously. By non-conflicting, we mean behaviors that do not require the same resource. This means separate parts of the robot can be controlled by separate pieces of code when desired and by a single piece of code when needed.

2 Behavior Architecture

Our behavior architecture is a hierarchical behavior-based system. The architecture is primarily reactive, but some behaviors have internal state to enable sequencing of behaviors and hysteresis. The input to the system is information about the objects seen (from the vision) and an estimate of the robots location (from the localization). Output from the behavior system consists of choosing between walking, kicking, getting up, looking at objects, etc. and selecting appropriate parameters for the motion chosen. Note that looking at objects requires the use of the head and can be run in parallel with walking but kicking (the robot actually heads the ball) and getting up require the use of the whole body. Our system intelligently makes trade-offs between walking/looking and kicking. The behavior architecture consists of three interconnected hierarchies for sensors, behaviors, and control (see Figure 1). The sensor hierarchy represents all that is known about the world. The behavior hierarchy makes all of the robot's choices. The control hierarchy encodes everything the robot can do.

The sensor hierarchy represents the knowledge that the robot has about the world. The lowest level of the sensor hierarchy are real sensors that are filled in from hardware sensors or from the other modules (vision, localization, motion). The other levels are virtual sensors for convenience that are computed based upon the real sensors. All behaviors have access to all of the sensors.

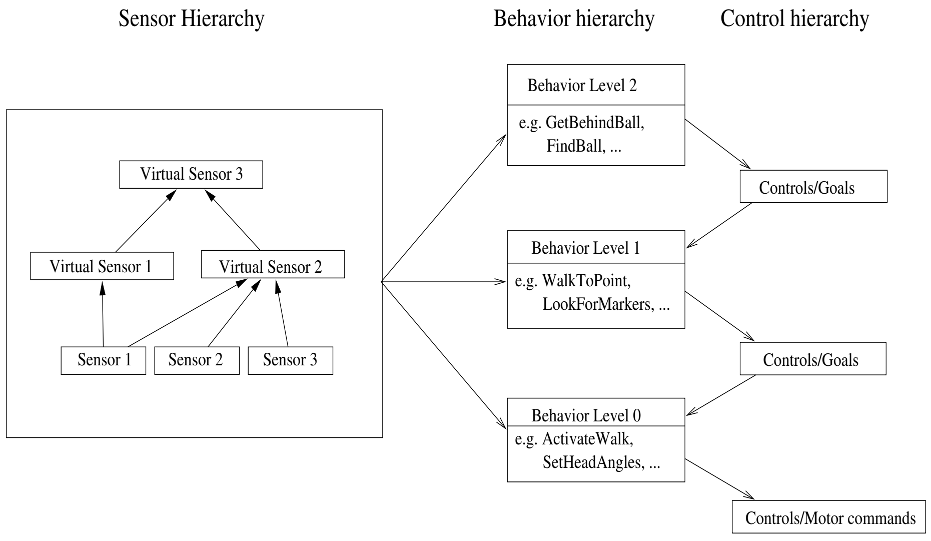

Fig. 1. Overview of the behavior system.

The behavior hierarchy controls what the robot does. Each level k of the behavior hierarchy, which we call a behavior set, is a subsumption architecture. The inputs of the level are the sensors and the control set at level $k + 1$. The outputs of the level are fed into behavior level $k - 1$ via the control set at level k. Together, the controls sets form the control hierarchy. The control set at level 0 then becomes the action(s) executed by the robot.

Each behavior set takes an abstract description of the task to be performed and creates a more concrete description of actions to be performed. The control hierarchy defines the interface between the various behavior levels. Each control set describes everything the robot can do at some level of detail. Each control set acts as a virtual actuator to the behavior level above it and as a goal set to the behavior level below it.

Behaviors have functions for calculating activation levels and outputs given the sensors and higher level controls. Each behavior looks at the sensors and the goals from higher levels and computes an activation value for the behavior (see middle of Figure 2). These activation values represent predictions of the future reward that will result if this behavior is run. These activations are used by the behavior set (via the combinator) to decide which behavior(s) to run. The processing function for each chosen behavior converts the sensors and control inputs into control outputs.

Each behavior drives a set of control outputs. Some behaviors within the behavior set will drive the same actuator/control, and thus conflict, while some will drive different actuators and can be run in parallel. Both cases occur frequently in our domain, walking/kicking vs. full body kicking motions. Conflicting be-

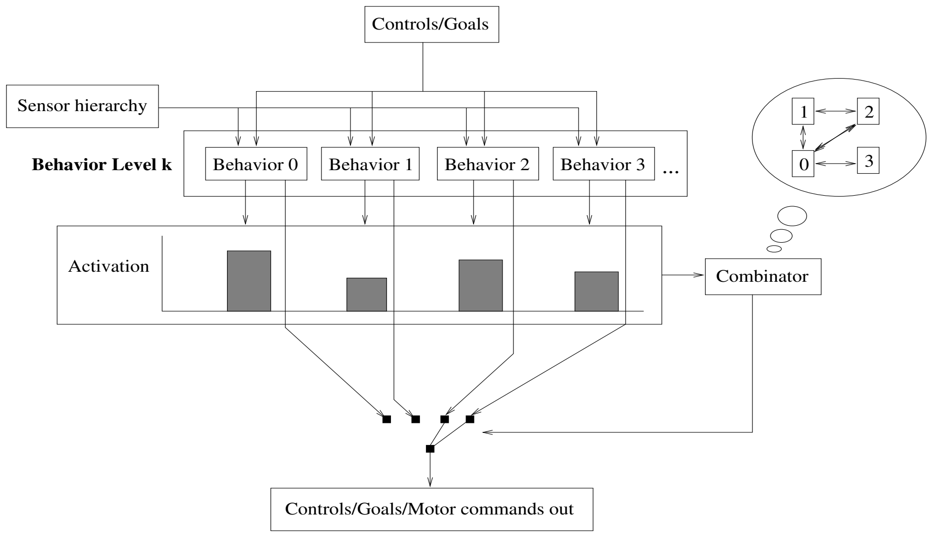

Fig. 2. Detail of a level in the behavior hierarchy.

haviors are mutually exclusive. We represent this constraint as a graph where behaviors are nodes and edges connect behaviors that conflict (see upper right part of Figure 2). We use a special combinator (described below) to choose a set of non-conflicting behaviors with near maximal total expected reward. For the example in the figure, the combinator has chosen behaviors 2 and 3 since this has more reward than executing behavior 0 alone. The chosen behaviors are run and use the sensors, the control set from above, and their memory of what they were doing to choose the controls to write directly into the goals of the next lower level.

3 Selection of Action(s)

The goal of the combinator is to find a set of non-conflicting behaviors that result in the maximal reward. This maximal reward set is the optimal policy under the assumption that the behavior activations are accurate estimates of future reward. Since the reward estimates(activations) and conflict net are given, this is the problem of finding maximal weight cliques in the dual of the conflict graph. Since this problem is NP-complete, we use an approximation algorithm. The basic idea is to find a suitably good approximation iteratively by suppressing weakly activated behaviors with many conflicts and reinforcing strongly activated behaviors with few conflicts.

To do this, we first produce an optimistic estimate for the total reward that can be achieved while running behavior k by assuming that all behaviors that are not in direct conflict with behavior k can be run in parallel. This is calculated by finding the total activation of all behaviors and subtracting the activation of all behaviors in direct conflict. We then treat this estimate as a gradient to change the activation of the k^{th} behavior. Behaviors that might be runnable with more reward than the behaviors they conflict with are reinforced while other behaviors are suppressed. Usually, at least one of the behaviors will have a negative gradient. We follow this gradient over all behavior activations until the activation of one of the behaviors becomes 0. Any behavior whose activation becomes 0 is removed from consideration. This process is repeated until the set of behaviors with non-zero activation contains no conflicts.

Small random perturbations are added to the activations to break any ties. In case all the gradients are positive, we double the amount subtracted for conflicts until one of the gradients becomes negative. In the worst case, it may take $O(n \lg n)$ iterations for the combinator to converge (where there are n behaviors) but for most cases it converges in a few iterations. The set of behaviors with non-zero activation at the end of this process are run completing the execution of the behavior set.

4 Discussion and Future Directions

The behavior system we developed has some interesting features. The system tends to be highly reactive. There is nothing in the system, however, that prevents the use of behaviors with internal state. In fact some of our behaviors use stateful systems internally. The core feature of the system is its modularity. This has many important consequences which are detailed below:

scalable - Since each behavior level is completely separated from the neighboring behavior levels by the control sets, changes to one behavior level do not require any changes to any other part of the system. Note that this is different from the FU-Fighters architecture where higher levels set activations for lower levels. In the FU-Fighters architecture, changes to one level potentially affect all behavior levels above that level and always involve at least two levels.

easy to add/remove behaviors - Behaviors can easily be turned on/off from a configuration file. Because behaviors compete amongst themselves for the right to run, the robot simply uses the best remaining behavior(s) to replace a behavior that has been removed. It is easy to replace a behavior with a different implementation or leave both implementations accessible and selectively disable one from a file.

easy to specialize a behavior - The system makes it easy to specialize behaviors. For example, a quicker but inaccurate shot behavior can be added without any other changes. With an appropriate activation function, the robot will then intelligently decide between quick and accurate shooting. This allows the flexibility to create several specialized behaviors instead of one general purpose behavior that has to handle every special case.

separates concerns when possible - Behaviors only interact through their activation functions and only with conflicting behaviors. So changes to behaviors controlling the head of our robot require no changes to behaviors controlling the legs and vice versa. The ability to separate the control of the head and the legs for most, *but not all*, behaviors allows the system to be conveniently decomposed into mostly independent parts, while preserving the ability for coordinated behavior. This same ability would also allow a multirobot team to work mostly independently except when collaboration, such as passing, requires working together.

Our behavior architecture is best understood as a step in the right direction. The architecture enables new capabilities in behaviors but as always we find that there is room for improvement. The activation functions present an opportunity for learning the future reward that will result from choosing an action now. Oscillations could be avoided by reasoning about the uncertainty present in the activation values and the cost of switching from one behavior to another. The cost of switching behaviors could also be better represented by each behavior providing an expected reward over time so that failure to produce the expected reward could be detected and taken into account.

5 Conclusion

We presented a field tested behavior system aimed at being more modular than existing systems. Behaviors can be removed, replaced, changed, or added extremely easily. The architecture also extends existing behavior-based systems to allow parallel control of multiple parts of a robot in a safe manner without disallowing full robot motions. The system was used on quadruped legged robots in the Sony Legged League of RoboCup-2000 where our team of robots came in third place only losing a single game.

References

1. R. C. Arkin. *Behavior-Based Robotics.* MIT Press, Cambridge, 1998.
2. S. Behnke, B. Frötschl, R. Rojas, et al. Using hierarchical dynamical systems to control reactive behavior. In *Proceedings of IJCAI-99*, pages 28–33, 1999.
3. R. Brooks. Elephants don't play chess. In P. Maes, editor, *Designing Autonomous Agents*, pages 3–15. MIT Press, Cambridge, 1990.
4. M. Fujita, M. Veloso, W. Uther, M. Asada, H. Kitano, V. Hugel, P. Bonnin, J.-C. Bouramoue, and P. Blazevic. Vision, strategy, and localization using the Sony legged robots at RoboCup-98. *AI Magazine*, 1999.
5. M. P. Georgeff and A. L. Lansky. Reactive reasoning and planning. In *AAAI87*, pages 677–682, Seattle, WA, 1987.
6. H. Jaeger and T. Christaller. Dual dynamics: Designing behavior systems for autonomous robots. In S. Fujimura and M. Sugisaka, editors, *Proceedings International Symposium on Artificial Life and Robotics.*, 1997.
7. H. Kitano, M. Asada, Y. Kuniyoshi, I. Noda, and E. Osawa. Robocup: The robot world cup initiative. In *Proceedings of the IJCAI-95 Workshop on Entertainment and AI/ALife*, 1995.
8. S. Lenser, J. Bruce, and M. Veloso. CMPack: A complete software system for autonomous legged soccer robots. In *Autonomous Agents*, 2001.