A Modular Hierarchical Behavior-Based Architecture*

Scott Lenser, James Bruce, 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. The system was also tested in another domain where it performed on par with a human operator.

1 Introduction

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 a bunch of levels which operate at different levels of detail and/or time scales. For instance, soccer playing robots might have one layer that decides whether the robot should attack or defend while a lower layer decides the best direction in which to move to accomplish this task. By behavior-based, we mean that the system is decomposed into behaviors, each of which accomplishes a specific task. 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 choosing 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) under certain assumptions detailed later.

* 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.

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. We also used this architecture to control a small remote controlled robot where it played on par with a human operator.

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 modification to the architecture that together are the major contribution of this paper:

- We give the behaviors access to all of the high and low level sensors so that the behaviors can access whatever information is appropriate. This avoids making decisions about which level to put particular information on without compromising the modularity of the system.
- We decouple behavior activations from the behavior level above them. We achieve this by having each behavior calculate its activation based upon goals set by the higher level behaviors instead of the higher levels setting activations directly. This increases the modularity of the system by allowing each behavior level to be treated as a separate unit. This is what enables us to easily add/remove/replace behaviors.
- We introduce a special combinator that allows 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 Domains

We tested the architecture in two robotic domains with very different robot capabilities. We chose one domain with global sensing and precise motion and one domain with local sensing and imprecise motion to ensure that the behavior system works well across a variety of domains. The rest of this paper focuses on the local sensing domain since this domain is more complex.

The global sensing domain consists of small wheeled robot playing soccer with a golf ball. The robot is radio controlled by an off-board computer. The computer gathers information about the state of the world via a camera placed above the field aimed downwards which provides global vision. The off-board computer makes decisions based on the camera and sends wheel velocity commands to the robot. Our robot uses bang-bang control and a differential drive design resulting in 9 possible motor commands.

Input to the behavior system consists of: position and velocity estimate for the ball, position and orientation estimates for each of our robots, and position

estimates for each opponent robot. Output consists of: forward, stop, or reverse for the left and right wheels.

The local sensing domain is the RoboCup Sony legged league where teams of quadrupeds play soccer against one another. The robots for this league are generously provided by Sony [4] specifically to be applied to the domain of robotic soccer. The robot consists of a quadruped designed to look like a small dog. The neck and four legs each have 3 degrees of freedom. The neck can pan almost 90° to each side, allowing the robot to scan around the field for markers. The camera has about a 55° field of view.

All teams in the RoboCup-00 legged robot league use this same hardware platform. The robots are autonomous, and have onboard cameras. The onboard processor provides image processing, localization and control. The robots are not remotely controlled in any way, and as of now, no communication is possible in this multi-robot system. The only state information available for decision making comes from the robot's onboard color vision camera and from sensors which report on the state of the robot's body. The field for these robots is 280cm long and 180cm wide. The goals are each 60cm wide and 30cm tall. Six unique colored landmarks are placed around the edges of the field (one at each corner, and one on each side of the halfway line) to help the robots localize themselves on the field.

Input to the behavior system consists of: egocentric distance and angle to visible objects (from vision), estimated position and uncertainty of robot's location (from localization), and various robot body state sensors. Output from the behavior system consists of a choice between kicking the ball (uses legs and head) or walking and using the head to independently do two separate things. If the robot chooses to kick, it also has to choose the type of kick to be performed (which direction to try to hit the ball). If it chooses to walk and use the head, it must provide further details specifying the exact motion to be performed. For walking, the robot selects among several different walk types (circular arcs, turning in place, etc.) plus parameters for the walk type chosen. Head motions required the joint angles to use for the tilt, pan, roll mechanism of the neck. This domain will be used for most of the examples in this paper. See our paper in Agents [8] for a description of the overall team.

3 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). The output of the behaviors is a motion to be executed. The motion can be a type of kick to execute (discrete) or a direction to walk (continuous) for example. 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

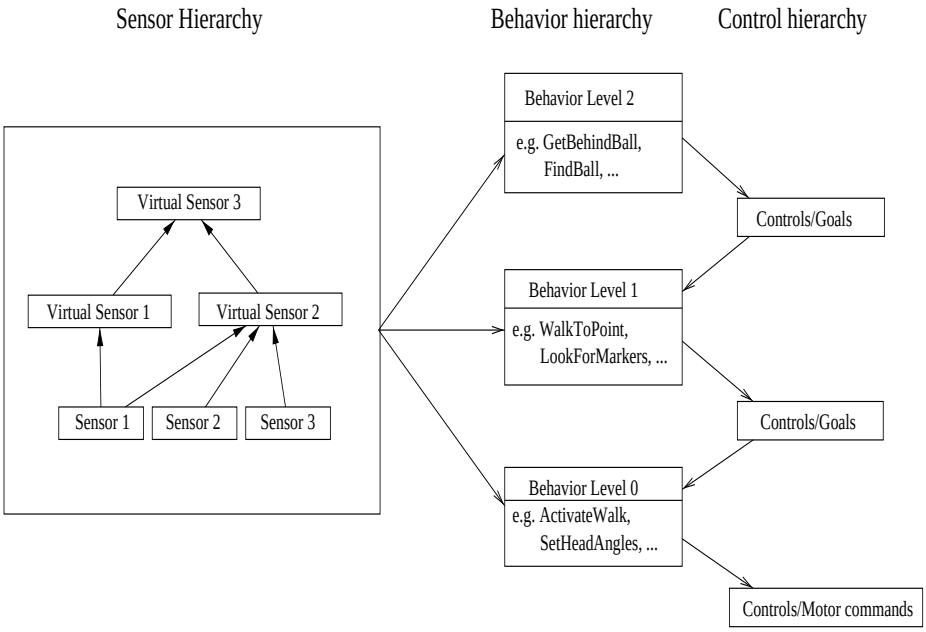

Fig. 1. Overview of the behavior system.

makes all of the robot's choices. The control hierarchy encodes everything the robot can do.

3.1 Sensors

The sensor hierarchy represents the knowledge that the robot has about the world. We divide the sensors into two categories, real sensors and virtual sensors. Real sensors are provided by hardware, virtual sensors represent processed versions of the real sensors. An example of a real (or base) sensor is the location of the ball returned by the vision module such as ahead 250mm 16° to the left. An example high level (or virtual) sensor is the estimated location of the ball in world coordinates. Notice this fuses information from the vision module and the localization module in a form more convenient for some tasks. An example very high level sensor is the average location of the ball on the field over the last 3 minutes. Virtual sensors provide a convenient way to fuse multiple hardware sensors. They allow sensing to be filtered over time or space to remove noise. They reduce duplication in the behaviors since information that multiple behaviors are interested in can be moved into a sensor. For example, we use a sensor which indicates the direction in which we would like the soccer ball to move in egocentric coordinates. This greatly simplifies dribbling and kicking behaviors. The sensor hierarchy includes storage for the current value of all of

the sensors plus code to update the virtual sensors in the data structure from the real sensors. All of the sensors in the sensor hierarchy provide input to all of the behaviors in a read only fashion.

3.2 Behaviors and Control

The behavior hierarchy and control hierarchies are tightly coupled and cooperate to decide on the action to perform. The behavior hierarchy contains all of the behaviors. The role of the behavior hierarchy is to make decisions. The behavior hierarchy consists of n behavior sets. Each behavior set represents a set of behaviors. Different behavior sets operate at different levels of detail, 0 being the lowest level and $n-1$ being the highest level.

The control hierarchy buffers the communication between the different behavior levels. The role of the control hierarchy is to describe the interface between higher level and lower level behaviors and buffer the communications between these levels. The control hierarchy consists of n control sets. Each control set is a data structure representing all the actions the robot could perform. Each control set represents the actions at different levels of detail. A control set can be viewed as a set of goals for lower level behaviors to achieve. Alternatively, it can be viewed as a virtual actuator that the higher level behaviors can control.

A behavior set at level k receives input from the the control set at level $k+1$ and the sensor hierarchy. The behavior set decides what action to perform and writes the decision into the more detailed control set at level k. The behavior set at level 0 writes the motor commands for the robot into control set 0. See Figure 1. Each behavior set at level k is a mapping from sensors and the control set at level $k+1$ to the control set at level k. If the behaviors are stateless, these mappings are functions. A hierarchy simplifies the implementation problem by grouping options that need to be considered together at the same level of decision making (in the behavior sets). The interface between higher levels and lower levels is clearly documented in the control set that stores the communication between the levels making the entire system highly transparent and easy to understand.

Remember that the role of each behavior set at level k is to provide a mapping from the sensors and the control set at level $k+1$ to the control set at level k (see Figure 2). In other words, the behavior set takes input from the sensors and controls directly above this level and produces controls directly below this level. To accomplish this goal, each behavior set is divided into a set of behaviors (these behaviors all operate at the same level of detail). Examples of some mid level behaviors are: move to a point on the field, kick the ball towards a point, and look around for landmarks. Examples of some high level behaviors are: get behind the ball, dribble the ball towards the goal, shoot the ball, and return to goalie position.

Behaviors have functions for calculating activation levels and outputs given the sensors and higher level controls. These functions are part of the domain knowledge given to the robot. Each behavior looks at the sensors and the goals

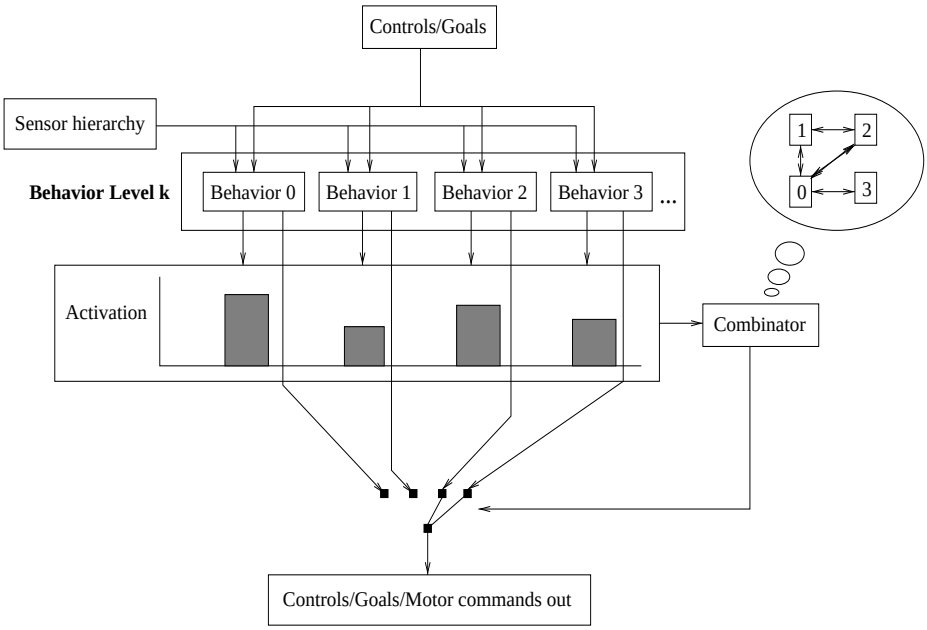

Fig. 2. Detail of a level in the behavior hierarchy.

from higher levels and decides how well it would perform. This measure of goodness is used as an activation value for the behavior (see middle of Figure 2). These activations represent predictions of the future reward that will result if this behavior is run. These activations are used by the behavior set to decide which behavior(s) to run. The processing function for each chosen behavior converts the sensors and control inputs into control outputs.

For example, the GetBehindBall behavior could be activated when the control set above this level indicates that it is good to get behind the ball and the ball location is known. This can be done by having the control set above this behavior set a variable (in the control set) with a range of 0–1 to indicate how good it is to get behind the ball. Then have a sensor with a range of 0–1 that indicates how likely we are to know the ball location. The behavior activation can then be found by multiplying these two values together. Many of the behavior activations can be specified this way by multiplying together values to achieve "and" preconditions and adding together conditions to achieve "or" preconditions. The values that are added or multiplied together can be simple functions. Ramps, sigmoids, piecewise linear and sinusoids are all useful functions for computing these bases. This allows the conditions to be fuzzy and the activation functions to be smooth. The behavior would then use the sensors to find out where the ball is and set a goal for the next level down to run along an arc that intersects a point behind the ball.

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 as we may want to walk somewhere while we are doing something useful with the head like scanning for the ball. Other times we want to use the legs and head together, usually to kick the ball. Behaviors that are in conflict must never be run together. We capture this constraint by constructing 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. So for the example activations in the figure, the combinator has chosen behaviors 2 and 3 since this has more reward than executing behavior 0 alone. The behaviors that are chosen for activation are then run and write there results directly into the control set for the next level of behaviors. The behaviors that are run use the sensors, the control set from above, and their memory of what they were doing to choose the controls to write into the goals of the next lower level.

Allowing the behaviors to use memory allows a behavior to sequence objectives. For example, when searching for the ball we pick random points on the field to walk to while looking for the ball. By keeping track of where we are walking to, we avoid oscillating between different random points on the field. Instead we go to one point then to another and so on. Memory can also be used in the activation functions to implement hysteresis and avoid oscillation.

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 provides storage for each level of the behavior hierarchy to write outputs and read inputs. The lowest level of the control hierarchy, level 0, is simply the set of low level actuators available to the robot, in our case the commands exported from the motion module.

4 Behavior Combination

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.

5 Discussion and Future Directions

The behavior system we developed, as presented in this paper, has some interesting features:

Reactiveness: The system tends to be highly reactive. This means that the architecture is primarily concerned with processing the sensory input. There is however nothing in the system that prevents the use of behaviors with internal state. In fact some of our behaviors do this, but the core assumption underlying the system is that most of the interesting behaviors in a highly dynamic environment are reactive in nature. If desired though, each behavior can execute a state machine or other stateful system. The behavior architecture system supports a general behavior definition. It mainly requires only the computation of an activation level and an output to the control system. If behaviors have some memory, they can sequence actions. In our domain, we use this feature when searching for the ball to allow the robot to remember where, on the field, it decided to go look for the ball at. Memory also allows us to add hysteresis to our behaviors where needed.

Scalability: Our experience tells us that our behavior system scales easily and easier than other behavior-based systems we have developed earlier. Since each behavior level is completely separated from the neighboring behavior levels by the control sets, changes to one behavior 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. In addition, behaviors within the same behavior level only interact with each other through their activation functions, and only with behaviors with which they conflict. So changes to behaviors controlling the head of our robots 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 decomposed into the

mostly independent parts the designer wishes to work with, 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.

Modularity: The core feature of the system is its modularity. This has many important consequences. Because the system consists of mostly independent parts, it is very easy to add or remove behaviors. The set of behaviors to include is easily selected from a file using a single binary flag for each possible behavior. Because behaviors compete amongst themselves for the right to run, removing a behavior that the robot used to run in a particular state results in the robot running the best remaining behavior for that state. It is easy to replace a behavior by plugging in a different implementation. Or leave both implementations accessible and selectively disable one from a file. The modularity also allows for easy specialization of behaviors. For example, normally the robot might want to carefully aim a shot on the goal so we might make this a behavior. But if the robot is right in front of the goal, it does not make sense to waste time carefully aiming the shot since just pushing the ball roughly forward is enough. We can add this special case simply by creating a new behavior that pushes the ball quickly towards the goal. If this new behavior's activation is higher than the normal carefully aimed shot when we do not want to bother aiming the shot, then the robot will slam the ball into the goal if the shot is wide open and carefully aim the shot otherwise. Sometimes, it is much easier as a designer to create several specialized behaviors than one general purpose behavior that has to handle every special case.

Our behavior architecture is best understood as a step in the right direction. The architecture concretely contributes several new approaches to representing and controlling robot behaviors. But while the system enables many great capabilities, which we believe were not easily possible before, as always we find that there is room for improvement. We discuss below directions along which we and other researchers can now contribute enhancements of the architecture. This would not have been possible without the development and contribution of this architecture, as presented in this paper.

The specification of behaviors require the definition of activation functions that approximate the reward the robot will receive from executing a behavior. The functions need to return a value with fidelity enough for decision making. There is a great opportunity *to learn* the future reward for executing an action from a particular state. This can be viewed as the familiar Q-table approximation problem in reinforcement learning. However it is also a well-known challenging task, due to the size of the state space. (Note that the action space is in principle reasonably small, since we are only deciding whether to run one out of a small number of behaviors in a particular state).

Another possible extension will be to provide more explicit support for reasoning about uncertainty about the future reward obtained by executing some action. By only switching behaviors that we are fairly certain are better than the ones we are executing now, we can avoid the common problem of oscillation

in reactive behavior systems. The reward estimate representation can also be changed to better reflect costs associated with switching behaviors. Often there is some time after switching behaviors before the robot is able to see the effect of its actions. By representing the estimated reward over time for each behavior, we could take into account the length of time (progression along reward graph) the behavior has already been running (and presumably getting closer to an observable improvement in state). This would model the cost of switching to a different behavior. This same representation would also allow us to monitor the behavior and abort it, if it is performing worse than expected.

6 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. Our team of robots, running a complete implementation of our behavior architecture, came in third place, only losing a single game. The behavior architecture is remarkably general and can be applied to different robotic platforms. We tested it on a prototype small size soccer-playing robot, where it played on par with a human playing an opposing robot by remote control.

References

1. R. C. Arkin. *Behavior-Based Robotics*. MIT Press, Cambridge, 1998.
2. S. Behnke, B. Fr otschl, 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.