An Architecture for Reactive Behavior*

Benny Rochwerger Claude L. Fennema† Bruce Draper Allen R. Hanson
Edward M. Riseman

Computer Vision Laboratory
Dept. of Computer Science
University of Massachusetts
Amherst, MA 01003

Abstract

Complex problems, such as driving, can be solved by decomposing them into simpler sub-problems, solving each sub-problem, and then integrating the solutions. In the case of an autonomous vehicle, the integrated system should be able to "react" in real time to a changing environment and to "reason" about ways to achieve its goals. This paper describes the approach taken on the UMass Mobile Perception Laboratory (MPL) to integrate independent processes (each solving a particular aspect of the navigation problem) into a fully capable autonomous vehicle.

1 Introduction

One of the goals of the autonomous vehicle effort at the University of Massachusetts is the development of a landmark-based navigation system capable of robust navigation both on-road and cross-country. While driving, the vehicle must exhibit classic driving behaviors, such as staying on the road or prescribed path, avoiding obstacles, etc., while using landmarks for determining the vehicle position.

The Mobile Perception Laboratory or MPL (Figure 1) is an experimental laboratory for testing and integrating different approaches to problems in autonomous navigation, including, but not limited to, landmark-based navigation, obstacle detection and avoidance, model acquisition and extension, road following, and path planning. It is therefore important that MPL have a software environment where multiple

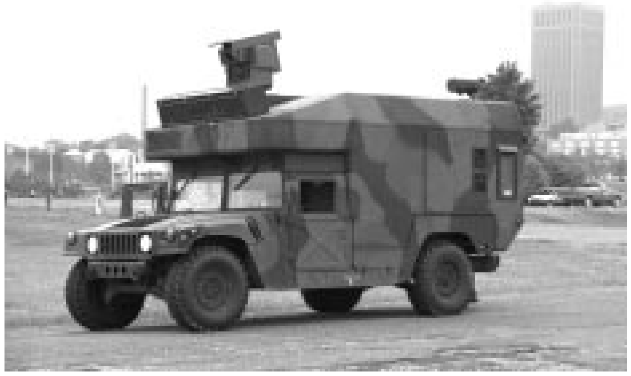
Figure 1: The Mobile Perception Laboratory, built on a modified HMMWV chassis, has computer controlled steering, throttle, and brakes, a complete research laboratory on the back.

visual modules, addressing different subtasks, can be easily integrated to provide autonomous driving functions, and where researchers can quickly experiment with different combinations and parameterizations of those modules. At the same time, MPL's software environment must be efficient enough to meet the demands of real-time navigation research. In addition, the system must be responsive to external events while executing pre-planned actions.

To achieve the desired flexibility we have implemented the control system as a "programable" finite state machine (FSM), where the states are the different modes of operation of the system (behaviors), and the transitions are the system's reactions to events (either external or internal). The composition of the states and the transitions is not fixed, i.e., the FSM can be tailored to the particular task the system is

*This work was supported by ARPA through U.S. Army TACOM under contract DAAE07-91-C-RO35 and the National Science Foundation under grant CDA-8922572.

†Computer Science Program, Mount Holyoke College - South Hadley, 01075 MA

trying to achieve. Each state is composed of several concurrent processes, each working on a particular aspect of the navigation problem.

The paper is structured as follow: first a brief review of behavior based systems, and the terminology used throughout this paper is introduced (section 2), followed by a discussion on our approach to behavior composition (section 3). Experiments on the MPL are described in section 4. Finally, section 5 summarizes the work.

2 What is a behavior?

In the robotics literature, *behavior* has generally been used to describe processes that connect perception to action, i.e., a behavior senses the environment and does something based on what was perceived. A combination of behaviors is also called a behavior, thus, a complex behavior can be achieved by combining simpler behaviors. In Brooks' subsumption architecture [3], the task of robot control is decomposed into levels of competence; each level, in combination with lower levels, defines a behavior. Payton, Rosenblatt and Keirsey [7] in their Distributed Architecture for Mobile Navigation system (DAMN), refer to behaviors as very low level decision-making processes which are guided by high level plans and combined through arbitration. In their DEDS work, Ramadge and Wonham [9] consider events to form the alphabet, Σ, of a formal language; a behavior is a sequence of events, or a string over Σ^*. Note that in this terminology every prefix of a string is also a behavior, i.e., the sequential combination of behaviors is a behavior.

These definitions, although consistent, can be confusing - the same term is used for individual processes and for the composition of these processes. To make the distinction clear, we have chosen to think of a behavior as a *mode of operation* [7], in which several *perception-action processes* [10] are executed concurrently. Each process converts sensory data into some kind of action (either physical or cognitive), and at any time may generate an *event* - a signal to let the system know that "something" significant has occurred. The system *reacts* to events by changing its behavior; hence the sequence of behaviors actually executed depends upon the sequence of events. Since the latter is unpredictable, so is the former. To program such a system, one must specify 1) which sets of processes constitute behaviors and 2) how the system should react to each event that a process can signal. We employ a finite state machine (FSM) - with states representing behaviors and transitions representing reactions to

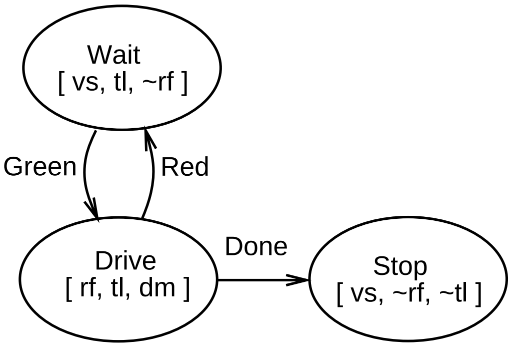

Figure 2: A driving system represented as a finite state machine (FSM): Listed below the state name are the perception-action processes that should be enabled in that state, and those that should be explicitly disabled (marked with a ˜). This example is composed of four perception-action processes: follow the road (*rf*), check for traffic lights (*tl*), monitor the distance traveled (*dm*), and stop the vehicle (*vs*).

events - to specify the system. A simple example of a system represented as a FSM is shown in Figure 2 [10]. This particular system will drive on the road a given distance while obeying traffic lights.

3 Behavior composition

In the FSM model presented in the previous section, the problem of system integration is tackled at two levels. First, perception-action processes are grouped together into behaviors. Then, behaviors are combined into a finite state machine. The composition of processes into behaviors presents a significant integration task by itself, however: the output of multiple concurrent processes must be combined into a single set of controller commands. This section presents the cascaded filter approach used for constructing behaviors for the UMass MPL, where each behavior corresponds to one state of the FSM.

One approach to constructing behaviors is the subsumption architecture proposed by [3]. In Brooks' original system higher level processes *subsume* lower level processes, i.e., whenever there is a conflict the higher level process takes control of the actuators (Figure 3). This approach for combining results can be problematic. Consider the case of a mobile robot with one process dedicated to keep the robot away from obstacles and another dedicated to move the robot towards a goal. Clearly, these two processes

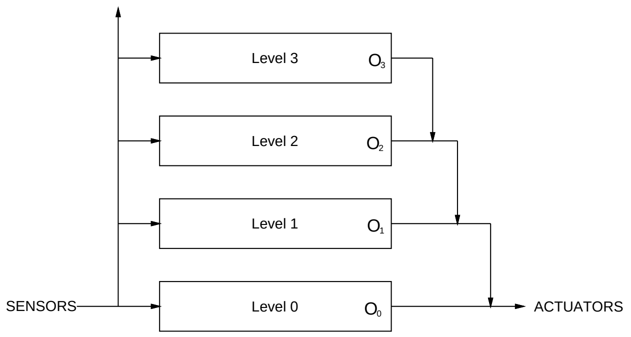

Figure 3: The subsumption architecture layered control. When process at level i wishes to take control, commands from process at levels below i are ignored.

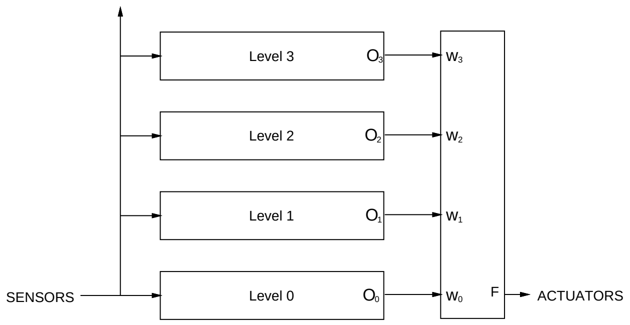

Figure 4: In the DAMN system each process votes on N possible values of a parameter. The votes are weighted and added: $vote_j = \sum_{l=0}^{L} w_l O_{lj}$. The value with the highest vote is selected and passed to the robot: $F = \{value_j \ : \ \forall i \ vote_j \geq vote_i\}$.

can generate contradictory commands. According to Brooks' model, the "higher" competence process wins. If the `avoid-obstacles` process is higher, then the robot may never achieve its goal because in the presence of obstacles, the commands generated by the `move-towards-goal` process are suppressed. On the other hand if the `move-towards-goal` is higher, then we may as well not have the `avoid-obstacles` process - its output is going to be ignored and the robot will go over obstacles.

The vertical decomposition of robot control, introduced with the subsumption architecture, does have advantages over more traditional methods (concurrency, flexibility, task independence, etc.), but a more flexible mechanism to combine outputs of different layers is needed. One possibility is simply adding a black box between the layered processes and the actuators. This box combines all the outputs in some rational way before passing them to the robot. In the DAMN [7] system, all controllable parameters are quantized into a discrete set of possible values for the parameter. Each process "votes" on each of the alternatives. The votes are combined by a weighted sum (processes are assigned weights according to their importance), and then the alternative with the highest vote is given to the actuators. In this case the black box is performing a selection of predefined values, based on a weighted sum of votes (Figure 4).

This approach, although better than pure subsumption, still presents some problems. In particular, general combination functions are hard to find, and have to take all the processes into account. This means that by selecting a general combination function the system loses part of the modularity present in the original architecture.

In our system we integrate the combination function idea of DAMN with the modularity of the subsumption architecture by cascading local combination functions, one at each level of the hierarchy (Figure 5). At each level the appropriate combination can be applied. Our motivation comes from the observation that a lot of relevant information, internal to each process, can't be used by a general combination technique. To utilize this information the combination must be done internally, but we still want a modular system. To achieve these two seemingly contradictory goals, each level, l, has a filter attached to it. This filter has two inputs and one output. The input coming from the level above (I_{l+1}) is in fact a value that can be given directly to the actuator; i.e. it is a legitimate control signal. The other input (I_l), as well as the filtering function (f_l) are level dependent (and as such can use any information locally available). The output of the filter (O_l) is the result of applying the combining function to both inputs:

$$O_l = f_l(I_l, I_{l+1}) \tag{1}$$

In order to keep the system modular the type of O_l is the same as that of I_{l+1}. An extra input (e) of the generic filter is the enabling signal. This input is useful to implement the *run* and *kill* tables of our FSM model. When a filter is disabled the input from the above level is passed unmodified. So the function computed by each filter is in fact:

$$O_l = (e \wedge f_l(I_l, I_{l+1})) \vee (\bar{e} \wedge I_{l+1}) \tag{2}$$

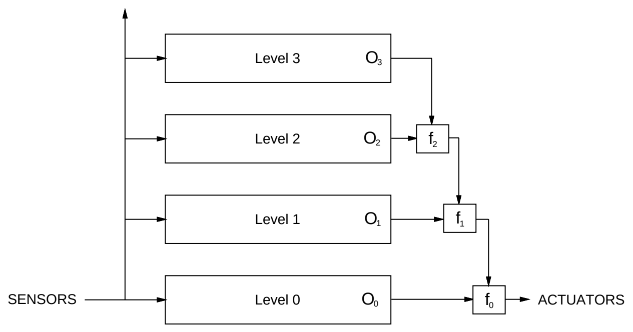

Figure 5: The filtering approach : At each level outputs are combined thru a different filter.

4 Experimental Results

Several experiments have been run to demonstrate the capabilities of the vehicle and the performance of the independent perception-action processes. First, each of the following perception-action processes was tested on the MPL:

- Precise pose determination (estimated bo be within 7% of real position) based on landmark model matching (this process is labeled `pe` in Figure 6) [2, 6].
- ALVINN - A neural-network based road follower (`rf`) [8] that produces a steering angle at up to 10Hz..
- Servo-based steering (`se`) [5, 4].
- Reflexive obstacle avoidance (`oa`) [1], capable, at 5MPH, of avoiding obstacles (at least 2 feet high) 30 feet away.
- A distance monitor (`dm`).
- Turning via dead reckoning (`dt`).

From these independent experiments several observations can be made:

1. Pose determination takes from 30 seconds to 1 minute. We believe the results to be within 7% of real position.
2. At 5 MPH, obstacles at 30 feet can be efficiently avoided. The obstacles must be at least 2 feet high.

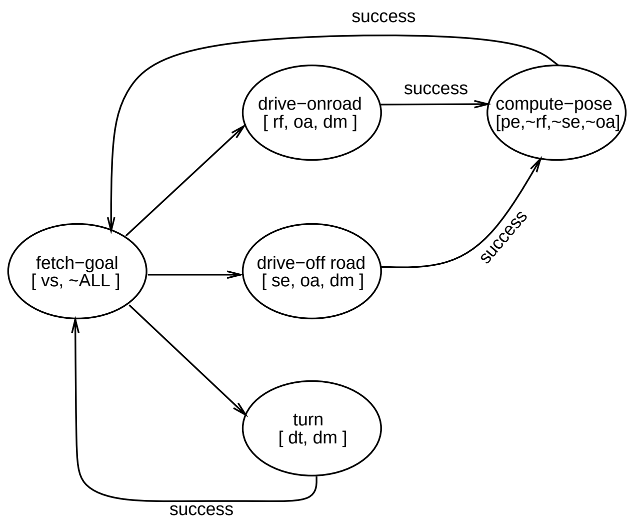

Figure 6: An augmented FSM for on/off road navigation : drive while avoiding obstacles for a specified distance, then check position. Each state in the FSM is a hierarchy of processes whose outputs are combined by cascaded filters, as shown in Figure 6.

With these results[1] in hand, these processes were combined and tested in the following script (Figure 6):

1. Drive on the road, while avoiding obstacles, for 100 meters.
2. Estimate vehicle position using landmarks.
3. Drive on the road, while avoiding obstacles, for 150 meters.
4. Estimate vehicle position using landmarks.
5. Turn left (at the experimental site, this command is a transition to off-road navigation).
6. Drive off road, while avoiding obstacles, for 50 meters.

The system has essentially two modes of operation, `drive-onroad` and `drive-offroad`. Although the avoid obstacles process, `oa`, is active in both modes, obstacles were introduced only in the off-road segment of the experiment. In each mode there is one "main driver" process controlling the steering angle, θ, and the velocity, v, of the vehicle. As in the DAMN system, the steering angle has been quantized into a discrete set of possible values. Both, the road following

[1]These results are discussed in detail in the respective papers.

process, `rf`, and the perceptual servoing process, `se`, produce a scalar angle from this set. The reflexive obstacle avoidance process, `oa`, produces a "vote" vector as follow:

$$vote_j = \begin{cases} d_j & \text{There is an obstacle at distance } d_j \\ & \text{in direction } \theta_j \\ \infty & \text{otherwise} \end{cases} \tag{3}$$

Clearly, the steering angle that should be given to the vehicle is one of those with infinity vote. The filter box attached to `oa` uses the steering angle from previous level, $I_l + 1$, as the initial point to search in the *vote* vector, i.e., the filter will output the free steering angle closest to the angle suggested by the "driver". In the case that an obstacle is present in all possible directions ($\forall j \; vote_j < \infty$) a more complex selection mechanism (based on distance to obstacle ($vote_j$), speed and suggested steering angle) must be used - currently in this situation the vehicle is stopped.

Note that in the experiment just described here, the vehicle is stopped before entering into the `compute-pose` mode. This is done to avoid contention for the color camera by the road following (`rf`) and pose estimation (`pe`) processes.

5 Conclusions

An autonomous vehicle should be able to react in real-time to a changing environment, but it should also be able to reason about its goals. In our system, these seemingly contradictory capabilities are achieved through the use of a"programable" finite state machine. In this model, reactions to events (either internal or external) are represented as state transitions. The system can react rapidly by following a transition table. Goal-directed reasoning is supported by the "programability" of the FSM - the system does not enforce what goes into a state or a particular transition table, instead it executes "scripts" which eventually will be written by an automated planner.

Within each state of the FSM, the outputs of different processes are combined thru a chain of "filters". The use of filters allows the system to use the information contained within each level, while keeping the system modular - any level can be taken out of the filtering chain and still leave a working system. The filters also allows us to easily switch the processes at each level making the embedding of the finite state machine model straightforward.

References

[1] S. 'Badal, S. Ravela, B. Draper, and A. Hanson. A practical obstacle detection and avoidance system. In *submitted to Workshop on Applications of Computer Vision*, 1994. Also available as a UMass Technical Report.

[2] R. Beveridge. Local search algorithms for geometric object recognition: Optimal correspondence and pose. Ph.D. Thesis CMPSCI TR93-71, University of Massachusetts at Amherst, 1993.

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

[4] C. L. Fennema. Interweaving reason, action and perception. Ph.D. Thesis COINS TR91-56, University of Massachusetts, 1991.

[5] C. L. Fennema and A. R. Hanson. Experiments in autonomous navigation. In *Proceedings of the Tenth International Conference on Pattern Recognition*, pages 24–31, 1990.

[6] R. Kumar. Model dependent inference of 3D information from a sequence of 2D images. Ph.D. Thesis COINS TR92-04, University of Massachusetts at Amherst, 1992.

[7] D. W. Payton, K. Rosenblatt, and D. M. Keirsey. Plan guided reaction. *IEEE Transactions on Systems, Man and Cybernetics*, pages 1370–1382, 1990.

[8] D. A. Pomerleau. Neural network based autonomous navigation. In Charles Thorpe, editor, *Vision and Navigation: The CMU Navlab*. Kluwer Academic Publishers, 1990.

[9] P. J. Ramadge and W. M. Wonham. The control of discrete event systems. *Proceedings of the IEEE*, 77(1):81–98, January 1989.

[10] B. Rochwerger et al. Executing reactive behavior for autonomous navigation. In *Proceedings of the 1994 IEEE Conference on Computer Vision and Pattern Recognition*, 1994. also in UMass technical report CMPSCI TR94-05.