# A Fine-Grained Alternative to the Subsumption Architecture for Mobile Robot Control

J. Kenneth Rosenblatt and David W. Payton

Hughes Artificial Intelligence Center
Hughes Research Laboratories
3011 Malibu Canyon Road
Malibu, California 90265

## ABSTRACT

We present an architecture for robot control which can be viewed as a very fine-grained layered architecture motivated by the principles underlying the subsumption architecture. The subsumption architecture provides a powerful means for defining intelligent robot control mechanisms through the layered composition of simple behaviors. However, we have found that there are basic limitations inherent in this architecture due to the inaccessibility of information internal to a behavior. While adhering to the basic concept of building a robot control system through successive layers of competence, task achieving behaviors in our system are fragmented into many smaller decision-making units. Each of these units simply has the task of transforming a set of input activations into an output activation, so that the role that any unit plays in the system is defined entirely by how it is connected to other units. This fine-grained nature of our architecture permits a more flexible arbitration of commands between behaviors and provides incrementally added behaviors with complete access to the internal state of existing behaviors. A detailed example of an operational robot control system for cross-country navigation is given.

## 1. Introduction

An autonomous mobile robot must be constantly involved in the processing of large amounts of disparate sensory data in order to produce meaningful actions in its environment. The ability of a control architecture to support this immense processing task in a timely manner is significantly affected by the organization of information pathways within the architecture. Some architectures have been explicitly designed to maximize the amount of parallel information flow from sensing to action so as to provide minimal delay in responding to a constantly changing environment. Our own experience with such a system has led us to a better understanding of the types of difficulties that can arise when action must be derived from the results of multiple independent decision-making processes.

We present an architecture for robot control which can be viewed as a very fine-grained layered architecture motivated by the principles underlying the subsumption architecture [1]. The positive aspects of the subsumption architecture design methodology are described in Section 2. In particular, we discuss the advantages of distributed control and perceptual knowledge and the benefits of a layered system where capabilities can be incrementally added. The limitations found in this architecture are then examined in Section 3. The inevitable loss of information caused by modules with internal state is identified as the root cause of these problems. A connectionist solution to these shortcomings is then proposed in Section 4. Behaviors in this architecture are constructed of units which maximize the amount of information available to other behaviors, so that the communication barriers between behaviors do not exist. In Section 5, we provide a detailed example of how a robot control system is constructed within this framework, illustrating how problems with our previous subsumption style architecture have been overcome. In this section, a mechanism is also introduced which is used to insure that the system will generate coherent action.

## 2. System Design Methodology

The subsumption architecture advocated by Brooks espouses principles of system design which we believe to be crucial to the successful construction of robot navigation systems [1]. A central tenet of the subsumption architecture is that a robot control system should be decomposed according to the desired external manifestations of the system, rather than divided along functional lines based on the structure of internal solutions.

### Horizontal Decomposition

As Brooks has argued, the traditional horizontal decomposition of the robot control problem, shown in Figure 1, has certain problems. In such architectures, sensory data is fused to build a centralized world representation, a plan is generated based on the current world representation, that plan is executed, and the process repeats. Such an organization creates bottlenecks by having centralized loci of representation and control. All sensor data must be fused and all tasks must be coordinated before any action can be taken, no matter how urgent or how insignificant that action may be. Besides introducing unnecessary delays, such a decomposition also leads to brittleness because the entire system degenerates if any part of the system is not functioning correctly.

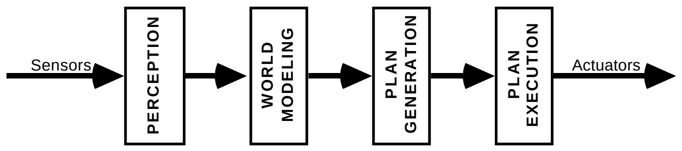


Figure 1. Traditional horizontal decomposition of robotic systems.

## Vertical Decomposition

To address the problems of horizontal decomposition, the subsumption architecture is decomposed vertically according to the activities which the robot is to exhibit. A *behavior* encapsulates the perception, planning, and task execution capabilities necessary to achieve one specific aspect of robot control. Because an individual component in such a system is capable of producing meaningful action, behaviors can be composed to form *levels of* competence [1], each of which endows the robot control system with a particular capability. Successive levels can be incrementally added in order to enhance the functionality of the robot. Level *i* in Figure 2 would endow the robot control system with a basic level of competence such as the ability to move without colliding. When new levels are added to provide greater functionality, the existing levels remain intact so that the capabilities they provide are still available.

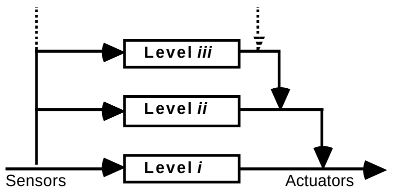


Figure 2. Vertical decomposition of robot control system yields successive levels of competence.

## Distributed Control

An important result of the vertical decomposition of the subsumption architecture is that control is distributed among a number of behaviors operating asynchronously and in parallel, each concerned with a very narrow aspect of the navigation problem. A behavior module is implemented as an augmented finite state machine which has internal instance variables as well as states. These modules are interconnected through a network of message-carrying wires, each receiving multiple inputs and generating multiple outputs as shown in Figure 3. A module may overwrite the data on the input or output line of another module; this is referred to as *subsumption* or *inhibition*, respectively.

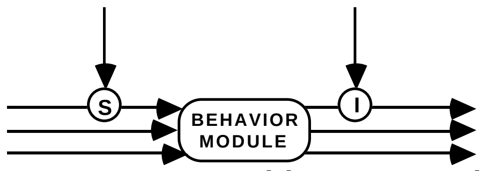


Figure 3. A module receives input data which may be subsumed, and outputs data which may be inhibited.

The navigation system used by Hughes for control of the Autonomous Land Vehicle (ALV) employs a distributed control mechanism very similar to that used within the subsumption architecture [2]. This system differs from the subsumption architecture in that behaviors are procedural units rather than finite state machines, and that subsumption and inhibition are effected through a blackboard system rather than being explicitly wired between modules. These differences allow the system to arbitrate commands according to data content as well as priority, and behavior priorities may be varied dynamically. The successful use of subsumption style systems to control both the ALV and the Artificial Insects [3] of the MIT AI Lab demonstrates that coherent action can be produced from the interaction of independent concurrent behaviors.

### Distributed World Representation

The vertical decomposition of the subsumption architecture also eliminates the bottleneck which occurs when sensor data must be fused with data from other sensors and integrated with previous data to form a central world representation. Instead, behaviors only receive that sensory data which is directly relevant to their particular decision-making needs. In this way, each layer requires only a partial view of the world, so there is no need to fuse all available data into a single coherent model. Attention can instead be focused on the production of coherent action. By appropriately fusing behavior commands through arbitration, a robot control system can respond to its environment without the delays imposed by sensor fusion. In effect, sensor fusion is supplanted by command fusion.

## 3. LIMITATIONS OF THE SUBSUMPTION ARCHITECTURE

Although the subsumption architecture has many good points, we have found through experience with a similar architecture that there are limitations. Brooks states that one of the requirements for a robot control system is that it be capable of satisfying multiple, possibly conflicting goals [1]. In many cases, however, this cannot be achieved. There can be situations in which the commands from two related behaviors are in direct conflict with one another. As the name "subsumption" implies, such conflicts are often resolved by having one behavior's commands completely override the other's. Even though it may be highly desirable for the system to act to simultaneously accommodate the needs of both behaviors, there is no way to arrive at a compromise solution.

### Inadequate Command Fusion

This inability to compromise stems from the fact that the priority-based arbitration of commands masks much of what is known within each individual behavior. In the process of selecting a command, behaviors may have to weigh several alternatives in order to make an appropriate choice. In other behaviors, it is more natural to rule out inappropriate action rather than to select the most suitable one, yet their only means of gaining influence is by completely subsuming or inhibiting the commands of other behaviors. A compromise between behaviors is not achievable because a behavior cannot express its relative preferences for the alternatives but can only indicate its first choice. Whenever one command is inhibited by another, the information contained within that command is completely lost.

Figure 4 illustrates the nature of the problem. Each behavior considers the five available command alternatives, labeled **A** through **E** in the figure. As indicated by the solid line originating from Behavior 1, it has chosen **A** as the alternative which best fulfills its requirements. As suggested by the dashed lines, Behavior 1 has also determined that **B** and **E** are less desirable yet still satisfactory, but it has no means of expressing this information. Likewise, Behavior 2 considers **D** to be the best choice but would also be satisfied with alternative **E**.

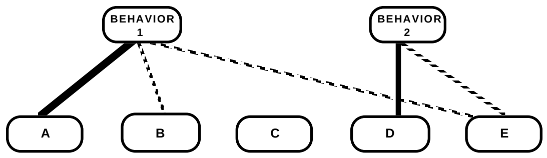


Figure 4. Behaviors must consider several alternatives in making a decision.

If the behaviors are configured as in Figure 5, then the output will be **A** if Behavior 1 is actively inhibiting the output line of Behavior 2, or **D** if Behavior 1 remains quiescent. In either case, the concerns of one of the behaviors will be disregarded. Since information has been lost, there is no way to properly arbitrate between the two disparate commands. This is unfortunate since there does exist a command option, **E**, which is consistent with the requirements of both behaviors.

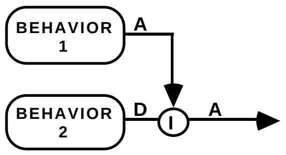


Figure 5. Information loss with priority-based arbitration of behaviors

In practice, during our development of behaviors for cross-country navigation [4], we found that this situation could only be dealt with by creating one larger behavior which issued commands based on the data and concerns of the two smaller behaviors, thus violating the goal of maintaining modularity and distributed control. The fundamental problem is that a behavior's output is an abstraction of all the information which that behavior computed in order to reach its conclusion.

### Inaccessibility of Internal State

Another difficulty found with encapsulating behaviors within a set of finite state machines is that the internal state of these behaviors becomes inaccessible to successive layers. The designer of a given layer in a subsumption network cannot anticipate the needs of future layers, and, ideally, the designer should not have to predict all future interactions with the current layer. In practice, however, each new layer places new demands on the lower layers, and, invariably, there is a need to modify behaviors in existing layers in order to provide for the addition of new levels of competence. An example of this occurring within the subsumption architecture can be found in [5], where a module (Motor) in the first level of competence had to be modified to provide a new output (Travel) for use by a second level module.

**Disruption of Levels of Competence**

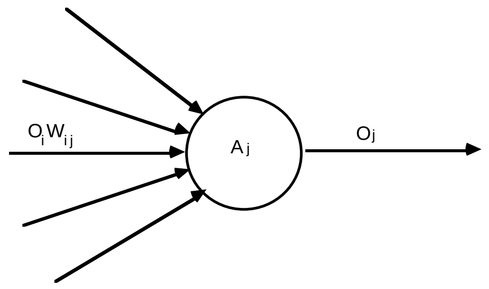
Another problem with one behavior completely inhibiting the output of another is that it undermines one of the strengths of the subsumption architecture: that a given level of competence can always be counted on to reliably provide certain capabilities regardless of subsequent levels. For example, a level of competence may be established which allows the robot to avoid obstacles, but the robot may collide with an obstacle when the output of that level is inhibited. Likewise, when the input to a module is subsumed, the information contained within the overridden data is lost. With the loss of this data, it can no longer be guaranteed that the behaviors will be able to adequately perform the functions for which their level of competence was defined.

## 4. Enhancement of Subsumption Architecture

Our solution to the problems described above is to make the behaviors as fine-grained as possible so that a module has no inaccessible internal state. Instead of finite state machines with internal states and instance variables, behaviors are comprised of atomic functional elements.

**Fine-Grained Behaviors**

In our alternative connectionist architecture, behaviors are divided into a collection of simple decision-making units. As shown in Figure 6, each unit receives multiple weighted inputs from other units and from external data sources, computes an activation level, and issues a single output [6, 7]. The activation and output functions for each unit can be any mapping from real numbers to a single real number value. The only constraint is that these functions be defined for inputs between -1 and 1, inclusive, and that the outputs be within this same range. The network is highly structured: there is no attempt to interconnect homogeneous units which form a distributed knowledge representation. Each unit represents a specific concept which the designer establishes through carefully chosen connections, and the functions for each unit are designated based on the desired output characteristics for that unit.



$O_i$ is the output of unit i.
$W_{ij}$ is the weight on the link from unit i to unit j.
$A_j = f_A(O_1W_{1j}, ..., O_nW_{nj})$, is the activation level of unit j with n weighted inputs.
$O_j = f_O(A_j)$, is the output of unit j.

Figure 6. A network unit.

## Distributed Command Arbitration

Each behavior is now distributed among several units so that the desirability for each choice can be explicitly expressed by the output value of the corresponding unit. In addition, since outputs may be negative, a behavior can directly indicate which choices it finds to be undesirable. To illustrate, let us consider how a compromise can now be reached in the situation described earlier where two behaviors are choosing between a set of command alternatives labeled **A** through **E**. As shown in Figure 7, behavior alternatives are represented by the appropriate place-coding of the component units.

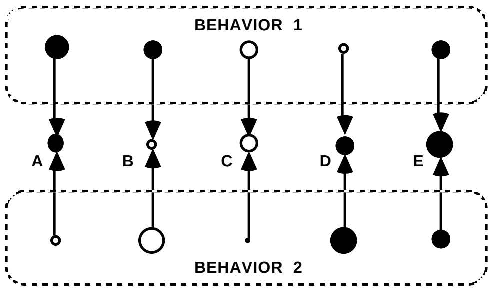


Figure 7. Fine-grained behaviors explicitly indicate preference for each alternative.

The level of activation for each unit in the figure is indicated by the diameter of the circle representing that unit, with a filled circle signifying positive activation and an open circle denoting a negative activation level. The output of each behavior unit serves as input to the corresponding command unit, as indicated by the arrows in the figure. The relative priority of each behavior is established by the weight on the links to the command units. Positive and negative activations are combined for each command unit, and the one with the highest activation (**E** in this example) is chosen as the option which best satisfies the needs of both behaviors. Thus, a connectionist architecture makes it possible to simultaneously satisfy the constraints of multiple independent behaviors.

## Additional Levels Of Competence

As we build up levels of competence, the connections between a new behavior and existing behaviors provide a rich means for expressing commands. New behaviors do not completely subsume the function of existing behaviors, but merely bias decisions in favor of different alternatives. Thus, an established level of competence remains intact and need not be suppressed before behaviors within higher levels of competence may express their concerns.

In keeping with our philosophy of reducing the grain-size of decision-making units, the function computed by each unit in the network may have no internal variables. Thus, the state of a behavior is easily accessible to any other behavior and there is no need to define an interface which might need subsequent revision.

## 5. An Instance of Fine-Grained Behaviors for Robot Control

To illustrate how fine-grained distributed behaviors can be constructed for robot control and how capabilities may be incrementally added, we present a portion of the network which has been used to control a simulated robotic vehicle.

### First Level of Competence

The first level of competence allows the robot to avoid obstacles while wandering aimlessly. This is accomplished by behaviors which react to sensors on-board the vehicle.

#### Sensory Input

One type of processed sensory data available to the planner is the *Vehicle Model Trajectory* (VMT) [8]. A VMT indicates how far the vehicle may safely travel along a linear trajectory in each of several headings, and whether an obstacle is known to exist in that direction. In Figure 8a, the distance known to be safe at each azimuth is proportional to the length of the line drawn, and the dark squares indicate obstacles at -x° and -y°.

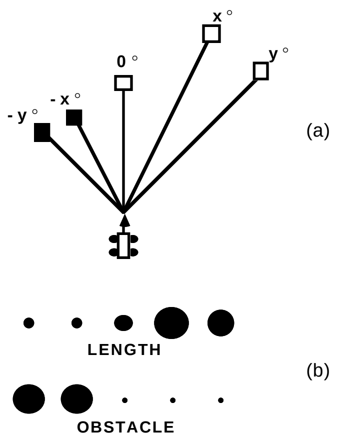


Figure 8. (a) Vehicle Model Trajectory output, and (b) corresponding input units.

#### Input Units

To make the data available for use by those behaviors which base their decisions on VMTs, the first step is to create input units which represent VMT data. To this end, two groups of units are created, one to represent the trajectory lengths and one to represent trajectory obstacles. These groups of units are labeled LENGTH and OBSTACLE, respectively, as shown in Figure 8b. Each group contains one unit for each trajectory in a VMT, and the direction that unit represents is indicated by its position within the group. Thus, if the directions of the trajectories were as in Figure 8, the first unit in each group would represent data concerning the trajectory at -y°, the second unit would correspond to the trajectory at -x°, the third would be 0° straight ahead, and so on. The activation of a LENGTH unit is proportional to the length of the corresponding trajectory, and the activation of an OBSTACLE unit is 1 if the corresponding trajectory terminates at a known obstacle or 0 otherwise.

The next step is to use the LENGTH and OBSTACLE input units described above to determine which trajectories pose hazards and which can be followed safely. This is accomplished by the SAFE TRAJECTORY group shown in Figure 9. Each unit in this group combines these inputs to produce an output value which reflects how safe it is to proceed in each direction. Only links for the center unit in a group are shown for clarity; the links for the other units are equivalent. A plus (+) or minus (-) sign next to a link indicates the sign of that link's weight. Thus, the activation of a SAFE TRAJECTORY unit increases with the activation of the corresponding LENGTH unit and decreases with the activation of the corresponding OBSTACLE unit.

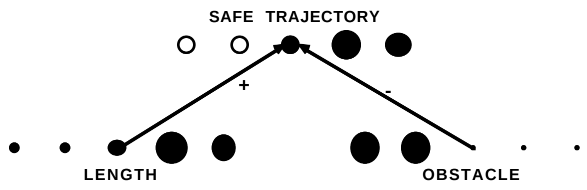


Figure 9. The combination of LENGTH and OBSTACLE inputs.

The SAFE TRAJECTORY group indicates the relative safety of traveling in each direction when considered in isolation, but it is also desirable to avoid traveling adjacent to an unsafe trajectory. Therefore, an intermediate set of units is introduced which represents the presence of undesirable neighbors. The UNSAFE LEFT NEIGHBOR and UNSAFE RIGHT NEIGHBOR groups, shown in Figure 10, provide negative activation whenever a neighbor is more hazardous than the trajectory being evaluated. The SAFE PATH group can then combine these to indicate how safe each trajectory is when its neighbors are taken into consideration as well. Thus, since the trajectory at 0° is relatively short, the UNSAFE LEFT NEIGHBOR unit for x° is activated. The result is that the SAFE PATH unit for y° is more activated than the one for x°, even though the opposite is true for the SAFE TRAJECTORY units. The units in Figure 10 constitute a behavior which expresses preferences for the turn commands based on VMT input.

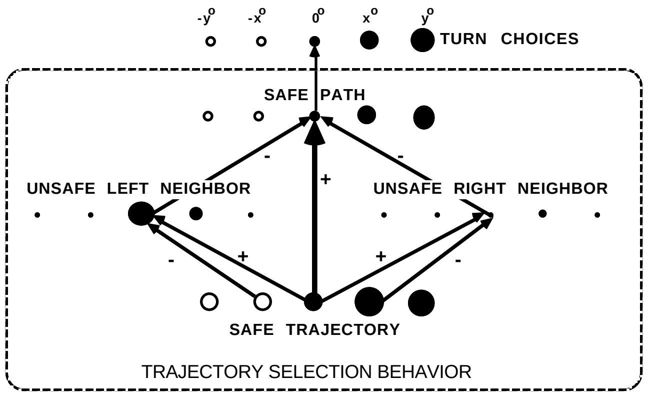


Figure 10. TRAJECTORY SELECTION behavior expresses preference for each turn choice.

The outputs of the TRAJECTORY SELECTION behavior are connected to the TURN CHOICES group of units, which represent the available turn choices. Since only one behavior is inputting preferences to these units, their activations correspond exactly with the activations of the SAFE PATH units; however, as other behaviors are added which influence the choice of a heading, the TURN CHOICES activations will reflect all these behaviors. The activation levels of the TURN CHOICES group are then fed into a winner-take-all network, which results in exactly one of the CHOSEN TURN units having an activation of 1 and all the rest having an activation of 0, as shown in Figure 11. The choice this one active unit represents is then used in sending turn commands to the vehicle.

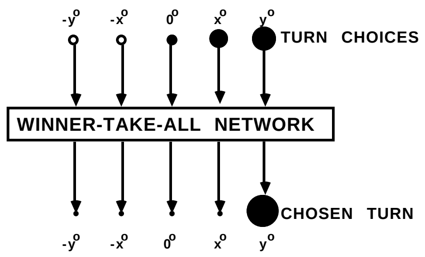


Figure 11. Winner-take-all network selects the most activated choice.

TRAJECTORY SPEED BEHAVIOR

The internal states of this behavior are also available for use by other behaviors. For example, once a choice has been made as to which direction the vehicle should head, it is necessary to select a suitable speed. In our previous non-connectionist implementation of behaviors, the length of the trajectory in the chosen direction was used in computing a speed. However, much more information is now accessible from units within the TRAJECTORY SELECTION behavior; the activation of a SAFE PATH unit reflects not only the length of the corresponding trajectory, but also the presence or absence of an obstacle and the traversability of the adjacent trajectories. Since all these factors influence how safe travel along a particular trajectory is, the value of the chosen SAFE PATH unit is a better measure for determining an appropriate speed. This value can be extracted by the "Chosen Value" subnetwork shown in Figure 12. The CHOSEN SAFE PATH units multiply the inputs from the SAFE PATH and CHOSEN TURN units, so that a CHOSEN SAFE PATH unit is nonzero only if the output of the linked CHOSEN TURN unit is 1, in which case the value is equal to the corresponding SAFE PATH unit. The outputs of these units are then input to the CHOSEN SAFE PATH VALUE unit, where a sum yields the desired result. The SPEED unit then sets the vehicle speed as a function of this value.

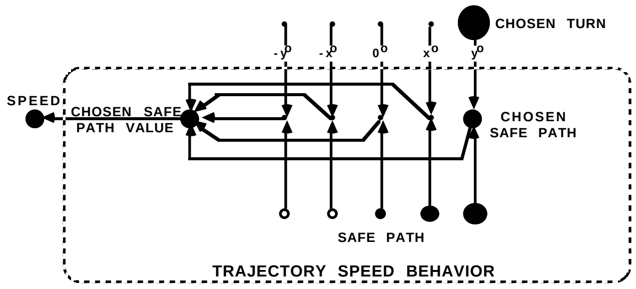


Figure 12. TRAJECTORY SPEED behavior.

ADJUST SPEED BEHAVIOR

As the vehicle proceeds along the chosen trajectory at the established speed, the rate of travel should diminish since the distance ahead known to be safe is reduced. This task can be accomplished very simply by making use of the existing units and adding in a single unit which represents the distance the vehicle has traveled since receiving the last VMT input. This MOVEMENT unit has a negatively weighted link to the SAFE TRAJECTORY units so that it decreases their activation, as shown in Figure 13. This leads to a decrease in the activation of the SAFE PATH units, so that they now represent the safety of travel at all times, rather than just at the time the VMT input is received. Thus, the functionality of existing behaviors can be extended by providing them with additional information, without a need to subsume existing data.

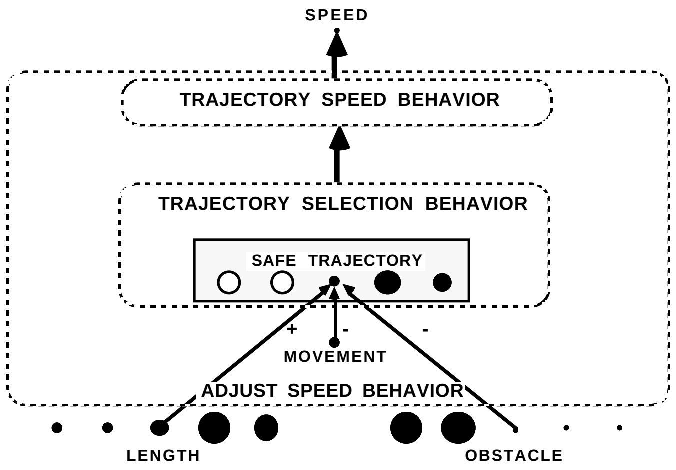


Figure 13. ADJUST SPEED behavior.

Thus, the fine-grained distributed implementation of a behavior not only provides access to all internal state, but it also allows for behaviors to share subparts and thereby avoid computational and functional redundancy. Once the vehicle is proceeding along in the chosen direction, the MOVEMENT unit's output will increase, leading to a decrease in the activation of each SAFE PATH unit. This has the desirable effect of reducing the output of the CHOSEN SAFE PATH VALUE unit so that the speed will be diminished. However, a deleterious side effect could occur: as the values of the SAFE PATH units change, the relative activations of these units could change so that the unit which originally had the greatest output, and therefore was chosen, might no longer have the largest output. While in the process of following a chosen trajectory, that choice could suddenly change, leading to unpredictable behavior.

### SELECTIVE UNIT UPDATE

To overcome this problem, a "latch" of some sort is needed so that a decision cannot be countermanded prematurely. The TURN CHOICES units need to receive new values from the SAFE PATH units when the changes reflect a new VMT input, but would like to ignore new values resulting from a change in the MOVEMENT input. This is accomplished by propagating markers through the network when new input arrives and allowing units to conditionally update their activation based on the arrival of these markers. When a new value is entered into an input unit, for one time step the links emanating from that unit not only carry a numerical value, but also transmit a tag which is unique to that unit (or unique to the group of units it is a member of). Thus, when vehicle movement is reported and a new value is input to the MOVEMENT unit, for the next time step the links from that unit will bear a *Movement* new input tag, as well as a new value from the unit's output. Next, each of the SAFE TRAJECTORY units will perform the default action, which is to receive the new data and tag, apply its unit functions to compute a new activation level and output, and propagate the tag through its outgoing links. The *Movement* tag will continue to propagate through the network until it reaches the TURN CHOICES group of units. The definition of these units specifies that their activation values should only be updated when a *Length* or *Obstacle* new input tag is received, so their values will remain unaffected when the *Movement* tag arrives. Thus, the outputs of the SAFE PATH units will have changed so that the activation of the CHOSEN SAFE PATH VALUE unit will change and the vehicle speed will be reduced, but the TURN CHOICES units will retain the activations which were computed when a VMT last arrived, so that the choice of a turn direction will not change until new VMT input is available. Propagation markers are also used to detect and avoid cycles in the network.

## Second Level of Competence

### GRADIENT FIELD

The above behaviors constitute a basic level of competence which endows the robot with the ability to avoid obstacles indicated in VMT input. Another source of information that is used in deciding upon a vehicle heading is the *gradient field* [9], which indicates the globally optimal route to a goal from any location. A representative gradient field is shown in Figure 14. The optimal path from any given point to the goal can be traced out by following the arrows. During the execution of a planned route, the gradient field is indexed by the current vehicle position and a desired heading is determined.

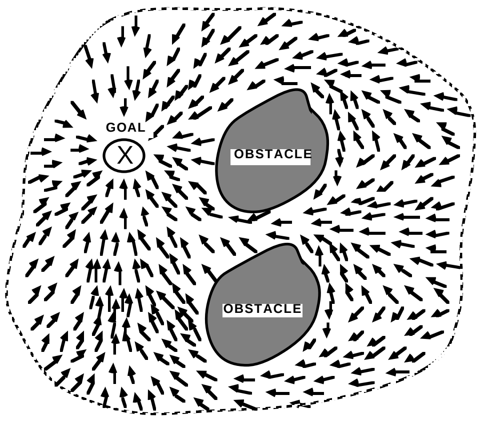


Figure 14. Gradient field.

### PURSUE GOAL BEHAVIOR

The units which receive the gradient field input are shown in Figure 15. In this example, the gradient field indicates that -x° is currently the preferred direction of travel, so the corresponding unit is the most highly activated. The activation for units representing each of the other directions is a function of the angle between that direction and the preferred direction. Note that a heading of y° significantly diverges from -x°, so that input unit has a negative activation. The FOLLOW GRADIENT units constitute a behavior which uses the GRADIENT FIELD units as input. This PURSUE GOAL behavior favors heading in a direction consistent with the gradient field. The choice of activation function for these units is designed so that the sensitivity to the divergence angle is initially small and increases as the divergence increases.

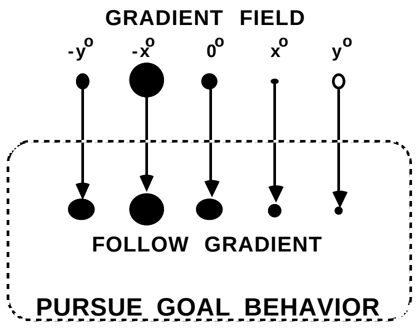


Figure 15. PURSUE GOAL behavior.

The use of a single desired heading as input from the route planner is a vestige from our previous system. This will be changed so that the desirability of each direction is evaluated independently so that the gradient field input will be more detailed. Thus, the explicit representation of each alternative also allows for greater communication between components of the planning system.

## COMMAND FUSION

The PURSUE GOAL behavior endows the robot control system with a higher level of competence by allowing the robot to achieve a specific goal. In our earlier subsumption style architecture, one behavior had to be constructed which combined gradient field input with VMT input, internally fusing commands in order to reach a decision. In our connectionist architecture, the command fusion can be achieved though the network itself via the TURN CHOICE units, as shown in Figure 16. Therefore, the TRAJECTORY SELECTION behavior can operate as always, utterly unaffected by the addition of the PURSUE GOAL behavior. The TURN CHOICES unit for x° is now the most activated, so that would be chosen instead of y° if both behaviors are used. Note that because the TURN CHOICES units have been defined to only update their activation based on new *Length* or *Obstacle* input, a change in the gradient field input will not have an effect on the chosen turn direction while a trajectory is being followed, but will have an effect once new VMT input arrives. Thus, the selective unit update mechanism once again provides a way to combine behaviors while maintaining the integrity of their decisions.

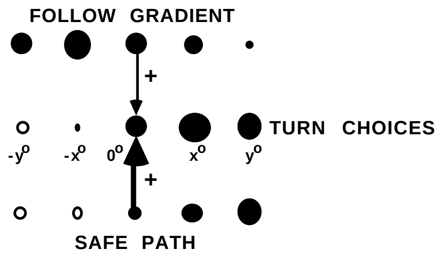


Figure 16. Fusion of commands from the TRAJECTORY SELECTION and PURSUE GOAL behaviors.

### GRADIENT SPEED BEHAVIOR

The FOLLOW GRADIENT units also influence the vehicle speed in the same manner as the SAFE PATH units do; i.e., a CHOSEN FOLLOW GRADIENT VALUE unit is created which has a link to the SPEED unit so that the vehicle speed will vary according to whether the robot is traveling with or against the gradient field. This configuration is shown in Figure 17.

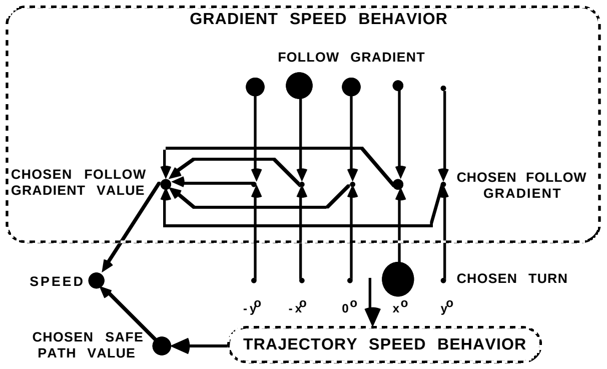


Figure 17. GRADIENT SPEED behavior influences vehicle speed.

The behaviors described above are part of a larger network which has been implemented and successfully used to navigate a vehicle in simulation. The vehicle simulation is a detailed and complex one, and behaviors previously developed using this simulation have been successfully transferred without modification for use on an actual autonomous vehicle [8]. In simulation, the vehicle winds its way around a variety of obstacles, always avoiding any path indicated as unsafe by a VMT; but the decisions of the robot control system are also influenced by the gradient field input so that it completely avoids mapped areas which are known to be difficult to traverse, and the vehicle ultimately reaches the goal point.

## 6. CONCLUSION

By identifying the difficulties resulting from information loss within a subsumption style architecture, we have been able to develop an alternative architecture which overcomes many of these problems. Our primary objective was to eliminate the strict priority-based scheme for arbitrating behavior commands, and allow the selection of commands which best meet the demands of all behaviors. The resulting connectionist architecture provides a very fine-grained methodology for constructing behaviors. Since the processing elements in the system are extremely simple, behaviors are defined by the connections between elements rather than by the properties of the elements themselves. Since no information is hidden within these elements, the internal representations used in one behavior are completely accessible to all other behaviors. The result is a system with no internal communication barriers.

Although our architecture takes a different approach to behavior modularity and command arbitration, our overall approach to system development is guided by the same principles which motivated the subsumption architecture. In particular ,we view the process of system development as one of building successive layers of competence. However, we find that by taking a finer grained approach, new layers can be introduced with minimal alteration of existing layers and without disrupting their functionality.

## 7. ACKNOWLEDGEMENTS

Many of the concepts presented in this paper came about through fruitful discussions with Thomas Caudell, Charlie Dolan, and Nigel Goddard. This work was funded in part by DARPA contract DACA76-85-C-0017.

## 8. REFERENCES

[1] R. A. Brooks, "A Robust Layered Control System for a Mobile Robot," IEEE Journal of Robotics and Automation, vol. RA-2, no. 1, pp. 14-23, April 1986.

[2] D.W. Payton, "An Architecture for Reflexive Autonomous Vehicle Control", in Proceedings of the IEEE Conference on Robotics and Automation, San Fransisco, CA, April 1986, pp. 1838-1845.

[3] R. A. Brooks, "Achieving Artificial Intelligence Through Building Robots," MIT AI Memo 899, May 1986.

[4] D.M. Keirsey, D.W. Payton, and J.K. Rosenblatt, "Autonomous Navigation in Cross Country Terrain", in Proceedings of the Image Understanding Workshop, Boston, MA, April 1988, pp. 411-416.

[5] R. A. Brooks, "A Layered Intelligent Control System for a Mobile Robot," in Proceedings of ISSR; Third International Symposium of Robotics Research, Gouvieux, France, October 1985.

[6] J. A. Feldman and D. H. Ballard, Connectionist Models and Their Properties. Cognitive Science, vol. 6, pp. 205-254, 1982.

[7] D. E. Rumelhart and J. L. McClelland, Parallel Distributed Processing, Cambridge: MIT Press, 1986, Vol. 1, ch. 2.

[8] M. J. Daily, J. G. Harris, D. M. Keirsey, K. E. Olin, D. W. Payton, K. Reiser, J. K. Rosenblatt, D. Y. Tseng, and V. S. Wong, "Autonomous Cross-Country Navigation with the ALV," in Proceedings of the IEEE Conference on Robotics and Automation, Philadelphia, PA., April 1988. (also in Proceedings of the DARPA Knowledge-Based Planning Workshop, Austin, Texas, December 1987, pp. 718-726)

[9] D. W. Payton, "Internalized Plans: A Representation for Action Resources", presented at the Workshop on Representation and Learning in an Autonomous Agent, Faro, Portugal, November 16-18, 1988.