Appeared in proceedings of 1995 AAAI Spring Symposium on Lessons Learned from Implemented Software Architectures for Physical Agents, Stanford, CA, March 27-29, 1995: AAAI Press, Menlo Park, CA.

DAMN: A Distributed Architecture for Mobile Navigation

Julio K. Rosenblatt
Robotics Institute
Carnegie Mellon University
Pittsburgh, PA 15213
jkr@cmu.edu

Abstract

An architecture is presented where a collection of distributed task-achieving modules, or *behaviors*, cooperatively determine a mobile robot's path by expressing their preferences for each of various possible actions. An arbiter then performs *command fusion* and selects that action which best satisfies the prioritized goals of the system, as expressed by the behaviors and their associated weights. Examples of implemented systems are given, and future research directions in command fusion are discussed.

Introduction

In order to function in unstructured, unknown, or dynamic environments, a mobile robot must be able to perceive its surroundings and generate actions that are appropriate for that environment and for the goals of the robotic system. To function effectively, an architectural framework for these sensing and reasoning processes must be imposed to provide a structure with which the system may be developed, tested, debugged, and understood. The system must also deal with uncertainty and incomplete knowledge of its environment and of the effects of its own actions. Another crucial consideration is the ability to respond to potentially dangerous situations in real-time while maintaining enough speed to be useful.

In addition, mobile robots need to combine information from several different sources. For example, the CMU Navlab vehicles are equipped with sensors such as video cameras, laser range finders, sonars, and inertial navigation systems, which are variously used by subsystems that follow roads, track paths, avoid obstacles and rough terrain, seek goals, and perform teleoperation. Because of the disparate nature of the raw sensor data and internal representations used by these subsystems, combining them into one coherent system which combines all their capabilities has proven to be very difficult. Many architectures espousing diverse principles of design methodology have been proposed over the years, but few have proved capable of integrating subsystems that have each been developed independently using whichever paradigm best achieves the task for which it is intended.

The earliest work in robot control architectures attempted to reason by manipulating abstract symbols using only pure logic (Nilsson, 1984). The limitations of this top-down AI approach led to a new generation of architectures designed in a bottom-up fashion to provide greater reactivity to the robot's surroundings, but sacrificed generality and the ability to reason about the system's own intentions and goals (Brooks, 1986; Agre & Chapman, 1987; Arkin, 1987).

It has been argued that a hierarchical approach is needed which allows slower abstract reasoning at the higher levels and faster numerical computations at the lower levels, thus allowing varying trade-offs between responsiveness and optimality as appropriate at each level (Payton, 1986; Albus, McCain & Lumia, 1987). While such an approach provides aspects of both deliberative planning and reactive control, the top-down nature of hierarchical structures tends to overly restrict the lower levels so that newly received information cannot be fully taken advantage of (Payton, Rosenblatt & Keirsey, 1990). In hierarchical architectures, each layer controls the layer beneath it and assumes that its commands will be executed as expected. Since expectations are not always met, there is a need to monitor the progress of desired actions and to report failures as they occur (Simmons, Lin & Fedor, 1990). In an unstructured, unknown, or dynamic environment, this approach introduces complexities and inefficiencies which could be avoided if higher level modules participated in the decision-making process without assuming that their commands will be strictly followed.

Experience over the years with different architectures and planning systems for mobile robots has led me to a distributed approach where an arbiter receives votes for and against commands from each subsystem and decides upon the course of action which best satisfies the current goals and constraints of the system. The architecture is designed with the underlying belief that centralized arbitration of votes from distributed, independent decision-making processes provides coherent, rational, goal-directed behavior while preserving real-time responsiveness to its immediate physical environment. Furthermore, a framework for developing and integrating independent decision-making modules communicating with such arbiters facilitates their development and leads to evolutionary creation of robust systems of incrementally greater capabilities.

The Distributed Architecture for Mobile Navigation has been successfully used to integrate the various subsystems mentioned above, thus providing systems that perform road following, cross-country navigation, or teleoperation while avoiding obstacles and meeting mission objectives. In addition to its use on the CMU Navlab vehicles, DAMN has also been used on outdoor test vehicles at Martin Marietta and on indoor robots and simulated environments at the Hughes Research Labs..

The Distributed Architecture for Mobile Navigation

Deliberative planning and reactive control are equally important for mobile robot navigation; when used appropriately, each complements the other and compensates for the other's deficiencies. Reactive components provide the basic capabilities which enable the robot to achieve low-level tasks without injury to itself or its environment, while deliberative components provide the ability to achieve higher-level goals and to avoid mistakes which could lead to inefficiencies or even mission failure. But rather than imposing an hierarchical structure to achieve this symbiosis, the Distributed Architecture for Mobile Navigation (DAMN) takes an approach where multiple modules concurrently share control of the robot. In order to achieve this, a common interface is established so that modules can communicate their intentions without regard for the level of planning involved (Langer, Rosenblatt & Hebert, 1994).

A scheme is used where each module votes for or against various alternatives in the command space based on geometric reasoning; this is at a higher level than direct actuator control, but lower than symbolic reasoning. This reasoning at the geometric level creates a bridge between the high-level goals of an AI planner and the low-level motor skills of a controller and is crucial to the successful operation of a robotic system in the real world, and yet it is the least understood.

Figure 1 shows the organization of the DAMN architecture, in which individual behaviors such as road following or obstacle avoidance send votes to the command arbitration module; these inputs are combined and the resulting command is sent to the vehicle controller. Each action-producing module, or *behavior*, is responsible for a particular aspect of vehicle control or for achieving some particular task; it operates asynchronously and in parallel with other behaviors, sending its outputs to the arbiter at whatever rate is appropriate for that particular function. Each behavior is assigned a weight reflecting its relative priority in controlling the vehicle. A mode manager may also be used to vary these weights during the course of a mission based on knowledge of which behaviors would be most relevant and reliable in a given situation.

DAMN is a behavior-based architecture similar in some regards to reactive systems such as the Subsumption Architecture (Brooks, 1986). In contrast to more traditional centralized AI planners that build a centralized world model and plan an optimal path through it, a behavior-based architecture consists of specialized task-achieving modules that operate independently and are responsible for only a very narrow portion of vehicle control, thus avoiding the need for sensor fusion. A distributed architecture has several advantages over a centralized one, including greater reactivity, flexibility, and robustness (Payton, Rosenblatt & Keirsey, 1990). However, one important distinction between this system and purely reactive systems is that, while an attempt is made to keep the perception and planning components of a behavior as simple as possible without sacrificing dependability, they can and often do maintain internal representations of the world. Brooks (1993) has argued that "the world is its own best model", but this assumes that the vehicle's sensors and the algorithms which process them are essentially free of harmful noise and that they can not benefit from evidence combination between consecutive scenes. In addition, disallowing the use of internal representations requires that all environmental features of immediate interest be visible to the vehicle sensors at all times. This adds unnecessary constraints and reduces the flexibility of the overall vehicle system

The DAMN architecture is designed to provide the basic capabilities essential to any mobile robot system, or *first level of competence* in the parlance of the Subsumption Architecture. In DAMN, this consists of safety behaviors which limit turn and speed to avoid vehicle tip-over or wheel slippage, obstacle avoidance behaviors to prevent collisions, as well as various auxiliary behaviors (see DAMN Behaviors section). As new functions are needed, additional behaviors can be added to the system without any need for modification to the previously included

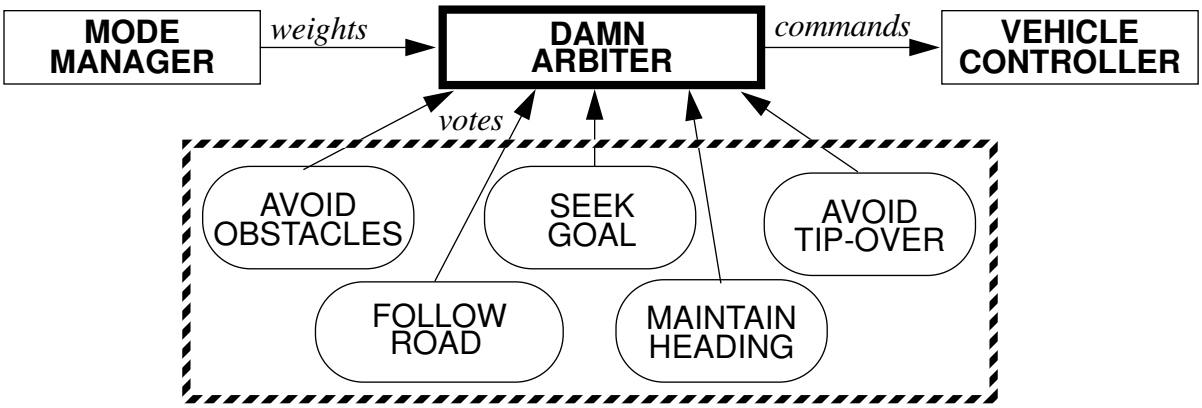

Figure 1: Behaviors sending votes to arbiter

behaviors, thus preserving their established functionality.

Since both deliberative and reflexive modules are needed, DAMN is designed so that behaviors can issue votes at any rate; for example, one behavior may operate reflexively at 10 Hz, another may maintain some local information and operate at 1 Hz, while yet another module may plan optimal paths in a global map and issue votes at a rate of 0.1 Hz. The use of distributed shared control allows multiple levels of planning to be used in decision-making without the need for an hierarchical structure. However, higher-level reasoning modules may still exert meta-level control within DAMN by modifying the voting weights assigned to behaviors and thus controlling the degree to which each behavior may influence the system's decision-making process and thus the robot's actions.

DAMN Arbiters

In a distributed architecture, it is necessary to decide which behaviors should be controlling the vehicle at any given time. In some architectures, this is achieved by having priorities assigned to each behavior; of all the behaviors issuing commands, the one with the highest priority is in control and the rest are ignored (Brooks, 1986; Rosenschein & Kaelbling, 1986). In order to allow multiple considerations to affect vehicle actions concurrently, DAMN instead uses a scheme where each behavior votes for or against each of a set of possible vehicle actions (Rosenblatt & Payton, 1989). An arbiter then performs *command fusion* to select the most appropriate action. While all votes must pass through the command arbiter before an action is taken, the function provided by the arbiter is fairly simple and does not represent the centralized bottleneck of more traditional systems.

Turn Arbiter

In the case of the turn arbiter, each behavior generates a vote between -1 and +1 for every possible steering command, with negative votes being against and positive votes for a particular command option. The votes generated by each behavior are only recommendations to the arbiter. The arbiter computes a weighted sum of the votes for each steering command, with the weights reflecting the relative priorities of the behaviors. The steering command with the highest vote is sent to the vehicle controller.

The arbiter collects the new votes from each behavior that has sent them, and performs a normalized weighted sum to find the turn command with the maximum vote value. In order to avoid problems with discretization such as biasing and "bang-bang" control, the arbiter performs sub-pixel interpolation. This is done by first convolving the votes with a Gaussian mask to smooth the values and then selecting the command option with the highest resulting value. A parabola is then fit to that value and the ones on either side, and the peak of the parabola is used as the command to be issued to the controller. This process is illustrated in Figure 2, where

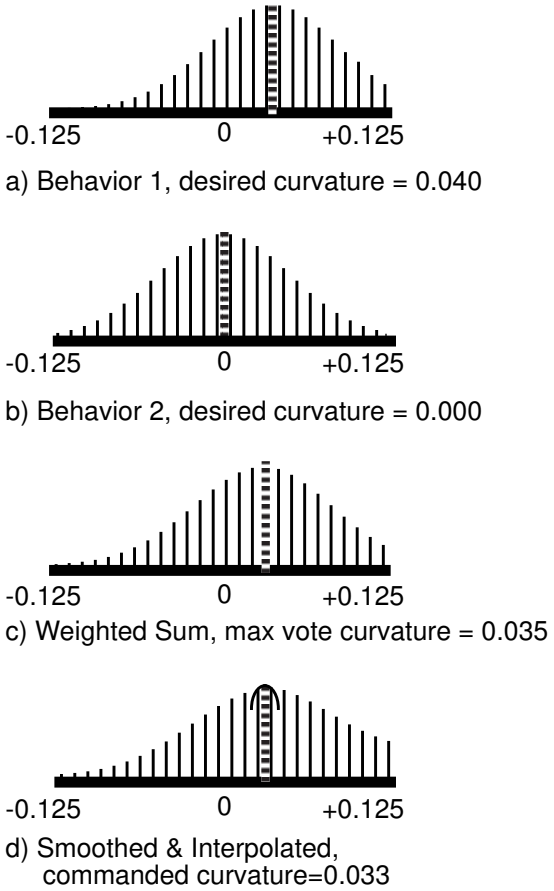

Figure 2: Command fusion process

the votes from two behaviors (a & b) are linearly combined (c), and then smoothed and interpolated to produce the resulting command (d).

Speed Arbiter

The emphasis in the research thus far has been in command fusion for the control of vehicle steering; until recently the commanded speed was decided in a very simplistic fashion based upon the commanded turn radius. The user-specified maximum vehicle speed was multiplied by the normalized weighted sum for the chosen turn radius; the result was the speed command issued.

An entirely separate speed arbiter with its own set of associated behaviors has now been developed. Thus, the turn behaviors can vote for turn commands without concern that the absolute magnitude of their votes will affect vehicle speed. At present each speed behavior votes for the largest speed possible which meets that behavior's constraints, and the arbiter simply chooses the minimum of those maxima, so that all speed constraints are satisfied.

Coordination of Arbiters

Because the choices of turn and speed commands are not completely independent and therefore must be coordi-

nated, many of the speed behaviors have as one of their inputs the output of the turn arbiter, so that the choice of an appropriate speed is influenced by the currently commanded turn radius. Other speed behaviors instead use the estimated actual turn radius of the vehicle so that they operate in a closed-loop fashion, albeit with greater delays. Likewise, some turn behaviors use the current vehicle speed in deciding upon allowable turn options.

DAMN Behaviors

Within the framework of DAMN, behaviors must be defined to provide the task-specific knowledge for controlling the vehicle. Each behavior runs completely independently and asynchronously, providing votes to the arbiter each at its own rate and according to its own time constraints. The arbiter periodically sums all the latest votes from each behavior and issues commands to the vehicle controller.

Safety Behaviors

A basic need for any mobile robot system is the ability to avoid situations hazardous to itself or to other objects in its environment. Therefore, an important part of DAMN is its "first level of competence" (Brooks, 1986), which consists of behaviors designed to provide vehicle safety. In contrast to priority-based architectures which only allow one behavior to be effective at any given moment, the structure of DAMN and its arbitration scheme allow the function of these safety behaviors to be preserved as additional levels of competence are added.

Obstacle Avoidance The most important behavior in the context of vehicle safety is the *Obstacle Avoidance* behavior. In order to decide in which directions the vehicle may safely travel, this behavior receives a list of current obstacles in vehicle-centered coordinates and evaluates each of the possible command options, as illustrated in Figure 3. The source of these obstacles may be intraversable regions of terrain determined by range image processing or stereo vision, by sonar detection of objects above the ground plane, or any other means of obstacle detection as appropriate to the current task and environment (Daily et al, 1986; Langer, Rosenblatt & Hebert, 1994).

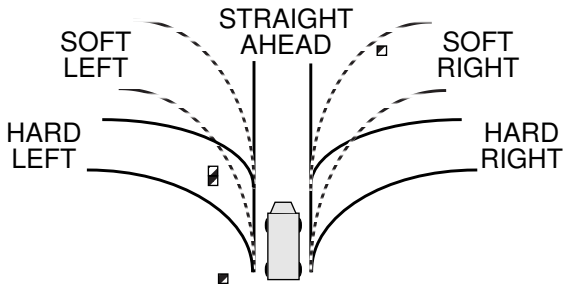

Figure 3: Arc evaluation in the Obstacle Avoidance behavior

If a trajectory is completely free of any neighboring obstacles (such as the Straight Ahead or Hard Right turns shown in Figure 3), then the obstacle avoidance behavior votes for travelling along that arc. If an obstacle lies in the path of a trajectory, the behavior votes against that arc, with the magnitude of the penalty proportional to the distance from the obstacle. Thus, the *Obstacle Avoidance* behavior votes more strongly against those turns that would result in an immediate impact (Hard Left in the figure) and votes less strongly against those turns which would only result in a collision after travelling several meters (Soft Right). In order to avoid bringing the vehicle unnecessarily close to an obstacle, the behavior also votes against those arcs that result in a near miss (Soft Left), although the evaluation is not as unfavorable as for those trajectories leading to a direct collision.

Vehicle Dynamics Another vital aspect of vehicle safety is insuring that the commanded speed and turn stay within the dynamic constraints of the vehicle as it travels over varying terrain conditions. The most important of these constraints is the one that insures that the vehicle will not tip over. Given a velocity of magnitude v, the maximum positive and negative curvatures κ to avoid tip-over would be:

$$\pm\kappa_{max} = \frac{\pm (\eta \cdot g \cdot \cos\rho) + g \cdot \sin\rho}{v^2}$$

where η is the ratio of the distance between the vehicle's center of gravity (c.g.) and the wheels to the c.g. height, g is the acceleration due to gravity, and ρ is the vehicle roll with respect to the gravity vector, as illustrated in Figure 4. Likewise, for a given vehicle turn curvature, the maximum velocity is:

$$v_{max} = \mathrm{Min}\left|\frac{\pm (\eta \cdot g \cdot \cos\rho) + g \cdot \sin\rho}{\kappa}\right|^{1/2}$$

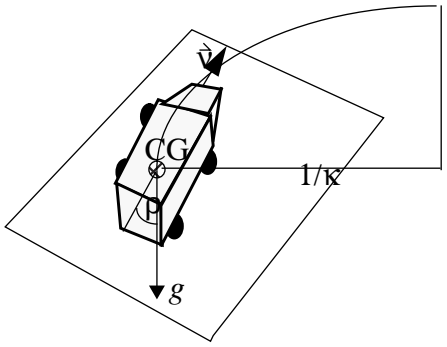

Figure 4: Vehicle dynamics

Similar constraints can be imposed on vehicle turn radius and speed in order to avoid tire slippage. The limit on curvature for slippage is:

$$\pm\kappa_{max} = \frac{(\mu \cdot g \cdot \cos\rho) \pm (g \cdot \sin\rho)}{v^2}$$

where μ is the dynamic coefficient of friction between the tire and the terrain, and for speed:

$$\pm v_{max} = \left| \frac{(\mu \cdot g \cdot \cos\rho) \pm (g \cdot \sin\rho)}{\kappa} \right|^{1/2}$$

Two behaviors, *Limit Turn* and *Limit Speed* send votes to the arbiter that implement these constraints, voting against commands that violate them.

Road Following

Once vehicle safety has been assured by the obstacle avoidance and dynamic constraint behaviors, it is desirable to add additional behaviors that provide the system with the ability to achieve the tasks for which it is intended., such as road following; one of the behaviors that have been implemented within DAMN to provide this function is ALVINN.

The ALVINN road following system is an artificial neural network that is trained, using backpropagation, to associate preprocessed low resolution input images with the appropriate output steering commands (Pomerleau, 1992). In the case of ALVINN, creating a behavior that independently evaluated each arc was relatively straightforward. The units of the neural network's output layer each represent an evaluation of a particular turn command, with the layer trained to produce Gaussian curves centered about those turns that would follow the road ahead. These units are simply resampled to the DAMN voting command space, using a Gaussian of the appropriate width. This process is illustrated in Figure 5.

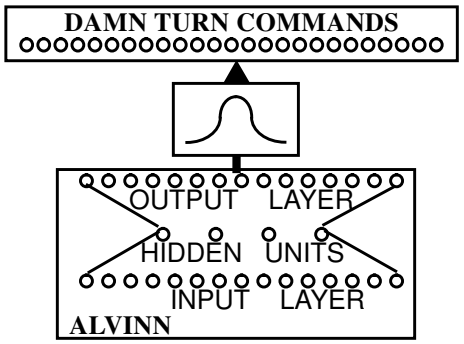

Figure 5: Resampling of ALVINN output layer

Goal-Directed Behaviors

Another important level of functionality that should be present in any general purpose robotic system is the ability to reach certain destinations using whatever global information is available. While the low-level behaviors operate at a high rate to ensure safety and to provide functions such as road following and cross-country navigation, high-level behaviors are free to process map-based or symbolic information at a slower rate, and periodically issue votes to the arbiter that guide the robot towards the current goal.

Subgoals The *Goal Seeking* behavior is one way to provide this capability. This simple behavior directs the vehicle toward a series of goal points specified in global coordinates either by the user (Langer, Rosenblatt & Hebert, 1994) or by a map-based planner (Keirsey, Payton & Rosenblatt, 1988). The desired turn radius is transformed into a series of votes by applying a Gaussian whose peak is at the desired turn radius and which tapers off as the difference between this turn radius and a prospective turn command increases. A goal is considered satisfied once the vehicle enters a circle centered at the goal location; then the next goal is pursued. Because of errors in goal placement and accumulated errors in vehicle positioning, a goal point may not be reachable. For this reason, an ellipse is defined with the current goal and the subsequent goal as foci; if the vehicle enters this ellipse, the current goal is abandoned and the next one becomes the current goal instead, thus allowing progress to continue.

Dynamic Programming Some more sophisticated map-based planning techniques have also been integrated and used within the DAMN framework. These planners use dynamic programming techniques based on the A* search algorithm (Nilsson, 1980) to determine an optimal global path. However, an important point is that they do not hand a plan down to a lower level planner for execution, but rather maintain an internal representation that allows them to participate directly in the control of the vehicle based on its current state. A* yields a set of pointers within the map grid that point toward the goal, as depicted by the small arrows in Figure 6. During execution, this grid may be indexed by the current vehicle position to yield a path towards the goal which is optimal based on the information available in the map at that time.

The Internalized Plans (Payton, 1990) approach uses a detailed map to perform an A* search from the goal(s) back toward the start point to create a "Gradient Field" towards the goal. The type and slope of the terrain, among other factors, is used to estimate the cost of traversal between grid cells. During run-time, the grid cell containing the current vehicle location is identified, and the Gradient Field pointers are followed forward to the point **G'** in Figure 6; the desired heading to reach the goal is that from the current location **S** to **G'**, and a series of votes with its peak at that value is sent to the turn arbiter.

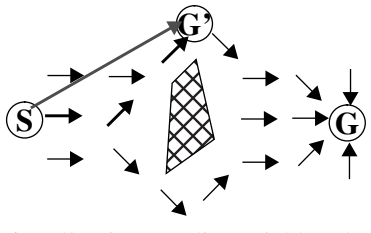

Figure 6: Following Gradient Field to determine intermediate goal heading

The D* planner (Stentz, 1993) also creates a grid with "backpointers" that represent information on how best to

reach the goal from any location in the map. The map may initially contain no information, but is created incrementally as new information becomes available during the execution of a mission, and the arc traversal costs and backpointers are updated to reflect this new knowledge. The resulting global plan is integrated into DAMN as a behavior by determining, for each possible turn command, the weight w of reaching the goal from a point along that arc a fixed distance ahead (the squares designated collectively as **S'** in Figure 7). If w_{max} and w_{min} are the maximum and minimum values of w, then the vote for each turn command is determined as: $(w_{max} - w) / (w_{max} - w_{min})$. In the case that a point S' is not represented on the grid, or if the goal cannot be reached from it, then the vote for that arc is set to -1.

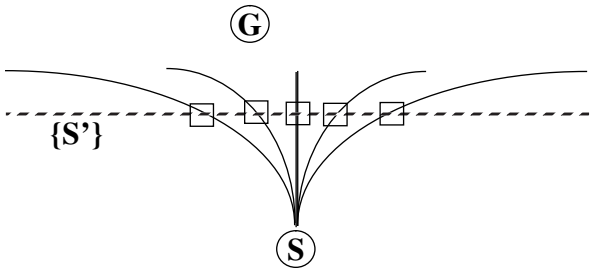

Figure 7: Using D* to evaluate distance to goal for each arc

Teleoperation

Teleoperation is another possible mode in which a robotic system may need to operate. The STRIPE teleoperation system (Kay & Thorpe, 1993) provides a graphical user interface allowing a human operator to designate waypoints for the vehicle by selecting points on a video image and projecting them on to the surface on which the vehicle is travelling. STRIPE then fits a spline to these points and uses pure pursuit to track the path. When used in isolation, it simply sends a steering command to the controller; when used as a DAMN behavior, it sends a series of votes representing a Gaussian centered on the desired command. This allows the dynamic constraints and obstacle avoidance behaviors to be used in conjunction with STRIPE so that the safety of the vehicle is still assured.

Auxiliary Behaviors

Various other auxiliary behaviors that do not achieve a particular task but issue votes for secondary considerations may also be run. These include the *Drive Straight* behavior, which simply favors going in whatever direction the vehicle is already heading at any given instant, in order to avoid sudden and unnecessary turns; and the *Maintain Turn* behavior, which votes against turning in directions opposite to the currently commanded turn, and which helps to avoid unnecessary oscillations in steering, the *Follow Heading* behavior which tries to keep the vehicle pointed in a constant direction, as well as various behaviors which allow user input to affect the choice of vehicle turn and speed commands.

Combining Behavior Votes

The voting strengths, or weights, of each behavior are specified by the user, and are then normalized by the arbiter so that their sum equals 1. Because only the relative values are important, and because the magnitude of each behavior‘s votes vary according to their importance, DAMN is fairly insensitive to the values of these weights and the system performs well without a need to tweak these parameters. For example, the *Obstacle Avoidance* behavior has been run in conjunction with the *Seek Goal* behaviors with relative weights of 0.75 and 0.25, respectively, and with weights of 0.9 and 0.1, and in both cases has successfully reached goals while avoiding obstacles.
The vote weights of each behavior can also be modified by messages sent to the arbiter from a mode manager module. It can reconfigure the weights according to whatever top-down planning considerations it may have, and potentially could use bottom-up information about the effectiveness and relevance of a behavior (Payton et al, 1993). Different modes of operation that exclude some behaviors can be constructed by setting the weights those behaviors to 0. A Mode Manager was developed at the Hughes Research Labs to be used with DAMN for this purpose, and at CMU Annotated Maps were integrated with DAMN to provide this capability (Thorpe et al, 1991).

As a simple example to illustrate the manner in which votes are issued and arbitrated within DAMN, consider the case in Figure 8 where two behaviors are active, one responsible for obstacle avoidance and the other for goal seeking (only five turn options are shown for simplicity). The magnitude of a vote is indicated by the size of a circle, with a large unfilled circle representing a vote of +1, a large striped circle a value of -1, and a small circle a value near 0. Thus, the goal-seeking behavior is voting most strongly in favor proceeding straight and less favorably for a soft left turn, and voting against hard left or any right turns; the obstacle avoidance behavior is voting against a hard left or soft right, and allowing the other turns as acceptable, with soft left being the most favorable.

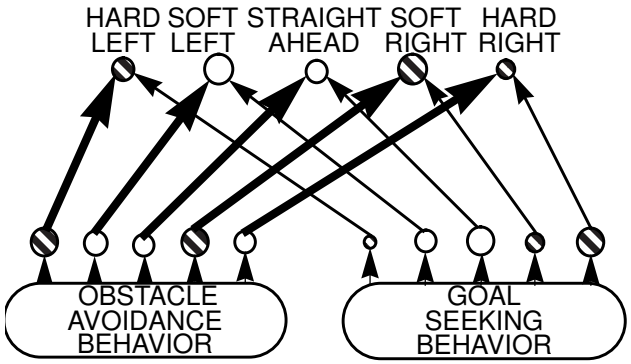

Figure 8: Command fusion in DAMN

Because avoiding obstacles is more important than taking the shortest path to the goal, the obstacle avoidance behav-