CHAPTER 6: CONCLUSION

6.1 Architectures for Mobile Robot Control

Mobile robots provide a useful and challenging domain in which to develop the thesis put forward here. Effective control of mobile robots and their associated sensors demands the synthesis and satisfaction of several complex constraints and objectives in real-time, particularly in unstructured, unknown, or dynamic environments such as those typically encountered by outdoor mobile robots. An architecture must connect the perception, planning, and control modules and provide a structure with which the system may be developed, tested, debugged, integrated and understood.

In addition to serving as an interesting case study for the development of control architectures, mobile robots have many uses, especially for gaining access to areas that are unreachable by or dangerous to humans such as cleanup of hazardous waste sites, inspection of nuclear power stations, military reconnaissance missions, subsea exploration, and planetary exploration. Other uses include the automation of routine tasks such as mail delivery and the inspection and waterproofing of heat tiles underneath the space shuttles. Recently, a new emphasis has been the incorporation of robotic technology in passenger vehicles in order to improve both the safety and efficiency of highway driving.

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. One of the key characteristics of most mobile robot domains is that uncertainty plays a large role, and that is the class of domains studied in this work. Prior knowledge of the environment may be incomplete or not exist at all, and the environment may be dynamic, so that reasoning must be non-monotonic in nature and occur rapidly enough to be able to respond to unexpected events. Posterior knowledge gained via sensing is incomplete, inaccurate, and uncertain, as is knowledge of the effects of actions decided upon and taken by the system. In addition, the dynamics of the vehicle itself often play an important role in determining which actions may be achieved and which actions are to be avoided.

Another important aspect of mobile robot systems is the need to combine information from several different sources such as video cameras, laser range finders, sonars, and inertial navigation systems; they must also be capable of combining objectives for a system that is to perform diverse tasks such as following roads, driving off-road, following designated paths, avoiding obstacles, and reaching goal destinations, as well as allowing for teleoperation. These sensors and the procedures that process their data operate at different rates, so they must be allowed to operate asynchronously to maximize the throughput and thus the responsiveness of the system. To function effectively, an architectural framework for these sensing and reasoning processes must be imposed to provide a structure for combining information from several different sources. The architecture should serve as an aid, not a burden, in the integration of modules that have been developed independently, so it must not be overly restrictive. It must allow for purposeful goal-oriented behavior yet retain the ability to respond to potentially dangerous situations in real-time while maintaining enough speed to be useful. While many general purpose architectures are computationally equivalent or nearly so, the ease with which various classes of systems may be instantiated within them varies greatly. A crucial yet often neglected consideration in the design of a mobile robot architecture is the ease with which a system may be developed, tested, debugged, and understood, as well as adapted to novel domains and applications.

When dealing with a physical system such as a mobile robot, it is also important to consider aspects of control such as stability and the limitations and constraints of the physical plant, such as the non-holonomic constraints of a wheeled vehicle, finite actuator capabilities, and the delays inherent in any system that arise from latencies in data acquisition, data processing, intermodule communications, and actuator response; together with the continuous motion of the vehicle, this implies that by the time the command is being executed the vehicle is no longer in the current state but actually in a future state, i.e. different position, heading velocity, turn rate, etc. An asynchronous distributed system presents an additional challenge in that, in general, the size of these latencies will be different for each module due to varying processing needs and sensor frame rates. If the various latencies of the system are not accounted for, the vehicle control will be unstable.

When designing a software architecture for the control of complex real-time systems such as mobile robots, there are many important issues that must be addressed. The architecture must provide the means by which the system may accomplish its multiple objectives efficiently; it must be able to satisfy real-time constraints, promote fault tolerance, and provide for the safety of the vehicle and of its surroundings. Some key issues to be considered in the design of a mobile robot control architecture are whether the architecture should be centralized or distributed, whether the reasoning should be reactive or deliberative, whether input combination should occur via sensor fusion or command arbitration, and whether control should be top-down or bottom-up. To be effective in dynamic, uncertain, and complex domains, DAMN combines both deliberative and reactive elements within a bottom-up structure, and centralized arbiters combine commands received from distributed, asynchronous behaviors.

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 mission failure. Because reactivity is essential for any real-time system, we must eschew the sensing and planning bottlenecks of centralized systems, but if we are to avoid sensor fusion, the system must combine command inputs to determine an appropriate course of action. Having decided upon a behavior-based architecture, it is then necessary to specify how information in a distributed system is to be combined and used to generate action in an intelligent, coherent manner. By appropriately combining behavior commands through arbitration, a robot control system can respond to its environment without suffering the problems inherent in sensor fusion. While sensor fusion creates a bottleneck, command arbitration runs the risk of losing information valuable to the decision-making process; therefore a careful balance must be struck between completeness and optimality on the one hand versus modularity and efficiency on the other. Systems that combine command inputs to determine an appropriate course of action fall into two broad categories, priority-based arbitration and command fusion.

In architectures which employ priority-based arbitration such as the Subsumption Architecture, action selection is achieved by assigning priorities to each behavior; of all the behaviors issuing commands, the one with the highest priority is in control and the rest are ignored. This allows for quick responses to new situations and stimuli, although, by definition, prioritization only allows one module to affect control at any given time. While this is an effective scheme for choosing among incompatible commands, it does not provide an adequate means for dealing with multiple goals that can and should be satisfied simultaneously. A compromise between behaviors cannot be achieved in such an all-or-nothing scenario; whenever one behavior's output is overridden by another, the information and knowledge represented by that behavior is completely lost to the system.

Architectures that perform *command fusion* combine the commands from individual behaviors so that decisions may be made based on multiple considerations while preserving the modularity and reactivity of distributed systems. Command fusion provides a mechanism for the concurrent satisfaction of multiple goals, and allows modules to be completely independent, thus allowing incremental, evolutionary system development. In systems that perform command fusion, the decision-making process is based on combining command inputs from multiple behaviors, and is therefore able to simultaneously satisfy multiple goals. Such systems are largely distributed, but they must also contain a central module that receives commands from the various behaviors and combines them in some manner. Motor schemas provide a general framework for command fusion, but because they are implemented as potential fields they are subject to the problems of command averaging and local minima, among others.

6.2 Contributions

6.2.1 Novel Paradigm for Command and Utility Fusion

The Distributed Architecture for Mobile Navigation (DAMN) is a planning and control architecture in which a collection of independently operating behaviors collectively determine a robot's actions. The organization of the DAMN architecture is shown in Figure 6-1; it consists of a group of distributed behaviors communicating with a centralized command arbiter, sending votes in favor of actions that satisfy its objectives and against those actions which do not. The arbiter is then responsible for combining the behaviors' votes and generating actions which reflects their objectives and priorities. Within the framework of DAMN, behaviors must be defined to provide the task-specific knowledge for controlling the vehicle. Each behavior in the system is responsible for a particular aspect of vehicle control or for achieving some particular task, receiving as input only that sensory information pertinent to the task, and processing that input to generate as output votes for and against possible actions. Each behavior operates asynchronously and in parallel with other behaviors, sending its outputs to the arbiter at whatever rate is appropriate for that particular function. A behavior is assigned a weight reflecting its relative priority in controlling the vehicle; however, this weighting is not a strict prioritization, thus votes from all behaviors are used in determining what the next action should be. 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. These behavior votes are then periodically combined by the arbiter and the resulting command is sent to the controller.

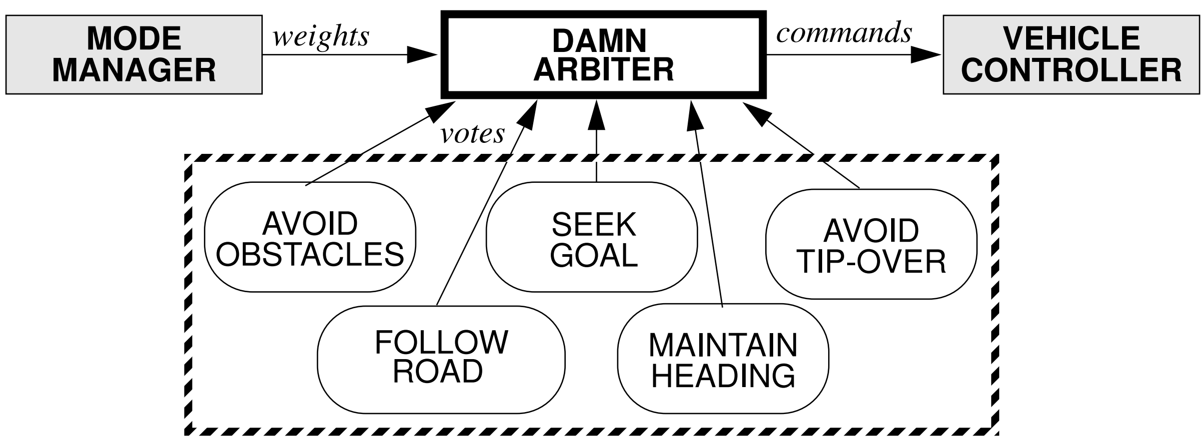

Figure 6-1 : Centralized arbitration of votes from distributed behaviors in DAMN.

The DAMN architecture reflects the position that some centralization is needed to provide coherent rational reasoning, but an appropriate level must be chosen so that it does not create a bottleneck. The capabilities of an agent should be divided up as finely as is practical among task-achieving behaviors, and the interfaces must be defined so as to avoid being overly restrictive. Thus, as shown in Figure 6-1, independent behaviors operate in a distributed fashion to generate votes for actions based on domain-specific knowledge, while a central arbiter combines their results to generate reasonable behavior that satisfies as many objectives as possible. Thus, DAMN performs centralized arbitration of votes from distributed, independent, asynchronous decision-making processes and in so doing provides coherent, rational, goal-directed behavior while preserving real-time responsiveness to its immediate physical environment.

Likewise, 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 mission failure. Unlike other behavior-based architectures, DAMN is designed so that behaviors provide both deliberative and reflexive capabilities; the use of distributed shared control allows multiple levels of planning to be used in decision-making without the need for an hierarchical structure. Input from all behaviors are used in determining what the next action should be, so that there are multiple influences on the decision-making process, and compromises are made to satisfy as many objectives as possible. 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. The distinction made in DAMN is not in the level of abstraction of a given module, but rather whether its domain is represented and acted upon in a discrete or continuous manner. All continuous servo-like activity is instantiated as a voting behavior without regard for the time or space scale in which it operates; votes from behaviors operating at all levels of reasoning cooperate in determining what the next action should be. Sequential activity is controlled by a mode manager which exerts 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.

In addition, the behaviors are completely modular and independent, so that new capabilities may be added to an existent system without a need to disrupt or modify previously established functionality; in particular, they operate asynchronously, each at their rate without the need to synchronize with a central clock, so that throughput and reactivity is maximized. Thus DAMN provides a framework for facilitates the evolutionary development and integration of independent decision-making modules to create robust systems of incrementally greater capabilities.

Command Fusion

DAMN was devised to provide a general framework for command fusion that overcomes the limitations of priority-based arbitration such as that used in the Subsumption Architecture, as well as other command fusion systems such as Motor Schemas.

DAMN command arbiters combine votes for and against each of a set of candidate commands, either directly in actuator space or indirectly in effect space. This method has the benefit of being very simple and straightforward, so that many different types of modules may easily be incorporated within the architecture. Voting takes places in a commonly defined action or state space, thus providing a uniform interface to the arbiter that is independent of the task the behavior is concerned with, the level of planning at which it operates, or the algorithms and paradigms which it uses for its decision-making process. DAMN is designed so that various behaviors can be easily added or removed from the system, depending on the current task at hand. The distributed, asynchronous nature of the architecture allows multiple goals and constraints to be fulfilled simultaneously, thus providing goal-oriented behavior without sacrificing real-time responsiveness. Unlike other behavior-based architectures, DAMN is designed so that behaviors provide both deliberative and reflexive capabilities; the use of distributed shared control allows multiple levels of planning to be used in decision-making without the need for an hierarchical structure.

Within DAMN, independent behaviors operate in a distributed fashion to generate votes for actions based on domain-specific knowledge, while a central arbiter combines their results to generate reasonable behavior which obeys all constraints and simultaneously satisfies as many objectives as possible by choosing that action which maximizes a function of the behavior votes. In order to preserve the respective advantages of centralized and distributed architectures, sufficient information must be communicated from the behaviors to allow the arbiter to make intelligent decisions, but the arbiter must not be so complex as to become a bottleneck for the system. Various points along this trade-off spectrum have been explored within DAMN, using different types of arbiters and vote structures. There are currently four different arbitration schemes in effect, illustrated in Figure 6-2.

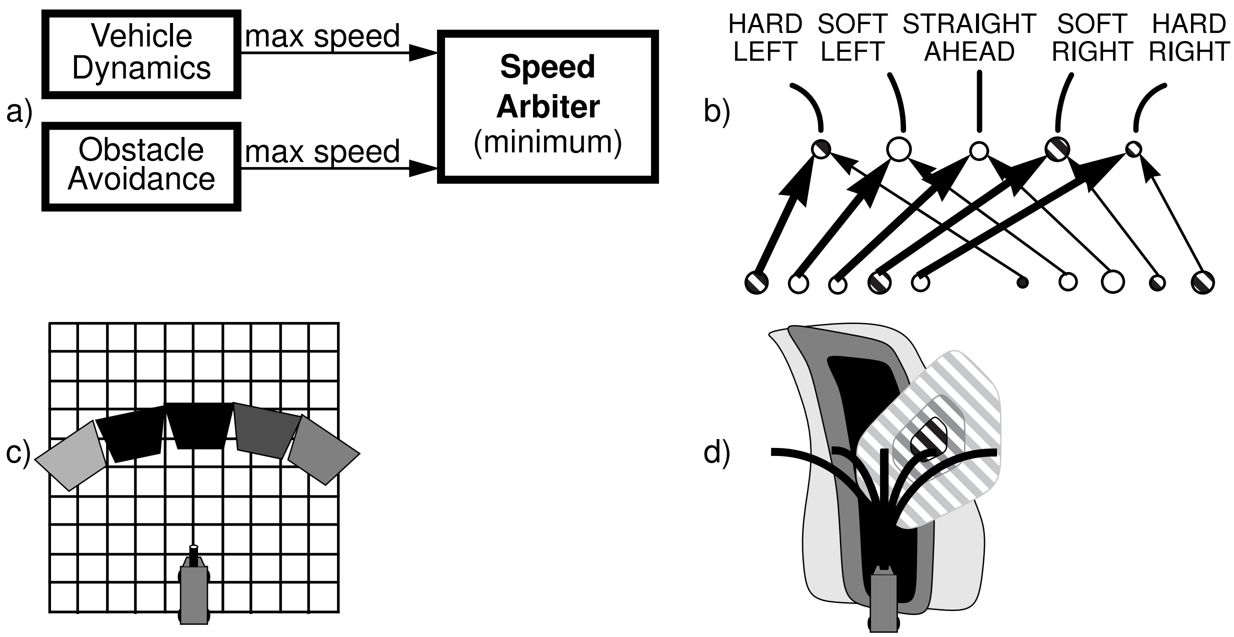

Figure 6-2 : DAMN arbitration schemes.
a) constraint, b) actuation, c) effect, and d) utility.

The simplest DAMN arbitration scheme is constraint arbitration, in which each behavior determines the highest value of the command being determined that satisfies the constraint which it seeks to enforce; the behaviors send that maximum value to the arbiter, which then takes the minimum of these values, thereby simultaneously satisfying all constraints. For example, speed behaviors determine their maximum speed vote based on constraints such as avoiding vehicle tip-over and avoiding obstacles, and the arbiter goes at the speed of the slowest vote, as shown in Figure 6-2a.

In the actuation arbitration scheme, each behavior votes for or against various alternatives in the actuator controller's command space; for example, the turn arbiter receives votes for a fixed set of vehicle turn radii, as shown in Figure 6-2b. The arbiter then sums these votes and selects that action which has the highest total score, and send the command to the controller.

In the effect arbitration scheme, each behavior votes for or against various alternatives in an abstract command space, i.e., they vote for the desired effect of the mechanism being controlled rather than the direct control of the mechanism's actuators. For example, a field of regard arbiter and its associated behaviors have been implemented and used for the control of a pair of stereo cameras on a pan/tilt platform, as shown in Figure 6-2c.

Utility Fusion

A new means of action selection via utility fusion is introduced as a solution to the shortcomings of behavior-based systems. Instead of voting for actions, behaviors indicate the utility of various possible world states, and it is the responsibility of the arbiter to determine which states are actually attainable and how to go about achieving them. This type of arbiter is no longer performing command fusion, nor is it performing sensor fusion; it is combining utilities to perform *utility fusion*.

This new approach strikes a balance between the extremes of action selection and sensor fusion and has been found to yield many benefits. The arbiter determines the next action based on the maximization of expected utility, thus providing a unified conceptual framework for defining the semantics of votes and for dealing with uncertainty. By explicitly representing and reasoning about uncertainty within the decision-making processes, a system can be created whose effects are well-defined and well-behaved. For example, a map-based path arbiter has been implemented as a means of voting for and producing steering control. Behaviors communicating with the path arbiter vote on the desirability of various possible vehicle locations, and the arbiter maintains a local map of these votes, as indicated in Figure 6-2d. The map-based utility space is not time-dependent, so that an arbiter using such a representation is capable of effectively synchronizing and maintaining a consistent interpretation of the votes received from asynchronous behaviors, thus providing coherent reasoning in a distributed system. The path arbiter evaluates the candidate trajectories within the utility map, and selects that one for which the total utility is the greatest. Behaviors can function without knowledge of the system dynamics, thus increasing their reusability for other systems.

The utility arbiter can use models of the system being controlled to determine which states are actually attainable, and to increase the accuracy and stability of control. In particular, the map-based utility arbiter gathers information from behaviors about the desirability of possible vehicle locations and then evaluates candidate trajectories to determine appropriate actions. The arbiter can then use kinematic models of the robot to determine which actions can be commanded without violating non-holonomic constraints, and use and dynamic models of the system to provide greater stability. Behaviors can function without knowledge of the system dynamics, thus increasing their reusability for other systems and types of vehicles.

The utility arbitration system does not perform sensor fusion, but instead the behaviors process data according to their tasks in a distributed fashion. However, the arbiter is more complex than those performing command fusion and resulting the system is thus more centralized than command fusion systems. As a result, systems implemented with utility fusion are less reactive but are better able to anticipate circumstances and to generate coherent action. In DAMN, the processing of sensory information is still distributed among the behaviors, and the utility fusion performed by the arbiter is much simpler and represents much less of a bottleneck than does sensor fusion. By centralizing the action evaluation process, the type of DAMN arbiter is able to address issues discussed such as synchronization of commands and stability of control. Utility theory also provides a semantic framework to define the meanings of votes, and to separate out the effects of uncertainty and reason about them explicitly.

Thus, through the appropriate use of command or utility fusion, DAMN performs centralized arbitration of votes from distributed, independent, asynchronous decision-making processes and in so doing provides coherent, rational, goal-directed behavior while preserving real-time responsiveness to its immediate physical environment. DAMN also provides a framework for developing and integrating independent decision-making modules communicating with such arbiters, thus facilitating their development and leading to evolutionary creation of robust systems of incrementally greater capabilities.

6.2.2 Integration of Heterogenous Modules

My work in building and testing DAMN has created a flexible and useful robot architecture that contributed significantly to the successes of the CMU Navlab project, the DARPA Autonomous Land Vehicle program, and the ARPA/OSD Demo II program. Vehicles under the control of DAMN have driven at highway speeds, navigated across stretches of off-road terrain some kilometers in length, cooperated with other robotic vehicles, and performed teleoperation, all while providing for the safety of the vehicle and meeting mission objectives; the types of data used in these systems include maps, color cameras, stereo cameras, sonars, ladars, and motion and position sensors.

DAMN has been used to combine various systems of differing capabilities on several mobile robots, at various sites; in addition to its use on the CMU Navlab vehicles, DAMN is also being used at the Lockheed Martin Corporation, the Hughes Research Labs, and the Georgia Institute of Technology. DAMN arbiters have been used to integrate navigation modules for the steering and speed control of single as well as multiple vehicles at these sites, and have also been used to select field of regard for the control of a pair of stereo cameras on a pan/tilt platform.

By providing a behavior general interface at the geometric level; the DAMN architecture has been able to integrate a wide variety of existing modules with minimal modification, regardless of those modules' internal representations, behavioral semantics, or the level of planning at which they operate. Thus, it provides new functionality to a system by synergistically combining existing capabilities. It does so without forcing the modules to be developed or reformulated to fit within a preconceived notion of how they should function.

6.3 Future Work

6.3.1 Definition of Uncertainties

In addition, the utility arbiter does not deal with uncertainty in an entirely satisfactory manner. As a behavior informs the arbiter of new utilities, they either replace old ones from that behavior or are included in the map together without any sort of combination taking place. When state space trajectories are evaluated, their expected utilities are summed. While this is correct for independent utilities corresponding to distinct objects, uncertainties for the same object should be combined via Bayes' theorem. However, this implies that the arbiter must know whether or not observations are independent, both from the same behavior and from different behaviors, and the complexity of the system would cause a significant bottleneck comparable to that created by sensor fusion methods.

Utility theory does provide a framework in which the uncertainties due to sensing, position and control can be expressed explicitly and reasoned about in a rational manner. However, the current system as implemented does not make full use of this capability. The uncertainty in sensing is specified as an arbitrary constant for a given behavior, rather than using measurements of the uncertainty in the sensor, which may also vary within an image, and of the sensor processing algorithm, which may vary depending upon the characteristics of the image.

Uncertainty in position and control should be used in the decision-making process as well. Currently trajectories are evaluated from a single predicted vehicle position rather than a distribution of possible locations due to uncertainty in position and control, as well as in the prediction process. Expected utility is defined by summing, for each consequence, the product of the utility of that consequence and the probability that it will occur. Thus, uncertainties can be convolved to calculate the probability of the various consequences which have been assigned utilities, and an integral performed to measure the true expected utility of an action.

6.3.2 Definition of Utilities

While it was hoped that utility theory would provide an objective means of combining information from disparate sources, utilities are still defined in a subjective manner, according to intuition and experience, just as weights are defined in a command fusion system. This appears to be a fundamental problem of multiple-objective systems, whether distributed or centralized, which is not completely solved by utility theory. However, user protocol studies could be conducted so that the utilities, while still subjective, may be defined within a rational framework in accordance with a model of the human decision-making process. Machine learning techniques such as reinforcement learning or genetic algorithms could also be employed in order to learn utilities and weights through experience, and to adapt parameters to the current situation.

In addition, the utilities as they are currently defined are valid from the particular pose from which the behavior performed the evaluation. For example, the utility expressed for a particular location specified in x-y space may be based on the assumption that the vehicle would go directly to that location from its current position; it is not necessarily the case that the same utility would be assigned if the vehicle were to approach it later from a different point with a different orientation. Other dimensions such as orientation could also be added to the representation, yielding a planner capable of planning in configuration space.

6.3.3 Time-Dependent Planning

Time is another dimension not explicitly represented by the map-based utility arbiter. Although the arbiter is able to explicitly reason about the movement of the vehicle, the utilities in its map are static so that it is necessary for behaviors to quickly update the location of a utility, even when the object to which it corresponds is moving with a known trajectory. In order to perform a limited form of time-dependent planning, the representation for utilities could be extended to include velocities when known, and the arbiter would then update the position of these utilities as time progresses, as well as in anticipation of system latencies as was done for predictive control. When evaluating a trajectory, the arbiter would also have to account for the time taken for the vehicle to move along the path, updating the predicted position of utilities as the position being evaluated progresses.

6.3.4 Coordination of Multiple Degrees of Freedom

There is usually a one-to-one mapping between an arbiter and a control degree of freedom such as steering or speed, although they are in fact interdependent and should be jointly controlled, as was the case with pan/tilt camera control. This is a challenge for higher degree of freedom systems such as robot manipulators or multiple coordinating robots. The coordination of multiple degrees of freedom may be accomplished by creating a multi-dimensional command space and requiring that behaviors vote for all combinations of their cross-product, as shown in Figure 6-3. However, the combinatorics require prohibitively expensive computations on the part of all behaviors, as well as the arbiter; this is exacerbated as the dimensionality or the resolution within any single dimension increases.

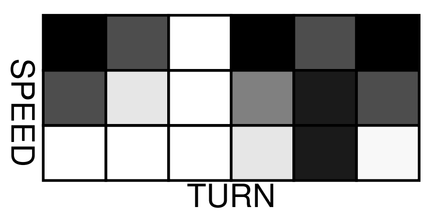

Figure 6-3 : Two-dimensional coordinated speed and turn arbitration.

The map-based utility arbitration scheme provides yet another possible means of coordinating multiple degrees of freedom. The trajectories evaluated depend upon the current vehicle speed. If instead the arbiter were to evaluate trajectories at several different speeds, it could then choose that turn and speed combination which maximizes expected utility overall. While this induces some combinatorics within the arbiter, it can be partially compensated for by precomputing the trajectories at the pre-determined speeds. One significant advantage of this scheme, however, is that behaviors incur no greater structural or computational overhead; as before, they simply express the utility of various external world states, and the arbiter exercises more control over which states may be achieved and how. This means of jointly determining turn and speed commands has not yet been implemented at this time.

It is also possible to place arbiters in nested control loops, with a map-based utility arbiter providing inputs to an effect arbiter which in turn feeds an actuation arbiter whose output is used by a constraint arbiter. For example, the path arbiter and turn arbiter could be nested as shown schematically in Figure 6-4. The path arbiter performs evidence fusion and evaluates candidate actions, sending the results as votes to the turn arbiter which performs command fusion and selects an action to be executed, and sends that turn command to the path arbiter as feedback.

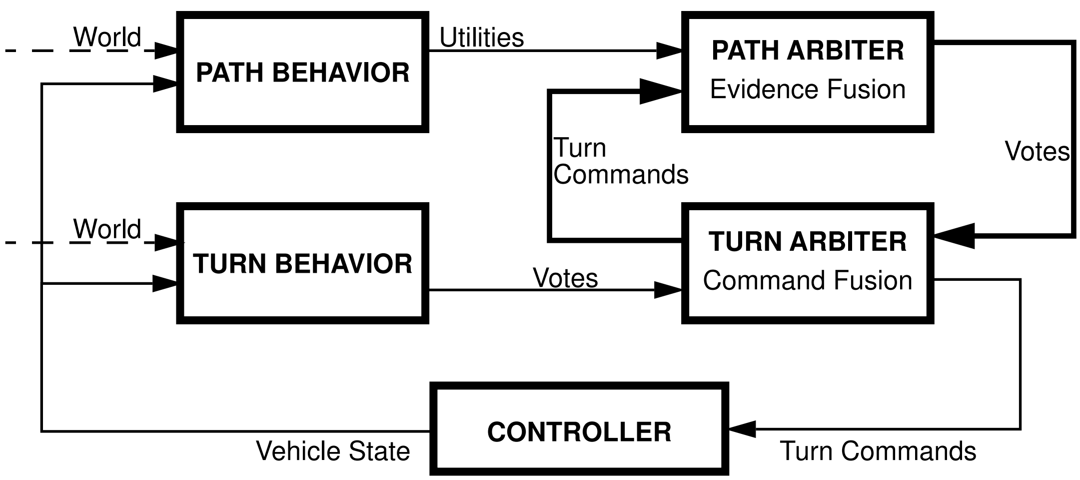

Figure 6-4 : Multiple arbiter nested control loop.
Path arbiter sends evaluations as votes to turn arbiter and receives turn commands as feedback.

6.4 Conclusion

Because reactivity is essential for any real-time system, we must eschew the sensing and planning bottlenecks of centralized systems, but if we are to avoid sensor fusion, the system must combine command inputs to determine an appropriate course of action. However, priority-based arbitration only allows one module to affect control at any given time. Command fusion provides a mechanism for the concurrent satisfaction of multiple goals, and allows modules to be completely independent, thus allowing incremental, evolutionary system development.

The Distributed Architecture for Mobile Navigation is a planning and control architecture in which a collection of independently operating behaviors collectively determine a robot's actions.

A command arbiter combines the behavior outputs and selects that action which best satisfies the prioritized goals of the system. The distributed, asynchronous nature of the architecture allows multiple goals and constraints to be fulfilled simultaneously, thus providing goal-oriented behavior without sacrificing real-time responsiveness. Unlike other behavior-based architectures, DAMN is designed so that behaviors provide both deliberative and reflexive capabilities; the use of distributed shared control allows multiple levels of planning to be used in decision-making without the need for an hierarchical structure.

Thus, DAMN performs centralized arbitration of votes from distributed, independent, asynchronous decision-making processes and in so doing provides coherent, rational, goal-directed behavior while preserving real-time responsiveness to its immediate physical environment. DAMN also provides a framework for developing and integrating independent decision-making modules communicating with such arbiters, thus facilitating their development and leading to evolutionary creation of robust systems of incrementally greater capabilities.

DAMN has been used to combine various systems of differing capabilities on several mobile robots, and has also been used for active sensor control. Various subsystems developed at CMU and elsewhere have been integrated within this architecture, creating systems that perform road following, cross-country navigation, map-based route following, and teleoperation while avoiding obstacles and meeting mission objectives.