Autonomous Robots vol. 9, no.1, pp. 17-25, 2000

Optimal Selection of Uncertain Actions by Maximizing Expected Utility

JULIO K. ROSENBLATT
Australian Centre for Field Robotics
University of Sydney, NSW 2006, Australia

julio@acfr.usyd.edu.au
http://www.acfr.usyd.edu.au

Abstract. A new means of action selection via *utility fusion* is introduced as an alternative to both sensor fusion and command fusion. Distributed asynchronous behaviors indicate the utility of various possible states and their associated uncertainty. A centralized arbiter then combines these utilities and probabilities to determine the optimal action based on the maximization of expected utility. The construction of a utility map allows the system being controlled to be modeled and compensated for; experimental results verify that this approach improves performance.

Keywords: behavior-based architectures, planning, decision theory

1. Introduction

In unstructured, unknown, and dynamic environments, such as those encountered by outdoor mobile robots, an intelligent agent must adequately address the issues of incomplete and inaccurate knowledge; it must be able to handle uncertainty in both its sensed and *a priori* information, in the current state of the agent itself, as well as in the effects of the agent's actions. In order to function effectively in such conditions, an agent must be responsive to its environment, proceeding in a data-driven manner, as well as goal-oriented, taking into account the higher level goals of the system. 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.

In order to achieve this desired symbiosis of deliberative and reactive elements, the Distributed Architecture for Mobile Navigation (DAMN) consists of a group of distributed behaviors communicating with a centralized command arbiter, as shown in Fig. 1. The arbiter is responsible for combining the behaviors' votes to generate actions which are then sent to the vehicle controller. A mode manager may also be used to vary these weights during the course of a mission based on knowledge of which behaviors are most relevant and reliable in a given situation. The distributed, asynchronous behaviors provide real-time responsiveness to the environment, while the centralized command arbitration provides a framework capable of producing coherent behavior (Rosenblatt, 1997).

Within the framework of DAMN, *utility fusion*, is presented here as a novel means of action selection for behavior-based systems. In this paradigm, behaviors do not issue commands or express preferences for actions but instead determine the utility of possible world states and their associated probabilities. These are collected by the arbiter and candidate actions are evaluated based on their expected utility. This approach deals explicitly with uncertainty, thus providing better defined vote semantics, and allows the central arbiter to use system models to improve performance.

2. Background

A mobile robot systems that perform sensor fusion gathers all available sensory data to create a complete model of its environment, plans a series of actions within the context of that model, and then executes that plan (Nilsson, 1984; Shafer et al., 1986). This approach has the advantage of being able to combine evidence to

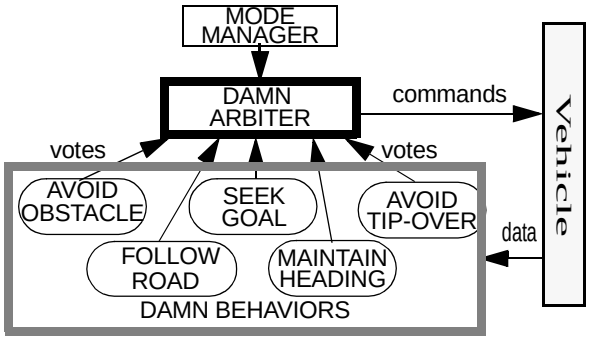

Figure 1. DAMN framework consists of centralized arbitration of votes from distributed behaviors.

overcome ambiguities and noise inherent in the sensing process (Durrant-Whyte, 1986), but has the disadvantage of creating a bottleneck. A monolithic world model is also more difficult to develop, maintain, and extend. A centralized architecture is also more likely to fail entirely if any single part of it is not functioning properly.

In architectures which employ priority-based arbitration such as the Subsumption Architecture (Brooks, 1986) and GAPPS (Rosenschein and Kaelbling, 1986), action selection is achieved by assigning priorities to each behavior; the behavior with the highest priority is in control and the rest are ignored. Thus, there is no means for dealing with multiple goals simultaneously.

Architectures that perform command fusion combine commands from various behaviors so that decisions are made based on multiple considerations, thus avoiding many of the pitfalls of sensor fusion and priority-based systems. Motor Schemas (Arkin, 1989) is a general framework for command fusion that uses potential fields (Khatib, 1990). However, it has been shown that, using potential fields, a robot cannot pass through closely spaced obstacles such as those surrounding a doorway, and there exist conditions under which potential fields suffer from oscillations and instability (Borenstein and Koren, 1991). In addition, vector addition essentially results in an averaged command that may not be satisfactory to any of the contributing schemas. The root of these limitations is that, like priority-based arbiters, each behavior simply outputs a single command, which is insufficient for effective command fusion.

In an arbitration scheme previously used in DAMN, each behavior voted for or against each of a set of candidate actions. For example, a turn arbiter received votes for a fixed set of arcs representing possible steering commands. The arbiter then performed command fusion by calculating a weighted sum of these votes and selecting the action with the highest total vote (Rosenblatt, 1997).

Many command fusion systems for mobile robot navigation have also been implemented using fuzzy logic (see Saffiotti, 1997). Uncertainty is dealt with implicitly by the fuzzification of crisp variables. This is similar to the DAMN command fusion arbiter described above, which in fact has been recast into a fuzzy logic framework (Yen and Pfluger, 1992).

These command fusion schemes provide mechanisms for the concurrent satisfaction of multiple goals as determined by independent behaviors, thus allowing incremental, evolutionary system development. However, command fusion in general still has shortcomings which reduce the system's overall effectiveness.

2.1. Limitations of Command Fusion

System Model. Command fusion systems generally do not take into account dynamic and kinematic constraints of the vehicle, thus producing commands that are feasible. For example, a given turn command may require a change in steering which exceeds the actuator's torque capabilities.

Another important consideration that is not usually dealt with is system latencies arising from delays in data acquisition and processing, communications, and actuator response. It is well known that the system must anticipate these latencies using a predictive model in order to achieve stable control (Kelly and Stentz, 1998).

Vote Synchronization and Semantics. Behavior synchronization allows reasoning to be coordinated and therefore coherent, but reduces the throughput of the system as modules must wait for a signal in order to remain synchronized. Allowing the modules in a distributed architecture to operate asynchronously, each at the greatest rate of which they are capable, maximizes throughput and therefore reactivity. However, if there is no synchronization between behaviors, then their votes are produced based on different system states, so that the semantics of combining the two sets of votes is ill-defined and may yield unpredictable results. An asynchronous system presents an additional challenge in that, in general, the latencies for each module will be different.

Representation of Uncertainty. In command fusion systems, uncertainties are often accounted for in an *ad hoc* manner, for example by "growing" the size of observed obstacles by some fixed amount or by "fuzzifying" the inputs to a system and using fuzzy reasoning to determine an approximately appropriate output (e.g., Kamada et al., 1990). Similarly, potential fields implicitly deal with uncertainty by virtue of the field's extension from a point. Although the fuzzy behavior blending described in (Saffiotti et al., 1995) is formally defined as a logic of graded preferences

(Rescher, 1967), there still exists no objective measure and treatment of uncertainty based on Bayesian probabilities.

3. Utility Fusion

A new means of action selection, *utility fusion*, is presented as an alternative to the methods described above. Utility theory provides a framework for defining votes and dealing with uncertainty. If assign a utility measure *U(c)* for each possible consequence of an action *a*, then the *expected utility U(a)* is:

$$U(a) = \sum_{c} U(c) \cdot P(c|a, e)$$

where *P(c|a,e)* is the probability that consequence *c* will occur given that we observe evidence *e* and take action *a* (Pearl, 1988). Thus, if behaviors provide these utilities and probabilities to an arbiter, it can then apply the *Maximum Expected Utility* (MEU) criterion to select the optimal action based on all current information.

Utility fusion does not create a world model as sensor fusion systems do. The information combined and stored by the utility arbiter does not represent sensed features of the world, but rather the desirability of being in a particular state according to some criterion defined within the behavior. The processing of sensory data is still distributed among behaviors, so the bottlenecks and brittleness associated with sensor fusion are avoided.

Unlike command arbitration or fusion systems, the utility arbiter does not simply select among or combine actions proposed by behaviors. Instead, behaviors indicate the utility of possible world states, together with estimates of uncertainty; thus, the arbiter is provided with much richer information from behaviors for intelligent decision-making. The arbiter maintains a map of these utilities, and evaluates candidate actions within it.

For example, consider a map-based utility arbiter for steering control, with behaviors for road-following and obstacle avoidance. Uncertainty is represented by a two-dimensional Gaussian distribution. The polygons in Fig. 2 show an area of positive utility associated with the road location, and an area of negative utility associated with a detected obstacle, with the lighter polygons suggesting the reduced probabilities as distance increases. The arbiter evaluates the possible trajectories, shown in the figure as arcs emanating from the vehicle, by summing the expected utilities along them, and selects that one for which the total is the greatest.

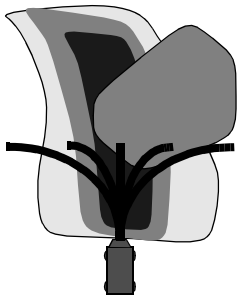

Figure 2. Map-based utility arbiter. Darker areas reflect higher probabilities. Arcs indicate trajectories evaluated by the arbiter.

3.1. Advantages of Utility Fusion

System Model. One important advantage of map-based utility fusion over command fusion is that the dynamics of the system being controlled can be modeled and accounted for by the arbiter, providing greater control accuracy and stability than in command fusion. For example, because the arbiter is evaluating candidate actions rather than the behaviors, it can use knowledge of its own processing latencies as well as delays inherent in the system and compensate for them via predictive control. Using current vehicle state, a history of recently issued commands, and knowledge of the effects of those commands, the arbiter can determine approximately where the vehicle will be when the next command is actually executed and assess feasible trajectories originating from that position, as indicated by the lighter vehicle outline in Fig. 3.

Non-holonomic and kinematic constraints can also be taken into account by the arbiter. For example, the steering mechanism of the vehicle imposes the constraint of a linear change in curvature when a new steering angle is commanded, so that the vehicle path follows a clothoid until the desired curvature is reached, and only then follows an arc (Kanayama and Miyake, 1985). Because it has sufficient state information, the utility arbiter can evaluate this more complex path, suggested in the diagram above, thus evaluating those actions which may be followed faithfully by the vehicle.

As described in Section 4., experiments were conducted comparing the utility arbiter to the previously used command arbiter. At slow vehicle speeds, both arbiters were able to achieve their mission, but at higher vehicle speeds the effects of system latency and dynamics became very apparent, and utility fusion performed much better than command fusion under those conditions.

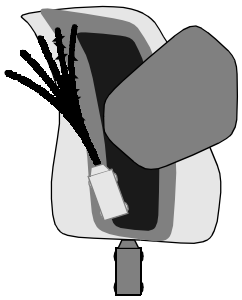

Figure 3. Utility arbiter with dynamics. Clothoid-arc trajectories are evaluated from the predicted vehicle position.

Vote Synchronization and Semantics. A map-based utility arbiter also solves the problem of unsynchronized behaviors because the information received from them is not time dependent. Command fusion involves combining behavior outputs which are only valid for a brief interval, so that their semantics are ill-defined unless executed immediately, but the utility arbiter receives votes for external world states whose meaning is well-defined independent of the current vehicle state. The utility map coordinates votes received at different times and from different vehicle locations, thus synchronizing the votes without imposing timing constraints on the behaviors, which would reduce system responsiveness.

It would be possible for each behavior in a command fusion scheme to reason about the constraints of the system and to perform predictive control, at the cost of greatly increasing the complexity and reducing the reusability of the behaviors. In addition, the behaviors would have to receive state information from the arbiter concerning the commands that have been sent to the controller but not yet executed, and the handshaking involved would further increase complexity. Perhaps the greatest limitation, however, is that behaviors would be forced to operate at a sufficiently high rate so that the prediction and decision-making process could be updated often enough to maintain stable control. Thus, many behaviors would be forced to separate into multiple independent processes, effectively duplicating the effort that currently takes place in the utility arbiter.

Representation of Uncertainty. Through the use of utility theory, uncertainty within the system is explicitly represented and reasoned about within the decision-making processes. Utility theory teases apart the value of the consequence of an action from the probability that the consequence will occur and provides a Bayesian framework for reasoning about uncertainty (Berger, 1985). Each behavior votes for the subjective utility of the agent's possible states, e.g., for a vehicle, locations where obstacles or a road may lie. The behavior also expresses the associated uncertainties as covariances in a multi-dimensional normal distribution. Uncertainty in the perception process, as well as in the position of the vehicle, can be generated by various means such as the covariance matrix of a Kalman filter or the residual of a linear regression algorithm.

The arbiter can then use utility theory to reason explicitly about the uncertainty in position and control of the vehicle and apply the MEU criterion to select the optimal action based on current information. 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. The resulting action selection mechanism generates a result which is Pareto-optimal (Pirjanian, 2000).

3.2. Implementation of Steering Utility Arbiter

The utility arbiter maintains a local map of the utilities and associated uncertainties sent to it by the behaviors. The utilities may be represented by geometric constructs (points, lines, and polygons) with associated two-dimensional gaussian distributions, as in Fig. 4(a) and (b); they may also be represented by a grid as in Fig. 4(c). Both representations are made available so that behaviors may use whichever is more appropriate and efficient; the arbiter maintains the two representations independently and later combines their utility estimates at the final stages.

For the sake of efficiency, the command choices to be evaluated are discretized; the results are smoothed and interpolated to avoid problems with bang-bang control, i.e., switching between discrete choices to achieve an intermediate value. The utility arbitration process is described below.

Step 0: *Initialize Vehicle Trajectories.* Given N possible steering commands, create an NxN matrix A of arc transition paths, with each entry A_{ij} containing a trajectory corresponding to commanding a curvature κ_j while the vehicle is currently executing an arc of curvature κ_i. For Ackerman steering, the vehicle path initially follows a clothoid pattern:

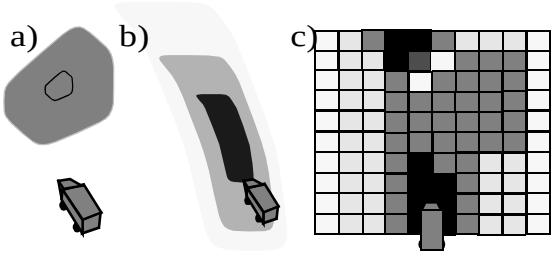

Figure 4. Continuous and discrete utility representations: a) point utility, b) polygonal utility, and c) grid utilities.

$\kappa = C(s) + \kappa_i$, where C is the rate of change of curvature while turning, s is the distance travelled along the path, and κ_i is the initial curvature. Once the desired curvature κ_j has been achieved, an arc of constant curvature is followed. Each trajectory contains $M = L / \Delta s$ points.

Step 1: *Collect Behavior Utilities.* Collect utilities, tagged by the sending behavior with the position of the vehicle at the time the utilities were generated.

Step 2: *Predict Vehicle State.* Get current vehicle state from controller, position (x_t, y_t, θ_t), speed v_t, and curvature κ_t. Compute predicted state based on estimated latency.

Step 3: *Transform Utility Coordinates.* For each point, line, or polygon utility, transform the coordinates to the predicted vehicle reference frame. Let these coordinates be (x_u', y_u'). If any of these lies more than three standard deviations behind the vehicle, i.e., (y_u' - y_n') > $3\sigma_y$ for all vertices, remove that utility object from the map.

Step 4: *Compute Expected Utilities.* For each utility u and each point (x_n', y_n') along a candidate vehicle trajectory, determine the transformed coordinates (x_u'*, y_u'*) of the point that is closest in Mahalanobis distance from (x_n', y_n'). Grid utilities are indexed by the vehicle position; zero is used if it is outside the bounds of the array.

The contribution $E(n',u)$ of utility u to the expected utility of point $n' = (x_n', y_n')$ is the product of the utility value υ and the probability as determined by the Mahalanobis distance between those two points in a bi-normal distribution:

$$E(n', u) = \upsilon \cdot \frac{\exp\left(-\left(\left(x'^{*}_{u} - x'_{n}\right)^2/\sigma_x^2\right) + \left(\left(y'^{*}_{u} - y'_{n}\right)^2/\sigma_y^2\right)\right)}{2\pi\sigma_x\sigma_y}$$

The utility values for grid behaviors is determined by indexing the grid by (x_n', y_n') and multiplying the result by υ; a value of zero is used if the indexed position is outside the bounds of the array.

Step 5: *Sum Expected Utilities on Trajectories.* The total expected utility for taking action a, i.e., following arc A_{ij}, is then computed by summing the utilities at each of the M points along the arc, multiplied by a discount factor λ ($0<\lambda<1$) used to account for the diminished expected returns of future actions:

$$U(a) = \sum_{u} \sum_{s=1}^{M} \lambda^s E(n', u)$$

Step 6: *Maximize Expected Utility.* Determine the maximum expected utility $U(\alpha)$ such that no other expected utility $U(a)$ has a greater value:

$$U(\alpha) \mid \forall \alpha : U(\alpha) \geq U(a)$$

Step 7: *Interpolate Commanded Curvature.* Locally approximate the Gaussian distribution of votes around the selected action α by fitting a parabola to $U(\alpha)$ and the utility values for the two adjacent commands, $U(\alpha-1)$ and $U(\alpha+1)$. The peak of this parabola is then chosen as the commanded action.

Step 8: *Send Controller Command.* Send the steering command as determined in the previous step to the controller, and add it to a queue for use in feedforward prediction.

Step 9: *Repeat Loop.* Return to Step 1.

3.3. Utility Behaviors

Within the framework of DAMN, behaviors must be defined to provide the task-specific knowledge for the domain. For the utility fusion arbiter in particular, the behaviors must provide utilities and probabilities of the form shown in Fig. 4.

Each behavior runs completely independently and asynchronously, using whichever sensor data and processing algorithms are best suited to the task. Behaviors can be easily added or removed from the system, depending on the task at hand.

Obstacle Avoidance Behavior. An important capability in any mobile robot system is avoiding collisions. The *Obstacle Avoidance* behavior detects intraversable regions of terrain determined by range image processing, by stereo vision, by sonar, etc. The obstacle detection system used most extensively with DAMN has been SMARTY (Langer et al., 1994). For each obstacle detected, the *Obstacle Avoidance* behavior sends to the arbiter a large negative utility with a small standard deviation representing the sensor uncertainty. Another negative utility with a smaller value and larger standard deviation is also assigned to the obstacle location to reflect the problems associated with getting too close to an obstacle, i.e., constrained mobility, occlusion of unknown areas, etc.

Follow Subgoals Behavior. Another important functionality that should be present in any general purpose robotic system is the ability to reach certain destinations. For example, the *Follow Subgoals* behavior associates a positive utility with each of the subgoals to be reached by the vehicle, shown as a point with circles of increasing uncertainty in Fig. 5. A positive line utility between subgoals defines a corridor between them; the vehicle tends to stay within the corridor, even in the presence of negative utilities due to factors such as obstacles. This behavior can simply send all utilities to the arbiter at once, and each attracts the vehicle in turn as it gets closer; if a subgoal is unreachable or inadvertently bypassed, then the utilities defined by the next corridor attract the vehicle to the next subgoal.

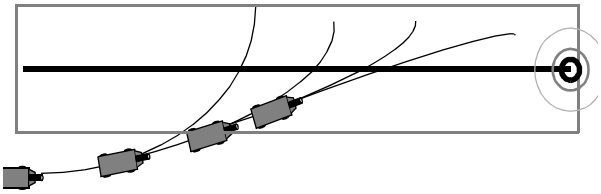

Figure 5. Follow Subgoals behavior: Positive point utilities correspond to subgoals, line utilities between create corridors.

Follow Path Behavior. More sophisticated map-based planning techniques which determine an optimal global path have also been integrated and used within the DAMN framework. For example, the D* planner creates a grid that represents how best to reach the goal from any location in the map (Stentz, 1995). D* is used to implement the Follow Path behavior, which simply sends a portion of its map around the current vehicle position as a grid to the arbiter, and sends a new portions of the map as information is updated or as the vehicle approaches a new region of the map. Only a small portion of the map needs to be sent to the arbiter, and updates need not occur often, so that the bandwidth requirements of this behavior are kept within a very reasonable level.

Each cell in the D* map has an estimated distance to the goal point, $\hat{h}_c$. Let the estimated goal distance from the current vehicle position be h_0. We then obtain the estimated utility of a cell by taking the difference of the two: $u_c = h_0 - \hat{h}_c$. This gives a positive value when the cell's goal distance is less than the current distance to the goal, negative when the distance is greater, and zero when there is no change in the distance to the goal. This results in grids sent to the arbiter like that shown in Fig. 6. The lighter stripes at the bottom of the grid indicate small positive utilities,

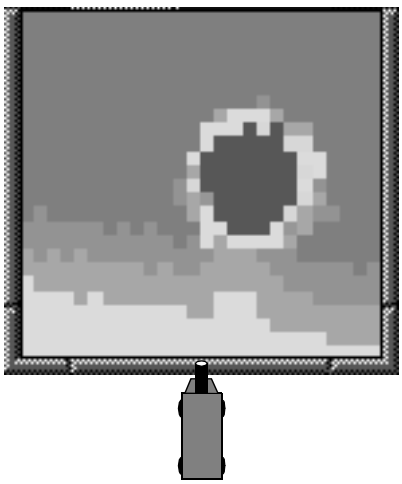

Figure 6. Grid-based utility based on distance to goal. Light to dark stripes indicate increasing positive utility, dark circle indicates large negative utility.

and they go from lighter to darker as the positive utility increases at the top of the grid, i.e. as the distance to the goal decreases. In this example D* has also been notified of a detected obstacle and placed it in the map; this appears in the grid as a dark circle surrounded by a lighter ring indicating the presence of a large negative utility.

4. Experimental Results

In this section we present some results from experiments conducted on the Navlab II HMMWV Vehicle at Carnegie Mellon University, as well as a simulation of that vehicle.

Experiments were run comparing the utility fusion arbiter to against the command fusion arbiter which accepted votes for various turn commands. Trials with the utility arbiter's predictive control capability turned off were also conducted so its effect could be observed independently.

4.1. Performance Metrics

Mean Obstacle Proximity. An important metric for a vehicle path is the average distance to obstacles along that path. The mean obstacle proximity metric for a path is defined by the inverse square of the distance l_o to the closest obstacle, integrated along the path and normalized by the total number of path points n:

$$l_o = min(\forall o \sqrt{(x_v - x_o)^2 + (y_v - y_o)^2})$$

$$\text{mean obstacle proximity} = \frac{\int_0^l \left(\frac{1}{l_o}\right)^2 ds}{n}$$

Lower mean obstacle proximity means that the vehicle was on the average further away from the nearest obstacle, and therefore the path was safer.

Roughness. Roughness is defined by the square of the change in vehicle curvature κ with respect to time, integrated along the path and normalized by the total time t:

$$\text{roughness} = \frac{\int_0^l \left(\frac{d\kappa}{dt}\right)^2 ds}{t}$$

A lower roughness measure means that curvature changed less over the course of the path, and therefore that the vehicle path was smoother.

Figure 7. Experimental vehicle and test area.

4.2. Utility Arbiter: Vehicle Run Results

The following tests were performed on the Navlab II HMMWV vehicle at Carnegie Mellon University, shown in Fig. 7 in the environment where the tests took place. The Obstacle Avoidance and Follow Subgoals behaviors described above were used in these vehicle experiments. The utility fusion combined the utilities from these behaviors into its vehicle-centered utility map, evaluated candidate trajectories within the map, and issued that steering command that maximized expected utility. The vehicle operated at speeds of approximately 0.8 m/s.

In order to study the effect of predictive control in isolation in these experiments, the utility arbiter was used both with and without that capability. As can be seen in Fig. 8, the vehicle oscillated quite a bit without predictive control, yielding a roughness measure of 0.00432, in contrast to the much more stable path with a roughness measure of 0.00003 generated using predictive control, a 150-fold improvement! The reduction in mean obstacle proximity, from 0.415 down to 0.145, while not as dramatic, still represents a significant improvement; the superior vehicle control afforded by utility fusion resulted in a greater margin of safety for the vehicle.

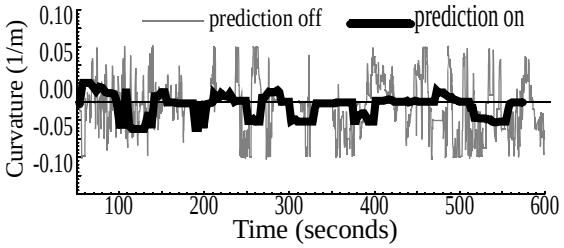

Figure 8. Vehicle curvature vs. time, with and without prediction.

4.3. Simulation Experiments

In this section we present some results from experiments conducted in simulation, demonstrating the benefits of utility fusion in compensating for vehicle dynamics. A vehicle simulator was used so that system parameters such as latency could be controlled, so that higher speeds could be used without causing damage, and so that trials could be repeated in identical conditions. The utility arbiter with and without predictive control, as well as the turn arbiter described previously that used command fusion, were compared at various vehicle speeds and system latencies.

For experiments at slow speeds, all arbiters successfully achieved their mission, both in actual vehicle experiments and in simulation. However, the effects of latency and dynamics became very apparent at higher vehicle speeds. The graphs of mean obstacle proximity as a function of speed in Fig. 9(a) and of path roughness vs. speed in Fig. 9(b) show that the turn arbiter does very badly at higher speeds; these runs are shown in Fig. 10(a).

The graphs also show that, at higher speeds, the utility arbiter without predictive control performed even worse than the turn arbiter, possibly due to the utility arbiter's greater complexity; these runs are shown in Fig. 10(b). However, when the utility arbiter made use of its predictive control capabilities, it was still able to go through this narrow corridor and reach the goal, in spite of the fact that a delay of 2 seconds at a speed of 6 m/s meant that the vehicle travelled 12m between the time that a command was issued and the time that it would actually be executed. These successful path traces are shown in Fig. 10(c), along with the trace of the position of the vehicle as predicted by the arbiter, which coincided well with the actual path.

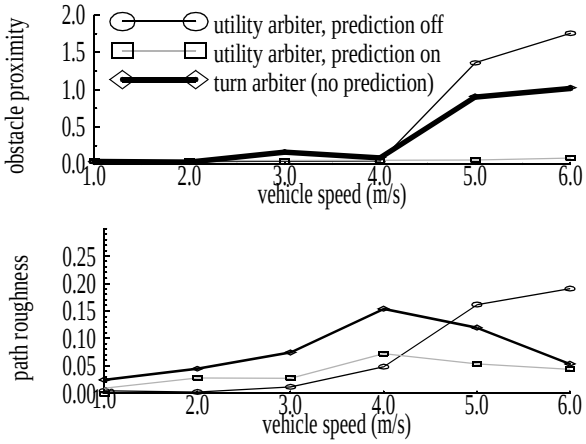

Figure 9. Path metrics as a function of speed: a) mean obstacle proximity, and b) path roughness.

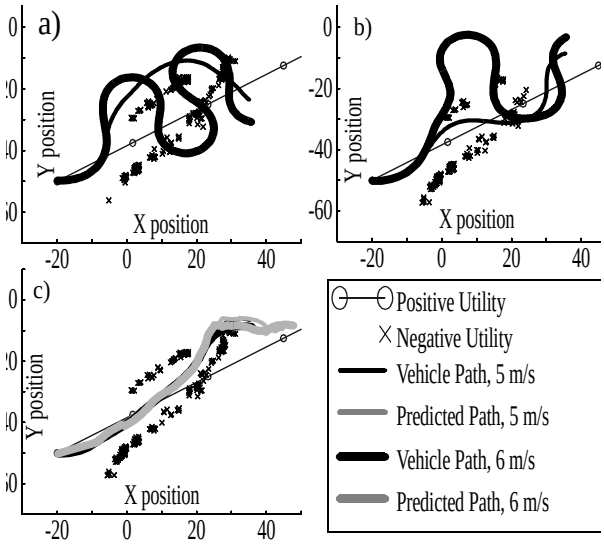

Figure 10. Paths executed at high speeds with 2 second latency: a) turn arbiter, b) utility arbiter without prediction, c) with prediction.

5. Conclusion

Utility fusion is introduced as a solution to the shortcomings of command fusion and sensor fusion systems. Instead of voting for actions, distributed, asynchronous behaviors indicate the utility of various possible world states and their probabilities based on domain-specific knowledge. The arbiter then evaluates various candidate actions, using system models to determine which actions can be taken without violating kinematic and dynamic constraints, and to provide greater stability. It then selects a Pareto optimal action based on the maximization of expected utility.

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, utility fusion provides coherent, optimal reasoning in a distributed, asynchronous system, combining the advantages of sensor fusion and command fusion while avoiding many of their drawbacks. It provides a well defined semantics of votes and uncertainty, and has been demonstrated experimentally to result in measurably better control.

For example, a utility arbiter has been implemented for vehicle steering control. Behaviors indicate the relative desirability of various possible vehicle locations, and the arbiter maintains a local map of these utilities. The arbiter then evaluates candidate actions by summing the expected utilities along each trajectory, taking into account uncertainty in perception and control. The arbiter then chooses that trajectory which maximizes expected utility and sends the corresponding steering command to the vehicle controller.

For experiments at low speeds, the command fusion and utility fusion arbiters all successfully achieved their mission, both in actual vehicle experiments and in simulation. However, the effects of latency and dynamics became very apparent at higher vehicle speeds, and the utility arbiter with predictive control performed much better than the other arbiters under those conditions.

A behavior-based system capable of performing a coral survey has also been developed. The autonomous underwater vehicle system is able to avoid collisions, maintain a proper standoff distance, and following a designated route using either a rope with video or targets with sonar. Initial experiments have been conducted in a pool and in a natural coastal inlet, and the system is to be soon demonstrated in the coral reef environment.

Acknowledgments

This research was supported in part by grants from ONR (N00014-J-91-1451), ARL (DAAH-049610297), ARPA (N00014-94-1090, DAST-95-C003, F30602-93-C-0039, DACA76-89-C-0014, DAAE07-90-C-R059), and NSF (BCS-9120655). The author was supported by a Hughes Research Fellowship.

References

Arkin, R. 1989. Motor Schema-Based Mobile Robot Navigation. *International Journal of Robotics Research*, 8(4):92-112.

Berger, J. 1985. *Statistical Decision Theory and Bayesian Analysis*, 2nd ed. New York: Springer.

Borenstein, J. and Koren, Y. 1991. Potential Field Methods and Their Inherent Limitations for Mobile Robot Navigation. In *Proc. of International Conference on Robotics and Automation*.

Brooks, R. 1986. A Robust Layered Control System for a Mobile Robot. *IEEE Journal of Robotics and Automation*, RA-2(1):14-23.

Durrant-Whyte, H. 1986-Whyte. Integration, Coordination, and Control of Multi-Sensor Robot Systems, Ph.D. Thesis, University of Pennsylvania, Philadelphia, PA.

Kamada, H., Naoi, S., and Gotoh, T. 1990. *A Compact Navigation System Using Image Processing and Fuzzy Control*. In Proc. of IEEE Southeastcon, New Orleans, April 1-4.

Kanayama, Y. and Miyake, N. 1985. Trajectory Generation for Mobile Robots. In *Proc. of Third International Symposium on Robotics Research*, pp. 333-340, Gouvieux, France.

Kelly, A. and Stentz, A. 1998. Rough Terrain Autonomous Mobility - Part 1: A Theoretical Analysis of Requirements. *Autonomous Robots*, 5(2):129-161.

Khatib, O. 1990, Real-Time Obstacle Avoidance for Manipulators and Mobile Robots. In *Proc. of International Conference on Robotics and Automation*.

Langer, D., Rosenblatt, R., and Hebert, M. 1994. A Behavior-Based System For Off-Road Navigation. *IEEE Journal of Robotics and Automation*, 10(6):776-782.

Nilsson, N. 1984. Shakey the Robot. SRI Tech. Note 323, Menlo Park, CA.

Pearl, J. 1988. *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. Morgan Kaufmann Publishers.

Pirjanian, P. 2000. Pirjanian, P. 2000. Multiple objective behavior-based control. *Journal of Robotics and Autonomous Systems*, 31(1/2):53-60.

Rescher, N. 1967. Semantic foundations for the logic of preference. In *The Logic of Decision and Action*, N. Rescher (ed.), Pittsburgh, PA.

Rosenblatt, J. 1997. The Distributed Architecture for Mobile Navigation. *Journal of Experimental and Theoretical Artificial Intelligence*, 9(2/3):339-360.

Rosenblatt, J. 1998. Utility Fusion: Map-Based Planning in a Behavior-Based System, *Field and Service Robotics*, Springer-Verlag.

Rosenschein, S. and Kaelbling, L. 1986. The Synthesis of Digital Machines with Provable Epistemic Properties. In *Proc. of Theoretical Aspects of Reasoning about Knowledge*, pp 83-98.

Saffiotti, A., Konolige, K., and Ruspini, E. 1995. A multivalued-logic approach to integrating planning and control. *Artificial Intelligence* 76(1-2):481-526.

Saffiotti, A. 1997. The Uses of Fuzzy Logic in Autonomous Robotics: a catalogue raisonne, *Soft Computing* 1(4):180-197, Springer-Verlag

Shafer, S., Stentz, A., and Thorpe, C. 1986. An Architecture for Sensor Fusion in a Mobile Robot. In *Proc. of IEEE International Conference on Robotics and Automation*, San Francisco, CA, April, pp. 2002-2011.

Yen, J. and Pfluger, N. 1992. A Fuzzy Logic Based Robot Navigation System. In *Proc. of AAAI Fall Symposium*.

Julio Rosenblatt is a Senior Research Associate at the Australian Centre for Field Robotics at Sydney University, and is an Australian Research Fellow. He received his Ph.D. and M.S. degrees in Robotics from Carnegie Mellon University in 1996 and 1991, respectively, and the S.B. in Computer Science and Engineering from the Massachusetts Institute of Technology in 1985. He was formerly Director of the Autonomous Mobile Robotics Laboratory at the University of Maryland, College Park, and a research scientist at the Hughes Research Labs in Malibu, CA. His research interests include behavior-based architectures for mobile robot control, artificial intelligence, intelligent systems and agents, decision theory, the combination of quantitative and qualitative methods for mobile robot navigation, and autonomous underwater vehicles.