Control-Driven Mapping and Planning

D. Wooden* , M. Powers† , D.C. MacKenzie§ , T. Balch† and M. Egerstedt*

* {wooden,magnus}@ece.gatech.edu
Electrical and Computer Engineering, Georgia Tech, Atlanta, GA 30332, USA

† {mpowers,tucker}@cc.gatech.edu
College of Computing, Georgia Tech, Atlanta, GA 30332, USA

§ doug@mobile-intelligence.com
Mobile Intelligence Corp., Livonia, MI 48150, USA

***Abstract*— Layered hybrid controllers typically include a planner at the top level with reactive control at the lower levels. The planner considers the state of the robot in a global context. The low-level controllers consider only the local environment of the robot and are able to operate at a high frequency to ensure the safety of the robot. Also, it is often the case that the low-level controllers consider more aspects of the robot's state (e.g. kinematic constraints) than the planner. The consideration of such constraints at the planning level would prohibitively increase the state space the planner must consider and, accordingly, its running time and complexity. In this paper, we investigate how we can take advantage at the planning level of domain knowledge encapsulated in the lower level controllers, and we introduce a feedback mechanism that enables low-level controllers to influence the high-level planner.**

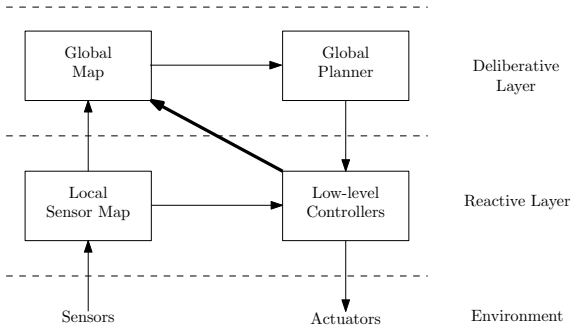

Fig. 1. Depicted is the Standard Hybrid Control System Block Diagram (without the thick arrow) and the new, Proposed Block Diagram (with the thick arrow) allowing for information to flow backwards between the two layers.

I. Introduction

Over the past decades, a canonical, two-layer control architecture has emerged for solving a number of mobile robot navigation tasks [2], [6], [8], as is illustrated in Fig. 1. In contrast to this canonical architecture, we, in this paper, introduce a novel approach for allowing feedback information to flow "backwards", i.e. from the controllers to the map. In fact, we will let a *veto mechanism* (which blocks unsafe directions for the robot) trigger feedback signals to the *deliberative* layer.

This feedback, depicted in Fig. 1, represents a new pathway for information flow in the layered architecture. In the standard architecture, information passes from the repository of map-based data to the controllers. Our architecture, however, is bidirectional, adding the ability of the controllers to pass information back into the global map. Hence, this paper describes a framework that simultaneously controls the robot and maps the environment.

The standard two-level architecture is employed in robotics applications for two reasons. First, the controllers must operate on a short time scale in order to guarantee that the robot is kept in a safe and allowable state. By decoupling the controllers from the mapping and planning processes, the deliberative layer is afforded more flexibility in terms of cycle regularity and frame rate. Second, from a complexity management point-of-view, the map is typically planar, i.e. contains a 2DOF description of the environment, while the controllers (which operate on a smaller spatial scale) can take the full kinematics and dynamics of the robot into consideration. For example, yaw, pitch and roll can be considered together with position, resulting in a higher dimensional configuration space. However, the planning process can barely keep up with real-time constraints in a planar world, and any attempt to plan paths through the full configuration space would be infeasible. As such we propose to project high-dimensional "obstacles" detected at the controller-level into the planar map.

Our overall approach is devised as follows. A global path planner continually re-plans based on updated sensory input incorporated into a global map and passes the path to the low-level controllers. We suggest the use of an "optimistic" planner, where the configuration space of the robot is reduced slightly, making the planner's conception of the robot holonomic. The controllers, on the other hand, are able to operate on a precise and accurate model of the robot, and with the corresponding configuration space make better informed decisions about what local regions are better to travel over.

This work was supported by DARPA through Grant number FA8650-04-C-7131.

II. Related Work: Dynamic Windows and Configuration Spaces

The dynamic window approach (DWA) is a method for accounting for the robot's velocity and acceleration capability in a reactive-layer framework [4], [5]. This approach considers a short time window and, using knowledge of the robot's current translational and rotational speed and maximum translational and rotational accelerations, a kind of velocity configuration space is computed. During this brief time period, the robot considers short arcs of constant curvature. Essentially, the DWA is a low-level controller that, as such, could be incorporated into the reactive layer described in this paper.

In [3], the authors describe a method of using the DWA in combination with a local objective function and partial global path planning. This approach is attractive because it directly addresses the need for global path (re)planning in an unknown environment, but path planning is only done on a limited portion of the entire global space. The major difference described in this paper is that we provide a mechanism for bidirectional communication between any global path planner (A^*, D^*, etc) and any suite of low-level controllers (possibly including DWA).

Central to the work in this paper is that different layers in a hybrid control architecture can use different configuration spaces. In fact, the configuration space is a geometric encoding of of all achievable poses, defined at a certain level of abstraction, with respect to the robot's kinematic constraints and, possibly, with respect to external constraints (i.e. obstacles). Changing the footprint of the robot necessarily changes the configuration space of the robot, as such a change forces the robot into a different set of configurations.

In order to reduce the computational burden associated with the navigation problem, the map-based planning layer uses a two-dimensional representation of the robot, which idealizes the robot's footprint as a circle with a diameter equal to the width of the robot at the drive wheels (Fig. 2(b)). This simplification of the configuration space is valid as long as the robot is driving straight ahead, but underestimates the size of the robot as it turns, especially as it turns in place. In fact, because of the choice of a circle as the idealized footprint, rotational kinematics are not taken into account at all. The effect is that the planning layer is "optimistic" about the robot's capabilities

Because the low-level control layer is working over a smaller spatial and temporal window, this can afford to use the full configuration space (Fig. 2(c)). (In other words, in evaluating the robot's kinematic constraints, one needs the robot's x and y Euclidean coordinates, as well as its orientation, θ.) This representation reflects the robot's actual kinematics and provides an accurate or even "pessimistic" (as a margin of safety is commonly added to the model of the robot) evaluation of the robot's capabilities.

An argument could be made that the planning layer could simply use a pessimistic over-estimate of the robot's footprint (perhaps a circle circumscribing the robot's actual footprint), always planning paths with wide safety margins. We find two problems with this strategy: First, it is possible that no solution exists for the pessimistic planner, when an accurate representation of the robot's kinematics would find a path. Second, especially when working with visual sensors, sensory data is less accurate far away from the robot. Using an optimistic planner effectively allows the sensory data a margin of error before closing off any path. While a pessimistic planner would plan around these borderline cases, an optimistic planner would bring the robot closer, allowing for better sensory input. It is this later problem – of reconciling an optimistic planner with the robot's true capabilities – that this work aims to address.

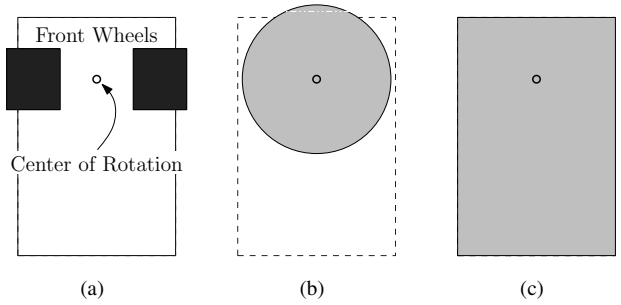

Fig. 2. Footprints used for the LAGR robot. (a) A diagram of the robot (pointed up) and its center of rotation. (b) The "optimistic" footprint used by the planning process. (c) The "pessimistic"/accurate footprint used by the low-level controllers.

III. Mapping, Planning, and Control

In this section, we briefly describe our robot platform, system integration, and planning and control tools. A more complete description of this system is provided in [12] and [14].

A. The LAGR Setup

The test bed for this paper's experiments is the LAGR robot. Learning Applied to Ground Robots (LAGR) is a DARPA-funded project with the goal "to develop a new generation of learned perception and control algorithms for autonomous ground vehicles, and to integrate these learned algorithms with a highly capable robotic ground vehicle" [1].

Fig. 3. The LAGR Robot.

The LAGR robot, depicted in Fig. 3, possesses four color cameras, a front bump switch, a Garmin GPS receiver, and an inertial measurement unit. The cameras are paired together so that each pair can provide stereo depth maps with a range of 6-10 meters. The robot's turning axis is centered just behind the front axle, with the rear unpowered wheels turning on casters. Its physical dimensions are 90 cm in length, 60 cm in width, and 60 cm in height, with a weight of about 90 kg.

Mapping, control and planning processes are run on a single Linux machine (1.4 GHz Pentium-M, 1 GB RAM, RedHat 9), the *planning computer*. There are three similar computers connected via Ethernet: 1 each for camera/stereo processing (called *eye* computers), and 1 for lower-level functions (e.g. radio-controller interfacing, GPS/IMU integration).

B. Map Processing

The two *eye* computers collect camera images, compute stereo depth maps, and using the robot's global position, transform a local terrain map onto a global coordinate frame. Then, this information is passed over Ethernet to the planning computer. On the same machines, the camera images are used to compute estimates of traversability of points in the global frame, as described in [7]. Both processes run steadily at 4Hz.

The planning computer receives the two streams of stereo and traversability information from both *eye* computers and incorporates it into global maps of terrain and traversability. This data is stored in grid cells of fixed size, with a resolution of 0.1 m. Under the assumption that the newest information is the most likely to be correct, previous information in a grid cell is overwritten with the new.

A separate map also stores the locations where the robot has encountered hits on its bumper, spikes in its motor amperes, and detections of wheel slippage.

C. Motion Planning

Incremental motion planning is executed over the global maps described above. The planning algorithm we use is either an A^*-type planner, or the combinatorial planner we described in [15]. However, for the purposes of this discussion, the particular planner to be used is of no real consequence. The mapper passes on points which have been modified by the stereo/traversability/etc processes to the planner, which incrementally updates its planned path.

D. Motion Control

Reactive control is implemented in a manner inspired by the DAMN [9] architecture. In this implementation, individual controllers, representing specific interests related to the robot's overall objective, are given an allotment of "votes" which they may cast for or against actions that will work to achieve their goals. An arbitrator sums the votes, choosing the action with highest tally. Similar implementations have been successfully deployed in several robotic navigation tasks [13], [10].

In our implementation, the actions evaluated are straight-line paths, at a resolution of 5 degrees around the robot. The controllers take the kinematic constraints of the robot into account by evaluating the effect of first turning to the desired direction (effecting the rotational component of the motor command) and then traveling in that direction (effecting the translational component of the motor command). This turns out to be a reasonable approximation of the robot's low-level controller, which implements a relatively aggressive rotational gain. The following voting controllers are used:

- **follow-plan** – casts positive votes in the direction of the next point on a list of waypoints provided by the planner. Votes are distributed according to a Gaussian function centered on the direction of next waypoint.
- **avoid-stereo-obstacles** – casts negative votes in the direction of any obstacles sensed by the stereo vision system. Votes are distributed according to a sum of Gaussian functions, each centered on a sensed obstacle.
- **avoid-color-obstacles** – casts negative votes in the direction of any obstacles sensed by the traversability (i.e. color-based) vision system. Votes are distributed according to a sum of Gaussian functions, each centered on a sensed obstacle.

One potential pitfall of arbitration over votes is misallocation of each controller's allotment of votes. If controllers' voting weights are not properly balanced, one controller may dominate the arbitration, either preventing the robot from making progress to higher-level goals or allowing the robot into undesirable states. Because this weighting is typically an empirical process and dependent on implementation and environment, we have added robustness in a manner similar to [11] by supplementing the voting scheme with "vetoes". Each controller, in addition to its allotment of votes is given the option to veto each of the available actions. The arbitrator respects the vetoes by ignoring actions that have been vetoed by at least one controller, regardless of how many votes those actions have received.

The strategy is that actions which are deemed to put the robot in imminent danger should be vetoed. What qualifies as "imminent danger" must be decided on a controller-by-controller basis. Because the burden is only to identify dangerous paths over a short distance, the full dynamics of the robot can be considered, including collision checking of rotations necessary to achieve the desired orientation and the feasibility of various maneuvers given the slope of the terrain.

Like the voting controllers, these vetoing controllers use a set of straight-line paths at 5-degree resolution as the action set:

- **veto-stereo-obstacles** – casts vetoes against any direction which will bring the robot into a collision with a stereo vision-sensed obstacle within one meter of the robot's current position, taking into account the robot's configuration space.
- **veto-color-obstacles** – casts vetoes against any direction which will bring the robot into a collision with a color vision-sensed obstacle within one meter of the robot's current position, taking into account the robot's

configuration space.

While the specific control mechanism is certainly an issue open for debate and research, it is not a central component to this work. In fact, the point should be made that any reactive controller (e.g. probabilistic voting, motor schemas) could be adapted to work in this architecture. The specific implementation presented here is given only as an example of how a system could be integrated into the larger architecture.

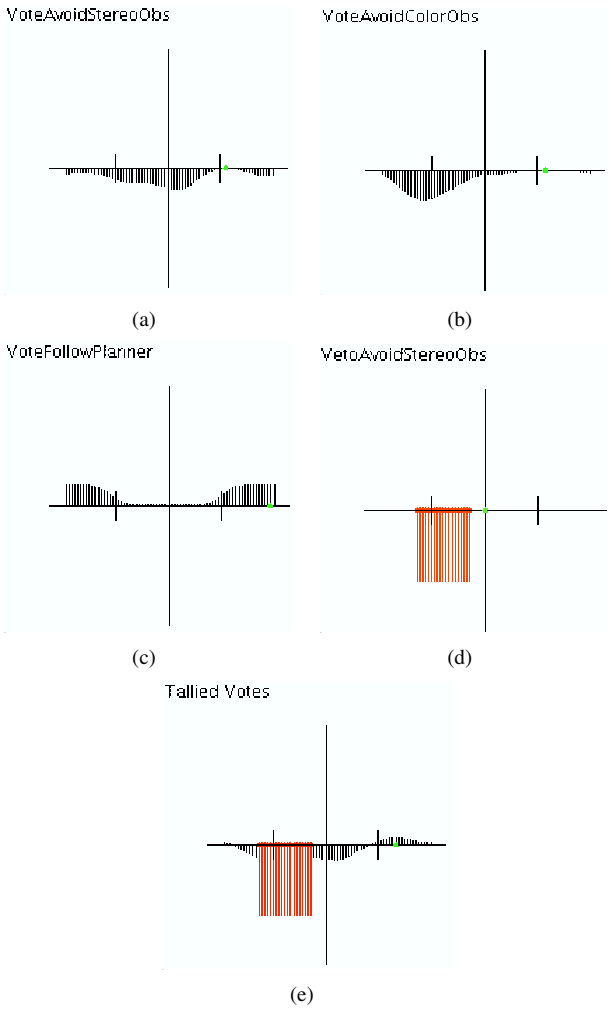

Fig. 4. A graphical representation of the voting scheme employed to navigate the robot. The x axis of each plot represents an ego-centric angular distribution of possible paths around the robot in the range $(-\Pi, +\Pi\ ]$, with 0 being in front of the robot. The y-axis represents the relative preference of each path, according to the respective controller. Vetoes are drawn as large negative values. The last plot represents the sum of the votes provided by all the controllers. The largest non-vetoed value is chosen for action by the robot. In this example, the first behavior resists a stereo-perceived obstacles to the front and left of the robot. A color-based obstacle is perceived to the left. The plan tells the robot to go backwards, and the left is vetoed as a result of stereo obstacles. The tallied votes tell the robot to go to the right.

IV. Feedback from Controller to Mapper

The main contribution of this paper is an architecture where controllers drive the robot based on maps, and the maps are informed (in part) by the controllers. The resulting bidirectional information flow between operational layers thus consists of the standard map-to-controller flow as well as the novel controller-to-map flow (see Fig. 1). Conceptually, this latter feedback mechanism from the controllers to the map, is executed by feeding back points to the map which correspond to conflict between the deliberative and reactive controllers.

Refer to Fig. 5; two obstacles leave a small opening, allowing a feasible path to pass between, given that the planner assumes some relatively optimistic configuration space for the robot. However, the controllers, which consider higher-dimensional kinematic/dynamical constraints do not allow this action. In the the proposed architecture, the controllers then detect this conflict and inform the map that the point (highlighted in the figure) should be marked as intraversable.

The points placed into the map can be set conservatively small, blocking only a small region in the map. This may then not block the passage through the offending region in a single step, but multiple planning/controller cycles through this region will place several points in the map, and eventually cover the kinematic obstruction.

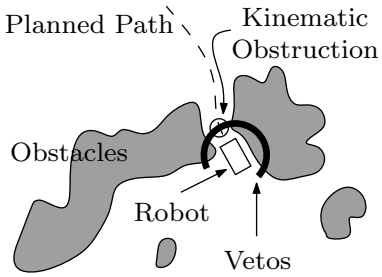

Fig. 5. Illustration of basic operation of the proposed framework.

V. Experimental Results

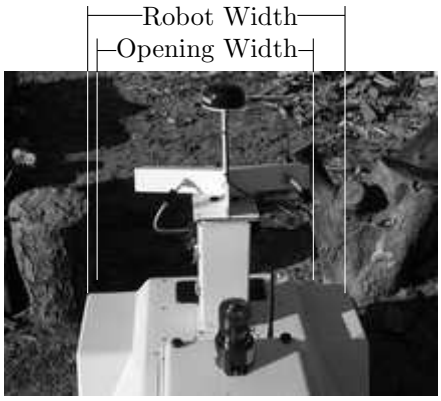

Fig. 7. Opening in Cul-de-sac.

In order to highlight the benefit associated with the proposed method and to illustrate its practical usefulness, we ran a series of controlled experiments in an outdoor terrain populated by small pine trees, fallen logs and other vegetative obstacles, as seen in Fig. 6. In the following subsections, we

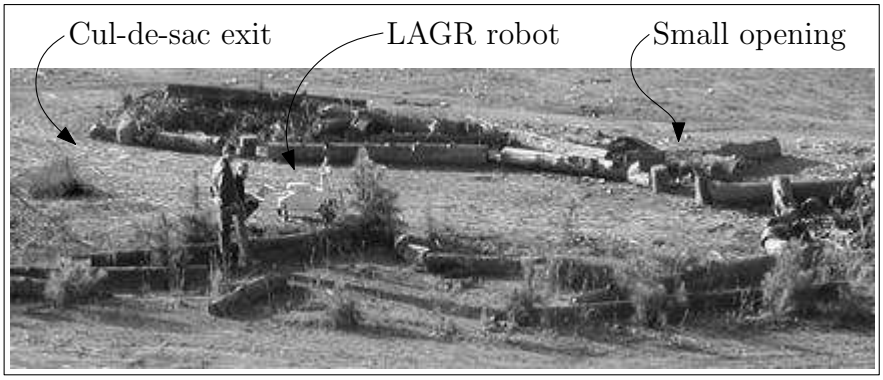

Fig. 6. Overview of Experiment Site.

summarize the outcomes of these experiments and highlight the differences in performance.

The particular experimental setup that we considered was the following: The robot was started up in the interior of a cul-de-sac with one small opening along one of its walls. This opening was wide enough to tell the aggressive planner that a feasible path exists through the opening. However, the reactive controllers will find the opening to be too narrow for safe passage, and as a result, they will veto any attempt to drive through it. For each experiment, the robot was started with no *a priori* information about the environment except its relative position to the global goal.

A. Planner Only

In the first run, only the planner was affecting the motion of the robot, and the only active low-level controller was a path-following controller. As was to be expected, the planner found the opening and tried to push through (Fig. 7), with the result that the robot crashed into one of the logs defining the boundary of the opening (Fig. 8). It should be noted that this problem can be remedied by making the planner less aggressive and allowing for a larger, explicit safety-footprint. However, one of the basic ideas behind the system framework is to let the planner be aggressive and optimistic, and let the reactive low-level controllers ensure safety and robustness if the planned path is deemed unsafe.

B. Reactive Controllers Only

In the second run, only the reactive, local controllers were active, and no global plan was provided from the planner. This control strategy exhibited the well-known and expected behavior of getting stuck in the cul-de-sac without any global information (aside from the heading to the goal) to guide the robot (Fig. 9). It should be noted though that the safety controller did in fact veto the opening that the planner-only controller tried to push through.

C. Planner Affecting the Reactive Controllers

In this scenario, there exists the potential for planning out of the cul-de-sac as well as proper maintenance of safe operation, but since the planner had no way of knowing that the opening was too narrow, it insisted on the robot driving through the opening. Meanwhile the safety-controller vetoed that action. As a result, the robot did not exhibit any improved behavior over the reactive-controller-only situation

Fig. 8. The map, trajectory and plan resulting from an experiment using only a planner. Because the planner is too optimistic for the configuration space of the physical robot, the robot collides with obstacles while trying to navigate through the narrow opening.

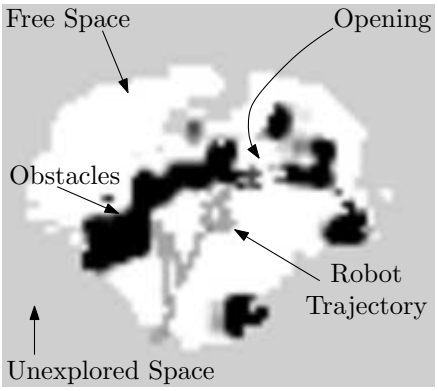

Fig. 9. The map and trajectory resulting from an experiment using only reactive controllers. The safety-minded controllers kept the robot a safe distance from all obstacles, but did not allow progress to the goal location.

(see Fig. 10). However, if the robot would have started outside the cul-de-sac, it is possible (but by no means guaranteed) that it would have eventually planned its way of the area based on the stored map information.

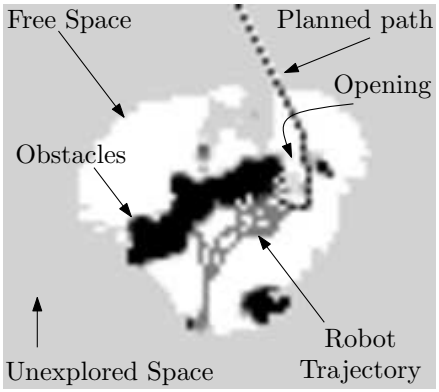

Fig. 10. The map, trajectory and plan resulting from an experiment using a planner which influences reactive controllers. The optimistic planner guides the robot toward the narrow opening, while the safety-minded controllers prevent the robot from entering. The result is that the robot loiters around the mouth of the opening.

D. Feedback From the Controllers to the Planner

Here, the planner once again tried to force the robot through the opening in the cul-de-sac wall. However, the safety-controller vetoed this action as well as encoded this veto through the feedback mechanism as an obstacle in the map, and the planner then re-planned its course of action. As seen in Fig. 11, after a bit of exploring of the cul-de-sac, the robot decided that there was no way forward through the cul-de-sac, and a path was planned out from the area, which enabled the robot to continue its mission.

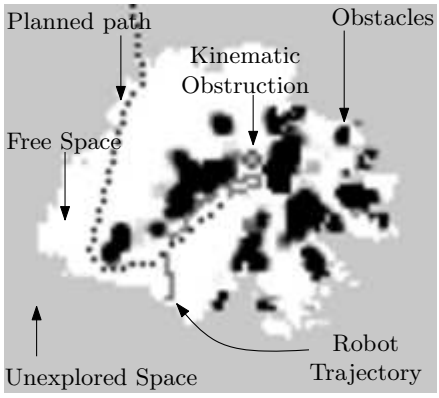

Fig. 11. The map, trajectory and plan resulting from an experiment using a planner which influences reactive controllers with feedback back to the global map. The planner initially guides the robot toward the narrow opening, but the reactive controllers veto this action, noting that action in the global map. Using this information, the planner finds a path through the only safe opening in the cul-de-sac.

VI. Conclusions

In this paper, we argue that it is beneficial to introduce a feedback mechanism from the low-level controllers to the high-level mapping and planning processes. In particular, based on the performance of the low-level controllers (and their interaction with the environment), kinematic obstructions are encoded in the global map even though they may not be perceived as obstacles by the planner. We argue that this is beneficial for the following reasons:

- The low-level controllers typically operate at a shorter time scale than the planner. This implies that the obstructions can be detected sooner by the controllers and, as such, they can notify the planner directly by short-cutting the time-consuming perception processing step.
- Due to the computational burden of global path planning, maps are typically planar (or at least low-dimensional), which means that high DOF kinematics, dynamic constraints, or complex configuration spaces cannot be handled directly. Through the feedback from the controller to the mapper/planner they can, however, be incorporated in the lower-dimensional descriptions of the environment.

We illustrate this view through an experiment in which a robot is trying to negotiate a cul-de-sac. This experiment shows that the introduction of feedback between layers has a beneficial impact on the robot performance.

References

[1] Learning applied to ground robots (lagr) proposer information pamphlet, May 2004. BAA # 04-25.

[2] R. C. Arkin. *Behavior-Based Robotics*. The MIT Press, 1998.

[3] O. Brock and O. Khatib. High-speed navigation using the global dynamic window approach. In *Proc. Int. Conf. on Robotics and Automation*, 1999.

[4] D. Fox, W. Burgard, and S. Thrun. The dynamic window approach to collision avoidance. *IEEE Robotics and Automation*, 4(1), 1997.

[5] D. Fox, W. Burgard, S. Thrun, and A. Cremers. A hybrid collision avoidance method for mobile robots. In *Proc. of the IEEE International Conference on Robotics & Automation*, 1998.

[6] E. Gat. Integrating planning and reacting in a heterogeneous asynchronous architecture for mobile robots. In *SIGART Bulletin*, volume 2, pages 70–74, 1991.

[7] D. Kim, J. Sun, S. M. Oh, J. Rehg, and A. Bobick. Traversability classification using unsupervised on-line visual learning for outdoor robot navigation. In *IEEE Int'l Conf. on Robotics and Automation*, 2006.

[8] A. Ranganathan and S. Koenig. A reactive robot architecture with planning on demand. In *Proc. of the IEEE Int'l Conf. on Intell. Robots and Systems*, 2003.

[9] J. Rosenblatt. Damn: A distributed architecture for mobile navigation. *Journal of Experimental and Theoretical Artificial Intell.*, 9(2):339–60, 1997.

[10] J. K. Rosenblatt. Maximizing expected utility for optimal action selection under uncertainty. *Autonomous Robots*, 9(1):17–25, 2000.

[11] R. Sukthankar, D. Pomerleau, and C. Thorpe. A distributed tactical reasoning framework for intelligent vehicles. In *Intelligent Systems and Manufacturing*, 1997.

[12] J. Sun, T. Mehta, D. Wooden, M. Powers, J. Regh, T. Balch, and M. Egerstedt. Learning from examples in unstructured, outdoor environments. *Journal of Field Robotics*, 2006.

[13] S. B. Williams, P. Newman, J. Rosenblatt, G. Dissanayake, and H. Durrant-Whyte. Autonomous underwater navigation and control. *Robotica*, 19(5):481–496, 2001.

[14] D. Wooden. A guide to vision-based mapping. *IEEE Robotics and Automation Magazine*, June 2006.

[15] D. Wooden and M. Egerstedt. Oriented visibility graphs: Low-complexity planning in real-time environments. In *IEEE Confernce on Robotics and Automation*, June 2006.