PROCEEDINGS

55. IWK

Internationales Wissenschaftliches Kolloquium

International Scientific Colloquium

13 - 17 September 2010

Crossing Borders within the ABC

Automation, Biomedical Engineering and Computer Science

Faculty of
Computer Science and Automation

www.tu-ilmenau.de

TECHNISCHE UNIVERSITÄT
ILMENAU

Impressum
Published by

Publisher:	Rector of the Ilmenau University of Technology Univ.-Prof. Dr. rer. nat. habil. Dr. h. c. Prof. h. c. Peter Scharff
Editor:	Marketing Department (Phone: +49 3677 69-2520) Andrea Schneider (conferences@tu-ilmenau.de) Faculty of Computer Science and Automation (Phone: +49 3677 69-2860) Univ.-Prof. Dr.-Ing. habil. Jens Haueisen
Editorial Deadline:	20. August 2010
Implementation:	Ilmenau University of Technology Felix Böckelmann Philipp Schmidt

USB-Flash-Version.

Publishing House:	Verlag ISLE, Betriebsstätte des ISLE e.V. Werner-von-Siemens-Str. 16 98693 Ilmenau
Production:	CDA Datenträger Albrechts GmbH, 98529 Suhl/Albrechts
Order trough:	Marketing Department (+49 3677 69-2520) Andrea Schneider (conferences@tu-ilmenau.de)

ISBN: 978-3-938843-53-6 (USB-Flash Version)

Online-Version:

Publisher:	Universitätsbibliothek Ilmenau ilmedia Postfach 10 05 65 98684 Ilmenau

Home / Index:
http://www.db-thueringen.de/servlets/DocumentServlet?id=16739

PILOT - MODULAR ROBOT NAVIGATION FOR REAL-WORLD APPLICATIONS

Erik Einhorn, Tim Langner

MetraLabs GmbH, Am Vogelherd 22, 98693 Ilmenau, Germany

ABSTRACT

For robot navigation the E*-Algorithm and the Dynamic Window Approach (DWA) have emerged as a de facto standard for path and motion planning. Based on these algorithms we present a generic solution for robot navigation that is applicable for both holonomic and non-holonomic robots. We propose a number of improvements and extensions that help to overcome several limitations of the original implementations and that are required for robots in daily operation. We introduce an adaptive Dynamic Window that allows a fine-grained control of the robot's actuators enabling the system to navigate with an inch-perfect precision while reducing the computational complexity. Furthermore, we present the flexible and modular design of our navigation approach that enables its usage in many different real-world applications.

***Index Terms*—** robot navigation, motion planning, path planning, adaptive dynamic window

1. INTRODUCTION

Accomplishing navigational tasks like path planing, obstacle avoidance and motion planning are essential capabilities for autonomous mobile robots allowing them to offer more complex services. Most current navigators support a "drive to" command only. However, for several industrial and real-world applications where autonomous mobile robots are used, this simple navigation task is no longer sufficient. Moreover, these navigators path planning is often limited to a single map. For that reason, they are not suitable for navigation in complex environments such as multi-level office buildings or factories where the use of elevators is required or the robot has to follow traffic rules like one way streets.

Our main focus is on the development of robots and robotic solutions for different commercial and industrial fields, like homeimprovement stores, fast-food restaurants and clean-room surveillance. Where the operational area varies from small rooms to large multilevel halls. This leads to a variety of different requirements such as:

- support for non-holonomic robots of different sizes,
- navigation with high precission (e.g. docking, handling of narrow passages),
- fast path planning and online dynamic replanning,
- taking moving obstacles into account,
- consideration of traffic rules (e.g. one way streets, forbidden areas and speed limits)

Therefore, we require a navigator that combines all these requirements in a modular architecture and that is able to change its behaviour according to new tasks at runtime. Additionally, in contrast to existing navigation algorithms which always consist of both, path and motion planning parts, we present an approach where each component can be enabled depending on the needs of the current task.

2. MODULAR ARCHITECTURE OVERVIEW

To accomplish the many different requirements and navigational tasks, we use a modular software design as seen in Fig. 1. The core component of our navigator receives the current navigational task from the user or the overlying application layer. The task can also be specified using a built-in scripting language. The scripting language allows remote procedure calls (RPC) and therefore enabling the control of the navigator from distributed applications. After a task is received by the navigator it is decomposed into distinct motion and velocity commands of the robot's wheels in order to process the desired task.

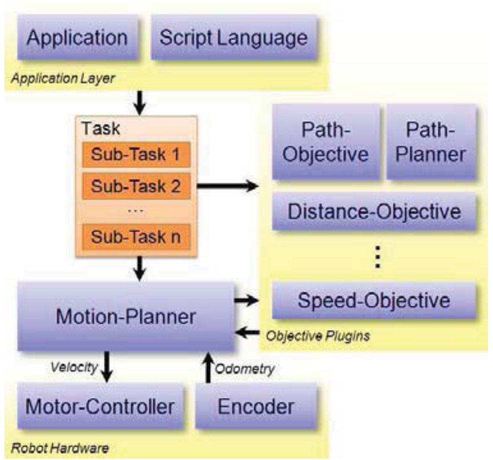


Fig. 1. *The architecture of our approach. A task received from the application layer is processed by the motion planner using multiple objectives in order to generate motion commands.*

The research leading to these results has received funding from the European Community's Seventh Framework Programme (FP7/2007-2011) under grant agreement #216487

For task processing, the motion planner and the objectives play a major role. Each objective is a separate software module specialized for certain tasks like following a person, driving at a certain speed or direction, etc. The objectives are realized as software plugins. This allows us to add new objectives easily when new tasks are necessary without changing other parts of the navigator. The output of the objectives is then used by the motion planner to generate motion commands that are then sent to the robot's motor controllers. Some objectives require additional information from other navigational modules such as localization and mapping algorithms or modules for user interaction like person trackers. Therefore, the objectives can access this information directly from these modules. Both, the motion planner and the objectives are described in section 4 in more detail.

3. NAVIGATION BY TASK

Based on the modular concept of objectives, we need a suitable interface for the navigator that allows us to specify complex tasks. In order to developing a highly generic navigator, we introduce a task based system which allows us to define jobs consisting of several sub-tasks and their corresponding parameters. To meet the requirements of different applications we have, so far, defined the following sub-tasks that can be combined to create complex tasks:

- docking with a charging station
- following a person
- explore the environment autonomously while avoiding obstacles
- driving to a specified position
- adjust the orientation at the goal point (e.g. to face a user)
- drive a maximum allowed distance

Each sub-task can be parametrized by numerous task-specific options, including:

- goal point to drive to,
- map to drive to
- preferred driving direction of the robot (backward, forward or both)
- accuracy for reaching a goal point
- accuracy for the orientation angle at a goal point
- maximum allowed driving distance (e.g. during exploration)

By specifying a combination of sub-tasks and their parameters the robot's navigational behavior can be completely modified at runtime. For example the complex task "Drive backward to the destination $(10, 0)$ in map 'Floor2' with an accuracy of ± 0.5 m and turn to the orientation of 70° with an accuracy of $\pm 15°$ " is easily handled by our navigation system. Using our built-in scripting languge that task can be described as follows:

```
Task t;
t.addSubTask(
  PreferredDirectionTask(BACKWARD));
t.addSubTask(PositionTask(10, 0, 0.5));
t.addSubTask(MapTask("Floor2"));
t.addSubTask(OrientationTask(70, 15));
Navigator.setTask(t);
```

To fulfill the job, the navigator must complete each sub-task by following the specified rules and options. Moreover, the task based system allows for the addition of new sub-tasks in the future, when required, without changing the interface of existing tasks or interfering with existing applications. This is an important advantage of this task based design since it reduces the effort necessary for software maintenance.

4. MOTION PLANNING

After a new task is set it is processed by the motion planner. Existing methods for motion planning and obstacle avoidance can be separated into reactive and anticipatory approaches [1]. Reactive approaches usually plan their movements in the workspace of the robot, i.e. the xy-plane which the robot can move on. One of the most frequently used reactive method is the VFH approach and its successor VFH+ [2]. These algorithms compute a polar histogram that represents the obstacle density in each direction around the robot. Based on this histogram, the approach identifies consecutive sectors where the obstacle density is below a certain threshold. These directions are classified as free, the other directions are blocked. Finally, it chooses a free direction that is closest to the target direction. However, the approach cannot handle non-holonomic robots with arbitrary shapes - one of our basic requirements. Additionally, the robot's dynamic is modelled insufficiently.

Anticipatory approaches usually plan their decisions in the robot's action or velocity space. They simulate the consequences for carrying out a certain action and choose the action that is expected to yield the highest benefit for reaching the target. One of the most often used algorithms for anticipatory motion planning is the Dynamic Window Approach [3]. This algorithm searches for the optimal motion command directly in the space of velocities. Therefore, the robot's two dimensional velocity space is discretized into regular two dimensional cells where each cell represents a certain trajectory. Each cell is defined a cost function. Finally, the velocities corresponding to the cell that yields the smallest costs are chosen as motion commands. In these computations the Dynamic Window Approach takes the robot's dynamic into account by reducing the search space to those velocities which are reachable under the dynamic constraints such as limited speed and acceleration. Additionally, only velocities are considered that are safe and do not lead to collisions. To check if certain velocities lead to collisions, the robot's trajectory is simulated for a small time frame using a kinematic model of the robot. The resulting circular trajectories are then tested for the distance they keep to the obstacles around the robot [1]. In this collision test non-holonomic robot shapes can also be taken into account. Beside other advatages, this makes the approach very attractive for our purposes.

In recent years several modifications and variants of the Dynamic Window Approach have been proposed. In [4] the robot's movements are simulated for larger time intervals to achieve a larger planning horizon. From the resulting circular trajectories, called "tentacles", the tentacle is chosen that yields the smallest costs computed according to a predefined cost function.

In [1] clothoids are used instead of circular trajectories, since they can approximate the robot's real trajectories better. However, since these curves, with linear variation of the curvature, are described by three parameters the resulting search space becomes three-dimensional and cannot be discretized into regular cells as this was done in the original Dynamic Window Approach. Instead, a particle filter is used to sample the search space in [1].

4.1. Adaptive Dynamic Window Approach

For our navigation framework we implemented a modified Dynamic Window Approach. Since we apply our navigator on robots with a differential drive, our velocity space i.e. the search space $S = [v_{min}, v_{max}] \times [v_{min}, v_{max}]$ is spanned by the velocities of the robot's left and right wheel (v_l, v_r). In contrast to most existing Dynamic Window Approaches we explicitly model negative velocities. This allows the robot to plan backwards motion in a seamless way. For most of our robots we choose $v_{min} = -0.5\frac{m}{s}$ and $v_{max} = 1.0\frac{m}{s}$ to achieve the maximum translation velocity of $1\frac{m}{s}$ in the forward direction while limiting the speed to $0.5\frac{m}{s}$ when driving backwards.

Since the robots must be able to move with very high precision, the velocity space of our Dynamic Window Approach must be discretized with a high resolution to produce fine-grained motion commands. However, this would result in enormous computational costs since the robot's movements must be predicted for a huge number of different speeds. In contrast to the original Dynamic Window Approach we therefore do not raster the velocity space regularly. Instead, we take into account that large changes in the robot's velocity cannot be performed and predicted accurately. When predicting the robot's motion, most approaches assume a linear motion model with constant acceleration. Due to unmodeled effects in the robot's hardware and its motion controllers, these models lose precision when the robot changes its velocity due to a large acceleration or deceleration. Hence it is not necessary to use a high resolution for large changes in the velocity's space. Therefore, we apply an adaptive discretization of the velocity space. While the resolution is high for velocities near the robot's current speed, it decreases for velocities near the border of the dynamic window (see Fig. 2b). On the one hand, this allows a fine-grained motion control since small changes in speed can be taken into account while on the other hand, it also reduces the computational costs since large regions of the velocity space that can not be reached by precise motion commands, are coarsly represented.

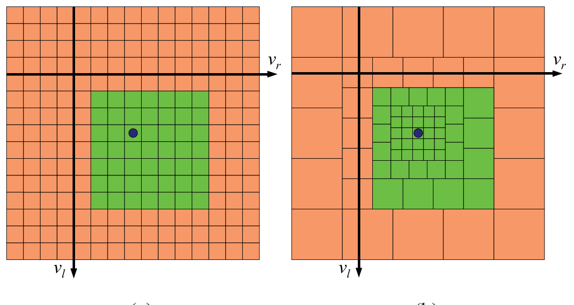


Fig. 2. *Discretization of the velocity space.* (a) *Regular rasterization.* (b) *Our proposed adaptive rasterization. Cells that cannot be reached due to the robot's dynamic constraints are colored in red.*

Our subdivision of the velocity space into discrete cells is done as follows. We place a cell with a given minimum size, i.e. a maximum resolution, at the position in the velocity space that corresponds to the robot's current position. Afterwards, more cells are added in a ring around that cell. That ring is again surrounded by another ring with cells and so on. The number of cells in each ring is controlled by a configuration $(r_1, r_2, \ldots, r_n)$, where r_i specifies the width of the i-th ring in cells. In Fig. 2b the configuration $(3, 5, 5, 5, \ldots)$ is shown. That configuration is used in most of our applications. However, depending on the requirements, the configuration and the adaptive resolution of the dynamic window can be changed arbitrarily. The classic regular decomposition of the velocity space in Fig. 2a can be obtained using the configuration $(3, 5, 7, 9, \ldots)$.

4.2. Objectives

As stated before, each cell covers a certain area of the velocity space and corresponds to a certain velocity command. For motion planning, a cost function is computed for each cell and therefore the velocity command that yields the smallest cost is chosen. In the original Dynamic Window Approach [3] that cost function is composed of three different functions, called objectives. One objective yields large costs when the robot would get too close to obstacles by choosing that certain action. The second objective prefers actions that lead to high speeds and the third one takes care of the robot's orientation. Additionally, each objective can forbid a certain action completely by marking it as "not admissible" when a certain requirement, like the minimal distance to an obstacle, is not met. If at least one objective marks an action as "not admissible" the action is excluded from the set of allowed actions.

In our approach we generalize this idea and decompose *all* navigational behaviors into objectives. For each cell of the adaptive dynamic window, that lies within the dynamic constraints of the robot, each objective is called to compute its cost value c_i for that cell or to mark the associated action as “not admissible”. In the latter case the computation for the cell is aborted immediately to save computation time. The final overall cost of each cell is then computed by the weighted sum of the returned cost values c_i of all objectives:

$$c = \sum w_i c_i \tag{1}$$

where w_i denotes the weight for each objective that controls the influence of each objective on the overall cost. Like in classical multi agent systems, the choice of the correct weights may be crucial. Therefore, it is reasonable to design the objectives in a way that they return their decision about a certain action using the “not admissible” marker instead of a cost value wherever this is applicable. The Distance Objective, the No-Go Objective and the Direction Objective which are described below are good examples of where the “not admissible” marker is sufficient. When using such a design the majority of objectives simply vote for the exclusion of an action from the set of allowed actions while only a few objectives finally compute a cost value for the remaining actions in order to choose the best one. In our applications the number of these objectives usually varies from one to three. The choice of the weights for such a small number of objectives is uncomplicated.

In order to reduce the computational costs the objectives are activated and deactived automatically depending on the current task. Each objective usually is specialized for one or two certain sub-tasks. Therefore, an objective is activated automatically if it “understands” at least one sub-task of the current task, otherwise it is deactived since it cannot contribute in the processing of that task.

After all active objective were processed for all cells in the dynamic window and the costs of all cells were computed according to Eq.1, from all admissible cells, the cell with the lowest cost value is chosen and the corresponding action is sent to the motor controllers in terms of a velocity command. Afterwards, the whole processing cycle is repeated until the current task and the specified goal is reached. This happens if all active objectives report that their sub-tasks have been reached.

4.3. Distance Objective

One of the most important objectives is the Distance Objective, that is responsible for avoiding collisions by calculating the distance between the robot and obstacles on the predicted trajectory. Similar to [3] the minimal braking distance d_{V_0} for the given velocity is calculated. It is the distance the robot would move before coming to a stop when performing an emergency brake. Furthermore, we compute the minimum distance d_{min} of the predicted trajectory to the closest obstacle. This is done efficiently by applying a distance transformation to the current local map and by searching for the minimum value in the intersection area of the robot’s shape and map. If $d_{min} \leq d_{V_0}$ the underlying action is discarded by marking it as “not admissible”.

4.4. Path Objective

To combine the proposed reactive method for local motion planning with a global goal oriented path planning stage, we use a special Path Objective. This objective is active whenever a certain goal position is specified that should be reached by the robot. Using this objective the motion planner is turned into a Global Dynamic Window Approach in a similar way as presented in [5]. The cost value c_2 for this objective is taken directly from the navigation function of the path planner. For each position in the global map this navigation function contains a value that resembles the distance to the goal. Hence, the path objective prefers actions that lead the robot closer to the specified target. In contrast to other navigators where the path planner is an integral part of the architecture, in our system it is a part of the Path Objective, allowing the activation and deactivation of the path planner depending on the current task. Our path planner is described in section 5.

4.5. Speed and No-Go Objective

In some scenarios there may be a request for the robot to abide speed limits (e.g. in areas with dense traffic) or to avoid forbidden areas. The Speed Objective fulfills this request by looking up speed limits in an additional grid based map where the grid cells encode the maximum translational speed at a certain position. The No-Go Objective uses another grid map that encodes forbidden areas where the robot is not allowed to drive. All actions leading to a violation of the speed limit or which let the robot enter a, so called, No-Go area are rejected.

4.6. Heading Objective

Looking in a predefined direction when arriving at the destination is an important factor for guiding customers to products in home improvement stores. Also a proper orientation at the end of a navigation process is necessary for docking with charging stations or delivering goods to delivery stations in a transport system. The Heading Objective will turn the robot by weighting the actions leading to orientation φ_r according to how close they get the robot to the specified target orientation φ_g by computing $c_5 = |\varphi_r - \varphi_g|$. To prevent the robot from turning to the desired direction while still driving to the goal, we choose a very small weight for this objective. This way the heading objective only gains influence when the robot is near its target location.

4.7. Person Follow Objective

Following and observing a person are essential tasks for robots assisting elderly people in their home. The Person Follow Objective can be parameterized to follow a person while taking privacy into account by keeping a given minimum distance between the robot and the user. Hypothesis about the users position obtained from a person tracker are constantly interpreted as new target positions. The robot shortens the distance to these positions until a minimum spacing is reached without the need of planning a path. The objective will also turn the robot to face the user.

4.8. ASTRoNAuT Objective

The term ASTRoNAuT stands for Accurate ShorT Range NAvigaTion. This name was chosen since this objective is used whenever the robot needs to move or be positioned very accurately at short range without planning a global path, like docking with a docking station. The target $t = (t_x, t_y, t_\varphi)^\top$ is specified by its position (t_x, t_y) and the target orientation t_φ. Furthermore, the desired precision is given as a covariance matrix S. The cost for an action that leads the robot to a predicted pose $p = (x, y, \varphi)$ can then be computed by the Mahalanobis distance: $c_7 = \sqrt{(p-t)^\top S^{-1}(p-t)}$.

4.9. Additional Objectives

As new objectives can be easily added due to the modularity of the system we are able to implement some simple but yet effective objectives such as a Mileage Objective whose goal it is to let the robot drive a specified distance. Combined with the Explore Objective which rewards actions corresponding to high velocities, a step aside behavior can be simulated where the robot moves out of the way. The Direction Objective is used to influence the driving direction of the robot by prohibiting velocities in the dynamic window that contradict to the prescribed moving directions. Finally, the User Objective enables a person to manually steer the robot remotely. Therefore, in the dynamic window the costs of the cells are weighted according to the desired actions of the remote controller. If used together with the Distance Objective the robot will follow the users commands while still avoiding obstacles automatically.

5. TOPOLOGICAL PATH PLANNING

Planning a path in complex scenarios including multiple maps on different floors implies that a topological representation of all known maps is used. In our approach we make use of a directed graph representation. Each map $M = \bigcup R_i$ consists of different regions $R_i, 1 \leq i \leq n$, where each region R_i contains nodes n_i^j that have certain positions within their region. Different nodes are connected via directed edges $E = (n_i^j, n_k^l)$ that have either predefined or calculated traversal costs. Each region R_i contains two special nodes n_i^{robot} and n_i^{goal}. The first one represents the robot's pose and is connected to all nodes n_i^j. The latter represents the goal and all nodes n_i^j are connected to it as well. Additionally all nodes n_i^j of region R_i are connected with each other bidirectional. For the calculation of the traversal costs we differentiate between different edge types. Costs for edges in the same region $E_i \subseteq E$, with $E_i = (n_i^j, n_k^l), i = k$ are calculated once in an initialization step using the E* planning algorithm. Edges between nodes in different regions $E_t \subseteq E$, with $E_t = (n_i^j, n_k^l), i \neq k$ are called transitions. The traversal costs of transitions must either be predefined or could be zero. Moving between nodes connected by edges from E_i requires the robot to drive along a path in R_i, while executing a transition from E_t always implies a change of the region or even the current map. Since maps may be based on different coordinate systems or could be of different resolution we use the relation between maps that are described as transformation matrices. They allow the conversion of the robot's pose from map to map. Additionally, the graph planner will wait for the execution of user context (e.g. calling an elevator, opening a door) which can be added to transitions. Transition nodes should always have corresponding coordinates. An example for this with three regions could be seen in the right of figure 3. Here the regions overlap in the area where the transition nodes are located. This way the pair of nodes n_1^1 and n_3^1 as well as n_2^1 and n_3^2 can have the same coordinates. Region $R3$ is used as a transit area. The robot will wait at the transition $E_1 = (n_1^1, n_3^1)$ for the door to open. In the left of figure 3 it is shown that the graph planner adds no further overhead to the simple case of having a single region in one map only. Our approach currently requires the user to split maps into regions manually. However, in future versions we plan to implement automatic map partitioning algorithms as proposed in [6].

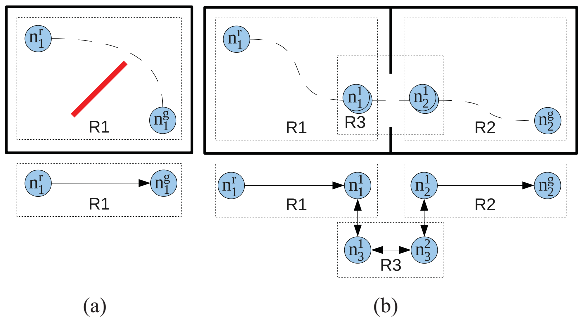
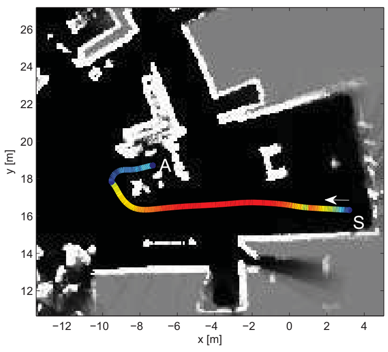
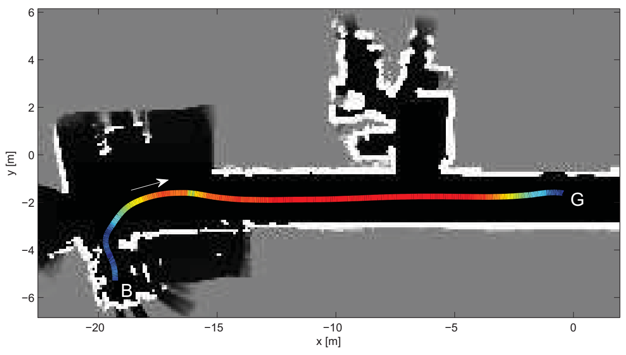


Fig. 3. *Examples for topological graphs. The graph is shown in the lower part of the figure, while the upper part shows the corresponding maps and regions.* (a) *If one map is used containing a single region only, no additional overhead is imposed by the graph planner.* (b) *A more complex scenario where a path is planned across three regions.*

Planning a path in topological maps involves several sub steps. Without loss of generality we assume that the robot is currently localized in region R_c and the goal resides in region R_g. First we update the costs of the edges connected to n_c^{robot} and the costs of the edges ending in n_g^{goal}. Then a path $P = (n_c^{robot}, n_c^k, \ldots, n_i^j, \ldots, n_g^{goal})$ from n_c^{robot} to n_g^{goal} is planned in the topological representation using one of the stan-



(a)



(b)

Fig. 4. *Trajectory of the robot that was driving from its start position (S) to a specified goal (G) on a different floor. The speed of the robot is indicated by the colors, where red corresponds to the robot's maximal speed of 1 $\frac{m}{s}$*. (a) *The partial trajectory on the first floor when driving to the elevator (A).* (b) *The partial trajectory on the second floor when driving to the goal position (G)*

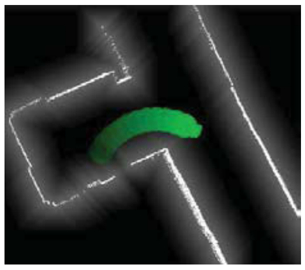
dard graph search algorithms such as Dijkstra or A*. After that, a local path from n_c^{robot} to n_c^j is planned using the grid based planner in R_c. This local path is used by the path objective as described in 4. When the robot reaches node n_c^j the transition given by P is executed and the process starts over again with the update of edge costs.

Using the topological representation we are able to model scenarios were the robot is only allowed to enter a room through one door and leave through another. Also one way corridors could be easily realized with the region based approach. As a side effect - dividing large maps into several regions and restricting the grid based planner to these regions reduces the computational costs for the planning process.

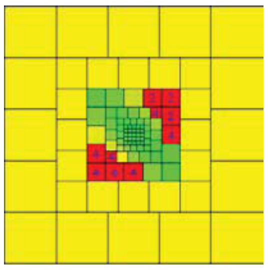
6. RESULTS

The navigation framework that is described in this paper is in long-term use in many different applications ranging from target navigation to person following in different scenarios like home improvement stores, fast food restaurants and factory buildings. In the following we show a small excerpt of its capabilities. Figure 4 shows the trajectory the robot took when it was adviced to drive from its position (S) to a goal position (G) on a different floor in an office building. The topological path planner first guided the robot to the elevator (A) that was used to get to the floor where the target is located (Fig. 4b). After the robot arrived on that floor, the path from the elevator (B) was planed to the goal position. The actual speed of the robot when following the driven trajectory is coded using different colors.

Fig. 5a shows the distance transformed local map of the robot that is used by the Distance Objective. The shades of gray indicates the distance to the closest obstacle. The predicted trajectory and the robot shapes along that trajectory are shown in green. These shapes are used for collision detection in the distance transformed map. Fig. 5b shows a part of the velocity space



(a)



(b)

Fig. 5. (a) *Distance transformed local map that is used by the Distance Objective.* (b) *Part of the velocity space showing the adaptive dynamic window. Red cells were classified as not admissible, while the colors green-yellow indicate the calculated costs for the different actions.*

while driving through a narrow corridor. Cells that were marked as “not addmissible” are shown in red, while cells that cannot be reached due to the robot's dynamic constraints are drawn using yellow color. The green colors indicate the calculated cost of admissible cells.

7. REFERENCES

[1] Ch. Schröter, M. Höchemer, and H.-M. Gross, “A Particle Filter for the Dynamic Window Approach to Mobile Robot Control,” in *Proc. 52nd Int. Scientific Colloquium (IWK)*, 2007, vol. 1, pp. 425–430.

[2] I. Ulrich and J. Borenstein, “VFH+: Reliable Obstacle Avoidance for Fast Mobile Robots,” in *Proc. IEEE Intl. Conf. on Robotics and Automation (ICRA98)*, 1998, pp. 1572–1577.

[3] D. Fox, W. Burgard, and S. Thrun, “The Dynamic Window Approach to Collision Avoidance,” 1997, vol. 4.

[4] F. Hundelshausen, et.al, “Driving with tentacles: Integral structures for sensing and motion,” *Journal of Field Robotics*, vol. 25, no. 9, pp. 640–673, 2008.

[5] O. Brock and O. Khatib, “High-speed Navigation using the Global Dynamic Window Approach,” in *Proc. IEEE Intl. Conf. on Robotics and Automation (ICRA99)*, 1999, pp. 341–346.

[6] S. Thrun, “Learning Metric-Topological Maps for Indoor Mobile Robot Navigation,” *Artificial Intelligence*, vol. 99, no. 1, pp. 21–71, 1998.