Robust Collision Avoidance in Unknown Domestic Environments

Stefan Jacobs[1], Alexander Ferrein[1 2], Stefan Schiffer[1], Daniel Beck[1], and Gerhard Lakemeyer[1]

[1] Knowledge-Based Systems Group
RWTH Aachen University
Aachen, Germany
stefan_j@gmx.de
{schiffer, dbeck, gerhard}@cs.rwth-aachen.de
[2] Robotics and Agents Research Lab
University of Cape Town
Cape Town, South Africa
alexander.ferrein@uct.ac.za

Abstract. Service robots operating in domestic indoor environments must be endowed with a safe collision avoidance and navigation method that is reactive enough to avoid contacts with the furniture of the apartment and humans that suddenly appear in front of the robot. Moreover, the method should be local, i.e. should not need a predefined map of the environment. In this paper we describe a navigation and collision avoidance method which is all of that: safe, fast, and local. Based on a geometric grid representation which is derived from the laser range finder of our domestic robot, a path to the next target point is found by employing A^*. The obstacles which are used in the local map of the robot are extended depending on the speed the robot travels at. We compute a triangular area in front of the robot which is guaranteed to be free of obstacles. This triangle serves as the space of feasible solutions when searching for the next drive commands. With this triangle, we are able to decouple the path search from the search for drive commands, which tremendously decreases the complexity. We used the proposed method for several years in RoboCup@Home where it was a key factor to our success in the competitions.

1 Introduction

One of the most important and most basic tasks for a mobile robot in domestic domains is safe navigation with reliable collision avoidance. In this paper we present our navigation and collision avoidance algorithm which we successfully deployed over many years in the domestic robot domain. The navigation scheme presented here was one of the key features for our success in the domestic robot competition RoboCup@Home. Our method relies on a distance measurement sensor from which a local map of the surrounding is constructed. In our case, we make use of a laser range finder. In this local map, which is in fact a grid representation, we search for a path to a target point. We employ A^* for this.

The such calculated path serves as an initial solution to the navigation task. For computing the path the robot's kinematic constraints have not been taken into account. This was decoupled in order to decrease the size of the search space. We integrate it in a second step where we construct the so-called *collision-free triangle*, where the path as it is calculated by A^* serves as the leg of this triangle. In particular, the current setting of the robot's parameters like speed and orientation are taken into account. By this, we explicitly take care of the robot's kinematic constraints. In the sequel, we prove that this triangle is obstacle-free and can be traversed safely.

The success of our method is founded on two ideas. (1) The first one is to represent the size of the surrounding obstacles depending on the speed of the robot, i.e. the faster the robot drives the larger the obstacles will become, since the robot needs more time to break in front of them. This way we can represent the robot as a mass point. (2) The second idea lies in decoupling the search for a path from its realization. In particular, we propose to construct a collision-free triangle which the robot can traverse safely.

In the past, many different approaches for this fundamental problem have been proposed. So why does this paper go beyond proposing yet another collision avoidance approach? The answer is three-fold:

1. We believe that extending the size of the obstacles depending on the speed of the robot is innovative and worth to be mentioned; with this the robot drives only as fast as possible not to collide with any obstacles. In narrow passages it reduces iteratively its speed until it can safely travel through, while in broad areas it will try to reach its maximal speed.
2. With the collision-free triangle we have an area in the motion area of the robot which is guaranteed to be collision-free.
3. Finally, we deployed this approach for many years in robotics competitions and it was a key to succeed in the domestic robot competition RoboCup@Home for the past three years.

The rest of this paper is organized as follows: In the next section we will present some of the huge body of related articles. In Sect. 3 we introduce our robot platform. In Sect. 4 we present how obstacles as perceived by the sensors of the robot are integrated into its local map, and how a path is calculated. We prove that the collision-free triangle is in fact without obstacles, and show how this triangle bounds the kinematic parameters of the robot. Sect. 5 discusses some implementation details and experimental results. We conclude with Sect. 6

2 Related Work

Approaches to mobile robot navigation can be categorized along several criteria. Some approaches make use of randomized planning techniques, e.g. [1, 2], other approaches use reactive schemes, for instance [3–5], and/or make use of a navigation function to follow a path like [6] or plan directly in the velocity space like [7–10]. Yet other approaches employ a search, some in physical space, some in configuration space.

In their early approach, Borenstein and Koren [7] proposed to use vector field histograms. The target point exerts an attractive force to the robot while obstacles impose repulsive forces. The trajectory of the robot is then formed by the sum of both these forces. They propose a special wall-following mode to avoid getting stuck in local minima which could otherwise cause oscillating behavior. The method was tested with robots equipped with sonar sensors driving with an average speed of 0.58 m/s.

In [3], Fox et al. proposed the dynamic window approach. It is directly derived from the motion equations. In the velocity space circular collision-free trajectories are searched. To handle the state space they define a *dynamic window* around the robot to only consider those velocities that the robot can reach in the next time interval. Finally, a trajectory is found by maximizing over the minimal target heading, maximal clearance around the robot, and maximal speed. The method was tested on an RWI B21 robot with a maximum speed of 0.95 m/s. A similar approach except for the dynamic window was proposed in [11].

Seraji and Howard describe a behavior-based navigation scheme in [12]. They distinguish between different terrain types such as roughness, slope, and discontinuity. A fuzzy controller selects between traverse-terrain, avoid-obstacles, and seek-goal behaviors. While their method aims at outdoor navigation, it is an example for a local reactive navigation scheme. Another reactive approach is presented in [13]. The difference to our work is that we use an optimal path as an initial solution to avoid nearby obstacles.

Besides range sensors, imaging sensors are commonly used to navigate a mobile robot. In [14] an approach to build an occupancy map from a stereo camera on an RWI B14 robot is presented. The map is used to navigate through previously unknown environments. We want to note that our method is different in the sense that we here present a reactive local method while the focus of [14] is on vision-based exploration. A large number other papers deals with navigation approaches using several sensors. Among those, the fusion of imaging and proximity sensors (cameras and LRFs) is popular, see for example [15–17].

Koenig and Likhachev [18] present a search heuristic called Lifelong Planning A*. They propose an incremental search heuristic where only the relevant parts of a path are recalculated. While Lifelong Planning A* is an interesting extension to the basic A* we use in this paper, we here focus on the obstacle representation and the decoupling of path and velocity planning to decrease the dimensionality of the search problem.

The most related research to our approach is the method proposed by Stachniss and Burgard [10]. They use a laser range finder as sensor and employ A* to find a shortest trajectory to a given target. The state space used here is a five-dimensional pose-velocity space consisting of the pose x, y, θ and the translational and rotational velocities v, ω. At first, a trajectory to the target is calculated using A* in the $\langle x, y\rangle$-space. This trajectory serves as the starting point for the search in the pose-velocity space. With a value iteration approach a 70 cm broad channel around the calculated trajectory is calculated which restricts the state space for the five dimensional search. Stachniss and Burgard tested their approach on a Pioneer I and an RWI B21 both having a maximum speed below 1 m/s. By restricting the search for velocities to the collision-free triangle which

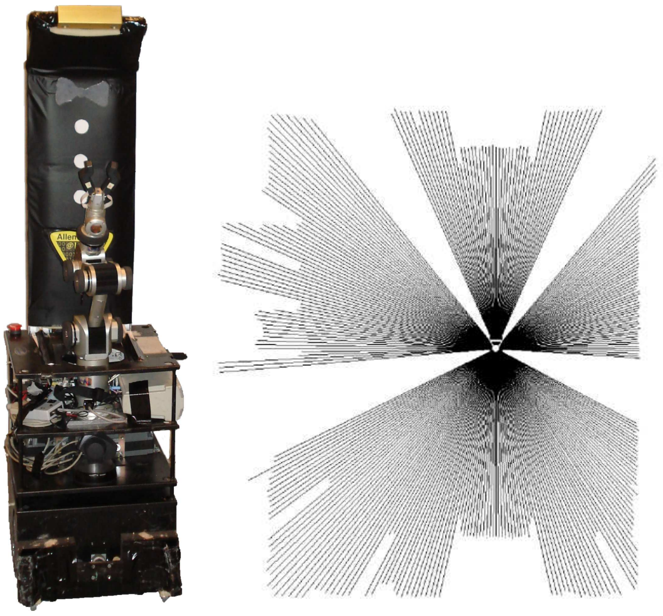

(a) Robot (b) Sensor data

Fig. 1. The robot "Caesar" and its sensor data. The four blind regions are due to the rods supporting the platforms above the laser range finder.

is known to be obstacle-free, we are able to avoid the search in the five dimensional pose-velocity space which leads to a much more reactive navigation and collision avoidance scheme. A similar method applied to holonomic small-size league robots in the robotic soccer domain can be found in [19].

3 Service Robot Platform

Our hardware platform has a size of 40 cm $\times$ 40 cm $\times$ 160 cm. It is driven by a differential drive, the motors have a total power of 2.4 kW and are originally developed for electric wheel chairs. With those motors the robot reaches a top speed of 3 m/s and a maximal rotational velocity of 1000°/s at a total weight of approximately 90 kg. On-board we have two Pentium III PCs at 933 MHz running Linux. Only one of these PCs is used for the navigation and collision avoidance approach presented in this paper. A 360° laser range finder (LRF) with a resolution of 1 degree provides new distance readings for each direction at a frequency of 10 Hz. Fig. 1(a) shows our robot.

We assume accelerated motion with our approach. Thus, the connection between pose, velocity and acceleration is given by $x(t) = \frac{1}{2} \cdot \ddot{x}(t) \cdot t^2 + \dot{x}(t) \cdot t + x_0$, $\dot{x}(t) = \ddot{x}(t) \cdot t + v_0$, and $\ddot{x}(t) = \mathit{const}$. As usual, $\ddot{x}(t)$ refers to acceleration, $\dot{x}(t)$ to the velocity, and $x(t)$ to the displacement at any given time t. In the algorithm we present in the next section, we assume that at each time instance the robot is accelerated only in the first time instance and from there on is driving with constant velocity till the next acceleration command is settled. We need this relation because, for efficiency reasons, we decouple the search in the pose-velocity space into a pose space and a velocity space in the following.

4 The Navigation Algorithm

The task of the navigation algorithm is to steer the robot on a collision-free path from its current location to a given target point. The algorithm we present does not rely on a global map as many other algorithms do but on a local map of its environment. The dimension of the local map corresponds to the area covered by the current reading from the LRF. It is updated every time new laser-readings are received. Although, in our implementation, we integrate new sensor readings into the previous local map if possible, we here assume that the local map is set up from scratch every cycle. In juxtaposition to approaches which rely on a global map a local map has the advantage that it allows to easily account for dynamic obstacles. Moreover, using a local map makes the successful execution of a planned path independent from the localization of the robot. In the following we will give a rough overview of our navigation algorithm and discuss the key aspects in greater detail thereafter.

```
Input: $\Delta x, \Delta y$ the target point in relative coordinates
while not reached target do
    $d_1, \ldots, d_n \leftarrow$ getLaserReadings() ;
    $v_t^{cur}, v_r^{cur} \leftarrow$ getCurrentVelocities() ;
    $map \leftarrow$ extendObstacles($d_1, \ldots, d_n, v_t^{cur}, v_r^{cur}$) ;
    $path \leftarrow$ findInitialPath($map$) ;
    if path.isEmpty() then
        sendMotorCommands(0, 0); break;
    end
    $v_t, v_r \leftarrow$ findVelocities($path$, $map$) ;
    if no $v_t, v_r$ then
        sendMotorCommands(0, 0);
        break;
    end
    sendMotorCommands($v_t, v_r$) ;
    $\Delta x, \Delta y \leftarrow$ updateTarget($v_t, v_r$) ;
end
sendMotorCommands(0, 0) ;
```

Algorithm 1: The navigation algorithm in pseudo-code.

If the robot is in close proximity to the given target, i.e., the target is reached, it stops. Otherwise the current distance readings from the LRF and the current translational and rotational velocities are obtained. This data is then used to build a (local) map. For this we employ a technique we refer to as *dynamic* *obstacle extension* (cf. Sect 4.1) which yields a (local) grid-based map of the robot's environment. The cells of the grid map may either be occupied or free.

With this representation of the surrounding of the robot we search for an initial path to the target, first. In a second step, we search for an approximation of the initial path which takes into account the kinematic constraints of the robot. In both steps we employ the A* search algorithm. Splitting up the path-planning problem into two independent search problems reduces the original problem of dimensionality four to two search problems over two-dimensional search spaces. Namely, finding a path in the xy-plane and appropriate velocities v_t, v_r. This is only possible since the search for an appropriate approximation of the initial path is restricted to a certain area which is guaranteed to be free of obstacles. We refer to this area as the *collision-free triangle*. More details on this are given in Sect. 4.2.

4.1 Dynamic Obstacle Extension

A technique often referred to as *obstacle growing* [20] extends the obstacles by the dimensions of the robot. This alleviates the problem of collision detection in the path-planning problem since the robot can now be treated as a mass point. We leapfrog on this idea and additionally extend the obstacles in dependence on their respective imminence of collision which takes into account the current speed of the robot as well as the position of the obstacle relative to the current trajectory of the robot. The intuition behind this is to mark potentially dangerous areas as occupied in the local map and thereby force the search to not consider paths leading through those areas.

The most threatening obstacle for the next step is the obstacle lying on the trajectory defined by the current and the projected position of the robot in the next step. We assume to recompute a new path every iteration and, consequently, it is not necessary to project further into the future then the next step. The next-step trajectory is computed from the current translational velocity v_r, the current rotational velocity v_r and the time between two iterations $\varDelta t$:

$$v_x = \frac{x(t+1) - x(t)}{\varDelta t} \qquad v_y = \frac{y(t+1) - y(t)}{\varDelta t} \qquad \alpha = \tan \frac{v_y}{v_x}$$

For each detected obstacle we place an ellipse centered at the reflection point of the laser beam in the local map and align the axes such that the semi-major axis is parallel to the laser beam. The radius for the semi-major axis r_1 and the radius for the semi-minor axis r_2 are computed as:

$$\begin{aligned} r_1 &= l + l_{sec} + |\cos(\theta - \alpha)| \cdot d \cdot n \\ r_2 &= l + l_{sec} \end{aligned}$$

where l is the radial extension of the robot[3], l_{sec} is an additional security distance, θ is the angle of the laser beam that hits the obstacle and d is the euclidean

[3] The formula could be further detailed to account for a rectangular shape of the robot, but this is omitted here for clarity reasons.

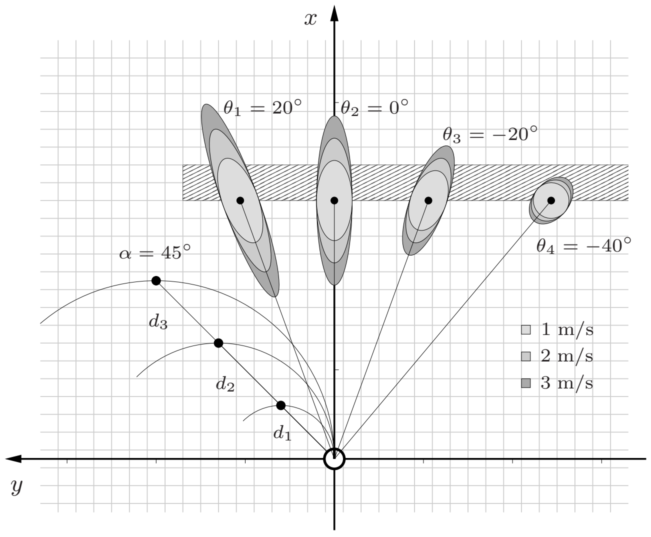


Fig. 2. In this illustrating example a wall is located in front of the robot. The extension of the obstacles is shown for the obstacles detected at 20°, 0°, -20°, and -40°. The translational velocities are 1 m/s, 2 m/s, and 3 m/s; the rotational velocity remains fixed at 1 rad/s. For illustrating purposes we chose $\Delta t = 0.25$ s and $n = 1$.

distance between the current position and the position projected for the next step

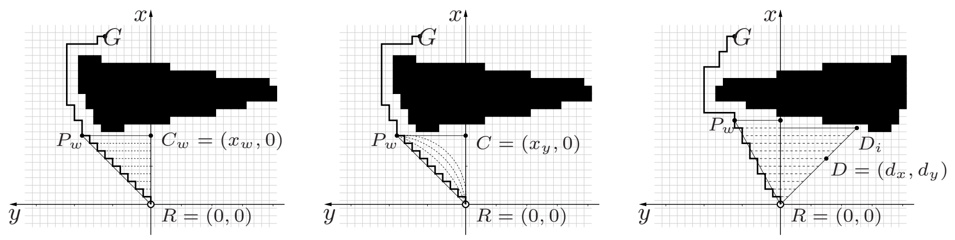
$$d = \sqrt{(v_x \cdot \Delta t)^2 + (v_y \cdot \Delta t)^2}$$

Then, the obstacles are extended in such a way that the robot will stay out of the "dangerous area" for the next n steps.

By means of extending the obstacles in such a way we capture the current obstacle configuration as well as the current configuration of the robot (in terms of translational and rotational velocity) in the local map. Fig. 2 illustrates the extension of the obstacles for different configurations of the robot: the rotational velocity remains fixed whereas the translational velocity is altered between 1 m/s and 3 m/s.

4.2 The Collision-free Triangle

As we pointed out in the introduction, within the search for an initial path we ignore the kinematic constraints of the robots as well as its current configuration in terms of velocity and orientation. With this, we are in good company with several other search-based methods like [10]. However, for successfully realizing a path on a real robot, the kinematic constraints need to be taken into account, of course. In our algorithm, we do so by performing an A* search on the possible accelerations in translation and rotation making use of the standard motion equations for accelerated motion. The kinematic constraints of the robot are only one part of the velocity planning problem of the robot. The other part



(a) Triangle construction (b) Finding a trajectory inside the triangle (c) Triangle for opposite rotational velocity

Fig. 3. Construction of the Collision-free Triangle

is to take the robot's current state into account, i.e. its current translational and rotational velocities. In the following, we therefor present the collision-free triangle. We prove that, by construction, each grid cell inside this triangle is free of obstacles. Poses inside this triangle are thus safe and therefore it can be used to restrict the search for the optimal trajectory the robot should follow.

Definition 1 (Collision-Free Triangle). *The robot is located in the origin of a Cartesian coordinate system, such that $R = (0, 0)$ is the position of the robot facing the positive x-axis. Let $C_k = (x_k, 0)$ be a grid cell in front of the first obstacle along the x axis. In case there is no obstacle along the x axis, the last cell within the perception range of the robot is taken.*

Let $p = \langle(0, 0), (x_1, y_1), \ldots, (x_{g-1}, y_{g-1}), (x_g, y_g)\rangle$ be the path to the target point G. The path is given as a sequence of grid cells. For each path point $P_i = (x_i, y_i)$, $1 \leq i \leq g$ we ray-trace to the point $C_i = (x_i, 0)$. A path point (x_i, y_i) is called safe *iff the ray $\overline{P_i\, C_i}$ does not hit any obstacle, otherwise it is unsafe. The* collision-free triangle *is now given by the points R, P_w, and C_w with w being the index of the last safe path point.*

Fig. 3 depicts the construction of the collision-free triangle as described in the definition. Note that the x axis always points into the direction of the robot's orientation as the map is given by a Cartesian coordinate system with the robot in its origin. Hence, the point C denotes the last free cell before the robot would collide with an obstacle if, from now on, it would drive only with translational velocities. Now, for each path point it is checked if the orthogonal projection of a path point onto the segment $|RC|$ will hit an obstacle. The robot's position R, the last safe path point P_w and the point C_w, the projection point of P_w onto $|RC|$, yields the corner points of the triangle.

Fig. 3(c) illustrates the situation in which the robot is turning away from the path. In that case, we span an additional triangle by projecting the robot's position according to its current velocity for the next step (cf. point D). We put a straight line from the robot's position through D until we hit an obstacle (cf. point D_i). Then we ray-trace analogous to the original triangle to ensure the additional triangle is also obstacle-free.

Theorem 1. *Each cell inside the triangle is obstacle-free.*

Proof. Suppose, P_i is the next path point to be considered in the construction of the collision-free triangle as described in the definition. Hence, the triangle given by $\triangle R, P_{i-1}, C_{i-1}$ is collision-free. Now we ray-trace from P_i orthogonal to the segment given by $|RC|$. If the ray hits an obstacle, the collision-free triangle is given by $\triangle R, P_{i-1}, C_{i-1}$, otherwise we conclude that the $\triangle R, P_i, C_i$ is collision-free. □

P_w is the next point for the robot to reach (safely). P_w closes in on G and the robot will thus eventually reach the target point.

4.3 Computing the Drive Commands

Lastly, a suitable sequence of velocities to reach the intermediate target point P_w needs to be determined. Again we employ A* search to find such a sequence. We restrict the search space by only considering velocity sequences which steer the robot to locations within the collision-free triangle—as soon as the robot leaves the collision-free triangle the search is aborted.

The initial state in the search space is $\langle 0, 0, 0, v_t, v_r \rangle$, i.e., the robot is located at the origin of the coordinate system, its orientation is 0°, and the current translational and rotational velocities are v_t and v_r, respectively. Possible successor states in each step are $\langle x', y', \theta', v'_t, v'_r \rangle$ where $v'_t = v_t + c_t \cdot a_t^{max} \cdot \Delta t$ with $c_t \in \{-1, -\frac{2}{3}, \ldots, 1\}$ and a_t^{max} being the maximal translational acceleration. Analogously for v'_r. (x', y', θ') is the projected pose at time $t + \Delta t$ when sending the drive commands $\langle v'_t, v'_r \rangle$ to the motor controller and the robot is located at $\langle x, y, \theta \rangle$ at time t. The change in position and orientation is computed according to the standard equations for differentially driven robots The heuristic value for each state is computed as the straight-line distance to the intermediate target; the costs are uniform. A goal state is reached if the distance between the projected position and the intermediate goal is smaller than a certain threshold.

The velocities returned by the function "findVelocities(·)" in Alg. 1 are the first translational and rotational velocities in the sequence of velocities that was determined by the search.

5 Implementation and Evaluation

Occupancy Grid. Although the LRF has a far longer range we limited the local map to a size of 6×6 m^2 for practical reasons. The local map is subdivided into grid-cells with a size of 5×5 cm^2. Consequently, the complete local map is made up of 14400 cells. Recomputing the ellipses that result from extending the obstacles and their respective rasterizations with every update is quite costly. Therefore, we pre-computed a library of rasterized ellipses of various sizes and at various angles. For a given obstacle we look-up the ellipse matching the current velocities of the robot and the angle at which the obstacle is detected and integrate it into the local map.

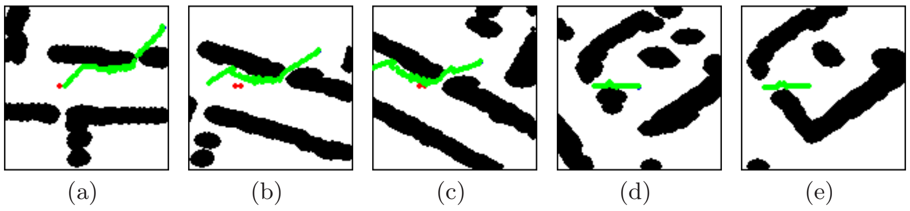
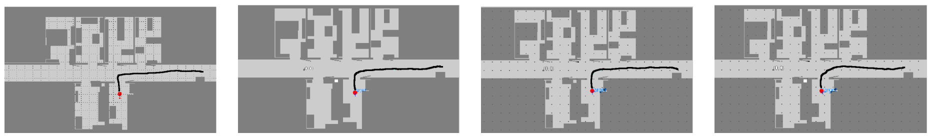


Fig. 4. Example traces

Searching for the Path. In order to avoid that the changed perception of the environment which is due to the movement of the robot leads to an oscillating behavior we accumulate the sensor readings for a short duration, i.e., we do not only consider the current distance readings but also a number of readings from the past. Each obstacle in this accumulated sensor reading is then extended in the same way described in Sect. 4.1. Further, for reasons of efficiency, we try to avoid re-computing a path and proceed with the remainder of the path computed previously, instead. Of course, this is only possible if the "old" path is still a valid path. This means we have to check whether the robot strayed too far from the projected path and whether the collision-free triangle computed for the "old" path is still free of any obstacles. Thus, we still maintain a high degree of reactivity. The implementation of the A^* search algorithm calculates a path of a length up to 300 grid cells (i.e. a path length of 15 m) in less than 10 ms on the Pentium-III 933 machine on the robot. Given that the frequency of the laser range finder is 10 Hz and the navigation module runs at 20 Hz (not to lose any update from the laser range finder) there are about 40 ms left for the other steps of the algorithm.

Evaluation. The method proposed in this paper was extensively tested during several RoboCup tournaments as well as with indoor demonstrations where the robot had to safely navigate through crowds of people. Fig. 4 shows the path visualization of a run of a robot in our department hallway. The red dot represents the robot, the green line represents the planned path, the black objects are the walls of the hallway. The robot should navigate from the hallway into a room. Note that the path is calculated in such a way that the shortest possible connection between robot and target is chosen. The second picture in the series shows the robot a few moments later. The calculation of the drive commands and the realization of these commands on the robot have the effect that the robot slightly deviates from the path. We remark that the position of the robot is still inside the collision-free triangle (which is not shown in the figure). In the fourth picture the robot entered the room.

We also tested our navigation algorithm on our B21 robot Carl as shown in Fig. 5. From the performance point of view we encountered no problems with the algorithm. We could observe that Carl reached higher velocities than with the Dynamic Window (DW) approach [3] which is part of the original control software of Carl. The DW approach has inherently problems with narrow doorways as well as with relatively sharp turns.



(a) B21 with DWA ($v_{\mathrm{max}} = 0.45$ cm/s) (b) B21 with A* ($v_{\mathrm{max}} = 0.45$ cm/s) (c) B21 with A* ($v_{\mathrm{max}} = 0.95$cm/s) (d) Caesar with A* ($v_{\mathrm{max}} = 3$ m/s)

Fig. 5. Comparison of the DWA with our method

6 Conclusion

In this paper we presented a navigation and collision avoidance method for service robots operating in domestic indoor environments. Particularly in these environments, a domestic robot must navigate carefully and be able to drive around obstacles that suddenly cross its path as it is interacting with humans. The core of our method is a grid representation which is generated from the sensory input of a laser range finder. Depending on the speed of the robot and several other security parameters, the detected obstacles are extended in the grid representation. This is done to speed up the collision detection when searching for a path to the target. For finding a path, we employ A* on the grid. Next, we construct a so-called collision-free triangle from the obstacle configuration, the current parameters of the robot (its actual speed) and the desired path. For each grid cell inside this triangle we can guarantee that it is collision-free. In a second step, we use this triangle to calculate the drive parameter for the next time step. Again, we employ A* for this task, this time we search for accelerations which keep the robot inside the triangle. The collision-free triangle relates the search for a path with the search for drive commands and allows to decouple both. Positions inside the triangle are safe and therefore feasible. This decreases the complexity of the search problem tremendously. This method allows for fast and adaptive navigation and was deployed for RoboCup@Home competitions over several years without ever colliding with the furniture or humans. For example, we were able to solve the Lost&Found task in 25 seconds while driving through a large part of the apartment looking for an object.

Acknowledgments
This work was supported by the German National Science Foundation (DFG) in the Priority Program 1125, Cooperating Teams of Mobile Robots in Dynamic Environments. A. Ferrein is currently funded by a grant of the Alexander von Humboldt foundation. We would like to thank the anonymous reviewers for their helpful comments.

References

1. Kavraki, L., Svestka, P., Latombe, J.C., Overmars, M.H.: Probabilistic roadmaps for path planning in high-dimensional configuration space. IEEE Transaction on Robotics and Automation **12**(4) (1996) 566–580

2. Petti, S., Fraichard, T.: Safe motion panning in dynamic environemts. In: Proc. of the IEEE/RSJ International Conference on Intelligent Robots and System (IROS-05). (2005) 2210 – 2215
3. Fox, D., Burgard, W., Thrun, S.: The dynamic window approach to collision avoidance. IEEE Robotics & Automation Magazine **4**(1) (1997) 23 – 33
4. Khatib, M., Bouilly, B., Simeon, T., Chatila., R.: Indoor navigation with uncertainty using sensor-based motions. In: Proc. of the IEEE/RSJ International Conference on Robotics and Automation (ICRA-97). (1997) 3379 – 3384
5. Fiorini, P., Shiller, Z.: Motion planning in dynamic environments using velocity obstacles. The International Journal of Robotics Research **17** (1998) 760–772
6. Brock, O., Khatib, O.: High-speed navigation using the global dynamic window approach. In: Proc. of the 1999 IEEE International Conference on Robotics and Automation. Volume 1. (1999) 341–346
7. Borenstein, J., Koren, Y.: The vector field histogram - fast obstacle avoidance for mobile robots. IEEE Transactions on Robotics and Automation **3**(7) (1991) 278–288
8. Ulrich, I., Borenstein, J.: Vfh*: Local obstacle avoidance with look-ahead verification. In: Proc. of the 2000 IEEE/RSJ International Conference on Robotics and Automation. (2000) 2505–2511
9. Large, F., Sekhavat, S., Shiller, Z., Laugier, C.: Towards real-time global motion planning in a dynamic environment using the nlvo concept. In: Proc. of the 2002 IEEE/RSJ International Conference on Robots and Systems. (2002) 607 – 612
10. Stachniss, C., Burgard, W.: An integrated approach to goal-directed obstacle avoidance under dynamic constraints for dynamic environments. In: Proc. of the 2001 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS-01). (2002) 508 – 513
11. Simmons, R.: The curvature-velocity method for local obstacle avoidance. In: Proc. of the 1996 IEEE International Conference on Robotics & Automation (ICRA-96). Volume 1. (1996) 1615 – 1621
12. Seraji, H., Howard, A.: Behavior-based robot navigation on challenging terrain: A fuzzylogic approach. In: Proc. of the IEEE Transactions on Robotics and Automation (ICRA-02). (2002) 308–321
13. Minguez, J., Montano, L.: "nearness diagram navigation (nd): Collision avoidance in troublesome scenarios". IEEE Transactions on Robotics and Automation **20**(1) (2004) 45–59
14. Murray, D., Little, J.: Using real-time stereo vision for mobile robot navigation. Autonomous Robots **8**(2) (2000) 161–171
15. Asensio, J.R., Montiel, J.M.M., Montano, L.: Goal directed reactive robot navigation with relocation using laser and vision. In: ICRA. (1999) 2905–2910
16. Dedieu, D., Cadenat, V., Soueres, P.: Mixed camera-laser based control for mobile robot navigation. In: Proc. of the 2000 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2000). Volume 2. (2000) 1081 – 1086
17. Wijesoma, W., Kodagoda, K., Balasuriya, A.: A laser and a camera for mobile robot navigation. In: 7th International Conference on Control, Automation, Robotics and Vision (ICARCV 2002). Volume 2. (2002) 740 – 745
18. Koenig, S., Likhachev, M.: Improved fast replanning for robot navigation in unknown terrain. In: Proc. of the 2002 IEEE International Conference on Robotics and Automation (ICRA-02). (2002) 968–975
19. Bruce, J., Veloso, M.: Safe multi-robot navigation within dynamics constraints. Proc. of the IEEE, Special Issue on Multi-Robot Systems **94**(7) (2006) 1398–1411
20. Meystel, A., Guez, A., Hillel, G.: Minimum time path planning for a robot. In: Proc. of the 1986 IEEE International Conference on Robotics and Automation (ICRA'86). (April 1986) 1678–1687