Robbie: A Fully Autonomous Robot for RoboCupRescue

Johannes Pellenz, David Gossow, and Dietrich Paulus

University of Koblenz-Landau
Institute for Computational Visualistics – Active Vision Group
Universitätsstr. 1, 56070 Koblenz, Germany
{pellenz,dgossow,paulus}@uni-koblenz.de

Abstract

One of the goals of the RoboCupRescue competition is to provide a standardized testbed for robots that can autonomously navigate in unknown and unstructured environments. This article describes in detail our contribution to the competition: *Robbie*. The robot uses an active sensing approach, so the sensors are adjusted or configured before the data is readout – depending on the task or other sensor readings. A map is generated on the fly using the 2D distance measurements of a gimbaled laser range finder (LRF). The measured data is fused in an occupancy grid using a combination of scan matching and particle filter. The same LRF generates 3D scans for the obstacle detection. The result of the obstacle detection and also the current 2D laser scan are blended into the 2D map. This new map is the basis for the path planning, where we developed the so-called Exploration Transform which combines the frontier based exploration with the path transform. While navigating, the planned path is constantly checked against the updated map, and replanning is started as soon as an obstacle is detected that blocks the calculated way. The victim detection relies on a handcrafted thermal camera that captures images with a field of view of up to 180° (horizontally). All components are combined using a strictly message based software architecture. *Robbie* was used at the RoboCup (German Open and World Championship) in 2007 and 2008 and won the "Best in Class Autonomy" award in all four competitions.

Keywords: autonomous robots, USAR, SLAM, RoboCupRescue, active sensing

1 Introduction

To support first responders to find survivors in collapsed buildings (e.g. after an earth quake), rescue robots can be used to collect sensor data such as thermal and video images in areas that are inaccessible for humans. These robots are mostly remote-controlled [Casper and Murphy, 2003]. To relieve the operator from the exhausting task of controlling the robot, the availability of partially or fully autonomous robots is desirable. Those robots must be able to explore an unstructured environment systematically. Therefore, they have to generate a map of the building to keep track of areas

that were already inspected. These maps also help the first responders to find the victims that the robot has located. In the RoboCupRescue competition, such a disaster site is simulated using standardized test elements that simulate walls, uneven ground or rubble piles, building structures such as stairs and ramps [Jacoff et al., 2003]. The task for the autonomous or teleoperated rescue robots is to search for victims within this area while localizing itself and mapping the environment concurrently. This problem is widely known as the SLAM (Simultaneous Localization and Mapping) problem [Thrun, 1998, Gutmann and Konolige, 1999, Montiel et al., 1999]. In the competition, that team wins which finds the most victims and acquires the most accurate data about the victim and the building. Special awards are given for the best autonomous robot and the robot that performs best in terms of mobility at places which are most difficult of access.

2 Hardware Overview

The mobile platform of our robot *Robbie* consists of a Pioneer 3 AT (P3AT) by MobileRobots.[1] The P3AT is a wheel based robot with a four-wheel drive system. To control the robot, the C++ software library ARIA (version 2.4.1) provided by MobileRobots is used. The robot accepts basic commands for driving and steering. The robot's firmware calculates odometry data, which is provided to the PC every 100 ms. This position estimation of the robot can be used as a first guess for the real robot location. An image of the mobile system *Robbie* is shown in Fig. 1(a).

The robot is equipped with the low cost, lightweight laser range finder (LRF) Hokuyo URG-04LX. The field of view is 240° and the distances are reported from 20 mm to 5600 mm with 10 Hz. The scanner is very well suited for mobile robots because of its low power consumption and the built-in USB port.

The mobile robot is controlled by a NEXOC Osiris S620II onboard laptop with an Intel Core 2 Duo (T7800, 2.6 GHz) and 2 GB RAM. This computer collects the sensor data, does the mapping, the path planning, and the navigation. In addition, it sends the collected data and the generated map to an operator station which consists of Centrino duo (1.8 GHz) laptop running exactly the same software. Details of the software architecture are given in section 7.

2.1 Active Laser Range Finder Adjustment and 3D scanning

Rescue robots are usually used in unstructured environments with uneven floors and piles of rubble. Therefore the sensor readings might be spurious if the sensors are rigidly coupled to the robot. The influence of uneven floor is simulated since the RoboCup German Open 2007 using 10° and 15° pitch/roll ramps. These ramps influence the LRF data in three different ways:

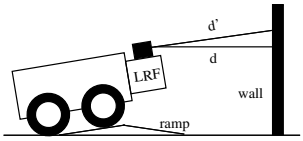
1. The measured length to an obstacle changes slightly by the factor of $(1/\cos\alpha)$ compared to the real distance, where α is the pitch angle of the robot relative to the flat ground.
2. The obstacle is not detected at all, since the sensor pointed above the object.

[1] `http://www.mobilerobots.com/`

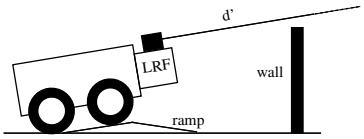
(a) Robbie found a victim

(b) Corrected 2-D LRF position

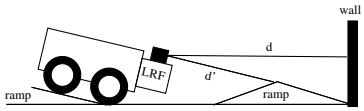
Figure 1: Robbie at the RoboCup 2007 in Atlanta (GA), USA



(a) LRF measures the wrong distance



(b) LRF misses the wall



(c) LRF finds a phantom wall

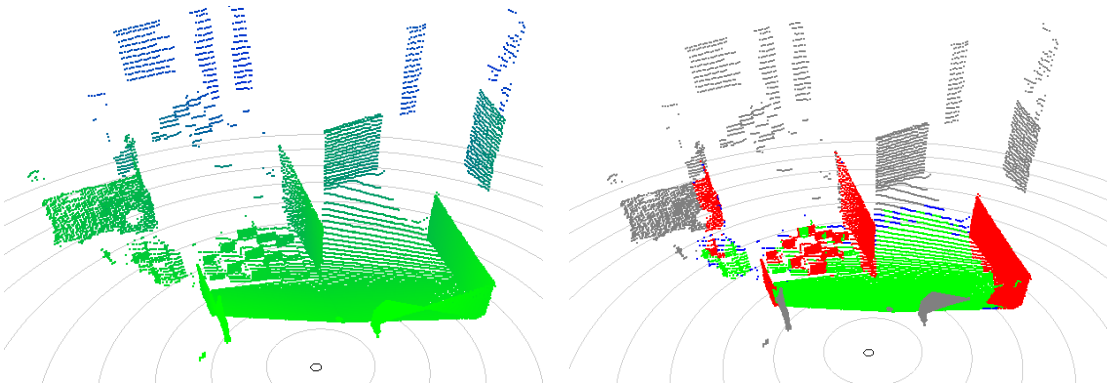
Figure 2: Correct and measured distances to walls due to ramps. (a) Slightly incorrect measurement (b) and (c) Incorrect measurements that cause serious problems for the localization and the mapping

3. "Phantom walls" are detected, since the sensor detected the next pitch ramp as an obstacle.

These problems are illustrated in Fig. 2. While (1) adds usually only a small error (about 1.5% at 10°), (2) and (3) result in inaccurate occupancy maps. In the last resort, case (3) can cause the robot to avoid an actually free path, because in the map its way is blocked by a phantom obstacle. An exploration of an unknown environment beyond this point would therefore fail.

One way to overcome these problems is to use a 3D LRF instead of a 2D LRF: The registration of the 3D scans would come up with a correct global map, and the exact pose of the robot can also be computed [Nüchter, 2006]. Anyway, taking a 3D scan and doing the registration of the scans with the global map in 6D space takes much more time than the scanning and matching in 2D.

Therefore, we adjust the orientation of the LRF based on the readings resulting from a low-cost gravity sensor [Pellenz, 2007]. The sensor measures the current orientation of the robot, and the position of the LRF is corrected so that it always scans horizontally. On our mobile system, we use a cheap dual-axis (2D), up to 2 g accelerometer [mis, 2000]. The chip generates two pulse width modulation (PWM) signals depending on the acceleration in $x-$ and $y-$direction. The lengths of the PWM signals are measured by a microcontroller and the times are sent to the laptop via USB. The PC converts the times into degrees and feeds them into a PID controller, which calculates the necessary correction angles.



(a) 3D scan of a stepfield

(b) Same scan; sectors classified as drivable or not

Figure 3: Example of a 3D scan of the laser range finder.

The correction information is sent back to the microcontroller which steers two servos that adjust the orientation of the LRF. The rotation/tilt unit mounted on the robot is shown in Fig. 1(b).

The same unit is also used to generate 3D scans. These local scans are used to classify the terrain around the robot. Fig. 3 gives an example of such a scan from the RoboCup 2008; it includes a stepfield on the left side. The 3D data is processed as follows:

1. Each point of the local scan is assigned to a grid cell (of size 100× 100 mm) on the x/y-plane (the x/y-plane is parallel to the ground).

2. For each grid cell c_i, the variance in z (the height), as well as the the lowest and the highest z value is computed ($\sigma^2_{z,i}$, $\min_{z,i}$ and $\max_{z,i}$). Also, the number of points in each cell is counted ($\#_i$).

3. A cell is assigned one of three classes, depending on the precalculated parameters:

$$\text{class}(c_i) = \begin{cases} \textit{unknown}, & \text{if } \#_i < \text{minCnt}, \\ \textit{occupied}, & \text{else if } (\sigma_i^2 > \text{maxVar}) \wedge (\max_{z,i} - \min_{z,i}) > \text{maxStep}, \\ \textit{free}, & \text{else.} \end{cases} \quad (1)$$

   The values for the thresholds were experimentally determined and are set to: minCnt = 10, maxVar = 18^2 and maxStep = 80 (all distances in mm).

4. Finally, using the current pose of the robot, the world coordinate of all cells of class *occupied* is calculated and stored in a 2D grid, the "inaccessible grid": inaccessible(localToWorld(c_i)) = *true*

So in addition to the occupancy grid in which the obstacles detected by the 2D LRF are stored (which will be described in section 3), the inaccessible grid keeps track of locations with obstacles detected by the 3D LRF.

3 Simultaneous Localization and Mapping (SLAM)

A 2D map is generated on the fly using the 2D distance measurements of the gimbaled laser range finder. The measured data is fused in an occupancy grid using a combination of scan matching and particle filter.

The 2D mapping uses the laser data in polar coordinates and the odometry data which is provided by the robot. The timestamp of the laser data is the reference time for the mapping step: The a–priori robot pose at that moment is interpolated by the odometry data reported right before and right after the LRF measurement. To improve the a–priori robot pose, the current laser scan is matched with the previous one (pairwise scan matching).

The data structure to store the map is a single occupancy map. In the context of RoboCupRescue, the grid usually has a size of 800 × 800 cells, which represents a map of an area of 40 × 40 meters with a grid cell size of 50 ×50 millimeters.[2] The grid is stored in two planes: One plane counts how often a cell was "seen" by a laser beam. This value is increased either if the laser beam reported the cell as free or if the cell was reported as occupied. A second plane stores the information how often a cell was seen as occupied. By dividing these two planes, the occupancy probability for a cell c_i is estimated by the following ratio:

$$p_{\mathrm{occ}}(c_i) = \frac{\mathrm{count}_{\mathrm{occ}}(\mathrm{c_i})}{\mathrm{count}_{\mathrm{seen}}(\mathrm{c_i})} \tag{2}$$

To solve the SLAM problem, a particle filter is used [Isard and Blake, 1998]. We use about 2000 particles; each particle represents a hypothesis for the pose of the robot: $(x, y, \Theta)^T$. The algorithm of the particle filter includes the following steps: *Resample*, *drift*, *measure* and *normalize* that we will describe in the following. The result is the most likely pose of the robot while the laser scan was taken. This pose is then used for the *map update*.

Resample: Depending on the weight of the particles (generated by the last measurement step), the probability distribution is adjusted. The resampling step is only done if the robot moved a certain distance (at least 20 mm) or turned (at least 5°) since the last mapping step. We use the common trick to go down the sorted list of particles (largest weight first) and to generate a new number of particles from the current particle, depending on its normalized weight. This way the resampling can be done very efficiently.

Drift: During the drift step, the particles are moved depending on the odometry data reported from the robot. It also models the noise, which is due to sensor inaccuracies and wheel slippage [Hähnel et al., 2003]. Three motion error sources are modelled:

- When the robot starts, the initial orientation might be wrong.
- The distance of the robot movement might be wrong.

[2]With noisy or distorted distance data, it is useful to increase the grid cell size, so that it is more likely that a measurement ends up in the same cell. We used a cell size of 100 × 100 millimeters to handle the pitch-/roll ramps without the gimbaled LRF during the RoboCup 2007 German Open in Hannover.

- The orientation of the robot when the move was finished might be wrong.

In our software, the initial rotation error is estimated with 5% of the reported rotation, the rotation error while driving with 5° per meter, and the translation error with 15% of the driven distance.

Measure: During the measurement step, new weights for each particle are calculated using the current laser scan and the already learned map. This step is extremely performance critical, since the current laser scan has to be matched against the occupancy map for each particle. For 2000 particles 600 successfull distance measurements contained in the laser scan, a naive approach would result in 1.200.000 coordinate calculations and random accesses to the map.

For this reason, a reduced set of measurement points is generated prior to the matching step. This is done by subsampling the laser scan such that two subsequent measurement points have a fixed minimum distance (150mm). Additionally, if the distance between two subsequent measurements in the original laser scan exceeds a fixed maximum (225mm), the two points are included as measurement points in any case. This way, the end points of closed segments within the laser scan are preserved. The advantage of preserving segment boundaries can be explained by considering a set of measurement points lying on a straight line (e.g. on a wall). If the two ends of the line are not used as measurement points, particles could shift to a certain amount along the direction of the line without affecting their weights. In our experiments, the use of a reduced set of measurement points while preserving segment boundaries did not cause a notable loss in mapping quality compared to using the full laser scan.

Our first implementation did not use the abovementioned optimizations, resulting in the measurement step consuming more than 95% of the CPU time used by the particle filter. The optimizations have speeded up the measurement routine by a factor of around 50. This way, much more particles can be used during real time mapping, resulting in a denser sampling of the pose space and thus increased accuracy and robustness.

Note that only measurements which contain a valid obstacle distance are considered in this step, ignoring parts of the laser scan for which the distance to the next obstacle exceeds the maximum range of the laser scanner or other errors are reported.

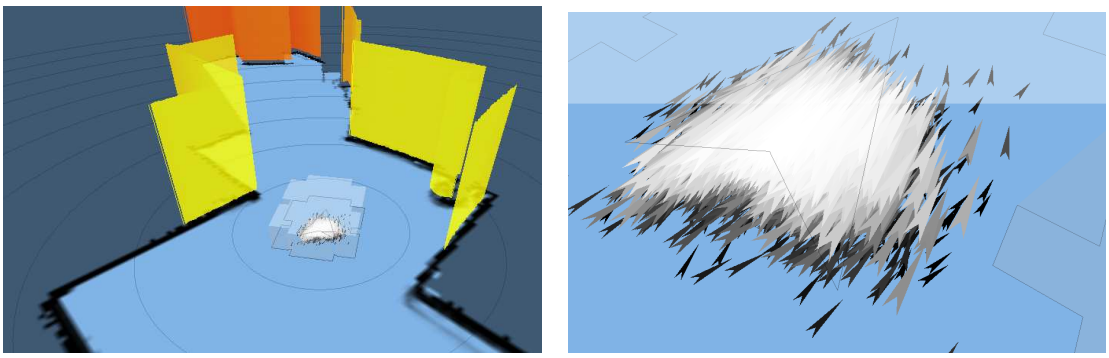
The measurement points are stored relative to the position of the robot. During the actual matching loop, a transformation matrix is calculated for each particle according to its pose hypothesis. The matrix is then used to efficiently transform the measurement points into map coordinates.

Let $c_{p,k}, k = 1, ..., K$ be the cells in the occupancy grid which correspond to the measurement points for particle p. The weight π_p for particle P is then computed as

$$\pi_p = \sum_{k=1}^{K} p_{\text{occ}}(c_{p,k}) \quad . \tag{3}$$

If the measurement points fall into cells with high occupancy probability (e. g. walls), the pose predicted by the particle is a good guess and therefore gains a high weight.

Normalize: The assigned weight of a particle is a sum of occupancy probability values. During the normalization step, the weight of each particle is divided by the sum of all particle weights.



(a) Particles (in the center), blended into the 3D view

(b) Detailed view of the particles.

Figure 4: Robot pose hypotheses, represented as particles. The better the particle represents the pose, the brighter the particle is shown. This view is available in realtime in the user interface.

Map update: The average pose of the top 5% particles with the largest weight is assumed to be the location of the robot. Using this pose, the current laser range scan is added to the global occupancy map by constructing a local map and "stamping" it into the global map, incrementing the counts for *seen* and *occupied* cells (cf. equation 2). Special attention is taken for cells that are touched by beams more than once during a single scan (these are cells that are close to the robot): If such a cell is seen as occupied, than other beams of this scan can not overwrite this cell as free any more. Fig. 4 illustrates the robot pose hypotheses, represented by particles.

In our implementation, the mapping algorithm runs in an extra thread. As soon as it has finished one mapping step, it waits for new laser range *and* new odometry data. Then, the next mapping step starts immediately. After each localization and mapping step, the estimated pose and the updated map is sent to the system core, so that the user interface and other modules can use this data.

4 Path Planning: The Exploration Transform 2.0

Beyond mapping the environment, autonomously exploring robots must decide where to go next to find out more about the environment [Gonzàles-Baños and Latombe, 2002]. This problem can be split into two questions:

1. *Select a target:* Where should the robot go next to extend the map or search for victims?
2. *Choose a path:* Which way should the robot take to reach this target?

By combining Yamauchi's frontier based exploration [Yamauchi, 1997] with Zelinsky's path transform [Zelinsky, 1988, Zelinsky, 1991] we achieved an elegant solution for the exploration problem, the so called *Exploration Transform* [Wirth and Pellenz, 2007]: The path transform is extended in a way that not the cost of a path to a *certain* target cell is calculated, but the cost of a path that goes to a close frontier. The path is not necessarily the shortest and the frontier not necessarily the closest, since the cost is determined by the distance *and* the safety of the path. The formula of the Exploration Transform is

(a) Short, but narrow path

(b) Longer, but safer path

Figure 5: Different paths chosen by the exploration transform (depending on the parameter α).

$$\Psi(c) = \min_{c_\mathrm{g} \in F} \left(\min_{C \in \chi_c^{c_\mathrm{g}}} \left(l(C) + \alpha \sum_{c_i \in C} c_\mathrm{danger}(c_i) \right) \right) \quad , \tag{4}$$

with c the start cell of the exploration, F the set of all frontier cells, $\chi_c^{c_\mathrm{g}}$ the set of all paths from c to c_g, $l(C)$ the length of the path C, $c_\mathrm{danger}(c_i)$ the cost function for the "discomfort" of entering cell c_i (e. g. because it is close to an obstacle), and α a weighting factor ≥ 0.

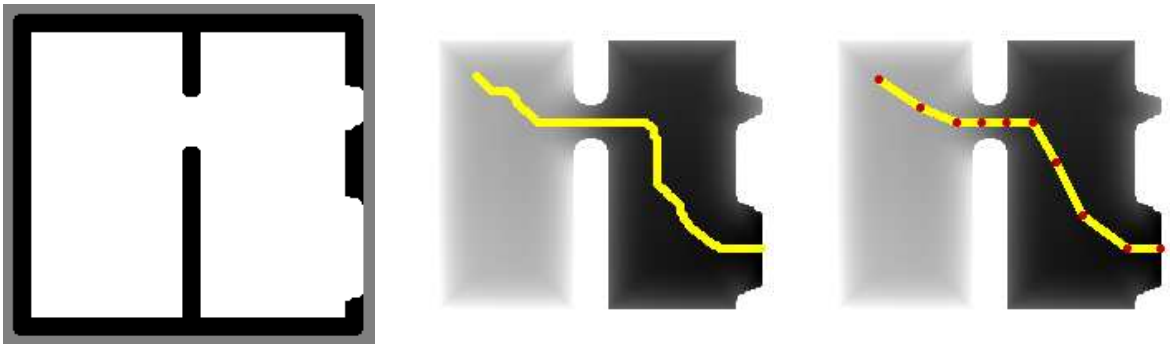
The Exploration Transform has the favorable property that by construction no local minima can occur:

1. The cost of all cells on a frontier are set to zero.
2. All other cells are assigned the cost of the neighbor cell with the lowest cost *plus* the Euclidian distance to this neighbor cell (1 or $\sqrt{2}$) *plus* the own discomfort cost is added.
3. Step 2 is repeated until the cost remain unchanged.

Using the calculated grid, from each cell a path to a close frontier can directly be extracted by descending this summed cost function.

Compared to the path transform, the Exploration Transform performs a search over all possible frontier cells. In the case that the set F contains only one element c_g, the Exploration Transform is identical to the path transform to the target cell c_g. Fig. 5 shows the influence of the weighting factor α: it determines how much a safer path is preferred over a shorter one.

A risk function that forces the robot to stay away as far as possible from obstacles bears the risk that the sensors (with their limited range) cannot see any landmarks. This should be avoided, because the robot needs landmarks to localize themselves and to extend the map. So in a first version of the Exploration Transform, the robot followed a "coastal navigation approach" [Roy et al., 1999] and tried to stay in a certain distance to the obstacles, typically 700 mm. However, this added a strong constraint that resulted in paths that are not optimal: The robot sticked to walls, even though a better way could have been used that was as safe as the chosen path. Therefore, the cost function was extended in a way that (a) the robot is encouraged to stay away from obstacles at a distance *range* of d_SafeMin to d_SafeMax and (b) never gets closer than $d_\mathrm{AllowedMin}$. This new discomfort cost c_danger of a cell c with distance d



(a) Floor plan (b) All adjacent free cells on the path (c) Optimized path

Figure 6: Path optimization: (a) shows the original floor plan, (b) the extracted path from the top left to the lower right, consisting of 212 free cells. In (c), the path is reduced to only 10 waypoints.

to the next obstacle is calculated as follows:

$$c_{\text{danger}}(c) = \begin{cases} \infty, & \text{if } d < d_{\text{AllowedMin}} \\ 0, & \text{else if } d \in [d_{\text{SafeMin}}, d_{\text{SafeMax}}] \\ (d - d_{\text{SafeMax}})^2, & \text{else if } d > d_{\text{SafeMax}} \\ (d_{\text{SafeMin}} - d)^2, & \text{else if } d < d_{\text{SafeMin}} \end{cases} \tag{5}$$

Using this function, there is no extra cost as long as the robot stays in a certain range to the closest landmark. A fixed distance is not enforced any more. The effect is that the chosen path is much more natural and leads to the target more directly. Still, it is guaranteed that a landmark is in sight. In a last step, the path is optimized by reducing the points that the robot navigates to. The following steps describe the path optimization:

1. Input: The complete list of waypoint cells $C_{all} = \{c_0, \ldots, c_m\}$, the obstacle map $O(c)$
2. Init: $C_{opt} = \{c_0\}$; $c_{\text{lastAdded}} = c_0$; $i = 1$.
3. dist $= ||c_{\text{lastAdded}} - c_i||$
4. if (dist $\geq \gamma\, O(c_{\text{lastAdded}})) \vee$ (dist $\geq \gamma\, O(c_{\text{i}})$) :
   $C_{opt} = C_{opt} \cup \{c_i\}$; $c_{\text{lastAdded}} = c_i$
5. i++
6. if $(i < m)$ go to step 3.
7. $C_{opt} = C_{opt} \cup \{c_m\}$

where C_{all} contains all cells from the start cell to the goal cell, the obstacle map $O(c)$ reports the distance to the closest obstacle for the cell c, and γ determines a safety factor usually set to $\gamma = 0.9$.

Using this optimization, in sections of the path with nearby obstacles more points are sampled than in sections that are far away from walls. Fig. 6 shows an example of the path optimization: a path that consists of the original 212 cells (Fig. 6(b)) and the reduced path of only 10 waypoints (Fig. 6(c)).

5 Navigation

For driving along the calculated path, the navigation module uses the localization data calculated by the SLAM module to correct the direction while moving. However, the SLAM position data is available with an update rate of about 2 Hz. Between these SLAM position messages, robot pose updates based on laser scan matching and on the odometry data are computed (which is done with about 10 Hz) and also used to correct the steering. This way it is possible to drive the robot continuously.

Dynamic obstacles are handled by adding them temporarily into the map: In addition to the cells with an occupation probability of more than 50% and to the cells that are marked as inaccessible by the 3D LRF, all cells that are seen as blocked in the *current* laserscan are marked as occupied in a temporary map that is used for the path planning (and path checking). Note that this does *not* influence the the actual occupancy grid, but only the temporary map.

The path checking is performed by verifying that no obstacles are present in the neighborhood of each map pixel of the calculated path. As this check can be performed very fast, it is done each time an updated map is generated (at about 2 Hz). The path planning, which is a more computing-intensive operation, is only done initially when a new target location is set and each time the path check fails. This way, the robot can dynamically react to obstacles appearing in its way (such as a moving person). While the path checking is done in realtime while driving, the path replanning – which takes usually about 500 ms and involves the full path planing described in the previous section – is done while the robot is standing.
Using the techniques described above, *Robbie* is now able to handle unknown, unstructured and uneven terrain, obstacles that are not visible in the 2D scan, and dynamic obstacles.

6 Victim Detection with a lowcost thermal camera

Robbie uses two thermal sensors (TPA 81) to detect victims autonomously by their body heat [Pellenz, 2007]. Two servos rotate the two 8-pixel sensors (see Fig. 7). The field of view of the thermal camera is selectable and depends on the task: For the initial search for victims, the field of view (FOV) is 180° and for the victim verification 60° only (horizontally). In both cases the vertical FOV is about 82°. A microcontroller steers the servos and collects the measured data and sends the information to the PC, where a 2D thermal image is created. For the visualization in the GUI the image is colored depending on the temperature values (see Fig. 8(a) and Fig. 8(c)). The automatic victim detection work as follows:

1. Input: The acquired thermal image (with heat measurements in °C) and the current environment (room) temperature (also in °C).
2. Filter the thermal image with a small Gaussian filter (with $\sigma = 1$).
3. Apply a threshold of (room temperature + $\delta\, t$) on the thermal image.
4. Find the connected components of the remaining "warm" regions, that have a minimum size SegmentSize_{Min} pixels.

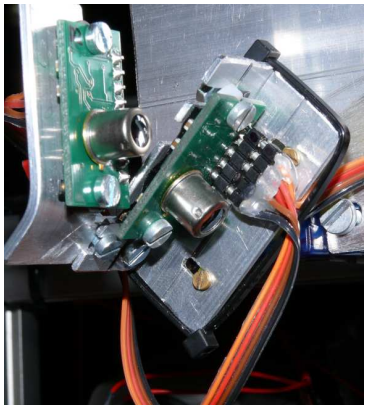
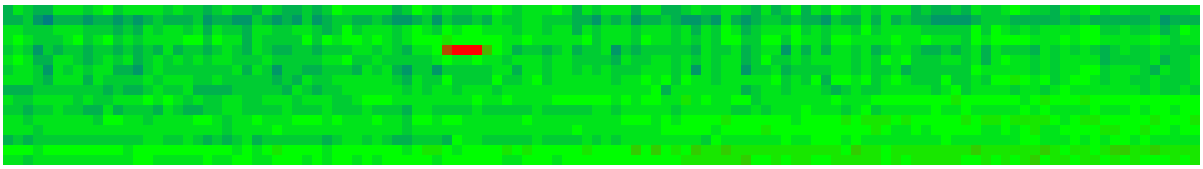
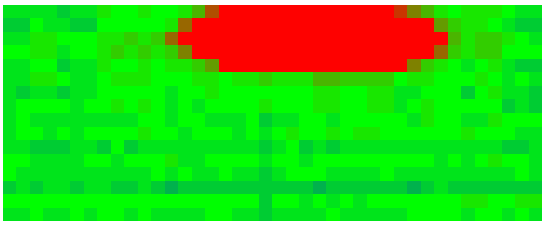


Figure 7: Self made thermal camera (built with two TPA 81 heat sensors)



(a) A thermal image that provides an overview of the scene (120 × 16 pixels; 180° hor./82° vert.)

(b) A color image taken from a Sony FireWire cameras



(c) A thermal image of the same scene (40 × 16 pixels; 60° hor./82° vert.)

Figure 8: Images of a simulated victims at the RoboCup 2008 in Suzhou (China)

The parameter δt is usually set to 5°C and SegmentSize_{Min} to 6 pixels in victim search stage and to 12 pixels in the victim verification stage. If a potential victim was found during the victim search stage, the robot estimates the position of the victim (using the current LRF scan), approaches the potential victim, and performs the victim verification by taking and analyzing another (smaller) thermal image.

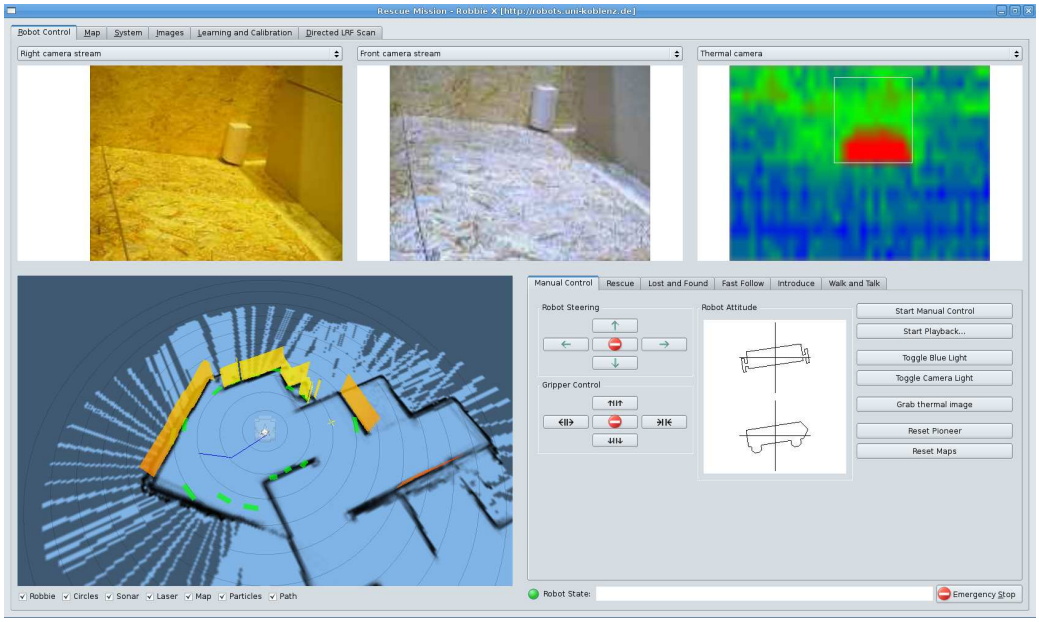


Figure 9: Monitoring software on the operator station of Robbie X. The current laser data is blended into the generated map (lower left widget).

7 Software Architecture

The software architecture of *Robbie* follows a loosly–coupled, strictly message–based software design: An application independent *System Core* forms the center of the system. So called *Modules* register themselves at this core, and subscribe to *Messages* that they are interested in. All types of data (sensor data, but also calculated position data and maps) are transported as messages via the system core. All messages have the capability to serialize and to deserialize themself. This way, the messages can be written to a log file or sent via a (wireless) network to an operator station that can be used to monitor and control the robot. A screenshot of the software running on the operator station is given in Fig. 9. Each Module works in its own thread and is connected to the core by two queues: The "outbound queue" of the module transports messages to the core, and the "inbound queue" transports the messages from the core to the module (Fig. 10). The idea of this architecture has its origin in the "Quasar windmill", but extends this concept by introducing a generic core [Siedersleben, 2002].
There are two types of Modules:

- *Passive Modules*: These modules sleep until a new message arrives. When a message arrives, the information is processed, possibly one or more messages are generated and sent to the core. Then, the module goes back to sleep. Passive Modules are used for processing data, e. g. filtering images or updating the map.

- *Active Modules*: These modules also wake up when a message arrives, but additionally become active periodically after a predefined period of time. Active Modules are used to request sensor data from external devices, e. g. to get the odometry data from the robot. The received data is

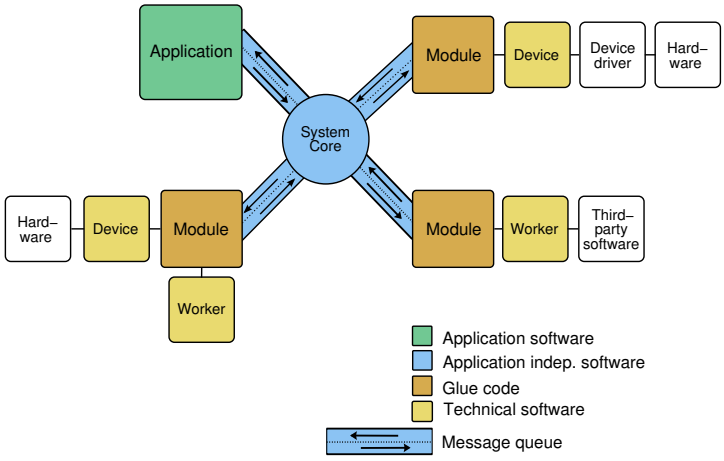


Figure 10: Overview of the Software architecture idea.

then wrapped into a message and sent to the core.

Other important components of the system are *Workers* and *Devices*.

- A *Worker* is a component that contains algorithms that fulfill a certain task for the application, e. g. the analysis of images for the victim detection. A Worker can use other (external) libraries (e. g. OpenCV for the image processing), but it does not know about Modules or Messages.

- A *Device* is a component that communicates with a certain piece of hardware, e. g. the self made thermal camera or a FireWire camera. A Device can use other (external) libraries (e. g. IEEE 1394 libraries to talk to the FireWire Cameras), but it also does not know about Modules or Messages.

A Module is a thin layer that connects Workers and Devices with the system core: It can interpret and create messages for the communication with the core, but it also knows how to talk to a particular Device or how to use the functionality of a particular Worker. The Modules implement so called "glue code" between these two worlds. Using this glue code, the Devices and Workers do not have to deal with the system specific objects (e. g. Messages), and are therefore very independent of the target (robotic) application. The use of the glue code helps also to isolate the generic system core from the application specific parts of the software.

The system has to deal with large data chunks, such as color images or 2D and 3D laser range data. Wrapping the data into messages would be very inefficient, especially if more than one receiver subscribed to the data. Therefore, only the pointers to the messages are sent to the Modules, and the core takes care of the memory management: After a Module is done with a Message, it signals the system core that the data is not used any more. When all the modules notified the system core that the message is handled, the data associated with the message is released and the message itself is deleted. Using a message based, event driven system for a robotic system has some remarkable advantages:

1. Clear interfaces, replacement of Modules and exchangeability of the robot task: Since all modules communicate via messages, it is very easy to understand what kind of data a module needs and

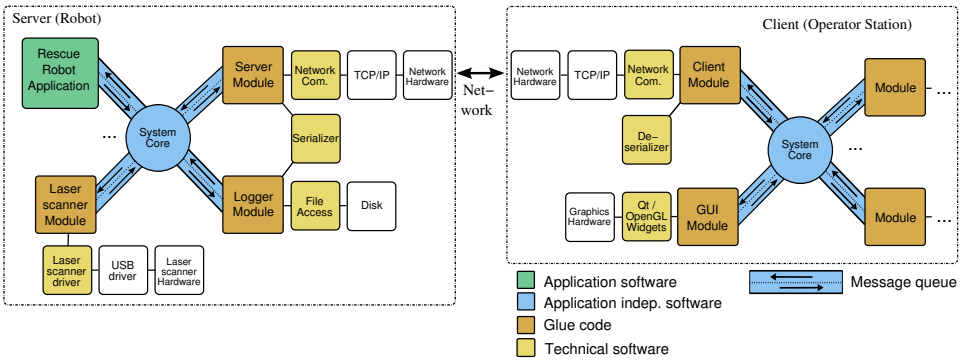


Figure 11: Example for a software configuration: Logging (on the server) and communication between two systems. Note that *exactly* the same software runs on both, the server and the client. The differ only in the configuration.

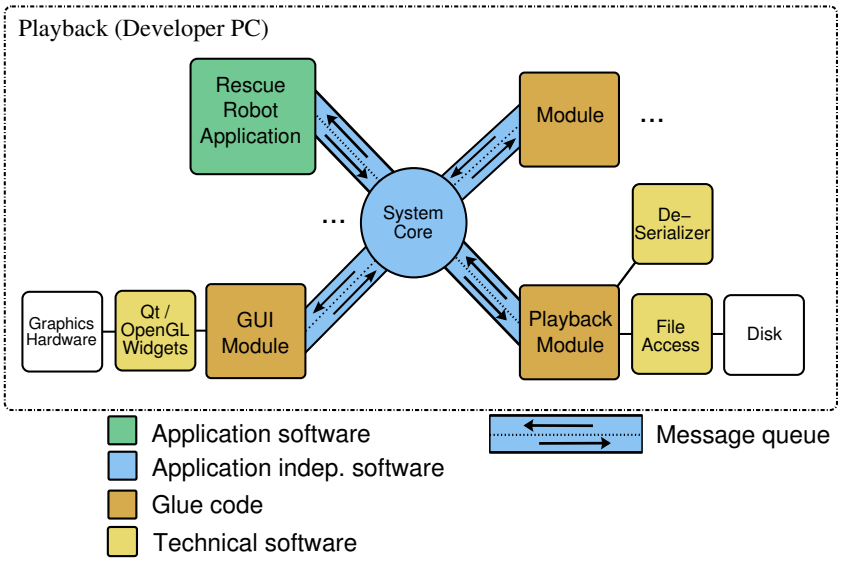


Figure 12: Example for a software configuration: Playback configuration. Again, exactly the same software is used to play back the logfiles. This enables to test the software with regression tests.

what data it generates. This makes improvements of the system easy. Improvements mean the addition of new modules (e. g. connecting a new sensor to the system), but also the removal of old, unused ("historic") code. Especially on a research platform, both forms of improvements are frequently applied. Modules can be temporarily replaced by other Modules without changing the behavior of the system, as long as the same messages are processed and produced. This way, the real sensors can easily be replaced by simulated devices. Also the application of the robot is implemented as a Module, so it is easy to change the task the robot has to fulfill by replacing this module. This way, *Robbie* was used as rescue robot *and* as a home robot in the RoboCup@Home league during RoboCup 2008.

2. Easy communication and data logging: The message objects have the ability to serialize them-

selves. These serialized messages can be sent through an arbitrary channel, like a TCP/IP socket which connects two cores. The connected second system instantiates the message by deserializing the data stream, and forwards the message to its own system core. This is depicted in Fig. 11. The serialized messages can also be written to disk. To replay the data, the stream has to be deserialized and the messages have to be sent to the core. Timestamps ensure that the messages can be replayed at the same speed. This is depicted in Fig. 12.

3. Parallel execution: Each module is executed in its own thread, therefore multiple processors or processor cores are automatically used. The application programmer (who for example adds a new sensor or writes new image handler) does not have to deal with the thread handling, the queue management or mutexes, because it is all inherited from the base class Module.

4. Simplified Active Sensing: Since the Modules that connect a sensor to the system core can also receive messages, they can use the information of other system components to adjust the sensor to improve its performance.

5. Testability of Devices and Workers: Since Devices and Workers are independent from the application (they don't know anything about Modules, Messages etc.) they can be easily tested in other testbeds such as unit tests.

6. Distributed computing: More than one instance of the software can run on different machines. Via the message exchange mechanism, a task can be assigned to another machine, and the result transferred back.

7. Simplicity: Even if from a user perspective a client and a server (here: an operator station that connects with a robot) is required, from a technical perspective exactly the same software is used on the client and the server. Only the configuration differs.

8. User interface on demand: For monitoring the robot's sensors and its behavior, an ergonomically designed GUI helps a lot to determine if the hard- and software is working correctly. On the other hand, this interface consumes system resources and is not always needed, especially on the robot while it is in a mission. Since the user interface in our architecture is just another Module, it can be instantiated when needed, and can be switched off if not required.

9. System Monitoring: The consistent architecture makes it easy to extract system parameters to monitor the system health. The number of messages that are waiting in the queues can be analyzed on the fly, and the processing time of the single threads. This overview of this monitoring data is also broadcasted (and logged) as a message. So remote monitoring and "post mortem" analysis of the system utilization can be done. A screenshot of the monitoring window is depicted in Fig. 13.

The software is developed on a Linux system (SuSE 10.3 and several Ubuntu versions) using the GNU C++ compiler `g++` (version 4.1.2). The user interface is implemented with Qt (version 4.2)[3] and

[3] `http://trolltech.com/products/qt`

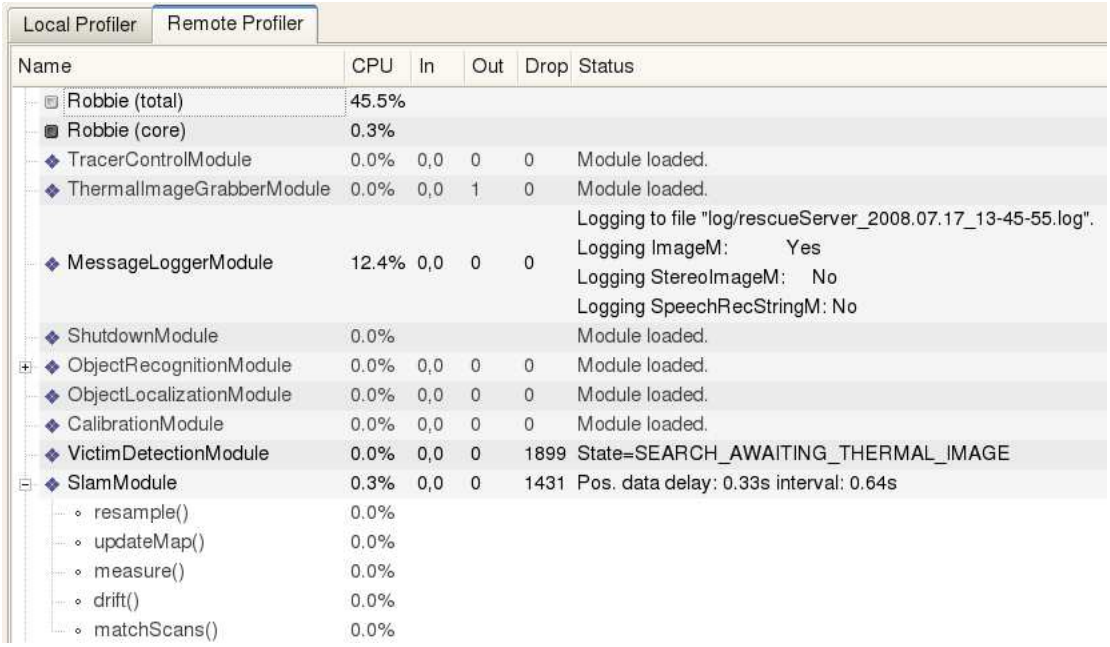


Figure 13: Monitoring window in the user interface: The CPU consumption of the application, the single threads and even from certain sections within the threads can be surveyed.

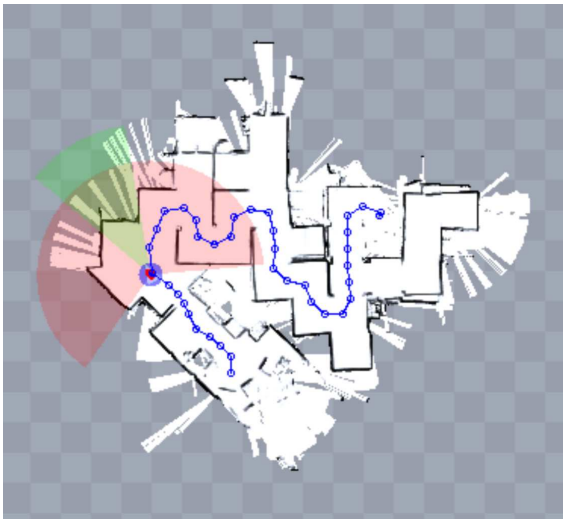
OpenGL. KDevelop is used as an integrated development environment (IDE).

8 Conclusions

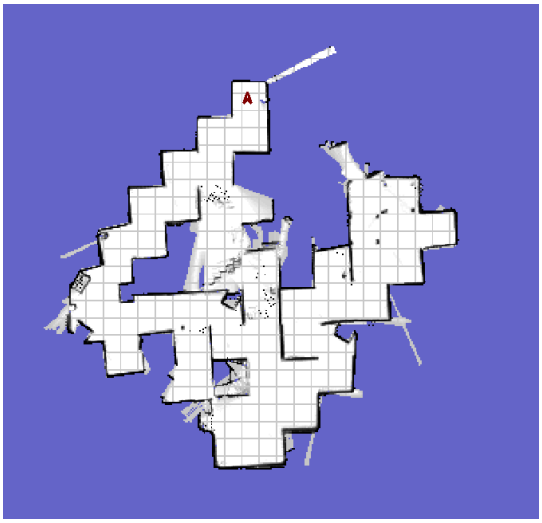
Our approach for the design of an autonomous rescue robot was presented. The robot consists of a wheel driven platform, a gimbaled 2D LRF that is also used as a 3D LRF, a self-made thermal camera, several color cameras and a laptop on the robot as well as an external laptop that serves as a remote control and monitoring station. It was shown that the gimbaled LRF enables a reliable 2D mapping, while the use of the same LRF as a 3D LRF enables the detection of low and overhead obstacles. Our obstacle detection method was presented, as well as our particle filter based SLAM technique. Based on the map that is generated by fusing the 2D occupancy grid, the obstacle information from the 3D scans and the current laser scan, the Exploration Transform 2.0 plans the path for the exploration. The thermal camera is used to search for victims and to verify a victim found. The complex system is organized by a message based, multi threaded software.

Using an unique Active Sensing approach to control the robot, the LRF, and the thermal sensor, *Robbie* demonstrates that the exploration of an unknown, unstructured and uneven terrain is possible without expensive and specialized 3D sensors. The Exploration Transform 2.0 guides the robot on a safe but still direct way through the environment. Dynamic obstacles are detected and included in the path planning.

To compare our approach with other solution, we used *Robbie* at the RoboCup (German Open and World Championship) for the participation in the RoboCupRescue league (in 2007 and 2008) *and*



(a) Map of the final of the RoboCup German Open 2008 in Hannover (Germany). Note that the map is not broken in the lower left part, but the environment really had this 45° orientation).



(b) Map of the final of the RoboCup 2008 in Suzhou (China)

Figure 14: Autonomously acquired maps during the finals at the RoboCup competitions 2008

the RoboCup@Home competition (in 2008). Maps generated during the fully autonomous exploration missions are shown in Fig. 14. The map 14(a) includes not only the RoboCupRescue arena, but also parts of the RoboCup@Home arena. The map shown in 14(b) was mapped within 7 minutes, while the second best team was able to map only half of the arena within 20 minutes.

Robbie won the "Best in class Autonomy" award in all four RoboCupRescue competitions.

References

[mis, 2000] (2000). Adxl202 low-cost 2 g dual-axis accelerometer with duty cycle output. Data sheet.

[Casper and Murphy, 2003] Casper, J. and Murphy, R. R. (2003). Human-robot interactions during the robot-assisted urban search and rescue response at the world trade center. *IEEE Transactions on Systems, Man, and Cybernetics, Part B*, 33(3):367–385.

[Gonzàles-Baños and Latombe, 2002] Gonzàles-Baños, H. H. and Latombe, J.-C. (2002). Navigation strategies for exploring indoor environments. *The International Journal of Robotics Research*.

[Gutmann and Konolige, 1999] Gutmann, J. and Konolige, K. (1999). Incremental mapping of large cyclic environments. In *Proceedings of the IEEE International Symposium on Computational Intelligence in Robotics and Automation (CIRA)*, pages 318–325, Monterey, California.

[Hähnel et al., 2003] Hähnel, D., Burgard, W., Fox, D., and Thrun, S. (2003). A highly efficient fastslam algorithm for generating cyclic maps of large-scale environments from raw laser range measurements. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*.

[Isard and Blake, 1998] Isard, M. and Blake, A. (1998). Condensation - conditional density propagation for visual tracking. *International Journal for Computer Vision*, 29(1):5–28.

[Jacoff et al., 2003] Jacoff, A., Messina, E., Weiss, B. A., Tadokoro, S., and Nakagawa, Y. (2003). Test arenas and performance metrics for urban search and rescue robots. In *Proceedings of the 2003 IEEE/RSJ International Conference on Intelligent Robotics and Systems*.

[Montiel et al., 1999] Montiel, J. M. M., Castellanos, J. A., Neira, J., and Tards, J. (1999). The spmap: A probabilistic framework for simultaneous localization and map building. *IEEE Transactions on Robotics and Automation*, 15(5):948–952.

[Nüchter, 2006] Nüchter, A. (2006). *Semantische dreidimensionale Karten für autonome mobile Roboter*. PhD thesis, Rheinische Friedrich-Wilhelms-Universität Bonn.

[Pellenz, 2007] Pellenz, J. (2007). Rescue robot sensor design: An active sensing approach. In *SRMED2007: Fourth International Workshop on Synthetic Simulation and Robotics to Mitigate Earthquake Disaster*, pages 33–37, Atlanta (USA).

[Roy et al., 1999] Roy, N., Burgard, W., Fox, D., and Thrun, S. (1999). Coastal navigation: Mobile robot navigation with uncertainty in dynamic environments. In *ICRA*, pages 35–40.

[Siedersleben, 2002] Siedersleben, J. (2002). Quasar: Die sd&m standardarchitektur. Technical report, sd&m Research.

[Thrun, 1998] Thrun, S. (1998). Learning metric-topological maps for indoor mobile robot navigation. *Artificial Intelligence*, 99(1):21–71.

[Wirth and Pellenz, 2007] Wirth, S. and Pellenz, J. (2007). Exploration transform: A stable exploring algorithm for robots in rescue environments. *Workshop on Safety, Security, and Rescue Robotics, http://sied.dis.uniroma1.it/ssrr07/*, pages 1–5.

[Yamauchi, 1997] Yamauchi, B. (1997). A frontier-based approach for autonomous exploration. *1997 IEEE International Symposium on Computational Intelligence in Robotics and Automation*, page 146 ff.

[Zelinsky, 1988] Zelinsky, A. (1988). Robot navigation with learning. *Australian Computer Journal*, 20(2):85–93.

[Zelinsky, 1991] Zelinsky, A. (1991). *Environment Exploration and Path Planning Algorithms for a Mobile Robot using Sonar*. PhD thesis, Wollongong University, Australia.