SOAVE 2004, Proc. 3rd Workshop on SelfOrganization of AdaptiVE Behaviour, Ilmenau, Germany, pp. 167-177, VDI Verlag Duesseldorf, ISBN 3-18-374310-8

Robust Map Learning with Low-Cost Sensors for an Autonomous Robot

Christof Schröter
Department of Neuroinformatics, Ilmenau Technical University
P.O. Box 100565, 98684 Ilmenau, Germany
christof.schroeter@tu-ilmenau.de

Abstract

The paper describes several approaches and experimental results for learning a map of an indoor environment, using a combination of odometry, sonar range sensors, and vision. The aim of the presented research work is consistent real-time map-learning in indoor-environments with low-cost sensors and limited computational resources and without installations in the environment itself. Therefore, we cannot afford high-complexity techniques like probabilistic SLAM. Instead, our approach involves a set of algorithms for odometry correction and automatic recalibration of position estimation. Results from different environments are presented that show that the methods we developed are appropriate for mapping large scale areas by means of low-cost sensors.

1 Introduction

Self-Localization and Map-Building are two of the key requirements to Autonomous Mobile Robotics. At any time, the robot must know where it is located, with respect to some reference frame, or at least must be able to aquire that information within a limited time by adequate behaviour.

The map constitutes the robot's awareness of its environment (or environment model). Therefore, it is preferable to learn the environment model from actual sensor readings instead of providing an external model. Furthermore, the map should contain those features that are relevant to the desired behaviour. The complexity and depth of the model directly imposes restrictions to the behaviour that can be derived from it and therefore to the tasks the robot is able to accomplish.

For a mobile robot, localization and mapping bear mutual interdependencies. With known pose, the map can be built by processing sensory inputs and integrating them into a common frame as the robot moves around its working area. Likewise, with a known map, it is possible to determine the robot pose by comparing actual sensor readings with expectations from the map. In the absence of accurate pose information as well as a model of the environment, both localization and mapping present complex tasks.

In this paper we will focus on the aim of learning a consistent geometric map from sonar range readings, using odometry (egomotion measurement) as main pose information and combining it with visual input and internal map alignment for increased accuracy. The remainder of the paper is structured as follows. In section 2 we give an overview of related research work in the field of pose

estimation. Section 3 outlines the navigation system of our experimental service robot PERSES and further defines the role of the work presented here. Section 4 illustrates in detail the steps taken to achieve consistent pose estimation, presenting three different methods for increasing the accuracy of odometry measurements and for correcting them on-line by evaluating visual input and internal models to determine pose estimation errors. Section 5 will present experimental results achieved with these methods in different environments. We close with an evaluation of the presented methods and results and our conclusions in Section 6.

2 Related Work

Usually, a mobile robot is equipped with wheel encoders, enabling it to measure its own motion and determining the current position by dead-reckoning, i.e. integrating incremental motion information over time. Odometry is widely used because it provides high sampling rates and good short-term accuracy while requiring relatively little effort. However, it inevitably leads to accumulation of systematic and non-systematic measurement errors, resulting in a growing difference between believed and real position. Therefore, uncorrected odometry alone is not a reliable method for tracking a mobile robot's pose and building consistent large-scale maps.

Borenstein et al. described calibration for a robots odometry system in [1], [2]. Their approach concentrated on identifying possible error sources for a differential drive robot and determining parameters to explicitly correct those errors. Martinelli [8] presented a strategy for experimentally determining the systematic and non-systematic errors in a synchro-drive robot, estimating error parameters from several observables.

Advanced localization techniques have been an area of intensive research for a long time and various different approaches have been developed. In scan matching, overlapping sensor readings at different locations are compared and matched to correct the odometry error aquired while moving from one position to the next one [6]. Scan matching obviously depends on high recognizability and therefore on high resolution and accuracy of the sensor for good matching results. Hence, usually laser scanners are utilized for this task.

As aforementioned, another approach to localization is using a given map to determine the current position relative to known features by comparison of sensor readings. Although in this paper, we are concentrating on a situation where no prior map is available, we give an overview of such algorithms for completeness and as introduction to section 3.

A simple method is to provide positions of easily recognizable landmarks that can be detected by either range sensors or visual input. Localization depends on observation of one or more landmarks, while between observations the position is maintained by odometry. Newer approaches estimate probability distributions for the robot state that are updated with motion and observations, using probabilistic motion and observation models. Kalman filters [9], grid-based Markov methods [4] and particle filters (Monte-Carlo-Localization, MCL) [3] have been used for probability estimation.

The chicken-and-egg problem of mapping and localization is adressed by a method called Simultaneous Localization and Mapping (SLAM). In SLAM, mapping and localization are not seen as separate tasks, but solved parallely observing and exploiting the mutual dependencies. The base of SLAM is an extended state space that contains the position as well as the map. A probability distribution over this common state space is maintained that converges with motion and observations [12], [10]. Most successful applications are using laser scanners, while visual SLAM and adaption to arbitrarily large environments are subjects of ongoing research.

3 PERSES Navigation Overview

In our PERSES project we are developing an autonomous service robot that in its first application will work as a shopping assistant in a home store environment, offering guidance and companion services to customers. Sensor and computational hardware equipment will be limited to allow for a low-cost robot, restricting us to visual and sonar range input.

For pathplanning, it contains a 2-dimensional grid map describing positions of objects and free space. The grid size is 0.2m * 0.2m while the overall area spans about 100m * 50m. Self-localization is done by Monte-Carlo-Localization using visual input from an omni-directional camera in combination with sonar range data [5]. Therefore, another map is needed containing features extracted from omni-camera images.

Both maps are built simultaneously in a "learning phase". During this phase, the robot is manually steered around the whole area, collecting sensor readings and integrating them into the map. The aim was to develop a pragmatic solution to consistent map building that is computationally efficient and does not need expensive sensoric equipment. Therefore, we use odometry for pose estimation, taking appropriate steps to increase accuracy as well as ensure long-term stability of the believed position.

With known position, a local grid map is built from sonar range readings and the global map is updated with this local map. Multiple observations of each grid cell from different observer positions at different times lead to a converging occupancy value, describing whether there is an object in this cell or if it contains free space [11]. At the same time, omni-camera images are taken along the path of the robot, processed and incorporated into a view-graph that preserves geometric relations of the view positions [5]. The methods described in this paper all refer to the learning phase where no prior map is available yet, but must be built from odometry.

4 Odometry Correction

4.1 Odometry Calibration

Currently, our experimental platform is a RWI B21r. The B21r is driven by 4 synchronously steered wheels (Synchro-Drive). Translation and rotation are measured independently by incremental encoders.

Our observations show that while odometry information is highly accurate concerning the driven distance, deviations in the believed orientation and motion direction can become quite high. Further experiments show that the robot is not driving a straight line when it should do, but instead is doing circular arcs. A possible explanation for this behaviour would be different wear on the wheels. In contrast to differential drive, in a synchro-drive the positions of the wheels with respect to the robot heading change with each rotation, altering their influence on the driving behaviour. Therefore, determining the systematic errors takes more effort for the synchro-drive than for the differential drive.

On the other hand, the experimental data suggests the error can really be described with a plain model mapping drive direction to arc radius, leading to a simpler algorithm than [8].

Fig. 1 (left) shows what happens when the robot should go straight ahead, but the wheels cover different distances. The actual driven path is a circular arc, but the wheel configuration with respect to the driving direction does not change, leading the robot to believe it has gone straight ahead. The measured distance d is (almost) exactly the length of the arc. During this motion, the robot's heading direction with respect to an external reference frame has changed by angle α.

If the radius of the resulting arc is known, we can compute the destination position from the driven distance d, by assuming a straight motion in direction ϕ' with distance d' (see Fig. 1 (center)).

The angle between the two legs of length r is the same as α because the radius is perpendicular

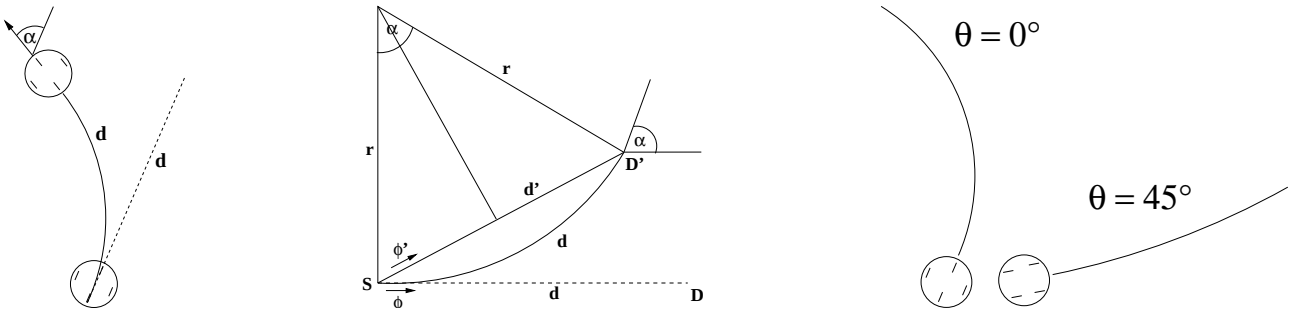

Figure 1. left: Illustration of circular arc motion instead of straight line. center: The circular motion can be calculated by assuming straight line motion with substitution direction and distance ϕ' and d'. right: Arc radius depends on motion direction with respect to the wheelbase

to the circle in any point by definition, therefore one leg is perpendicular to the tangent in starting point S (start orientation), the other is perpendicular to the tangent in destination point D (destination orientation). We can compute α using the circumference formula:

$$\frac{d}{2\pi r} = \frac{\alpha}{2\pi} \tag{1}$$

$$\alpha = \frac{d}{r} \tag{2}$$

In the next step, we compute the distance d' between S and D using relations in the bisected triangle:

$$sin(\frac{\alpha}{2}) = \frac{\frac{d'}{2}}{r} \tag{3}$$

$$d' = 2r * sin(\frac{\alpha}{2}) \tag{4}$$

We can also compute ϕ' by

$$\phi' = \phi + \frac{\alpha}{2} \tag{5}$$

which becomes obvious if we move the connection d' to a tangent to the circular arc.

Substituting d by d' and ϕ by ϕ' we can do a regular position update now. However, we have to do an additional correction for the robot heading, as the drive direction no longer is the same as the heading direction.

Note that without the described error behaviour, the orientation of the wheel base with respect to the external reference frame never changes during regular robot operation and therefore we only need one orientation parameter to describe the robot heading. However, now we need 2 parameters, one describing orientation in the external frame, which is the actual direction the robot is facing and moving, and another parameter describing the direction with respect to the wheel base (also called torsion), which will affect the wheels influence on motion behaviour and will be used in odometry correction.

In order to apply the correction algorithms, we need to know the radius of the arc the robot is driving. Fig. 1 (right) illustrates that we should expect the radius to depend on the drive direction *with respect to the wheel base* (the torsion), as the varying wheel configuration determines the drive behaviour. That means we need not only measure the radius once, but need to find a function mapping

drive direction to arc radius. Our first approach here was to sample the radius at a number of directions and interpolate between those sample points.

The ideal method of measuring the radius would be to have the robot move in a full circle and measure the diameter of course. Unfortunately, as the actually occuring radii are up to 2000m, this is not feasible in the absence of a very large perfectly even area. Instead, we can only do fragments of full circles and try to measure observables that enable us to compute the arc radius. We propose 2 methods here. The first one is illustrated in Fig. 2 (left). The robot is commanded to drive a straight line and the motion in x and y direction is measured (ideally it should only move in x direction). Without loss of generality we can assign coordinates (0,0) to the start point and (dx, dy) to the end point, which places the center of the circle at (0, r). Inserting the end point into the circle equation we get

$$(dx - 0)^2 + (dy - r)^2 = r^2 \tag{6}$$

$$r = \frac{dx^2 + dy^2}{2dy} \tag{7}$$

This method only requires the measurement of 2 ranges, but also depends on very exact alignment of the robot and measurement, as we must ensure the measurement coordinate system matches the inital heading direction of the robot.

The second method needs no alignment at all, but requires the measurement of angles. Using equation 2, we can calculate r by measuring the difference between initial and final heading along the arc path.

$$r = \frac{d}{\alpha} \tag{8}$$

For direct angle measurement, the resolution that can be achieved with simple methods is lower than for measuring the ranges as in the first method. However, with increasing distance d the resulting error in the radius r is decreasing. Therefore, the second method should be preferred if a reasonably large area is available for experiments.

Figure 2 (center, right) shows measurement samples taken with method 1 and method 2. Note that we have plotted the inverse 1/r as a function of the heading here. In both plots it is obvious that the function approximately matches a sine. This supports the hypothesis that the errors stem from different conditions of the wheels, as those should become more or less significant with the wheels taking a place near the center of the robot or closer to the edge (observing drive direction), following a sine curve as well. An additional best match sine curve is displayed in both plots.

It is also visible that both methods yield comparable results both in the mean as well as the variance of measurements. We must be aware that the results contain impreciseness of measurements as well as non-systematic errors in the robot motion itself. Perfect deterministic specification of the robot motion is illusory of course even with very good error measurement and compensation.

4.2 Odometry Correction with External Visual Reference

As already stated in 4.1, distance measurements of the odometry system are quite exact while orientation estimation may differ from actual orientation. Due to the incremental nature of odometry, correct position estimation strongly depends on true orientation and even slight errors may cause the mapping to fail after a short time without a means of correction. Looking for a source of reference, we found that the floor pattern in our experimental area contains easily recognizable features that provide (partial) orientation information.

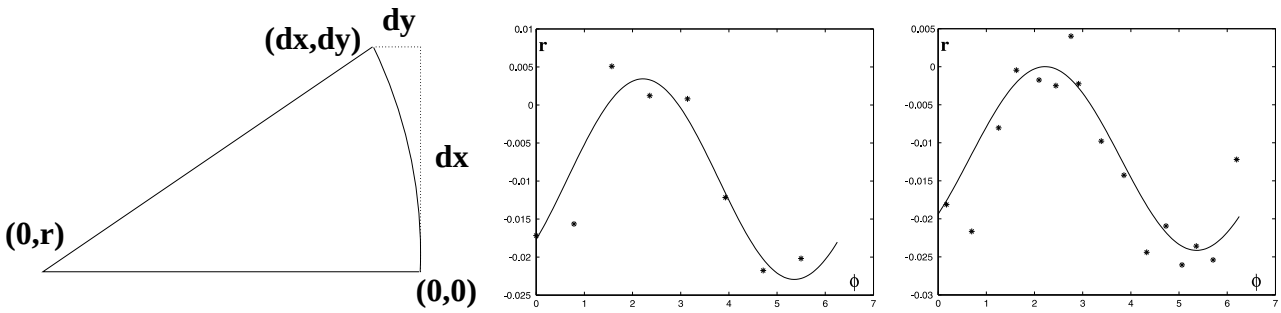

Figure 2. left: Determination of r by measuring dy and dy. center/right: 1/r as function of the heading direction, measured with method 1 and 2, respectively.

The floor consists of 30cm * 30cm tiles with easily recognizable darker joints between them, spanning the whole area with constant orientation. The idea here is to detect the orientation of these floor joints and use it as an external reference to correct odometry.

The image taken by a camera looking at the floor in front of the robot is slightly distorted radially. To correct this distortion we apply the correction algorithm

$$r_{new} = r_{old} \cdot (1 + r_{old}^2 \cdot k) \tag{9}$$

$$r = \sqrt{(x - x_{center})^2 + (y - y_{center})^2} \tag{10}$$

to each pixel (x,y), moving it away from the image center, with an empirically determined radial correction factor $k = 2 * 10^{-6}$. This is a very simple approach to correcting camera abberation, but it is sufficient for our problem. In the corrected image the lines are running straight and parallelly.

Now we calculate a local orientation for each pixel by applying an orientation tensor (inertia tensor method) [7]. The result of the orientation tensor are fields of angle and power of local orientation for each point of the image. As to be seen in Fig. 3 (top right, bottom left), at the edges of the tiles there are strong local orientations with angles aligned to the direction of these edges while on the surface of the tiles the orientations are unaligned but with low power. In a histogram of local orientations, weighted with respective power, the maximum shows the actual orientation of the robot with respect to the main orientation of the floor. Of course, the grid only enables us to determine the orientation within the interval $0°$ to $90°$, but this is sufficient if correction is done fast enough to avoid the robot turning more than $90°$ between two update steps.

To avoid objects covering the floor (e.g. obstacles) disturbing the histogram too much, the image is split into 4 sectors (upper left, upper right, lower left, lower right) and histograms evaluated indepedently for each sector. If one of the histograms differs too much from the others, it is discarded. The remaining sectors are integrated into a common histogram, the histogram maximum shows the main orientation of the image, at the same time being the actual robot orientation.

Given the true orientation of the robot, we can now correct odometry readings, using that external reference. The correction algorithm is similar to the one presented in 4.1 in handling of differences between believed and real orientation. However, since image processing is done on a higher level in the architecture than odometry processing, the resulting position is not written back to the odometry system, but a position in an independent coordinate system is maintained instead. Whenever a change between believed and sensed orientation occurs, the correspondance between odometry and position coordinate system is adapted. The independent reference system has the additional advantage that

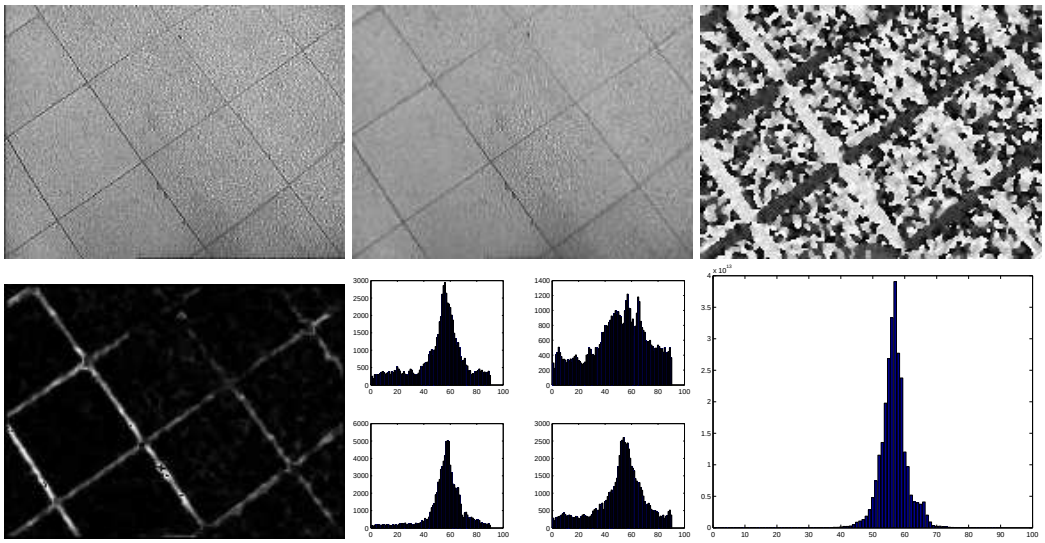

Figure 3. Top row from left to right: initial image as captured from the camera, image after radial correction, angle of local orientations (gray values coding orientation angles in a range of -90° to 90°), bottom row: power of local orientations (smoothed by box filter), 4 sector histograms, overall histogram showing strong peak at main orientation

initialization to an arbitrary position is possible without interference with the odometry system, e.g. to initialize the pose when a map is loaded.

4.3 Odometry Correction by Map Matching

With the correction methods presented in 4.1 and 4.2 we are able to improve pose estimation performance significantly. However, since we only use an external reference for a part of the robot state, the main problem of odometry is still present, although on a lower scale: the position error can still grow unbounded in the long run. To prevent errors in the map growing too large, up to now, we need to manually reset the position in certain intervalls. To this purpose, we mark a number of calibration points with known absolute position and return to one of these points frequently during mapping. Of course this is inconvenient as well as it prevents automatic map building by an appropriate exploration strategy.

Therefore, an automatic recalibration scheme is needed. The idea here is that in case of a position error, whenever the robot enters an area it has mapped before, the wrong believed position leads to a displacement between the map and the environment as it is perceived by the robot's sensors. The idea is somewhat similar to scan matching, with the difference that we do not match consecutive measurements. Furthermore, since we are working with unreliable sonar range sensors, single scans do not seem very promising.

Instead, it seems reasonable to match maps against each other. The purpose of the map building algorithm is to compensate for erroneous measurements, making the resulting map much more reliable than single scans. To enable matching, we need to discriminate between a temporary map, containing only recent measurements within a certain travel distance window and a static map, containing all older measurements that are not part of the temporary map. This is implemented by using a queue where all measurements are stored together with the respective odometry position. The scans leave the queue and are integrated into the static map when the robot has moved away from that position by a certain distance. The temporary map is rebuilt each time from all the scans that are still in the

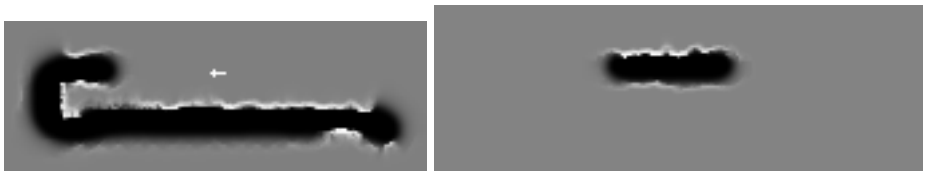

Figure 4. left: static map, right: temporary map (black is free space, white are obstacles, grey is unknown)

queue.

As the robot moves along it's path, the temporary map will run ahead of the static map, always covering the most recently crossed area. When the robot enters an area it has seen before, the static map will already contain that area. Only in that case there is an overlap between the static and temporary map. If the robot has aquired an error in position belief since the area was mapped last time, it will now assume a wrong position with reference to the static map, which leads to a displacement between the static and temporary map. By determining this displacement, we can calculate and then correct the position error.

In order to find the best match between the static and temporary map, we use a simple cross-correlation algorithm. Since errors can occur as shift in x- or y-direction or as rotation, the temporary map needs to be be shifted and rotated by different values within certain bounds and a similarity measure has to be computed each time. The scalar product is used for that purpose here. The result is a 3-dimensional matrix containing a fit value for each combination of x-shift, y-shift and rotation $(dx, dy, d\phi)$.

$$C(dx, dy, d\phi) = \sum_{x,y} static(x, y) \cdot temporary(x', y') \tag{11}$$

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} cos(d\phi) & sin(d\phi) \\ -sin(d\phi) & cos(d\phi) \end{bmatrix} \cdot \begin{bmatrix} x - x_0 \\ y - y_0 \end{bmatrix} + \begin{bmatrix} dx + x_0 \\ dy + y_0 \end{bmatrix} \tag{12}$$

One problem with the scalar product is that for "flat" matrices it produces wide maxima instead of narrow peaks. Since the maps usually consist of wide areas of free space (nearly same value), the results are very unspecific. If the matrices to match were merely shifted and rotated copies of each other, we would still get a distinctive peak for the true match. The mapping procedure may lead to slight differences between the maps though, especially since the temporary map does not contain very many scans. Overall, this often leads to bad matches especially in the rotation component.

Therefore it is preferrable not to use the map for matching as it is, but first extract certain features, avoiding areas of equal value. We chose the edges between obstacles and free space as features here (see Fig. 4). With the resulting sparse matrices, maxima are defined better in the correlation matrix.

Another problem is introduced by the typical structure of indoor environments. These mostly consist of long corridors with parallel walls. When trying to determine the best match between the maps, the correlation is very badly defined in the direction of the corridor. This may lead to the correlation maximum not reflecting the real position error. In such cases, the correlation matrix usually shows a wide maximum that stretches in one direction but is very narrow in the perpendicular direction. If it is possible to identify such situations we can apply correction only in the direction where the offset is well known (usually across the corridor) while delaying correction in the other direction until we get a more reliable estimation. The distribution in the correlation matrix can be described by calculating its eigenvectors and corresponding eigenvalues. Here we first look for the best rotation match, then calculate eigenvectors for the remaining 2d matrix (Fig. 5). The first

eigenvector points at the direction of main distribution of matrix elements, the eigenvalue represents the length of this distribution. If the eigenvalue is very high, the correlation is badly defined. In that case, correction is only applied in the direction of the second eigenvector.

With the map gridsize known, the correlation can directly be converted into a position offset that is used for correction of the position belief. Since the scans in the queue are stored with the erroneous position belief, they need to be corrected too, with a correction factor decreasing over the distance to the current position.

Figure 5. Extracted features in the static (left) and temporary map (center), correlation matrix with mean and eigenvectors (right)

5 Results

Fig. 6 shows corrected learned maps for all correction approaches presented in this paper.

In the left example the odometry calibration is illustrated. In fact, here the robot traversed a corridor from one end to the other, then went back to the starting point. Due to massive odometry errors, it seems like there are 2 different corridors. It is also evident that the robot believes to drive an arc when it actually goes straight ahead. Obviously, applying correction the accuracy of position belief is improved greatly. However, a noticeable offset still remains, increasing the width of the corridor near the end of the path and making the adjacent room appear twice. Overall, the position error grows slower, but still unbounded. Without further correction odometry precision is not sufficient for learning of large maps.

The second example shows mapping using the visual orientation reference, additionally to odometry calibration. In the top map only the calibrated odometry was used. The map seems deformed, the 2 corridors are not parallel as they should be. In contrast to that, the structure of the bottom map, where the visual reference was used, is correct. Experiments show, that the remaining error is about 0.2m for a driven distance of 50m (lower than 0.5%)

The right example demonstrates the effect of the map matching procedure described in section 4.3. Here the corridor from the first example is used again, but this time the robot went from one end to the other 2 times. During this time, it aquired a serious error in heading orientation, leading to wrong position belief and map as is clearly visible in the top image. Note that this experiment was conducted in an area with no distince floor structure, so the the visual reference was not available. Nevertheless, with the correction method presented in 4.3, we are still able to maintain correct position belief and build a consistent map.

In Figure 7 we show results of a combination of all 3 methods for mapping a part of our experimental area, a homestore. The increase in accuracy of position belief and the resulting map is demonstrated here.

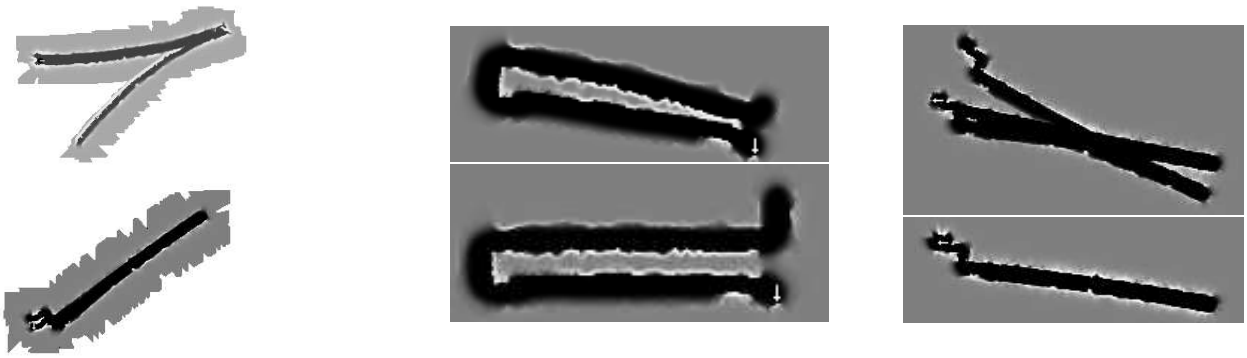

Figure 6. Results of different correction methods, map without (top) and with the proposed correction (bottom). left to right: Odometry calibration (see section 4.1), correction by visual reference (see section 4.2), map matching (see section 4.3)

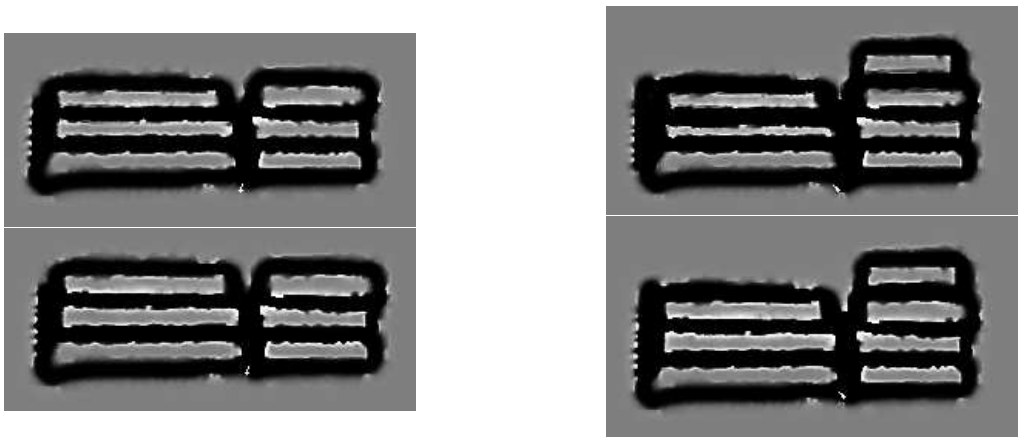

*Figure 7. Results of mapping our experimental area, a homestore. The area shown is about 45m * 20m. We followed a special pattern when driving the robot around to take a map: After adding a new shelf to the map, we always returned to a known area, before moving on to unknown territory. That way, we could correct serious position errors aquired during mapping of a new part of the map when re-entering the known area.*
top left: Here, only Odometry Calibration and Visual Floor Reference have been used. After a path of 350m the robot returns to the starting point to determine the aquired position error. The difference between the robot's belief (shown as the small white arrow) and the true position is -0.8m in x and 0.4m in y. bottom left: Operating on the same data, Map Matching is applied additionally. The difference between belief and true position is 0.05m in x and 0.2m in y only. The position belief is greatly improved by the Map Matching correction.
top right: Shown here is the map after 500m, without Map Matching again. In the top left corner the map is erroneous because a large position error was aquired. As a result, the left corridor is too wide in the map. Furthermore, the second shelf from the bottom seems too narrow, because the map was updated with wrong position belief while driving through this hallway. The robot also returned to the home position, position error was measured as -1.1m in x and 1.3m in y.
bottom right: Again, this map was generated from the same data as the one in the image above, but with Map Matching. No errors are visible in the map. The difference between robot belief and true position at home still was no higher than 0.05m in x and 0.2m in y.

6 Conclusions

With the odometry calibration we can correct odometry data, reducing errors significantly. Adding the visual reference frame we ensure true orientation belief at any time. Since distance measurement is very accurate compared to orientation, this further increases the accuracy of the resulting maps. However, the odometry error still grows unbounded, although on a lower scale. Therefore, manual re-calibration was still needed from time to time.

The novel map matching algorithm overcomes this limitation and allows automatic map building for large areas. A limitation of the algorithm is that the position belief is only recalibrated when a known area is re-entered. Position errors that occured between recalibration points may still have an effect on the map. Therefore, the robot should not move too far without closing a loop, returning to a known place.

References

[1] J. Borenstein and L. Feng. Correction of systematic odometry errors in mobile robots. In *Proc. 1995 Intl. Conf. on Intelligent Robots and Systems (IROS '95*, pp. 569–574.

[2] J. Borenstein and L. Feng. Measurement and correction of systematic odometry errors in mobile robots. *IEEE Transactions on Robotics and Automation*, 12(5), 1996.

[3] D. Fox, W. Burgard, F. Dellert, and S. Thrun. Monte carlo localization: Efficient position estimation for mobile robots. In *Proc. of the AAAI Nat. Conf. on Artifical Intelligence*, 1999.

[4] Dieter Fox, Wolfram Burgard, and Sebastian Thrun. Markov localization for mobile robots in dynamic environments. *Journal of Artificial Intelligence Research*, 11, 1999.

[5] Horst-Michael Gross, Alexander Koenig, Hans-Joachim Boehme, and Christof Schroeter. Vision-based monte carlo self-localization for a mobile service robot acting as shopping assistant in a home store. In *Proc. of the 2002 IEEE/RSJ Intl. Conf. on Intelligent Robots and System*, pp. 265–262.

[6] Jens-Steffen Gutmann and Christian Schlegel. Amos: Comparison of scan matching approaches for self-localization in indoor environments. In *Proceedings of the 1st Euromicro Workshop on Advanced Mobile Robots (EUROBOT '96)*, 1996.

[7] Bernd Jaehne. *Practical Handbook on Image Processing for Scientific Applications*. CRC Press LLC, Boca Raton, Florida, 1997.

[8] Agostino Martinelli. The odometry error of a mobile robot with a synchronous drive system. *IEEE Transactions on Robotics and Automation*, 18(3):399–405, 2002.

[9] P.S. Maybeck. The kalman filter: an introduction to concepts. In I. Cox and G. Wilfong, editors, *Autonomous Robot Vehicles*, pages 194–204. Springer-Verlag, 1990.

[10] Michael Montemerlo, Sebastian Thrun, Daphne Koller, and Ben Wegbreit. FastSLAM:A factored solution to the simultaneous localization and mapping problem. In *Proceedings of the AAAI Nat. Conf. on Artificial Intelligence*, 2002.

[11] Hans Moravec. Sensor fusion in certainty grids for mobile robots. *AI Magazine*, 9(2):61–77, 1988.

[12] Randall Smith, Matthew Self, and Peter Cheeseman. A stochastic map for uncertain spatial relationships. In *4th Intl. Symposium on Robotic Research*. MIT Press, 1987.