Triangulation-Based Fusion of Sonar Data with Application in Robot Pose Tracking

Olle Wijk, *Student Member, IEEE,* and Henrik I. Christensen, *Member, IEEE*

***Abstract*—In this paper a sensor fusion scheme, called triangulation-based fusion (TBF) of sonar data, is presented. This algorithm delivers stable natural point landmarks, which appear in practically all indoor environments, i.e., vertical edges like door posts, table legs, and so forth. The landmark precision is in most cases within centimeters. The TBF algorithm is implemented as a voting scheme, which group sonar measurements that are likely to have hit a mutual object in the environment. The algorithm has low complexity and is sufficiently fast for most mobile robot applications. As a case study, we apply the TBF algorithm to robot pose tracking. The pose tracker is implemented as a classic extended Kalman filter, which use odometry readings for the prediction step and TBF data for measurement updates. The TBF data is matched to pre-recorded reference maps of landmarks in order to measure the robot pose. In corridors, complementary TBF data measurements from the walls are used to improve the orientation and position estimate. Experiments demonstrate that the pose tracker is robust enough for handling kilometer distances in a large scale indoor environment containing a sufficiently dense landmark set.**

***Index Terms*—Localization, pose tracking, sensor fusion, sensor modeling, sonars.**

I. Introduction

In THE authors' opinion the sonar is an attractive range sensor. It is cheap compared to other popular range sensors like laser scanners and range cameras. While the field of view of a laser scanner is usually limited to a plane, a set of sonars placed at strategic positions on a mobile robot gives a complete coverage of the surrounding world. Moreover, if disregarding outliers caused by specular reflections and cross talk, a standard Polaroid 6500 sonar sensor gives quite accurate range readings ($\pm 1\%$).

The bad angular resolution and the frequent number of outliers in sonar data can be overcome using different techniques [1], [2], [5]. In advanced systems, arrays of sonars are being used for listening to their own and the other sensors echoes in order to find object positions through triangulation. The echoes are then fast sampled (1 MHz) and a large amount of signal processing is performed to obtain very accurate range readings, which is a prerequisite if triangulating objects with a small base line between the sensors. In this kind of research, high confidence classification of points, edges, and planes is reported [10], [12]. The accuracy in position of the classified objects is claimed to be within millimeters, where the error comes down to being dependent on environmental factors like temperature, humidity and wind fluctuations.

An appealing method, which is presented in this paper, is to manage with less signal processing, i.e., less range accuracy, and still be able to reliably detect discriminating features, like vertical edges. The method we propose is called triangulation-based fusion (TBF) of sonar data, and buffers sonar scans that are triangulated against each other using a simple and low complexity voting scheme. Since the sonar scans in the buffer are taken from different robot positions, the base line between the sensor readings being triangulated are only limited by the odometry drift. Hence, the range accuracy is not of major importance in this approach. This paper contains a detailed description of the TBF algorithm as well as a case study of how it can be applied for robot pose tracking. The paper is organized as follows.

Section II contains a detailed description of the TBF algorithm and discusses implementation issues. It is also explained how the TBF data uncertainty can be obtained using local grid maps. A real world example, where a sonar equipped mobile robot operates in a living room, is used throughout the section to illustrate the characteristics of the TBF algorithm.

Section III describes how the TBF algorithm can be applied to robot pose tracking when using an extended Kalman filter. For related work, see [3], [12], and [16]. A lot of details, such as tuned parameter values, are reported in order to document the implementation of the pose tracker. The idea is to use TBF data that have been validated against reference maps of TBF data for subsequent measurement of the robot pose. In corridors, the method is complemented with TBF data from the walls. A large scale indoor environment is used in the experiment to demonstrate the strength and weaknesses of the approach.

II. TBF Algorithm

TBF of sonar data is a novel a computationally efficient voting scheme for grouping sonar readings together that have hit a mutual vertical edge object in the environment. The sonars are assumed to be placed in the same horizontal plane and the data obtained from the sensors is interpreted in the two-dimensional world representation given by this plane. The performance of the algorithm is illustrated in Fig. 1, which shows a complete 3-D model of a 9×5 m living room which

Manuscript received October 15, 1999; revised September 15, 2000. This paper was recommended for publication by Associate Editor J. Laumond and Editor V. Lumelsky upon evaluation of the reviewers' comments. The research was carried out at the Centre for Autonomous Systems at the Royal Institute of Technology and sponsored by the Swedish Foundation for Strategic Research. This paper was presented in part at the IEEE International Conference on Robotics and Automation, Leuven, Belgium, May 1998, and in part at the Seventh International Symposium on Intelligent Robotic Systems, Coimbra, Portugal, July 1999.

The authors are with the Centre for Autonomous Systems, Royal Institute of Technology, SE-100 44 Stockholm, Sweden (e-mail: olle@s3.kth.se; hic@nada.kth.se).

Publisher Item Identifier S 1042-296X(00)11578-2.

1042–296X/00$10.00 © 2000 IEEE

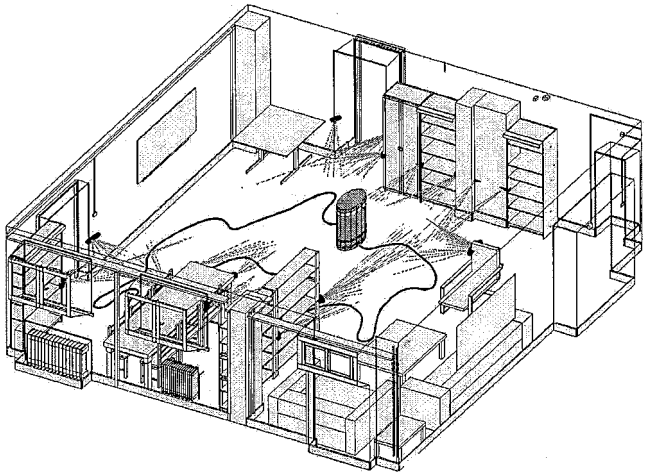

Fig. 1. The TBF algorithm is a voting scheme that solves a data association problem, i.e., it clusters sonar measurements that originate from a mutual edge in the environment. This figure illustrates how the TBF algorithm performs when a sonar equipped Nomad 200 robot traverses a living room. All data presented in the figure are real.

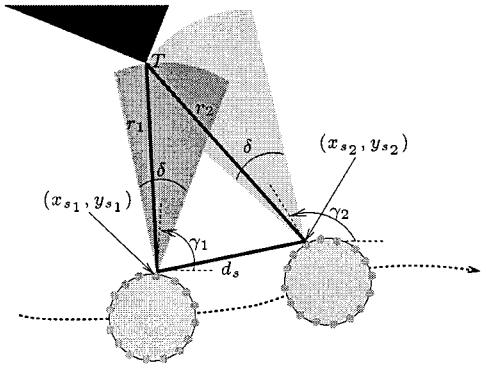

Fig. 2. Basic triangulation principle of an edge. Given two sonar readings from an edge, the position T of the edge can be obtained by taking the intersection between the beam arcs.

was hand measured within centimeter precision. The thick line on the floor is the robot trajectory and the clusters of lines represent sonar readings that have been grouped by the TBF algorithm. The mutual end point for each line cluster represents an object position estimate delivered by the TBF algorithm. As seen from the figure, the TBF method is good at detecting vertical edges in the environment, for instance door posts, shelf corners, table legs and so forth. Before discussing the TBF method in detail, we will spend the following section on discussing the basic component used in the TBF algorithm, i.e., triangulating two sonar measurements.

A. Basic Triangulation Principle

Consider two sonar readings taken from different positions during robot motion (Fig. 2). The readings are assumed to originate from a vertical edge at position $T = (x_T, y_T)$ in the environment. If only one of the range readings is considered, the sonar physics limit the object position to be somewhere along the associated beam arc.[1] When using the information from both sonar readings we can get the object position T by computing

[1]Neglecting the presence of side lobes.

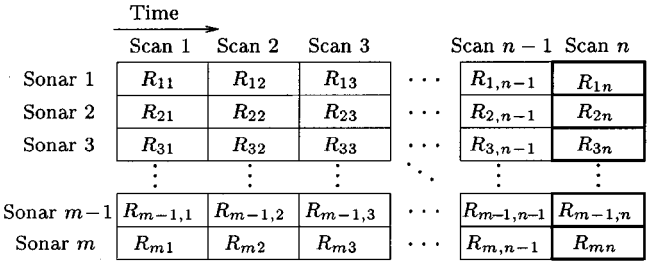

Fig. 3. Sliding window for storage of sonar readings. Each column contain a complete sonar scan taken during robot motion.

the intersection point $T = (x_T, y_T)$ between the two beam arcs. The following equations have to be solved:

$$(x_T - x_{s_i})^2 + (y_T - y_{s_i})^2 = r_i^2, \qquad i = 1, 2 \tag{1}$$

$$\arctan\left(\frac{y_T - y_{s_i}}{x_T - x_{s_i}}\right) \in \left[\gamma_i - \frac{\delta}{2}, \gamma_i + \frac{\delta}{2}\right], \qquad i = 1, 2. \tag{2}$$

Here (x_s, y_s) denotes the sensor position, r is the range reading, γ is the sensor heading angle, and δ is the opening angle of the center sonar lobe. For the sonar type used in this presentation (Polaroid 6500), δ is about 25°. The solutions $(\hat{x}_T, \hat{y}_T)$ of (1) with $i = 1, 2$ can be written

$$\hat{x}_T = x_{s_1} + \frac{1}{d_s^2}\left(d_{x_s} d_r^2 \pm |d_{y_s}| \sqrt{r_2^2 d_s^2 - d_r^4}\right) \tag{3}$$

$$\hat{y}_T = y_{s_1} + \frac{1}{d_s^2}\left(d_{y_s} d_r^2 \pm |d_{x_s}| \sqrt{r_2^2 d_s^2 - d_r^4}\right) \tag{4}$$

where

$$\begin{aligned} d_{x_s} &= x_{s_1} - x_{s_2} \\ d_{y_s} &= y_{s_1} - y_{s_2} \\ d_s^2 &= d_{x_s}^2 + d_{y_s}^2 \\ d_r^2 &= \frac{r_1^2 - r_2^2 - d_s^2}{2}. \end{aligned}$$

False solutions $(\hat{x}_T, \hat{y}_T)$ in (3) and (4) are removed when verified against (1) and (2).

B. Implementation

Consider a mobile robot equipped with m sonars distributed at arbitrary positions in a horizontal plane. The TBF algorithm is then implemented as a sliding window (Fig. 3) with m rows and n columns. Each entry R_{ij} in the window contains sonar data necessary for performing a triangulation, i.e.,

$$R_{ij} = (x_{s_{ij}}, y_{s_{ij}}, \gamma_{ij}, r_{ij}).$$

Each column in the window represents a complete sonar scan taken during robot motion. The right column represent the most recent sonar scan, and the sliding window is updated with a new scan when *all* sensor position stamps with respect to the old scan have moved more than a certain minimum distance $d_t = 0.05$ m. Depending on the sonar scan sampling frequency (2–3 Hz) and

```
/********************* THE TBF ALGORITHM ***************************/
for i = 1 → m{                                                          (1)
 if r_in < r_max{                                                       (2)
  G := {R_in}                                                           (3)
  n_t := 0,   x̂_T := x_{s_in} + r_in cos(γ_in),   ŷ_T := y_{s_in} + r_in sin(γ_in)   (4)
  x_min := y_min := maxInt,   x_max := y_max := −maxInt                 (5)
  for j = (n − 1) → 1                                                   (6)
   for k = 1 → m                                                        (6)
    if r_kj < r_max                                                     (7)
     if (x̂_T, ŷ_T) ∈ beam of R_kj                                       (7)
      r_e := sqrt((x̂_T − x_{s_kj})^2 + (ŷ_T − y_{s_kj})^2)              (7)
      if |r_e − r_kj| < d_1/(n_t + 1)                                   (7)
       if TriangulationPossible (In:R_in, R_kj; Out:x_T^tri, y_T^tri)   (8)
        G := {G, R_kj}                                                  (9)
        x̂_T := 1/(n_t+1) (n_t x̂_T + x_T^tri)                            (9)
        ŷ_T := 1/(n_t+1) (n_t ŷ_T + y_T^tri)                            (9)
        n_t := n_t + 1                                                  (9)
        if x_T^tri < x_min then x_min := x_T^tri                        (10)
        if x_T^tri > x_max then x_max := x_T^tri                        (10)
        if y_T^tri < y_min then y_min := y_T^tri                        (10)
        if y_T^tri > y_max then y_max := y_T^tri                        (10)
 }
 if n_t ≥ 1                                                             (11)
  if (x_max − x_min) + (y_max − y_min) > d_2 then n_t := −n_t            (12)
  RefineTriangulationPoint(In: G, Out: x̂_T, ŷ_T, P_T)                   (13)
  Store n_t, T̂, P_T and possibly G                                      (14)
}                                                                       (15)
```

Fig. 4. Pseudocode for the TBF algorithm. Each step in the right column is explained in detail in the text.

the current robot speed (0–0.5 m/s), this distance will vary, typically between 0.05–0.25 m. When a new scan is inserted into the window, the oldest scan is shifted out (left column). Hence, the column size of the sliding window is kept constant to n sonar scans. In the present implementation $n = 10$ is used. Between each update of the sliding window, the TBF algorithm searches for triangulation hypotheses between the right column (new data) and the rest of the window. From a programming point of view, the TBF algorithm is implemented according to the pseudocode given in Fig. 4. The different steps in the pseudocode have been numbered in the right column, and are explained in detail below.

1) Loop over the last column in the sliding window, i.e., the most recently acquired sonar scan. Let us from here on consider the first lap in this loop ($i = 1$). This means that we consider the most recent reading taken with sonar 1, i.e., $R_{1n} = (x_{s_{1n}}, y_{s_{1n}}, \gamma_{1n}, r_{1n})$.
2) Check if the range reading r_{1n} possibly could be from an edge. From experiments we have concluded that a Poloroid 6500 sensor is able to detect edges up to at least 5 m. Hence we check that $r_{1n} < r_{\max} = 5$ m.
3) Form a set G that will contain all the readings which should be associated with the reading R_{1n}. Initially $G = R_{1n}$.
4) Initialize a zero hypothesis ($n_t = 0$) about the object that was hit by the reading R_{1n}. The zero hypothesis correspond to an object position estimate $\hat{T} = (\hat{x}_T, \hat{y}_T)$ at the middle of the corresponding beam arc.
5) Initialize variables that keep track of the maximum deviation between the triangulation measurements.
6) Loop over all but the last column in the sliding window to find triangulation partners for the reading R_{1n}. A triangulation partner R_{kj} is only approved if it fulfills three constraints (step 7).
7) First repeat the check done in step 2 for the range reading r_{kj}, second check that the current object position estimate $\hat{T}$ belongs to the sonar beam of R_{kj}, and third that the expected range reading r_e does not differ too much from the actual range reading r_{kj}. The allowed difference $d_1/(n_t + 1)$ decreases with the number of successful triangulations n_t that have been done (step 8). In our implementation $d_1 = 0.3$ m.
8) If this step is reached, there is a high probability that the readings R_{1n} and R_{kj} originate from the same object. Hence, it is worth using (1)–(4) to check if an intersection point (x_T^{tri}, y_T^{tri}) exists.
9) For a successful triangulation (step 8), add the reading R_{kj} to the set G, recursively update the object position estimate $\hat{T}$ and finally increase the hypothesis counter n_t (the number of successful triangulations done so far). Note that if $n_t = 0$ the zero hypothesis is just replaced by (x_T^{tri}, y_T^{tri}).
10) Update the maximum deviation between the successful triangulations done so far.
11) Only consider hypotheses supported with at least one successful triangulation.
12) If the maximum deviation between the successful triangulations is large ($d_2 = 0.1$ m in our implementation) we classify the position estimate $\hat{T}$ as belonging to an object in the environment which is *not* well represented

as an edge, for instance a wall. This classification is done by switching the sign of the n_t variable. The sign switch secures that triangulation points with high positive n_t values will have a high probability of originating from an edge such as door posts, shelf edges, table legs etc.

13) Use the set G of grouped readings to compute a refined position estimate $(\hat{x}_T, \hat{y}_T)$ and the associated covariance matrix P_T. This step is discussed in detail in the next section.
14) Save the triangulation point $(n_t, \hat{T}, P_T)$, and optionally the set G.
15) Go back to step 1 and increase i.

Continuing commenting on the TBF code in Fig. 4, it is noted that it generates at most m triangulation points between each update of the sliding window. Each triangulation point correspond to a position of an object that was hit by one of the readings in the last column of the sliding window. At line 6 in the TBF code (*for* $j = (n-1) \rightarrow 1$), the window is swept backward column wise to find triangulation partners for a particular element in the last column. The reason for not sweeping the columns in forward direction is that it is more probable that nearby sonars scans have hit a mutual object than scans that are further apart. Hence a zero hypothesis (step 4) is more probable to evolve into a correct 1-hypothesis ($n_t = 1$) if first considering the most adjacent sonar scan. In step 13 of the TBF code, the triangulation point $(\hat{x}_T, \hat{y}_T)$ is refined by a local grid map computation. In this computation the uncertainty of the estimate is also obtained. The next section describes how this is done in detail.

C. Refining a Triangulation Point Using a Local Grid Map

Steps 2–10 in the TBF algorithm provide a triangulation point estimate that has been formed by recursively taking the mean value of beam arc intersections (step 9). It is clear that this is not the best way to compute this estimate.[2] Moreover, the uncertainty in the position is unknown. What is good, though, is that the triangulation point produced by steps 2–10 is a good initial guess of the true position. Hence, by centering a local grid map around this initial guess we can recompute the triangulation point and its uncertainty by using a sonar sensor model. In the present implementation a simple sensor model is used for this purpose, where the range reading r is assumed to be normally distributed around the true range $\overline{r}$ as

$$r \sim N(\overline{r}, 0.01\overline{r} + 0.01 \text{ m}).$$

The first standard deviation term, $0.01\overline{r}$, is in agreement with what the data sheets specify for the a Polaroid 6500 sensor and the extra centimeter is used to cover up for the uncertainty between the sensor positions where the readings were taken, caused by odometry drift. Concerning the angular modeling of the sonar, an uniform distribution is used, i.e., an object is assumed to be detectable anywhere in the main lobe with equal probability, but not detectable at all outside the main lobe. This model is supported by experiments done with a sonar pulse interacting with an edge and is documented in [21].

As stated above the local grid map is centered around the initial position estimate provided by steps 2–10 in the TBF algorithm. The grid size depends on the quality (n_t) of the initial guess. For triangulation points with $n_t \geq 4$, a 31×31 grid map with 0.01-m resolution is used.[3] For points with $n_t < 4$, a 24×24 grid map (cell size 0.02 m) is used. Hence more computational effort is put on estimating triangulation points supported by many readings. By using the sensor model, the grouped sonar readings can be fused in the grid map. Treating the readings as independent, this fusion process is straightforward (pure multiplication in each cell). An example of a grid map with fused sonar readings is shown in Fig. 5. By using the cell with maximum probability as a triangulation point estimate $(\hat{x}_T, \hat{y}_T)$, the covariance P_T of this estimate is readily obtained as

$$P_T = \sum_{i,j} \begin{pmatrix} (x_{ij}-\hat{x}_T)^2 & (x_{ij}-\hat{x}_T)(y_{ij}-\hat{y}_T) \\ (x_{ij}-\hat{x}_T)(y_{ij}-\hat{y}_T) & (y_{ij}-\hat{y}_T)^2 \end{pmatrix} p_{ij}$$

where (x_{ij}, y_{ij}) is the position corresponding to cell (i, j) and p_{ij} is the associated probability.

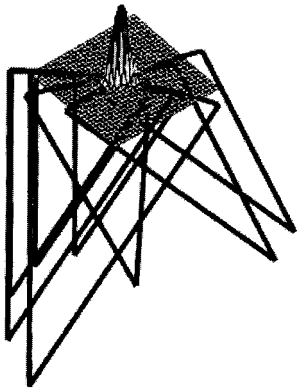

Fig. 5. Refined triangulation point estimate using a local grid map. The beams represent grouped sonar readings by the TBF algorithm. This triangulation point actually corresponds to one of the shelf corners from the experiment presented in Fig. 1.

D. Computational Complexity

The computational complexity of the TBF algorithm essentially relies on steps 8 and 13 (see code). Step 13, the local grid map computation, is by far the most expensive step and is executed for each found triangulation point. The complexity of the grid map fusion process is $o(n_c^2(n_t+1))$, where n_c is the number of cells along a side of the grid map, but can be reduced by truncating the range sensor model after a couple of standard deviations. When running the TBF algorithm on a 450-MHz PC, step 13 takes on average 2.6 ms to process. The TBF algorithm can of course be speeded up by simply disregarding step 13, but then it will provide less accurate triangulation point estimates, formed by taking recursive means of intersection points. Furthermore, the uncertainty in these estimates will not be available. Under these circumstances, the complexity of the algorithm depends on step 8, i.e., triangulating two sonar readings. This operation can be done in less than a 100 operations and takes about 10 μs to process on a 450-MHz PC. Note that step 8 is only performed when necessary (step 7).

[2]But it is cheap and often gives surprisingly good estimates!

[3]The grid size can probably be reduced, but this is the size we use at the moment. The grid size is limited from below by the grid resolution and the accuracy of the initial guess.

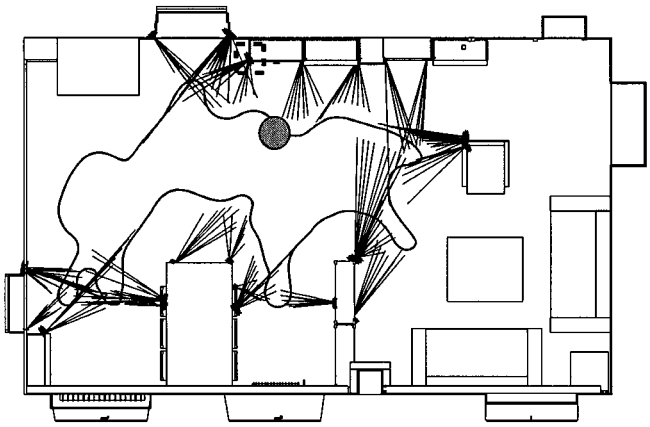

Fig. 6. The experiment in Fig. 1 from a top view. The triangulation point estimates are plotted with their uncertainty ellipses. Fig. 7 shows a closeup of the dashed rectangle at one of the bookshelf corners.

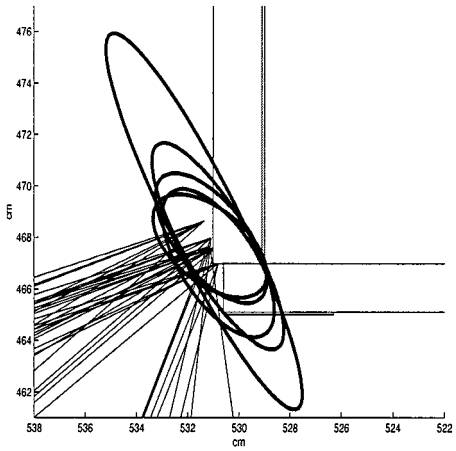

Fig. 7. Close-up of dashed rectangle in Fig. 6. The ellipses in the picture represent the uncertainty, two standard deviations in size, of a triangulation point produced by the TBF algorithm. From this picture it is clear that the accuracy of the triangulation points can reach centimeter level. One of these ellipses originate from the refined triangulation point computation shown in Fig. 5.

During a complete iteration of the TBF algorithm (step 1–16), the sliding window is swept m times (step 1 and 6). Hence step 7 is executed $m^2(n-1)$ times (disregarding step 2). This is usually not a real time problem for a moderate size window ($m = 16$, $n = 10$). However, when increasing the size of the window, the complexity explodes. This will for instance happen when increasing the number of sonars (m), or worse, using multiple echoes from each sonar. To circumvent this problem, it is possible to store the sonar scans in a coarse grid map (0.5-m resolution) rather than in a sliding window. Each sonar reading R_{ij} is then placed in the cell corresponding to the mid beam position (mentioned zero hypothesis in step 4) *and* in the adjacent eight cells. Each cell can then be treated as a sliding window of its own. For each new reading R_{in}, the mid beam position of R_{in} is then used to find the grid cell containing all the adjacent readings. This reduces complexity substantially.

E. Experiments

In the introduction of this section, a real world experiment with the TBF algorithm was presented. Let us take a closer look at this experiment. In Fig. 6, a top view of Fig. 1 is shown. From this view it is clear that the TBF algorithm is good at detecting vertical edges in the environment. Only triangulation points fulfilling $n_t \geq 6$ and $\sqrt{\rho(P_T)} < 0.05$ m are shown in this plot, where $\rho(P_T)$ means the spectral radius of the covariance matrix P_T. Besides this "high" thresholding it may seem like that some obvious edges are missed, for instance the sofa table, but then one has to consider the height of the objects. The sonars we use only have a beam width of 25° and hence the sight in vertical direction is rather limited. The sofa table is usually never detected by the sonars, because of its low height. Moreover, the sonars we use only give the first echo response back. With the ability to detect multiple echoes, more edges could have been seen. Another fact that is seen from Fig. 6, is that the TBF algorithm is best at detecting edges when the robot moves side passed them. This is because of the base line between the readings then come in a favorable orientation with respect to the object being triangulated.

In Fig. 7, a close up of the dashed rectangle of Fig. 6 is shown. From this figure it is clear that the edges are not just detected once, but several times when the robot moves along. The precision of the estimates can be seen to be within centimeters. In a general situation the precision depends on the distance to the object as well as the base line between the sonar readings. Preferably, one would like to have as large base line as possible. Since the TBF algorithm passively stores and analyzes sonar scans there is no base line restriction, except for the odometry drift between the scans. This makes it possible to produce triangulation points with acceptable precision (0.1 m) using sonars with low range resolution. In a previous implementation of the algorithm, a 25-mm resolution in the sonar data was used successfully. In the experiments presented in this paper the sensor resolution is however at millimeter level, though not calibrated to that level.

Other interesting facts, which are not shown in the experiment presented here, is that the TBF algorithm is very efficient for removing outliers in sonar data such as specular reflections and cross talk [21], furthermore, moving objects are efficiently rejected by the method. This has to do with that the algorithm is being built on a static world assumption. Triangulations on a moving target which in the algorithm is assumed to be static will clearly not get much support when it comes to voting (n_t).

III. Robot Pose Tracking

In this section we present a case study of how the TBF algorithm can be used for robot pose tracking. However, to be able to track the position, a reference map is needed. In the first section we therefore discuss how such map could be built manually. In fact the TBF data presented in the previous section was actually generated with this map building strategy.[4]

A. Map Acquisition Using Natural Point Landmarks

Consider a robot that is manually operated around in an indoor setting which it later should operate autonomously in. The TBF algorithm is supposed to be active during the movement of

[4]Please do not confuse this step with what popularly is called SLAM (simultaneous localization and map building). This is a map built with human robot interaction.

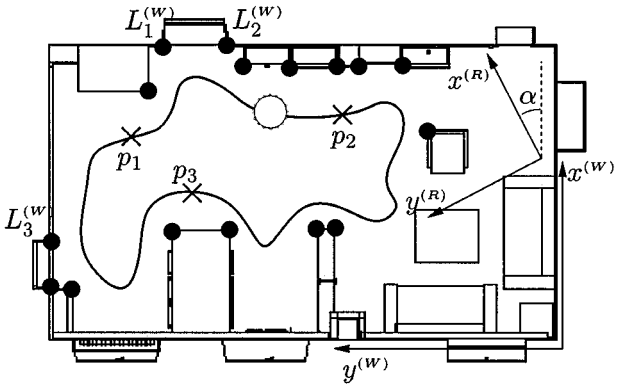

Fig. 8. Using positions $p_1, p_2, p_3, \ldots$ that are known in both (R)obot coordinates (odometry) and (W)orld coordinates (hand measured), a recorded landmark $L^{(\mathrm{R})}$ can be translated to world coordinates $L^{(\mathrm{W})}$.

the robot. Each time a triangulation point it detected that satisfies the thresholds $n_t \geq 6$ and $\sqrt{\rho(P_T)} < 0.05$ m, the corresponding position $\hat{T}$ is registered as a *natural point landmark* L. To limit the number of natural point landmarks, we keep track of triangulation points that cluster (closer than 0.1 m). The clustered triangulation points are regarded as *one* natural point landmark by only keeping the most recent detected one. Because of the odometry drift, it is important not to record landmarks over too large distances, otherwise landmarks collected at the beginning of the robot motion would be represented in a significantly different coordinate system compared to landmarks collected at the end of the motion. A suitable way to represent all landmarks in *one* coordinate system with acceptable precision, is to introduce a (W)orld coordinate system, and then measure up some positions $p_1^{(\mathrm{W})}, p_2^{(\mathrm{W})}, \ldots, p_p^{(\mathrm{W})}$ that cover the area the robot is going to operate in (Fig. 8). By placing the robot at one of these positions, let say $p_1^{(\mathrm{W})}$, the corresponding (R)obot coordinate $p_1^{(\mathrm{R})}$ can be obtained from odometry information. The robot can then be moved to another position, let say $p_2^{(\mathrm{W})}$, and while this is done the TBF algorithm records natural point landmarks. When the robot reaches $p_2^{(\mathrm{W})}$, the corresponding robot coordinate $p_2^{(\mathrm{R})}$ is given from odometry information. The collected natural point landmarks can then be converted into world coordinates since the transformation between robot and world coordinates is known once $p_1^{(\mathrm{W})}$, $p_1^{(\mathrm{R})}$, $p_2^{(\mathrm{W})}$, and $p_2^{(\mathrm{R})}$ are known. The distance between the positions $p_1^{(\mathrm{W})}$ and $p_2^{(\mathrm{W})}$ should not be too long because this could cause a significant odometry drift effect between the landmark positions. In our implementation we use a separation distance of about 4–7 m, but this distance is dependent on the kind of ground surface the robot moves on as well as the odometry precision of the robot. Summing up, the procedure of moving the robot between pairs of positions $p_1^{(\mathrm{W})}, p_2^{(\mathrm{W})}, \ldots, p_p^{(\mathrm{W})}$ can be repeated so that a complete reference map of landmarks is obtained.

B. Pose Tracking

Consider a robot navigating in an indoor environment with the TBF algorithm running in the background, and that a reference map of landmarks $\mathcal{L}^{(\mathrm{W})} = \{L_1^{(\mathrm{W})}, L_2^{(\mathrm{W})}, \ldots, L_n^{(\mathrm{W})}\}$ is available in world coordinates. The robot position in world coordinates is denoted $(x_r^{(\mathrm{W})}, y_r^{(\mathrm{W})})$ and the orientation is captured by an angle α between the world- and robot-x-coordinate-axis (see Fig. 8). Hence the *robot pose* is captured by the state vector

$$x \triangleq \left(x_r^{(\mathrm{W})}, y_r^{(\mathrm{W})}, \alpha\right)^T.$$

The pose tracking problem studied here is to maintain an estimate of the state x while the robot is moving. For that purpose we propose an extended Kalman filter operating on the nonlinear state model

$$x_{k+1} = f(x_k, u_k) + w_k \tag{5}$$

$$y_k^{(i)} = H^{(i)} x_k + v_k^{(i)}, \qquad i = 1, 2 \tag{6}$$

$$E(w_k w_k^T) = Q_k \tag{7}$$

$$E\left(v_k^{(i)} v_k^{(i)} T\right) = R_k^{(i)}, \qquad i = 1, 2. \tag{8}$$

Here the input signal u_k influence how the robot is moving. The robot pose measurement $y_k^{(1)}$ is obtained from triangulation points that have been validated against the reference map $\mathcal{L}^{(\mathrm{W})}$ (Sections III-B-1 and III-B-2). Finally, $y_k^{(2)}$ is a partial measurement of the robot pose based on triangulation points that have been matched against corridor walls (Section III-C). Assuming white and uncorrelated noise, the extended Kalman filter for the robot pose model becomes

prediction:

$$\hat{x}_{k+1|k} = f(\hat{x}_{k|k}, u_k) \tag{9}$$

$$P_{k+1|k} = \frac{\partial f}{\partial x_k} P_{k|k} \left(\frac{\partial f}{\partial x_k}\right)^T + Q_k \tag{10}$$

measurement update:

$$\epsilon_{k+1} = y_{k+1}^{(i)} - H^{(i)} \hat{x}_{k+1|k}$$

$$S_{k+1} = H^{(i)} P_{k+1|k} H^{(i)T} + R_{k+1}^{(i)} \tag{12}$$

$$K_{k+1} = P_{k+1|k} H^{(i)T} S_{k+1}^{-1} \tag{13}$$

$$\hat{x}_{k+1|k+1} = \hat{x}_{k+1|k} + K_{k+1} \epsilon_{k+1} \tag{14}$$

$$P_{k+1|k+1} = \left(I - K_{k+1} H^{(i)}\right) P_{k+1|k}. \tag{15}$$

Let us now consider these equations in more detail. The nonlinear function $f(x_k, u_k)$ that appears in the odometry model (9) can for a synchro-drive robot be taken as

$$f = \begin{pmatrix} \left(x_r^{(\mathrm{W})}\right)_k + \frac{D_k}{\Delta\beta_k}\left(\sin(\beta_k + \alpha_k + \Delta\beta_k) - \sin(\beta_k + \alpha_k)\right) \\ \left(y_r^{(\mathrm{W})}\right)_k + \frac{D_k}{\Delta\beta_k}\left(\cos(\beta_k + \alpha_k + \Delta\beta_k) - \cos(\beta_k + \alpha_k)\right) \\ \alpha_k \end{pmatrix}$$

where D_k is the relative distance movement between time step k and $k+1$ and $\Delta\beta_k$ is the change in motion direction. The input signal u_k of the Kalman filter is a vector containing these odometry signals, i.e., $u_k = (D_k, \Delta\beta_k)^T$. From prediction (9) it is seen that the orientation angle α is predicted to remain constant. A more accurate robot model would include an extra state representing the drift in orientation. In this approach, the orientation drift is assumed to be estimated and compensated for enough by many triangulation point measurements.

In (10), the state covariance is predicted. Here it is important to model the process noise matrix Q_k appropriately. Preferably Q_k should be chosen so that if the robot moves from point A to point B with no other sensing than odometry, then the state covariance prediction when reaching point B should be the same independently of the odometry sampling frequency. The matrix Q_k should also be chosen so that the predicted state covariance become realistic not only over short distance movements, but also for long distances. In [11] it is described how to model the process noise matrix Q_k for a differential-drive robot. This model technique can be extended to also encompass a synchro-drive robot. In the experiments presented in Section III-D a synchro-drive Nomad 200 robot is used, and hence the process noise matrix Q_k is modeled based on this fact [7]. Three parameters need to be set in the synchro-drive model, k_D, k_β^β, and k_β^D where

$$\sigma_{D_k}^2 = k_D|D_k| \qquad \sigma_{\Delta\beta_k}^2 = k_\beta^\beta|\Delta\beta_k| + k_\beta^D|D_k|.$$

In our implementation these parameters are set to

$$\begin{aligned} k_D &= 0.005 \quad \mathrm{m}^2/\mathrm{m} \\ k_\beta^\beta &= 6.3 \cdot 10^{-6} \quad \mathrm{rad}^2/\mathrm{rad} \\ k_\beta^D &= 0.0005 \quad \mathrm{rad}^2/\mathrm{m}. \end{aligned}$$

The parameter k_D reflects the variance in the distance traveled while k_β^β and k_β^D captures the variance in motion direction when the robot is steering and moving, respectively. The parameter choices are conservative to be sure to capture the odometry drift, for instance $k_D = 0.005\ \mathrm{m}^2/\mathrm{m}$ means a drift of about 0.07 m/m.

From (10) and (15) it is evident that the prediction part of the Kalman filter will increase the state covariance matrix P_k while the measurement part will decrease it. When P_k decreases, the Kalman filter converges and measurements are gradually weighted less, which implies that the state estimate $\hat{x}_k$ becomes static. To prevent this, the diagonal elements of P_k are bounded from below as

$$p_{11},\, p_{22} \geq (0.1\ \mathrm{m})^2 \qquad p_{33} \geq \left(2^\circ \frac{\pi}{180^\circ}\right)^2.$$

The Kalman filter is obviously represented in world coordinates, since the state x is in world coordinates. The data obtained from odometry and sonars is however given in robot coordinates, and hence it need to be converted to world coordinates. Therefore, a state-dependent transformation is introduced that take an arbitrary point p from robot to world coordinates

$$p^{(\mathrm{W})} = R_r(\hat{\alpha}) \left[p^{(\mathrm{R})} - \begin{pmatrix} x_r^{(\mathrm{R})} \\ y_r^{(\mathrm{R})} \end{pmatrix} \right] + \begin{pmatrix} \hat{x}_r^{(\mathrm{W})} \\ \hat{y}_r^{(\mathrm{W})} \end{pmatrix} \quad (16)$$

$$R_r(\hat{\alpha}) = \begin{pmatrix} \cos(\hat{\alpha}) & -\sin(\hat{\alpha}) \\ \sin(\hat{\alpha} & \cos(\hat{\alpha}) \end{pmatrix}. \quad (17)$$

Here $(\hat{x}_r^{(\mathrm{W})}, \hat{y}_r^{(\mathrm{W})}, \hat{\alpha})$ is the current pose estimate $\hat{x}$ and $(x_r^{(\mathrm{R})}, y_r^{(\mathrm{R})})$ is the most recent robot position obtained from sampled odometry data. In the following sections we will use (16) and describe, in detail, how the robot pose measurements $y_k^{(1)}$ and $y_k^{(2)}$ are performed.

1) Robot Pose Measurement: The measurement $y_k^{(1)}$ appearing in (6) is a direct measurement of the robot pose, i.e., $H^{(1)} = I$. In fact, $y_k^{(1)}$ is a nonlinear measurement of the state, but we have chosen to hide these nonlinearities within the measurement covariance matrix $R_k^{(1)}$. The state measurement $y_k^{(1)}$ is obtained by solving a maximum-likelihood estimation problem which is formulated as

$$y_k^{(1)} = \operatorname*{argmin}_x \sum_{i=1}^{q} f^{(i)}(x), \qquad q \geq 2 \quad (18)$$

$$f^{(i)}(x) = \left(\hat{T}_i^{(\mathrm{W})} - L_i^{(\mathrm{W})}\right)^T P_{T_i}^{-1(\mathrm{W})} \left(\hat{T}_i^{(\mathrm{W})} - L_i^{(\mathrm{W})}\right). \quad (19)$$

Here $(\hat{T}_i^{(\mathrm{W})}, P_{T_i}^{(\mathrm{W})})$ denotes a recent triangulation point produced by the TBF algorithm and $L_i^{(\mathrm{W})} = (x_{L_i}^{(\mathrm{W})}, y_{L_i}^{(\mathrm{W})})$ is a natural point landmark in the reference map $\mathcal{L}^{(\mathrm{W})}$. The triangulation point $\hat{T}_i^{(\mathrm{W})}$ has been passed through two types of validation gates to assure that $\hat{T}_i^{(\mathrm{W})}$ is a measurement of the natural point landmark $L_i^{(\mathrm{W})}$. These validation gates will be discussed in more detail later. In the objective function of (18) it is seen that $\hat{T}_i^{(\mathrm{W})}$ and $P_{T_i}^{(\mathrm{W})}$ are given in world coordinates, but these quantities will in practice be given in robot coordinates by the TBF algorithm. The transformation (16) could be used to relate $\hat{T}_i^{(\mathrm{W})}$, $P_{T_i}^{(\mathrm{W})}$ with $\hat{T}_i^{(\mathrm{R})}$, $P_{T_i}^{(\mathrm{R})}$ as

$$\hat{T}_i^{(\mathrm{W})}(x) = R_r(\alpha) \left[\hat{T}_i^{(\mathrm{R})} - \begin{pmatrix} x_r^{(\mathrm{R})} \\ y_r^{(\mathrm{R})} \end{pmatrix} \right] + \begin{pmatrix} x_r^{(\mathrm{W})} \\ y_r^{(\mathrm{W})} \end{pmatrix} \quad (20)$$

$$P_{T_i}^{(\mathrm{W})}(x) = R_r(\alpha) P_{T_i}^{(\mathrm{R})} R_r^T(\alpha). \quad (21)$$

From these expressions it is seen that the function $f^{(i)}(x)$ is state dependent. If we denote the elements of $\hat{T}_i^{(\mathrm{R})}$ and $P_{T_i}^{(\mathrm{R})}$ by

$$\hat{T}_i^{(\mathrm{R})} = \begin{pmatrix} \hat{x}_{T_i} \\ \hat{y}_{T_i} \end{pmatrix} \qquad P_{T_i}^{(\mathrm{R})} = \begin{pmatrix} p_1^{(i)} & p_3^{(i)} \\ p_3^{(i)} & p_2^{(i)} \end{pmatrix}$$

and substitute (20) and (21) into the objective function (18), we obtain

$$\begin{aligned} f^{(i)}(x) &= \frac{p_2^{(i)} h_1^{2(i)}(x) + p_1^{(i)} h_2^{2(i)} - 2p_3^{(i)} h_1^{(i)}(x) h_2^{(i)}(x)}{p_1^{(i)} p_2^{(i)} - p_3^{2(i)}} \\ h_1^{(i)}(x) &= \hat{x}_{T_i} - x_r^{(\mathrm{R})} + \cos(\alpha)\left(x_r^{(\mathrm{W})} - x_{L_i}^{(\mathrm{W})}\right) \\ &\quad + \sin(\alpha)\left(y_r^{(\mathrm{W})} - x_{L_i}^{(\mathrm{W})}\right) \\ h_2^{(i)}(x) &= \hat{y}_{T_i} - y_r^{(\mathrm{R})} - \sin(\alpha)\left(x_r^{(\mathrm{W})} - x_{L_i}^{(\mathrm{W})}\right) \\ &\quad + \cos(\alpha)\left(y_r^{(\mathrm{W})} - y_{L_i}^{(\mathrm{W})}\right). \end{aligned}$$

The minimization of (18) can now be done numerically with a Newton–Raphson search.[5] As an initial estimate of the state x, the current Kalman filter state prediction $\hat{x}_{k+1|k}$ is used. This estimate can be expected to be close to the optimal solution if the pose tracking is working accurately. When the Newton–Raphson search is run in practice, it usually converges

[5]The necessary gradient and Jacobian expression involved in the Newton–Raphson search are given in the Appendix.

to an optimal solution x^{opt} within two to three iterations. As convergence criterion we use

$$\left|x^{(k+1)} - x^{(k)}\right| < \begin{pmatrix} 0.01\text{ m} \\ 0.01\text{ m} \\ 1^\circ \dfrac{\pi}{180^\circ} \end{pmatrix}.$$

The optimal solution x^{opt} is then taken as a state measurement $y_k^{(1)}$, i.e.,

$$y_k^{(1)} = x^{\text{opt}}.$$

Concerning the measurement covariance matrix $R_k^{(1)}$ a conservative approach is taken by setting constant variances

$$R_k^{(1)} = \begin{pmatrix} (0.84\text{ m})^2 & 0 & 0 \\ 0 & (0.84\text{ m})^2 & 0 \\ 0 & 0 & \left(10^\circ \dfrac{\pi}{180^\circ}\right)^2 \end{pmatrix}.$$

Since the measurements $y_1^{(1)}, y_2^{(1)}, y_3^{(1)}, \ldots,$ in fact are correlated,[6] the noise variances are kept high to avoid the Kalman filter putting too much attention on the measurements. This is of course not an optimal solution, but it has proven to work surprisingly well. Another alternative would of course be to extend the state model to include a correlated noise model, but aside from the problem of finding such appropriate model, this would increase the computational cost, which is something we want to avoid.

2) Landmark Validation: In Section III-B-1, validated landmark pairs $(\hat{T}_i^{(\text{R})}, L_i^{(\text{W})})$ are used to obtain a measurement of the robot pose. In this section it is explained how these landmark pairs have been validated to assure that $\hat{T}_i^{(\text{R})}$ is a triangulation point which represent a measurement of a natural point landmark $L_i^{(\text{W})} \in \mathcal{L}^{(\text{W})}$.

First of all, only triangulation points satisfying $n_t \geq 3$ are considered to be potential measurements of natural point landmarks in the reference map. The reason for choosing a lower threshold compared to when recording the reference map (Section III-A) is that we want to be sure that no landmarks are missed by the TBF algorithm. TBF data that has no correspondence to the reference map landmarks is expected to be sorted away in a two-stage validation process. The first type of validation gate is based on the predicted distance d_{RL} and angle ψ_{RL} between the robot position $(\hat{x}_r^{(\text{W})}, \hat{y}_r^{(\text{W})})$ and a reference landmark $L_i^{(\text{W})}$. If a triangulation point measurement $\hat{T}_i^{(\text{R})}$ should be associated with this landmark it is required that

$$|d_{RL} - d_{RT}| < d_{th}$$
$$|\psi_{RL} - (\psi_{RT} + \hat{\alpha})| < \psi_{th}$$

where d_{RT} and ψ_{RT} is the distance and angle between the robot position $(x_r^{(\text{R})}, y_r^{(\text{R})})$ and the triangulation point measurement $\hat{T}_i^{(\text{R})}$. The thresholds d_{th} and ψ_{th} are state covariance dependent as

$$d_{th} = \min\left\{0.1\text{ m} + \sqrt{\max\{p_{11}, p_{22}\}}, 0.4\text{ m}\right\}$$

[6]Because of a sliding landmark pair set $\mathcal{P}$ described later in text.

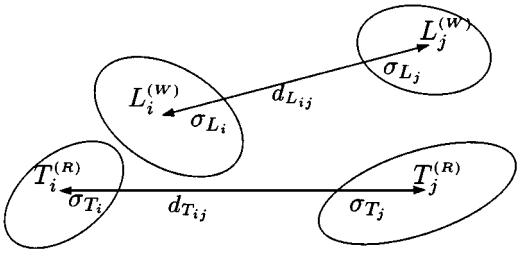

Fig. 9. Using the one standard deviation uncertainty ellipses of the points $\hat{T}_i^{(\text{R})}$, $\hat{T}_j^{(\text{R})}$, $L_i^{(\text{W})}$ and $L_j^{(\text{W})}$ the distribution of the distances $d_{T_{ij}}$ and $d_{L_{ij}}$ can be approximated as in (22) and (23).

$$\psi_{th} = \min\left\{\frac{1}{2}\left(3^\circ \frac{\pi}{180^\circ} + \sqrt{p_{33}} + \frac{d_{th}}{d_{RL}}\right), 15^\circ \frac{\pi}{180^\circ}\right\}.$$

If the triangulation point measurement $\hat{T}_i^{(\text{R})}$ happens to fall within several validation gates, then the closest reference map landmark is chosen as a partner. Using this technique to pair triangulation point measurements with reference map landmarks, a landmark candidate pair set $\mathcal{P}$ is generated

$$\mathcal{P} = \left\{\left(\hat{T}_1^{(\text{R})}, L_1^{(\text{W})}\right), \left(\hat{T}_2^{(\text{R})}, L_2^{(\text{W})}\right), \cdots, \left(\hat{T}_n^{(\text{R})}, L_n^{(\text{W})}\right)\right\}.$$

Here $\mathcal{P}$ is a sliding set where a pair $(\hat{T}_i^{(\text{R})}, L_i^{(\text{W})})$ is removed if a) the robot has moved more than 2 m since detecting the triangulation point $\hat{T}_i^{(\text{R})}$, b) a new measurement of $\hat{T}_i^{(\text{R})}$ appears (closer than 0.1 m), or c) the pair does not fulfill a validation based on relative distances. Step c) is an efficient way to remove most of the bad data associations in the set $\mathcal{P}$. Indeed if the pairs $(\hat{T}_i^{(\text{R})}, L_i^{(\text{W})}), (\hat{T}_j^{(\text{R})}, L_j^{(\text{W})}) \in \mathcal{P}$ are correct data associations, then the distances

$$d_{T_{ij}} = \left|\hat{T}_i^{(\text{R})} - \hat{T}_j^{(\text{R})}\right| \qquad d_{L_{ij}} = \left|L_i^{(\text{W})} - L_j^{(\text{W})}\right|$$

should be almost the same. This fact can be used to remove false data associations from the set $\mathcal{P}$. However, better distance measures can be obtained using the covariance matrices of $\hat{T}_i^{(\text{R})}$, $L_i^{(\text{W})}$, $\hat{T}_j^{(\text{R})}$, and $L_j^{(\text{W})}$ (Fig. 9) to approximate the distributions of the distances $d_{T_{ij}}$ and $d_{L_{ij}}$ as

$$d_{T_{ij}} \sim N\left(\overline{d}, \sqrt{\sigma_{T_i}^2 + \sigma_{T_j}^2 + \sigma_{\text{od}}^2}\right) \tag{22}$$

$$d_{L_{ij}} \sim N\left(\overline{d}, \sqrt{\sigma_{L_i}^2 + \sigma_{L_j}^2}\right). \tag{23}$$

Here $\overline{d}$ is the true distance between the landmarks and $\sigma_{\text{od}} = 15$ mm is an extra term covering up for the odometry drift that may be present between detecting $\hat{T}_i^{(\text{R})}$ and $\hat{T}_j^{(\text{R})}$. Under these circumstances the distribution of $d_{T_{ij}} - d_{L_{ij}}$ is

$$d_{T_{ij}} - d_{L_{ij}} \sim N\left(0, \underbrace{\sqrt{\sigma_{T_i}^2 + \sigma_{T_j}^2 + \sigma_{\text{od}}^2 + \sigma_{L_i}^2 + \sigma_{L_j}^2}}_{\sigma}\right).$$

Hence using the normalized (Mahalanobis) distance

$$d_{M_{ij}} = \frac{|d_{T_{ij}} - d_{L_{ij}}|}{\sigma}$$

a better distance measure is obtained. For a correct data association the threshold

$$d_{M_{ij}} < 3 \tag{24}$$

is used. If this condition is not met, it is interpreted as that either of $(\hat{T}_i^{(\mathrm{R})}, L_i^{(\mathrm{W})})$ and $(\hat{T}_j^{(\mathrm{R})}, L_j^{(\mathrm{W})})$ is an incorrect data association, or both. To find out which landmark pairs in $\mathcal{P}$ that are incorrect, it is necessary to loop over all distances $d_{M_{ij}}$, $i \neq j$. The landmark pair that is most frequent in not satisfying condition (24) should be removed from the set $\mathcal{P}$. This procedure is repeated until all pair combinations in $\mathcal{P}$ meet the condition (24).

After validating the landmark pairs in $\mathcal{P}$ with respect to relative distances, there is a good chance that the surviving landmark pairs of $\mathcal{P}$ are correct data associations. Hence, they could be used to measure the robot pose as explained in Section III-B-1. Note however that $\mathcal{P}$ might just contain a single element $(\hat{T}^{(\mathrm{R})}, L^{(\mathrm{W})})$ after passing the validation based on relative distances. In those cases we cannot tell if $(\hat{T}^{(\mathrm{R})}, L^{(\mathrm{W})})$ is a correct data association or not, and hence no robot pose measurement is performed in that case.

To guarantee that $\mathcal{P}$ is appropriate for updating the robot pose with respect to orientation, there should be some distance between the landmark pairs in $\mathcal{P}$. In the present implementation, $\mathcal{P}$ is approved for robot pose measurement only if

$$\max\{d_{T_{ij}}\} > 1.5 \text{ m}.$$

C. Pose Tracking in Corridors

The just described robot pose measurement $y_k^{(1)}$ is sufficient for pose tracking in most rooms of office or living room size. Usually these kind of rooms contains a dense set of natural point landmarks, making it easy for the robot to update its position. In larger areas, with a sparse set of landmarks, like corridors for instance, these kind of measurements are insufficient. This has to do with the orientation estimate $\hat{\alpha}$ becoming crucial when the robot moves long distances in the same direction. Only a slight error in the orientation estimate will propagate into a large position error if the robot has to move a long distance before detecting new landmarks. In a corridor, where the robot task often is to move from one room to another, the natural point landmarks of the reference map can be limited to door posts. In the worst case, the corridor only contain smooth walls and hence no point landmarks at all are available. Because of these reasons, a second type of measurement of the robot pose $y_k^{(2)}$ is introduced to handle the case when the robot visits a corridor.

The idea is to estimate the robot pose based on measurements of the corridor width and orientation. Suppose the corridor is modeled as a rectangle with a local coordinate system placed at $(x_c^{(\mathrm{W})}, y_c^{(\mathrm{W})})$ (Fig. 10). To be general, the $x^{(\mathrm{LW})}$-axis is assumed to be tilted an angle ϕ relative to the world coordinate x-axis. The relation between these two coordinate systems is then given by

$$\begin{pmatrix} x^{(\mathrm{W})} \\ y^{(\mathrm{W})} \end{pmatrix} = \begin{pmatrix} \cos(\phi) & \sin(\phi) \\ -\sin(\phi) & \cos(\phi) \end{pmatrix} \begin{pmatrix} x^{(\mathrm{LW})} \\ y^{(\mathrm{LW})} \end{pmatrix} + \begin{pmatrix} x_c^{(\mathrm{W})} \\ y_c^{(\mathrm{W})} \end{pmatrix}. \tag{25}$$

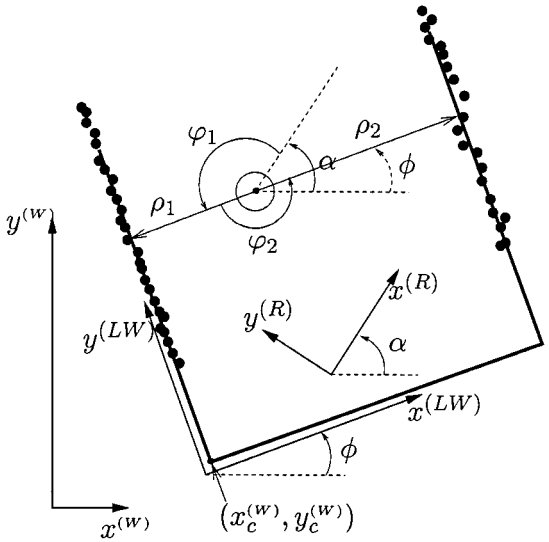

Fig. 10. Corridor wall measurements (ρ_1, φ_1) and (ρ_2, φ_2) can be obtained using local Hough transforms based on triangulation points.

Using the current state estimate $\hat{x}$, the corridor wall positions and orientation can be predicted in robot coordinates in terms of the polar coordinates $(\hat{\rho}_1, \hat{\varphi}_1)$ and $(\hat{\rho}_2, \hat{\varphi}_2)$. Using two (ρ, φ)-grids centered locally around $(\hat{\rho}_1, \hat{\varphi}_1)$ and $(\hat{\rho}_2, \hat{\varphi}_2)$, respectively, a Hough transform [18] based on triangulation points can be performed to actually measure (ρ_1, φ_1) and (ρ_2, φ_2) (Fig. 10). In our implementation a buffer of the 50 most recent detected triangulation points is used in the Hough transforms.[7] This kind of measurement is repeated every time ten new triangulation points are available. Given for instance the measurement (ρ_1, φ_1), the orientation angle α and the local x-coordinate of the robot $x_r^{(\mathrm{LW})}$ can be measured as

$$y_k^{(2)} = \begin{pmatrix} x_r^{(\mathrm{LW})} \\ \alpha \end{pmatrix} = \begin{pmatrix} \rho_1 \\ \pi + \phi - \varphi_1 \end{pmatrix}.$$

Using the transformation (25) this can be expressed in terms of the state variable x as

$$y_k^{(2)} = \begin{pmatrix} \cos(\phi) & \sin(\phi) & 0 \\ 0 & 0 & 1 \end{pmatrix} x \triangleq H^{(2)} x. \tag{26}$$

These kind of corridor wall measurements need of course to be validated before actually updating the Kalman filter. For this purpose the ρ- and φ-validation gates

$$|\rho_1 + \rho_2 - d_{\mathrm{corr}}| < 4\Delta\rho$$
$$\big||\varphi_1 - \varphi_2| - \pi\big| < \Delta\varphi$$

are used where d_{corr} is the corridor width and $\Delta\rho$, $\Delta\varphi$ is the resolution of the local (ρ, φ)-grids. In our implementation $\Delta\rho = 0.05$ m and $\Delta\varphi = 1^\circ$ and the local grids are of size $(d_{\mathrm{corr}}/\Delta\rho) \times 20$. The large ρ-validation gate $(4\Delta\rho)$ is due to the fact that the corridor width actually may vary a bit. The co-

[7]No restriction is put on the parameters n_t and P_T in this case since a triangulation point with low n_t value and large P_T actually is likely to originate from a smooth object, e.g., a corridor wall.

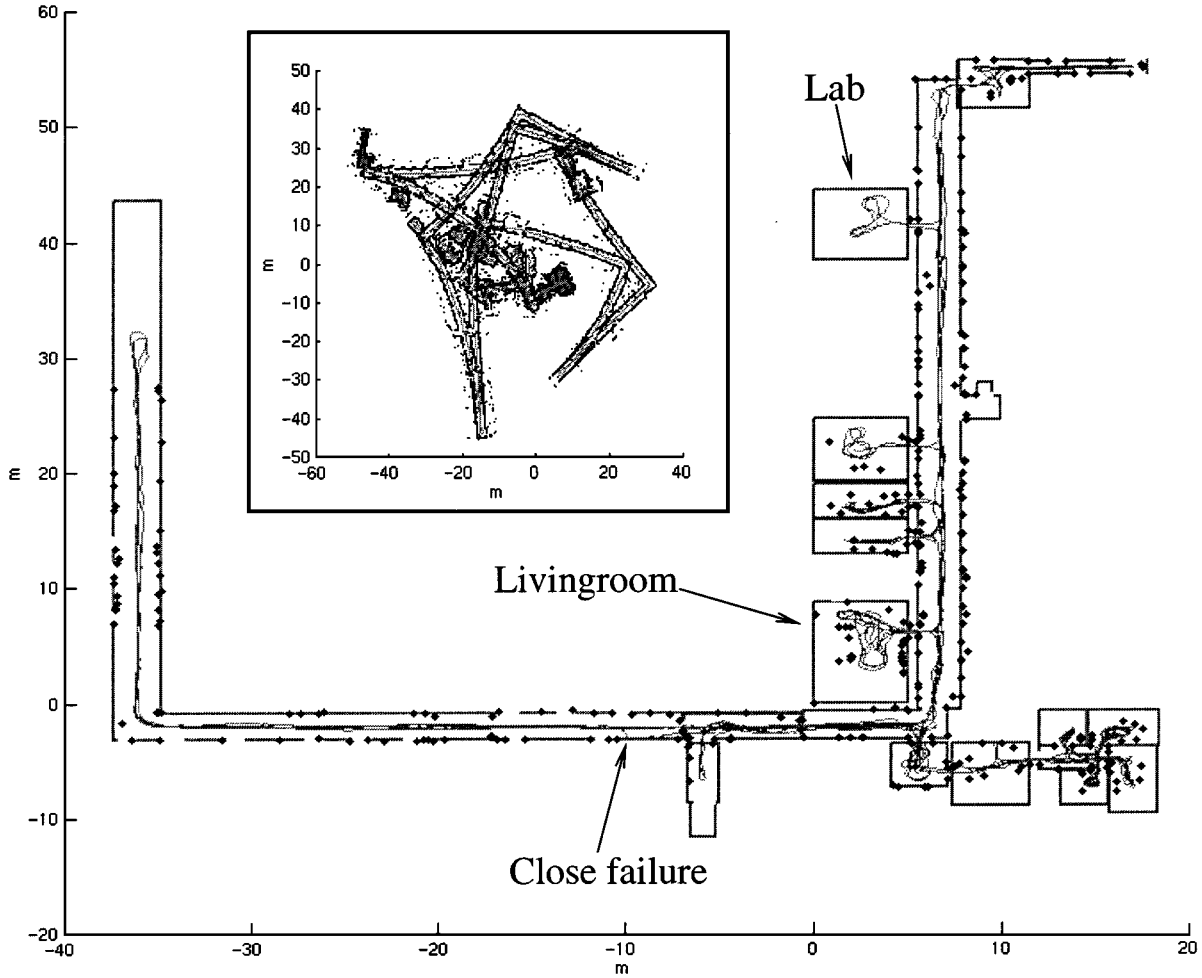

Fig. 11. This figure illustrates the performance of the pose tracker when being applied to a Nomad 200 robot moving over 1 km in a large scale indoor environment. Middle boxed picture shows pure odometric data and the outer picture shows the estimated trajectory in world coordinates placed on top of a hand measured drawing of the building. The dark dots in the outer picture are reference map landmarks that have been used in the pose tracking. See text for more details.

variance matrix $R_k^{(2)}$ of a measurement of type $y_k^{(2)}$ is set to

$$R_k^{(2)} = \begin{pmatrix} (|\rho_1 + \rho_2 - d_{\text{corr}}| + 0.2\,\text{m})^2 & 0 \\ 0 & \left(3^\circ \frac{\pi}{180^\circ}\right)^2 \end{pmatrix}$$

where the 0.2 m again has to do with covering up for varying width of the corridor.

Finally it is worth noting that the complexity of a measurement of type $y_k^{(2)}$ is low because of the Hough transform complexity being linear with the number of points used.

D. Experiments

To test the proposed pose tracker, a large scale indoor environment was chosen, containing 15 rooms and three long corridors. A world coordinate system was defined with the origin in the living room (see Fig. 8). Landmarks were then recorded for each room/corridor according to the guidelines of Section III-A. After this, a 1.8-km odometry and sonar data set was collected for off-line tuning of the pose tracker. The various parameters (validation gates etc.) were then trimmed and chosen as specified in Sections III-B and III-C. A fact that simplified the tuning was that data from a well established laser pose tracker [6] was available which gave centimeter precision. Hence, "ground truth" was known when doing the tuning. After one year (!), a 1-km new odometry, sonar and laser data set was collected from the environment. The pose tracker was then again run off-line using the one year old reference maps (one map per room). Some of the reference maps were up to date and some were not because of refurnishing. The result from this experiment is documented in Fig. 11. The middle boxed picture shows the collected odometry and sonar data. The grey solid line is the robot trajectory and the black dots building up the walls of the rooms and corridors are all the triangulation points produced by the TBF algorithm. In total the orientation angle α drifted about 300° during this data collection, i.e., on average $0.3^\circ/\text{m}$. The drift was, however, not constant since it depended on the configuration of the three wheels of the robot with respect to the current motion direction. In the worst case, the drift was about $0.8^\circ/\text{m}$.

The outer picture shows how the pose tracker has compensated the robot trajectory, which has been plotted on top of a drawing of the environment (hand measured in world coordinates). The black circles appearing in this figure are landmarks in the reference maps that have been used during the pose tracking. As seen from this picture only a few landmarks were matched in certain areas, like for instance the left corridor,

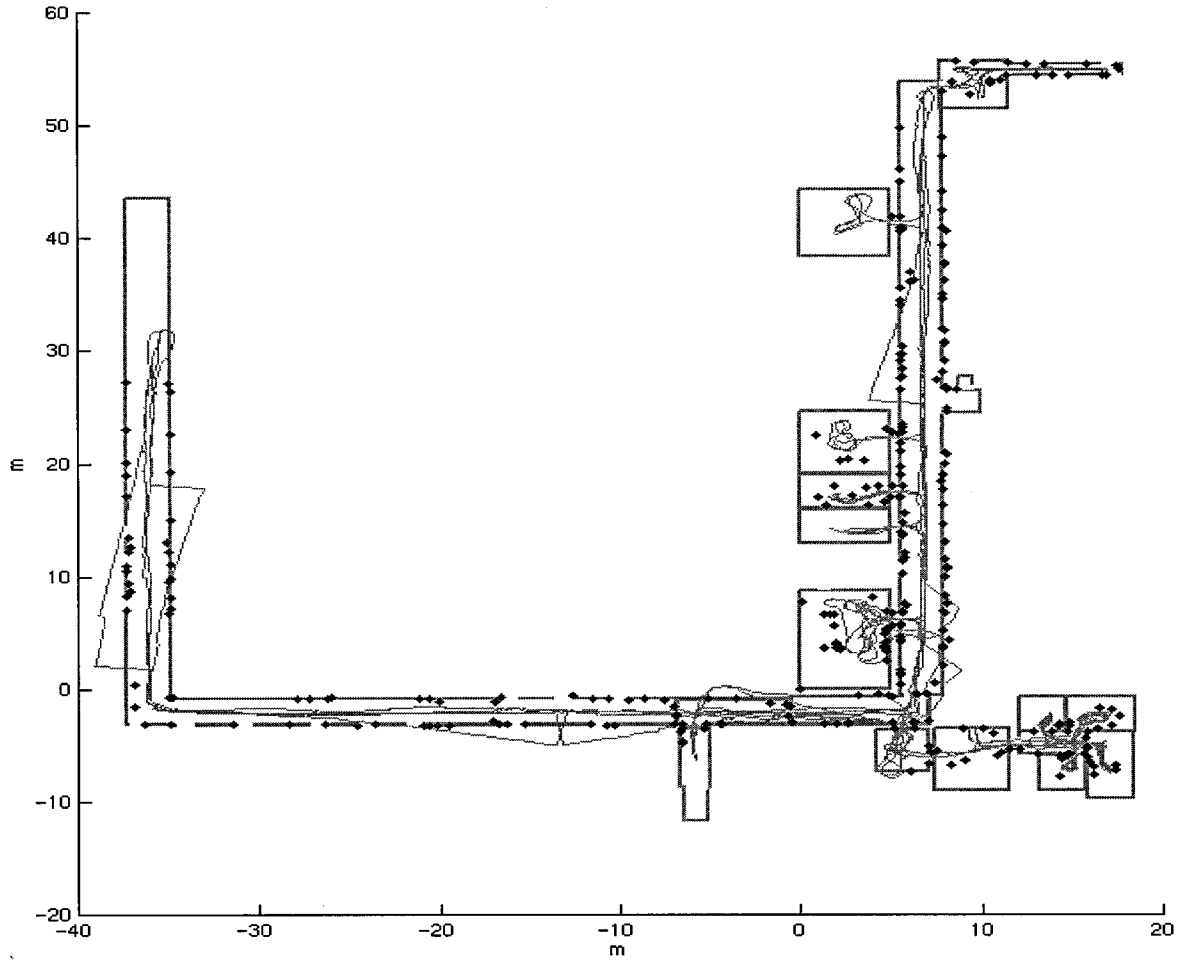

Fig. 12. Repeated experiment from Fig. 11 but without using the wall measurements in the corridors. The degrade in performance is clear. When the estimated robot pose differed from the true pose more than 3 m, the sonar pose tracker was reset. In this run the sonar pose tracker was reset seven times.

which contains few doors, and the room marked "Lab," which had been completely refurnished since the acquisition of the reference map. In the left corridor the pose tracker was very dependent on the Hough transform-based measurements of the orientation angle and robot position. In the Lab the robot had to survive on pure odometry information.

The pose tracker was implemented in a way so that the robot not only used the reference map corresponding to the room it was currently visiting, but also the reference maps of the adjacent rooms. This made it possible to track landmarks in several rooms at the same time, which is important when the robot should change room. The presence of thresholds in door openings can easily cause sudden changes in the robot pose and hence it is important to have access to as many landmark measurements (door posts) as possible in such cases. Since the pose tracker had information about where the doors should be in world coordinates, it could estimate when it actually crossed a door opening. When a room change occurred, or was believed to occur, extra uncertainty was added to the state covariance matrix to cover up for slippage and rotation errors when crossing a threshold

$$p_{11} := (\sqrt{p_{11}} + 0.15\,\mathrm{m})^2$$
$$p_{22} := (\sqrt{p_{22}} + 0.15\,\mathrm{m})^2$$
$$p_{33} := \left(\sqrt{p_{33}} + 1^\circ \frac{\pi}{180^\circ}\right)^2.$$

When comparing the sonar pose tracker data with corresponding data from the laser pose tracker, the state estimates was found to be almost overlapping, differing only a few centimeters in most cases. However, at one instance the sonar pose tracker failed badly in its estimate. If looking closer at Fig. 11 an arrow marks a place where the sonar pose tracking estimate almost diverged (almost crossing corridor wall) but being saved by Hough measurement corrections against the corridor walls. It was the lack of Hough measurements earlier on that caused a somewhat bad orientation estimate propagating into a large position error. The lack of hough measurements could be blamed on that in an earlier part of this corridor one of the walls was missing, which had not been modeled in our map.

To illustrate how the corridor wall measurements influence the performance of the pose tracker, the experiment was repeated, but with the Hough component removed. The result is illustrated in Fig. 12 where the degrade in performace is clear. In this experiment, the sonar pose tracker was reseted by the laser pose tracker whenever the robot pose estimate differed more than 3 m. This happened seven times during the run. Hence, wall measurements are necessary for acceptable performance in corridors. Note that this result implies that the presented pose tracker may perform badly in a large room with a sparse set of landmarks and no wall measurements available. From the experiment in Fig. 12, the just mentioned example is best illustrated

in the left vertical corridor. The pose tracker diverged twice in this corridor when not having access to wall measurements, both times due to an orientation error propagated into a large position error when the robot passed areas with only a few landmarks. Although this corridor contain an area with a dense population of landmarks, the position error had already become too large when the robot entered this area. Basically the robot becomes lost when the position error has grown above the maximum size of the landmark validation gates. This fact can be used to trig local re-localization of the robot. Indeed it is strong evidence for divergence of the Kalman filter if the landmark validation gates have grown to maximum size and the robot still fails to validate landmarks, although being in a dense landmark area (implied by state estimate and reference maps). When re-localizing the robot locally or globally, one could for instance use old Monte Carlo techniques that lately have become popular in mobile robotics [4], [8], [9]. Contrary to the Kalman filter, these methods can handle a multimodal non-Gaussian distribution of the robot pose, but at the expense of an increased computational cost.

IV. Conclusions

A sensor fusion scheme called TBF of sonar data has been presented. The method produce stable estimates of static (vertical) edges in the environment and can be run at a low computational cost. There are many applications for the TBF algorithm in mobile robotics, for instance mapping and localization. In this article the case of robot pose tracking was studied. The performance of the pose tracker was shown to be robust in most parts of a large scale indoor office environment. The experimental section also indicated under what circumstances the pose tracker might fail, i.e., in large rooms with areas containing only a sparse set of landmarks. For more information about the TBF algorithm and its application areas, see [8], [20], and [21].

Appendix
Gradient and Jacobean Expressions

In the Newton–Raphson minimization of the object function (18) the gradient and Jacobian expression of $f^{(i)}(x)$ is needed. These formulas are given as follows:

$$\nabla f^{(i)}(x) = \frac{2}{p_1^{(i)} p_2^{(i)} - p_3^{2(i)}} \cdot \begin{pmatrix} g_1^{(i)}(x)\cos(\alpha) - g_2^{(i)}(x)\sin(\alpha) \\ g_1^{(i)}(x)\sin(\alpha) + g_2^{(i)}(x)\cos(\alpha) \\ g_1^{(i)}(x)\dfrac{dh_1^{(i)}}{d\alpha} + g_2^{(i)}(x)\dfrac{dh_2^{(i)}}{d\alpha} \end{pmatrix}$$

$$g_1^{(i)}(x) = p_2^{(i)} h_1^{(i)}(x) - p_3^{(i)} h_2^{(i)}(x)$$
$$g_2^{(i)}(x) = p_2^{(i)} h_2^{(i)}(x) - p_3^{(i)} h_1^{(i)}(x)$$

$$\frac{dh_1^{(i)}}{d\alpha} = -\sin(\alpha)\left(x_r^{(\mathrm{W})} - x_{L_i}^{(\mathrm{W})}\right) + \cos(\alpha)\left(y_r^{(\mathrm{W})} - y_{L_i}^{(\mathrm{W})}\right)$$
$$\frac{dh_2^{(i)}}{d\alpha} = -\cos(\alpha)\left(x_r^{(\mathrm{W})} - x_{L_i}^{(\mathrm{W})}\right) + \sin(\alpha)\left(y_r^{(\mathrm{W})} - y_{L_i}^{(\mathrm{W})}\right)$$

$$J_f^{(i)}(x) = \frac{2}{p_1^{(i)} p_2^{(i)} - p_3^{2(i)}} \begin{pmatrix} J_{11}^{(i)} & J_{12}^{(i)} & J_{13}^{(i)} \\ J_{21}^{(i)} & J_{22}^{(i)} & J_{23}^{(i)} \\ J_{31}^{(i)} & J_{32}^{(i)} & J_{33}^{(i)} \end{pmatrix}$$

$$J_{11}^{(i)} = \frac{dg_1^{(i)}}{dx_r^{(\mathrm{W})}}\cos(\alpha) - \frac{dg_2^{(i)}}{dx_r^{(\mathrm{W})}}\sin(\alpha)$$
$$J_{21}^{(i)} = \frac{dg_1^{(i)}}{dx_r^{(\mathrm{W})}}\sin(\alpha) + \frac{dg_2^{(i)}}{dx_r^{(\mathrm{W})}}\cos(\alpha)$$
$$J_{31}^{(i)} = \frac{dg_1^{(i)}}{dx_r^{(\mathrm{W})}}\frac{dh_1^{(i)}}{d\alpha} - g_1^{(i)}(x)\sin(\alpha) + \frac{dg_2^{(i)}}{dx_r^{(\mathrm{W})}}\frac{dh_2^{(i)}}{d\alpha} - g_2^{(i)}(x)\cos(\alpha)$$
$$J_{12}^{(i)} = \frac{dg_1^{(i)}}{dy_r^{(\mathrm{W})}}\cos(\alpha) - \frac{dg_2^{(i)}}{dy_r^{(\mathrm{W})}}\sin(\alpha)$$
$$J_{22}^{(i)} = \frac{dg_1^{(i)}}{dy_r^{(\mathrm{W})}}\sin(\alpha) + \frac{dg_2^{(i)}}{dy_r^{(\mathrm{W})}}\cos(\alpha)$$
$$J_{32}^{(i)} = \frac{dg_1^{(i)}}{dy_r^{(\mathrm{W})}}\frac{dh_1^{(i)}}{d\alpha} + g_1^{(i)}(x)\cos(\alpha) + \frac{dg_2^{(i)}}{dy_r^{(\mathrm{W})}}\frac{dh_2^{(i)}}{d\alpha} - g_2^{(i)}(x)\sin(\alpha)$$
$$J_{13}^{(i)} = \frac{dg_1^{(i)}}{d\alpha}\cos(\alpha) - \frac{dg_2^{(i)}}{d\alpha}\sin(\alpha)$$
$$J_{23}^{(i)} = \frac{dg_1^{(i)}}{d\alpha}\sin(\alpha) + \frac{dg_2^{(i)}}{d\alpha}\cos(\alpha)$$
$$J_{33}^{(i)} = \frac{dg_1^{(i)}}{d\alpha}\frac{dh_1^{(i)}}{d\alpha} + g_1^{(i)}(x)\frac{dh_2^{(i)}}{d\alpha} + \frac{dg_2^{(i)}}{d\alpha}\frac{dh_2^{(i)}}{d\alpha} - g_2^{(i)}(x)\frac{dh_1^{(i)}}{d\alpha}$$

$$\frac{dg_1^{(i)}}{dx_r^{(\mathrm{W})}} = p_2^{(i)}\cos(\alpha) + p_3^{(i)}\sin(\alpha)$$
$$\frac{dg_1^{(i)}}{dy_r^{(\mathrm{W})}} = p_2^{(i)}\sin(\alpha) - p_3^{(i)}\cos(\alpha)$$
$$\frac{dg_1^{(i)}}{d\alpha} = p_2^{(i)}\frac{dh_1^{(i)}}{d\alpha} = p_3^{(i)}\frac{dh_2^{(i)}}{d\alpha}$$
$$\frac{dg_2^{(i)}}{dx_r^{(\mathrm{W})}} = -p_1^{(i)}\sin(\alpha) - p_3^{(i)}\cos(\alpha)$$
$$\frac{dg_2^{(i)}}{dx_r^{(\mathrm{W})}} = p_1^{(i)}\cos(\alpha) - p_3^{(i)}\sin(\alpha)$$
$$\frac{dg_2^{(i)}}{\alpha} = p_1^{(i)}\frac{dh_2^{(i)}}{d\alpha} - p_3^{(i)}\frac{dh_1^{(i)}}{d\alpha}.$$

References

[1] J. Borenstein and Y. Koren, "The vector field histogram—Fast obstacle avoidance for mobile robots," *IEEE Trans. Robot. Automat.*, vol. 7, pp. 278–288, June 1991.

[2] ——, "Error eliminating rapid ultrasonic firing for mobile robot obstacle avoidance," *IEEE Trans. Robot. Automat.*, vol. 11, pp. 132–138, Feb. 1995.

[3] J. L. Crowley, "World modeling and position estimation for a mobile robot using ultrasonic ranging," in *Proc. IEEE Int. Conf. Robotics and Automation*, vol. 2, May 1989, pp. 674–680.

[4] F. Dellaert, D. Fox, W. Burgard, and S. Thrun, "Monte Carlo localization for mobile robots," in *Proc. IEEE Int. Conf. Robotics and Automation*, vol. 2, May 1999, pp. 1322–1328.
[5] A. Elfes, "Sonar-based real-world mapping and navigation," *IEEE J. Robot. Automat.*, vol. RA-3, pp. 249–265, June 1987.
[6] P. Jensfelt, "Localization using laser scanning and minimalistic environmental models," Licentiate thesis, Automat. Contr. Dept., Royal Inst. Technol., Stockholm, Sweden, Apr. 1999.
[7] P. Jensfelt, D. Austin, and H. I. Christensen, "Toward task oriented localization," in *Proc. IAS-6*, Venice, Italy, July 2000.
[8] P. Jensfelt, D. Austin, O. Wijk, and M. Andersson, "Feature based condensation for mobile robot localization," in *Proc. IEEE Int. Conf. Robotics and Automation*, vol. 3, San Francisco, CA, Apr. 2000, pp. 2531–2537.
[9] P. Jensfelt, O. Wijk, D. Austin, and M. Andersson, "Experiments on augmenting condensation for mobile robot localization," in *Proc. IEEE Int. Conf. Robotics and Automation*, vol. 3, San Francisco, CA, Apr. 2000, pp. 2518–2524.
[10] L. Kleeman and R. Kuc, "An optimal sonar array for target localization and classification," in *Proc. IEEE Int. Conf. Robotics and Automation*, vol. 4, Los Alamitos, CA, 1994, pp. 3130–3135.
[11] K. S. Chong and L. Kleeman, "Accurate odometry and error modeling for a mobile robot," in *Proc IEEE Int. Conf. Robotics and Automation*, vol. 4, Albuquerque, NM, Apr. 1997, pp. 2783–2788.
[12] L. Kleeman, "Fast and accurate sonar trackers using double pulse coding," in *Proc. IEEE/JRS Int. Conf. Intelligent Robots and Systems*, vol. 3, Piscataway, NJ, 1999, pp. 1185–1190.
[13] R. Kuc and M. W. Siegel, "Physically based simulation model for acoustic sensor robot navigation," *IEEE Trans. Pattern Anal. Machine Intell.*, vol. PAMI-9, no. 6, pp. 766–778, 1987.
[14] B. Barshan and R. Kuc, "Differentiating sonar reflections from corners and planes by employing an intelligent sensor," *IEEE Trans. Pattern Anal. Machine Intell.*, vol. 12, no. 6, pp. 560–569, 1990.
[15] O. Bozma and R. Kuc, "A physical model-based analysis of heterogeneous environment using sonar—ENDURA method," *IEEE Trans. Pattern Anal. Machine Intell.*, vol. 16, no. 5, pp. 497–506, 1994.
[16] J. Leonard and H. Durrant-Whyte, *Directed Sonar Sensing for Mobile Robot Navigation*. Norwell, MA: Kluwer, 1992.
[17] H. Peremans, K. Audenaert, and J. M. van Campenhout, "A high-resolution sensor based on tri-aural perception," *IEEE Trans. Robot. Automat.*, vol. 9, pp. 36–38, Feb. 1993.
[18] T. Risse, "Hough transform for line recognition," *Comput.Vision Image Processing*, vol. 46, pp. 327–345, 1989.
[19] T. Yata, A. Ohya, and S. Yuta, "A fast and accurate sonar-ring sensor for a mobile robot," in *Proc. IEEE Int. Conf. Robotics and Automation*, vol. 1, Detroit, MI, 1999, pp. 630–636.
[20] O. Wijk, P. Jensfelt, and H. I. Christensen, "Triangulation based fusion of ultrasonic sensor data," in *Proc. IEEE Int. Conf. Robotics and Automation*, vol. 4, Leuven, Belgium, May 1998, pp. 3419–3424.
[21] O. Wijk, "Triangulation based fusion of sonar data with application in mobile robot mapping and localization," Ph.D. dissertation, Automat. Contr. Dept., Royal Inst. Technol., Stockholm, Sweden, 2001, submitted for publication.

Olle Wijk (S'00) received the M.Sc. degree from the Royal Institute of Technology, Stockholm, Sweden, in 1996. He is currently pursuing the Ph.D. degree at the Centre for Autonomous Systems, Royal Institute of Technology.

His research interests are primarily sensor fusion, mobile robot localization, and navigation.

Henrik I. Christensen (M'87) received the M.Sc. and Ph.D. degrees from Aalborg University, Aalborg, Denmark, in 1987 and 1989, respectively.

He is a Chaired Professor of Computer Science at the Royal Institute of Technology, Stockholm, Sweden. He is also Director of the Centre for Autonomous Systems that does research on mobile robotics for domestic and outdoor applications. He has held appointments at Aalborg University, University of Pennsylvania, and Oak Ridge National Laboratory. His primary research interest is now in system integration for vision and mobile robotics.