# Real-Time Loop Closure in 2D LIDAR SLAM

Wolfgang Hess[1], Damon Kohler[1], Holger Rapp[1], Daniel Andor[1]

*Abstract*— **Portable laser range-finders, further referred to as LIDAR, and simultaneous localization and mapping (SLAM) are an efficient method of acquiring as-built floor plans. Generating and visualizing floor plans in real-time helps the operator assess the quality and coverage of capture data. Building a portable capture platform necessitates operating under limited computational resources. We present the approach used in our backpack mapping platform which achieves real-time mapping and loop closure at a 5 cm resolution. To achieve real-time loop closure, we use a branch-and-bound approach for computing scan-to-submap matches as constraints. We provide experimental results and comparisons to other well known approaches which show that, in terms of quality, our approach is competitive with established techniques.**

## I. Introduction

As-built floor plans are useful for a variety of applications. Manual surveys to collect this data for building management tasks typically combine computed-aided design (CAD) with laser tape measures. These methods are slow and, by employing human preconceptions of buildings as collections of straight lines, do not always accurately describe the true nature of the space. Using SLAM, it is possible to swiftly and accurately survey buildings of sizes and complexities that would take orders of magnitude longer to survey manually.

Applying SLAM in this field is not a new idea and is not the focus of this paper. Instead, the contribution of this paper is a novel method for reducing the computational requirements of computing loop closure constraints from laser range data. This technique has enabled us to map very large floors, tens-of-thousands of square meters, while providing the operator fully optimized results in real-time.

## II. Related work

Scan-to-scan matching is frequently used to compute relative pose changes in laser-based SLAM approaches, for example [1]–[4]. On its own, however, scan-to-scan matching quickly accumulates error.

Scan-to-map matching helps limit this accumulation of error. One such approach, which uses Gauss-Newton to find local optima on a linearly interpolated map, is [5]. In the presence of good initial estimates for the pose, provided in this case by using a sufficiently high data rate LIDAR, locally optimized scan-to-map matching is efficient and robust. On unstable platforms, the laser fan is projected onto the horizontal plane using an inertial measurement unit (IMU) to estimate the orientation of gravity.

Pixel-accurate scan matching approaches, such as [1], further reduce local error accumulation. Although computationally more expensive, this approach is also useful for loop closure detection. Some methods focus on improving on the computational cost by matching on extracted features from the laser scans [4]. Other approaches for loop closure detection include histogram-based matching [6], feature detection in scan data, and using machine learning [7].

Two common approaches for addressing the remaining local error accumulation are particle filter and graph-based SLAM [2], [8].

Particle filters must maintain a representation of the full system state in each particle. For grid-based SLAM, this quickly becomes resource intensive as maps become large; e.g. one of our test cases is $22{,}000\,\mathrm{m}^2$ collected over a $3\,\mathrm{km}$ trajectory. Smaller dimensional feature representations, such as [9], which do not require a grid map for each particle, may be used to reduce resource requirements. When an up-to-date grid map is required, [10] suggests computing submaps, which are updated only when necessary, such that the final map is the rasterization of all submaps.

Graph-based approaches work over a collection of nodes representing poses and features. Edges in the graph are constraints generated from observations. Various optimization methods may be used to minimize the error introduced by all constraints, e.g. [11], [12]. Such a system for outdoor SLAM that uses a graph-based approach, local scan-to-scan matching, and matching of overlapping local maps based on histograms of submap features is described in [13].

## III. System overview

Google's *Cartographer* provides a real-time solution for indoor mapping in the form of a sensor equipped backpack that generates 2D grid maps with a $r = 5\,\mathrm{cm}$ resolution. The operator of the system can see the map being created while walking through a building. Laser scans are inserted into a *submap* at the best estimated position, which is assumed to be sufficiently accurate for short periods of time. Scan matching happens against a recent submap, so it only depends on recent scans, and the error of pose estimates in the world frame accumulates.

To achieve good performance with modest hardware requirements, our SLAM approach does not employ a particle filter. To cope with the accumulation of error, we regularly run a *pose optimization*. When a submap is finished, that is no new scans will be inserted into it anymore, it takes part in scan matching for loop closure. All finished submaps and scans are automatically considered for *loop closure*. If they are close enough based on current pose estimates, a scan matcher tries to find the scan in the submap. If a sufficiently good match is found in a search window around the currently estimated pose, it is added as a *loop closing constraint* to the

[1]All authors are at Google.

optimization problem. By completing the optimization every few seconds, the experience of an operator is that loops are closed immediately when a location is revisited. This leads to the soft real-time constraint that the loop closure scan matching has to happen quicker than new scans are added, otherwise it falls behind noticeably. We achieve this by using a branch-and-bound approach and several precomputed grids per finished submap.

## IV. Local 2D SLAM

Our system combines separate local and global approaches to 2D SLAM. Both approaches optimize the pose, $\xi = (\xi_x, \xi_y, \xi_\theta)$ consisting of a $(x, y)$ translation and a rotation $\xi_\theta$, of LIDAR observations, which are further referred to as scans. On an unstable platform, such as our backpack, an IMU is used to estimate the orientation of gravity for projecting scans from the horizontally mounted LIDAR into the 2D world.

In our local approach, each consecutive scan is matched against a small chunk of the world, called a submap $M$, using a non-linear optimization that aligns the scan with the submap; this process is further referred to as scan matching. Scan matching accumulates error over time that is later removed by our global approach, which is described in Section V.

### A. Scans

Submap construction is the iterative process of repeatedly aligning scan and submap coordinate frames, further referred to as frames. With the origin of the scan at $\mathbf{0} \in \mathbb{R}^2$, we now write the information about the scan points as $H = \{h_k\}_{k=1,\dots,K}, h_k \in \mathbb{R}^2$. The pose $\xi$ of the scan frame in the submap frame is represented as the transformation $T_\xi$, which rigidly transforms scan points from the scan frame into the submap frame, defined as

$$T_\xi p = \underbrace{\begin{pmatrix} \cos \xi_\theta & -\sin \xi_\theta \\ \sin \xi_\theta & \cos \xi_\theta \end{pmatrix}}_{R_\xi} p + \underbrace{\begin{pmatrix} \xi_x \\ \xi_y \end{pmatrix}}_{t_\xi}. \tag{1}$$

### B. Submaps

A few consecutive scans are used to build a submap. These submaps take the form of probability grids $M : r\mathbb{Z} \times r\mathbb{Z} \to [p_{\min}, p_{\max}]$ which map from discrete grid points at a given resolution $r$, for example 5 cm, to values. These values can be thought of as the probability that a grid point is obstructed. For each grid point, we define the corresponding *pixel* to consist of all points that are closest to that grid point.

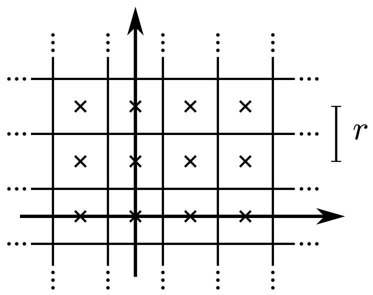

Fig. 1. Grid points and associated pixels.

Whenever a scan is to be inserted into the probability grid, a set of grid points for *hits* and a disjoint set for *misses* are computed. For every hit, we insert the closest grid point into the hit set. For every miss, we insert the grid point associated with each pixel that intersects one of the rays between the scan origin and each scan point, excluding grid points which are already in the hit set. Every formerly unobserved grid point is assigned a probability $p_{\text{hit}}$ or $p_{\text{miss}}$ if it is in one of these sets. If the grid point $x$ has already been observed, we update the odds for hits and misses as

$$\text{odds}(p) = \frac{p}{1-p}, \tag{2}$$

$$M_{\text{new}}(x) = \text{clamp}(\text{odds}^{-1}(\text{odds}(M_{\text{old}}(x)) \cdot \text{odds}(p_{\text{hit}}))) \tag{3}$$

and equivalently for misses.

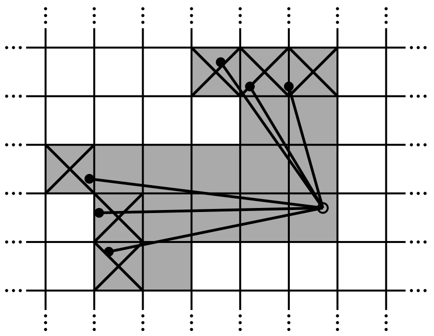

Fig. 2. A scan and pixels associated with *hits* (shaded and crossed out) and *misses* (shaded only).

### C. Ceres scan matching

Prior to inserting a scan into a submap, the scan pose $\xi$ is optimized relative to the current local submap using a *Ceres*-based [14] scan matcher. The scan matcher is responsible for finding a scan pose that maximizes the probabilities at the scan points in the submap. We cast this as a nonlinear least squares problem

$$\underset{\xi}{\text{argmin}} \quad \sum_{k=1}^{K} \left(1 - M_{\text{smooth}}(T_\xi h_k)\right)^2 \tag{CS}$$

where $T_\xi$ transforms $h_k$ from the scan frame to the submap frame according to the scan pose. The function $M_{\text{smooth}} : \mathbb{R}^2 \to \mathbb{R}$ is a smooth version of the probability values in the local submap. We use bicubic interpolation. As a result, values outside the interval $[0, 1]$ can occur but are considered harmless.

Mathematical optimization of this smooth function usually gives better precision than the resolution of the grid. Since this is a local optimization, good initial estimates are required. An IMU capable of measuring angular velocities can be used to estimate the rotational component $\theta$ of the pose between scan matches. A higher frequency of scan matches or a pixel-accurate scan matching approach, although more computationally intensive, can be used in the absence of an IMU.

## V. Closing loops

As scans are only matched against a submap containing a few recent scans, the approach described above slowly accumulates error. For only a few dozen consecutive scans, the accumulated error is small.

Larger spaces are handled by creating many small submaps. Our approach, optimizing the poses of all scans and submaps, follows Sparse Pose Adjustment [2]. The relative poses where scans are inserted are stored in memory for use in the loop closing optimization. In addition to these relative poses, all other pairs consisting of a scan and a submap are considered for loop closing once the submap no longer changes. A scan matcher is run in the background and if a good match is found, the corresponding relative pose is added to the optimization problem.

### A. Optimization problem

Loop closure optimization, like scan matching, is also formulated as a nonlinear least squares problem which allows easily adding residuals to take additional data into account. Once every few seconds, we use Ceres [14] to compute a solution to

$$\underset{\Xi^{\mathrm{m}},\Xi^{\mathrm{s}}}{\operatorname{argmin}} \quad \frac{1}{2}\sum_{ij} \rho\big(E^2(\xi_i^{\mathrm{m}}, \xi_j^{\mathrm{s}}; \Sigma_{ij}, \xi_{ij})\big) \tag{SPA}$$

where the submap poses $\Xi^{\mathrm{m}} = \{\xi_i^{\mathrm{m}}\}_{i=1,\ldots,m}$ and the scan poses $\Xi^{\mathrm{s}} = \{\xi_j^{\mathrm{s}}\}_{j=1,\ldots,n}$ in the world are optimized given some *constraints*. These constraints take the form of relative poses $\xi_{ij}$ and associated covariance matrices $\Sigma_{ij}$. For a pair of submap $i$ and scan $j$, the pose $\xi_{ij}$ describes where in the submap coordinate frame the scan was matched. The covariance matrices $\Sigma_{ij}$ can be evaluated, for example, following the approach in [15], or locally using the covariance estimation feature of Ceres [14] with (CS). The residual $E$ for such a constraint is computed by

$$E^2(\xi_i^{\mathrm{m}}, \xi_j^{\mathrm{s}}; \Sigma_{ij}, \xi_{ij}) = e(\xi_i^{\mathrm{m}}, \xi_j^{\mathrm{s}}; \xi_{ij})^T \Sigma_{ij}^{-1} e(\xi_i^{\mathrm{m}}, \xi_j^{\mathrm{s}}; \xi_{ij}), \tag{4}$$

$$e(\xi_i^{\mathrm{m}}, \xi_j^{\mathrm{s}}; \xi_{ij}) = \xi_{ij} - \begin{pmatrix} R_{\xi_i^{\mathrm{m}}}^{-1}(t_{\xi_i^{\mathrm{m}}} - t_{\xi_j^{\mathrm{s}}}) \\ \xi_{i;\theta}^{\mathrm{m}} - \xi_{j;\theta}^{\mathrm{s}} \end{pmatrix}. \tag{5}$$

A loss function $\rho$, for example *Huber loss*, is used to reduce the influence of outliers which can appear in (SPA) when scan matching adds incorrect constraints to the optimization problem. For example, this may happen in locally symmetric environments, such as office cubicles. Alternative approaches to outliers include [16].

### B. Branch-and-bound scan matching

We are interested in the optimal, pixel-accurate match

$$\xi^\star = \underset{\xi\in\mathcal{W}}{\operatorname{argmax}} \sum_{k=1}^{K} M_{\text{nearest}}(T_\xi h_k), \tag{BBS}$$

where $\mathcal{W}$ is the search window and $M_{\text{nearest}}$ is $M$ extended to all of $\mathbb{R}^2$ by rounding its arguments to the nearest grid point first, that is extending the value of a grid points to the corresponding pixel. The quality of the match can be improved further using (CS).

Efficiency is improved by carefully choosing step sizes. We choose the angular step size $\delta_\theta$ so that scan points at the maximum range $d_{\max}$ do not move more than $r$, the width of one pixel. Using the law of cosines, we derive

$$d_{\max} = \max_{k=1,\ldots,K} \|h_k\|, \tag{6}$$

$$\delta_\theta = \arccos\left(1 - \frac{r^2}{2d_{\max}^2}\right). \tag{7}$$

We compute an integral number of steps covering given linear and angular search window sizes, e.g., $W_x = W_y = 7\,\mathrm{m}$ and $W_\theta = 30°$,

$$w_x = \left\lceil \frac{W_x}{r} \right\rceil, \quad w_y = \left\lceil \frac{W_y}{r} \right\rceil, \quad w_\theta = \left\lceil \frac{W_\theta}{\delta_\theta} \right\rceil. \tag{8}$$

This leads to a finite set $\mathcal{W}$ forming a search window around an estimate $\xi_0$ placed in its center,

$$\overline{\mathcal{W}} = \{-w_x, \ldots, w_x\} \times \{-w_y, \ldots, w_y\} \times \{-w_\theta, \ldots, w_\theta\}, \tag{9}$$

$$\mathcal{W} = \{\xi_0 + (rj_x, rj_y, \delta_\theta j_\theta) : (j_x, j_y, j_\theta) \in \overline{\mathcal{W}}\}. \tag{10}$$

A naive algorithm to find $\xi^\star$ can easily be formulated, see Algorithm 1, but for the search window sizes we have in mind it would be far too slow.

**Algorithm 1** Naive algorithm for (BBS)

```
$best\_score \leftarrow -\infty$
for $j_x = -w_x$ to $w_x$ do
    for $j_y = -w_y$ to $w_y$ do
        for $j_\theta = -w_\theta$ to $w_\theta$ do
            $score \leftarrow \sum_{k=1}^{K} M_{\text{nearest}}(T_{\xi_0+(rj_x,rj_y,\delta_\theta j_\theta)} h_k)$
            if $score > best\_score$ then
                $match \leftarrow \xi_0 + (rj_x, rj_y, \delta_\theta j_\theta)$
                $best\_score \leftarrow score$
            end if
        end for
    end for
end for
return $best\_score$ and $match$ when set.
```

Instead, we use a branch and bound approach to efficiently compute $\xi^\star$ over larger search windows. See Algorithm 2 for the generic approach. This approach was first suggested in the context of mixed integer linear programs [17]. Literature on the topic is extensive; see [18] for a short overview.

The main idea is to represent subsets of possibilities as nodes in a tree where the root node represents all possible solutions, $\mathcal{W}$ in our case. The children of each node form a partition of their parent, so that they together represent the same set of possibilities. The leaf nodes are singletons; each represents a single feasible solution. Note that the algorithm is exact. It provides the same solution as the naive approach, as long as the $score(c)$ of inner nodes $c$ is an upper bound on the score of its elements. In that case, whenever a node is *bounded*, a solution better than the best known solution so far does not exist in this subtree.

To arrive at a concrete algorithm, we have to decide on the method of *node selection*, *branching*, and computation of *upper bounds*.

*1) Node selection:* Our algorithm uses depth-first search (DFS) as the default choice in the absence of a better alternative: The efficiency of the algorithm depends on a large part of the tree being pruned. This depends on two things: a good upper bound, and a good current solution. The latter part is helped by DFS, which quickly evaluates many leaf nodes. Since we do not want to add poor matches as loop closing constraints, we also introduce a score threshold below which we are not interested in the optimal solution. Since in practice the threshold will not often be surpassed, this reduces the importance of the node selection or finding an initial heuristic solution. Regarding the order in which the children are visited during the DFS, we compute the upper bound on the score for each child, visiting the most promising child node with the largest bound first. This method is Algorithm 3.

*2) Branching rule:* Each node in the tree is described by a tuple of integers $c = (c_x, c_y, c_\theta, c_h) \in \mathbb{Z}^4$. Nodes at height $c_h$ combine up to $2^{c_h} \times 2^{c_h}$ possible translations but represent a specific rotation:

$$\overline{\overline{\mathcal{W}}}_c = \left( \left\{ (j_x, j_y) \in \mathbb{Z}^2 : \begin{array}{l} c_x \le j_x < c_x + 2^{c_h} \\ c_y \le j_y < c_y + 2^{c_h} \end{array} \right\} \times \{c_\theta\} \right), \tag{11}$$

$$\overline{\mathcal{W}}_c = \overline{\overline{\mathcal{W}}}_c \cap \overline{\mathcal{W}}. \tag{12}$$

**Algorithm 2** Generic branch and bound

```
$best\_score \leftarrow -\infty$
$\mathcal{C} \leftarrow \mathcal{C}_0$
while $\mathcal{C} \neq \emptyset$ do
    Select a node $c \in \mathcal{C}$ and remove it from the set.
    if $c$ is a leaf node then
        if $score(c) > best\_score$ then
            $solution \leftarrow n$
            $best\_score \leftarrow score(c)$
        end if
    else
        if $score(c) > best\_score$ then
            Branch: Split $c$ into nodes $\mathcal{C}_c$.
            $\mathcal{C} \leftarrow \mathcal{C} \cup \mathcal{C}_c$
        else
            Bound.
        end if
    end if
end while
return $best\_score$ and $solution$ when set.
```

**Algorithm 3** DFS branch and bound scan matcher for (BBS)

```
$best\_score \leftarrow score\_threshold$
Compute and memorize a score for each element in $\mathcal{C}_0$.
Initialize a stack $\mathcal{C}$ with $\mathcal{C}_0$ sorted by score, the maximum score at the top.
while $\mathcal{C}$ is not empty do
    Pop $c$ from the stack $\mathcal{C}$.
    if $score(c) > best\_score$ then
        if $c$ is a leaf node then
            $match \leftarrow \xi_c$
            $best\_score \leftarrow score(c)$
        else
            Branch: Split $c$ into nodes $\mathcal{C}_c$.
            Compute and memorize a score for each element in $\mathcal{C}_c$.
            Push $\mathcal{C}_c$ onto the stack $\mathcal{C}$, sorted by score, the maximum score last.
        end if
    end if
end while
return $best\_score$ and $match$ when set.
```

Leaf nodes have height $c_h = 0$, and correspond to feasible solutions $\mathcal{W} \ni \xi_c = \xi_0 + (rc_x, rc_y, \delta_\theta c_\theta)$.

In our formulation of Algorithm 3, the root node, encompassing all feasible solutions, does not explicitly appear and branches into a set of initial nodes $\mathcal{C}_0$ at a fixed height $h_0$ covering the search window

$$\begin{aligned} \overline{\mathcal{W}}_{0,x} &= \{-w_x + 2^{h_0} j_x : j_x \in \mathbb{Z}, 0 \le 2^{h_0} j_x \le 2w_x\}, \\ \overline{\mathcal{W}}_{0,y} &= \{-w_y + 2^{h_0} j_y : j_y \in \mathbb{Z}, 0 \le 2^{h_0} j_y \le 2w_y\}, \\ \overline{\mathcal{W}}_{0,\theta} &= \{j_\theta \in \mathbb{Z} : -w_\theta \le j_\theta \le w_\theta\}, \\ \mathcal{C}_0 &= \overline{\mathcal{W}}_{0,x} \times \overline{\mathcal{W}}_{0,y} \times \overline{\mathcal{W}}_{0,\theta} \times \{h_0\}. \end{aligned} \tag{13}$$

At a given node $c$ with $c_h > 1$, we branch into up to four children of height $c_h - 1$

$$\begin{aligned} \mathcal{C}_c = \Big( \big( \{c_x, c_x + 2^{c_h - 1}\} &\times \{c_y, c_y + 2^{c_h - 1}\} \\ &\times c_\theta \big) \cap \overline{\mathcal{W}} \Big) \times \{c_h - 1\}. \end{aligned} \tag{14}$$

*3) Computing upper bounds:* The remaining part of the branch and bound approach is an efficient way to compute upper bounds at inner nodes, both in terms of computational effort and in the quality of the bound. We use

$$\begin{aligned} score(c) &= \sum_{k=1}^{K} \max_{j \in \overline{\overline{\mathcal{W}}}_c} M_{\text{nearest}}(T_{\xi_j} h_k) \\ &\ge \sum_{k=1}^{K} \max_{j \in \overline{\mathcal{W}}_c} M_{\text{nearest}}(T_{\xi_j} h_k) \\ &\ge \max_{j \in \overline{\mathcal{W}}_c} \sum_{k=1}^{K} M_{\text{nearest}}(T_{\xi_j} h_k). \end{aligned} \tag{15}$$

To be able to compute the maximum efficiently, we use precomputed grids $M_{\text{precomp}}^{c_h}$. Precomputing one grid per possible height $c_h$ allows us to compute the score with effort linear in the number of scan points. Note that, to be able to do this, we also compute the maximum over $\overline{\overline{\mathcal{W}}}_c$ which can be larger than $\overline{\mathcal{W}}_c$ near the boundary of our search space.

$$score(c) = \sum_{k=1}^{K} M_{\text{precomp}}^{c_h}(T_{\xi_c} h_k), \tag{16}$$

$$M_{\text{precomp}}^{h}(x, y) = \max_{\substack{x' \in [x, x + r(2^h - 1)] \\ y' \in [y, y + r(2^h - 1)]}} M_{\text{nearest}}(x', y') \tag{17}$$

with $\xi_c$ as before for the leaf nodes. Note that $M_{\text{precomp}}^{h}$ has the same pixel structure as $M_{\text{nearest}}$, but in each pixel storing the maximum of the values of the $2^h \times 2^h$ box of pixels beginning there. An example of such precomputed grids is given in Figure 3.

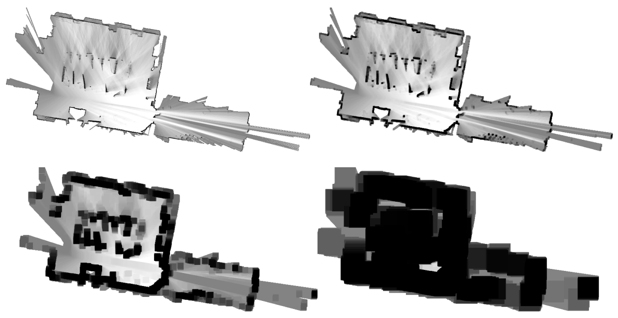
Fig. 3. Precomputed grids of size 1, 4, 16 and 64.

To keep the computational effort for constructing the precomputed grids low, we wait until a probability grid will receive no further updates. Then we compute a collection of precomputed grids, and start matching against it.

For each precomputed grid, we compute the maximum of a $2^h$ pixel wide row starting at each pixel. Using this intermediate result, the next precomputed grid is then constructed.

The maximum of a changing collection of values can be kept up-to-date in amortized $\mathcal{O}(1)$ if values are removed in the order in which they have been added. Successive maxima are kept in a deque that can be defined recursively as containing the maximum of all values currently in the collection followed by the list of successive maxima of all values after the first occurrence of the maximum. For an empty collection of values, this list is empty. Using this approach, the precomputed grids can be computed in $\mathcal{O}(n)$ where $n$ is the number of pixels in each precomputed grids.

An alternative way to compute upper bounds is to compute lower resolution probability grids, successively halving the resolution, see [1]. Since the additional memory consumption of our approach is acceptable, we prefer it over using lower resolution probability grids which lead to worse bounds than (15) and thus negatively impact performance.

## VI. Experimental results

In this section, we present some results of our SLAM algorithm computed from recorded sensor data using the same online algorithms that are used interactively on the backpack. First, we show results using data collected by the sensors

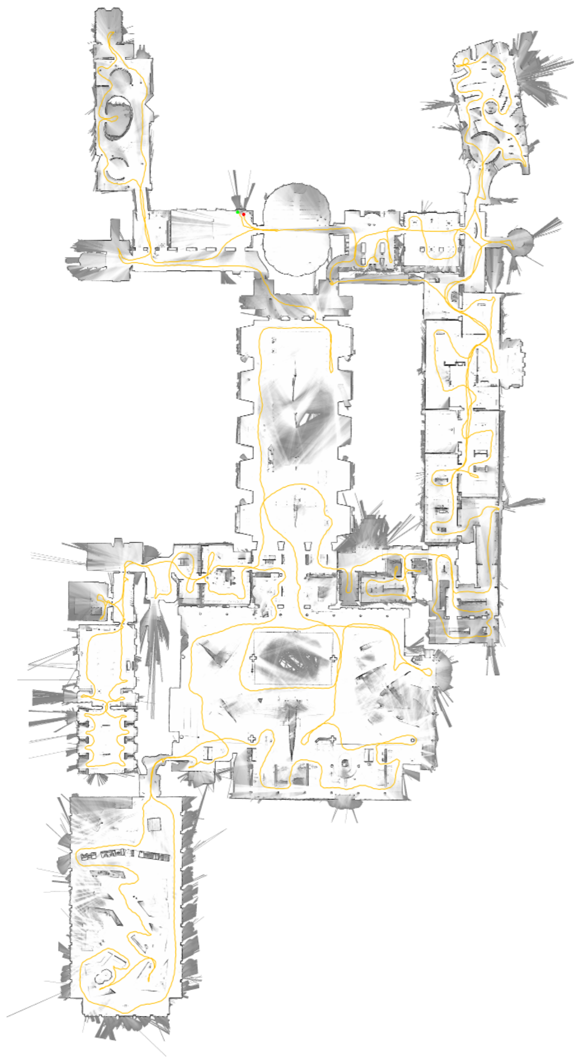
Fig. 4. Cartographer map of the 2nd floor of the Deutsches Museum.

of our Cartographer backpack in the Deutsches Museum in Munich. Second, we demonstrate that our algorithms work well with inexpensive hardware by using data collected from a robotic vacuum cleaner sensor. Lastly, we show results using the Radish data set [19] and compare ourselves to published results.

### A. Real-World Experiment: Deutsches Museum

Using data collected at the Deutsches Museum spanning 1,913 s of sensor data or 2,253 m (according to the computed solution), we computed the map shown in Figure 4. On a workstation with an Intel Xeon E5-1650 at 3.2 GHz, our SLAM algorithm uses 1,018 s CPU time, using up to 2.2 GB of memory and up to 4 background threads for loop closure scan matching. It finishes after 360 s wall clock time, meaning it achieved 5.3 times real-time performance.

The generated graph for the loop closure optimization consists of 11,456 nodes and 35,300 edges. The optimization problem (SPA) is run every time a few nodes have been added to the graph. A typical solution takes about 3 iterations, and finishes in about 0.3 s.

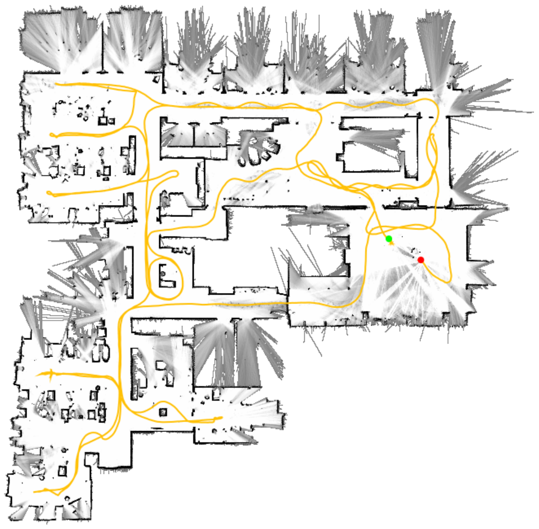

Fig. 5. Cartographer map generated using Revo LDS sensor data.

TABLE I
QUANTITATIVE ERRORS WITH REVO LDS

| Laser Tape | Cartographer | Error (absolute) | Error (relative) |
|---|---|---|---|
| 4.09 | 4.08 | −0.01 | −0.2 % |
| 5.40 | 5.43 | +0.03 | +0.6 % |
| 8.67 | 8.74 | +0.07 | +0.8 % |
| 15.09 | 15.20 | +0.11 | +0.7 % |
| 15.12 | 15.23 | +0.11 | +0.7 % |

### B. Real-World Experiment: Neato's Revo LDS

Neato Robotics uses a laser distance sensor (LDS) called Revo LDS [20] in their vacuum cleaners which costs under $ 30. We captured data by pushing around the vacuum cleaner on a trolley while taking scans at approximately 2 Hz over its debug connection. Figure 5 shows the resulting 5 cm resolution floor plan. To evaluate the quality of the floor plan, we compare laser tape measurements for 5 straight lines to the pixel distance in the resulting map as computed by a drawing tool. The results are presented in Table I, all values are in meters. The values are roughly in the expected order of magnitude of one pixel at each end of the line.

### C. Comparisons using the Radish data set

We compare our approach to others using the benchmark measure suggested in [21], which compares the error in relative pose changes to manually curated ground truth relations. Table II shows the results computed by our Cartographer SLAM algorithm. For comparison, we quote results for Graph Mapping (GM) from [21]. Additionally, we quote more recently published results from [9] in Table III. All errors are given in meters and degrees, either absolute or squared, together with their standard deviation.

Each public data set was collected with a unique sensor configuration that differs from our Cartographer backpack. Therefore, various algorithmic parameters needed to be adapted to produce reasonable results. In our experience, tuning Cartographer is only required to match the algorithm to the sensor configuration and not to the specific surroundings.

TABLE II
QUANTITATIVE COMPARISON OF ERROR WITH [21]

| | Cartographer | GM |
|---|---|---|
| Aces | | |
| Absolute translational | $0.0375 \pm 0.0426$ | $0.044 \pm 0.044$ |
| Squared translational | $0.0032 \pm 0.0285$ | $0.004 \pm 0.009$ |
| Absolute rotational | $0.373 \pm 0.469$ | $0.4 \pm 0.4$ |
| Squared rotational | $0.359 \pm 3.696$ | $0.3 \pm 0.8$ |
| Intel | | |
| Absolute translational | $0.0229 \pm 0.0239$ | $0.031 \pm 0.026$ |
| Squared translational | $0.0011 \pm 0.0040$ | $0.002 \pm 0.004$ |
| Absolute rotational | $0.453 \pm 1.335$ | $1.3 \pm 4.7$ |
| Squared rotational | $1.986 \pm 23.988$ | $24.0 \pm 166.1$ |
| MIT Killian Court | | |
| Absolute translational | $0.0395 \pm 0.0488$ | $0.050 \pm 0.056$ |
| Squared translational | $0.0039 \pm 0.0144$ | $0.006 \pm 0.029$ |
| Absolute rotational | $0.352 \pm 0.353$ | $0.5 \pm 0.5$ |
| Squared rotational | $0.248 \pm 0.610$ | $0.9 \pm 0.9$ |
| MIT CSAIL | | |
| Absolute translational | $0.0319 \pm 0.0363$ | $0.004 \pm 0.009$ |
| Squared translational | $0.0023 \pm 0.0099$ | $0.0001 \pm 0.0005$ |
| Absolute rotational | $0.369 \pm 0.365$ | $0.05 \pm 0.08$ |
| Squared rotational | $0.270 \pm 0.637$ | $0.01 \pm 0.04$ |
| Freiburg bldg 79 | | |
| Absolute translational | $0.0452 \pm 0.0354$ | $0.056 \pm 0.042$ |
| Squared translational | $0.0033 \pm 0.0055$ | $0.005 \pm 0.011$ |
| Absolute rotational | $0.538 \pm 0.718$ | $0.6 \pm 0.6$ |
| Squared rotational | $0.804 \pm 3.627$ | $0.7 \pm 1.7$ |
| Freiburg hospital (local) | | |
| Absolute translational | $0.1078 \pm 0.1943$ | $0.143 \pm 0.180$ |
| Squared translational | $0.0494 \pm 0.2831$ | $0.053 \pm 0.272$ |
| Absolute rotational | $0.747 \pm 2.047$ | $0.9 \pm 2.2$ |
| Squared rotational | $4.745 \pm 40.081$ | $5.5 \pm 46.2$ |
| Freiburg hospital (global) | | |
| Absolute translational | $5.2242 \pm 6.6230$ | $11.6 \pm 11.9$ |
| Squared translational | $71.0288 \pm 267.7715$ | $276.1 \pm 516.5$ |
| Absolute rotational | $3.341 \pm 4.797$ | $6.3 \pm 6.2$ |
| Squared rotational | $34.107 \pm 127.227$ | $77.2 \pm 154.8$ |

Since each public data set has a unique sensor configuration, we cannot be sure that we did not also fit our parameters to the specific locations. The only exception being the Freiburg hospital data set where there are two separate relations files. We tuned our parameters using the local relations but also see good results on the global relations.

TABLE III
QUANTITATIVE COMPARISON OF ERROR WITH [9]

| | Cartographer | Graph FLIRT |
|---|---|---|
| Intel | | |
| Absolute translational | $0.0229 \pm 0.0239$ | $0.02 \pm 0.02$ |
| Absolute rotational | $0.453 \pm 1.335$ | $0.3 \pm 0.3$ |
| Freiburg bldg 79 | | |
| Absolute translational | $0.0452 \pm 0.0354$ | $0.06 \pm 0.09$ |
| Absolute rotational | $0.538 \pm 0.718$ | $0.8 \pm 1.1$ |
| Freiburg hospital (local) | | |
| Absolute translational | $0.1078 \pm 0.1943$ | $0.18 \pm 0.27$ |
| Absolute rotational | $0.747 \pm 2.047$ | $0.9 \pm 2.0$ |
| Freiburg hospital (global) | | |
| Absolute translational | $5.2242 \pm 6.6230$ | $8.3 \pm 8.6$ |
| Absolute rotational | $3.341 \pm 4.797$ | $5.0 \pm 5.3$ |

TABLE IV
LOOP CLOSURE PRECISION

| Test case | No. of constraints | Precision |
|---|---|---|
| Aces | 971 | 98.1 % |
| Intel | 5786 | 97.2 % |
| MIT Killian Court | 916 | 93.4 % |
| MIT CSAIL | 1857 | 94.1 % |
| Freiburg bldg 79 | 412 | 99.8 % |
| Freiburg hospital | 554 | 77.3 % |

TABLE V
PERFORMANCE

| Test case | Data duration (s) | Wall clock (s) |
|---|---|---|
| Aces | 1366 | 41 |
| Intel | 2691 | 179 |
| MIT Killian Court | 7678 | 190 |
| MIT CSAIL | 424 | 35 |
| Freiburg bldg 79 | 1061 | 62 |
| Freiburg hospital | 4820 | 10 |

The most significant differences between all data sets is the frequency and quality of the laser scans as well as the availability and quality of odometry.

Despite the relatively outdated sensor hardware used in the public data sets, Cartographer SLAM consistently performs within our expectations, even in the case of MIT CSAIL, where we perform considerably worse than Graph Mapping. For the Intel data set, we outperform Graph Mapping, but not Graph FLIRT. For MIT Killian Court we outperform Graph Mapping in all metrics. In all other cases, Cartographer outperforms both Graph Mapping and Graph FLIRT in most but not all metrics.

Since we add loop closure constraints between submaps and scans, the data sets contain no ground truth for them. It is also difficult to compare numbers with other approaches based on scan-to-scan. Table IV shows the number of loop closure constraints added for each test case (true and false positives), as well as the precision, that is the fraction of true positives. We determine the set of true positive constraints to be the subset of all loop closure constraints which are not violated by more than 20 cm or $1^\circ$ when we compute (SPA). We see that while our scan-to-submap matching procedure produces false positives which have to be handled in the optimization (SPA), it manages to provide a sufficient number of loop closure constraints in all test cases. Our use of the Huber loss in (SPA) is one of the factors that renders loop closure robust to outliers. In the Freiburg hospital case, the choice of a low resolution and a low minimum score for the loop closure detection produces a comparatively high rate of false positives. The precision can be improved by raising the minimum score for loop closure detection, but this decreases the solution quality in some dimensions according to ground truth. The authors believe that the ground truth remains the better benchmark of final map quality.

The parameters of Cartographer's SLAM were not tuned for CPU performance. We still provide the wall clock times in Table V which were again measured on a workstation with an Intel Xeon E5-1650 at 3.2 GHz. We provide the duration of the sensor data for comparison.

## VII. CONCLUSIONS

In this paper, we presented and experimentally validated a 2D SLAM system that combines scan-to-submap matching with loop closure detection and graph optimization. Individual submap trajectories are created using our local, grid-based SLAM approach. In the background, all scans are matched to nearby submaps using pixel-accurate scan matching to create loop closure constraints. The constraint graph of submap and scan poses is periodically optimized in the background. The operator is presented with an up-to-date preview of the final map as a GPU-accelerated combination of finished submaps and the current submap. We demonstrated that it is possible to run our algorithms on modest hardware in real-time.

## ACKNOWLEDGMENTS

This research has been validated through experiments in the Deutsches Museum, Munich. The authors thank its administration for supporting our work.

Comparisons were done using manually verified relations and results from [21] which uses data from the Robotics Data Set Repository (Radish) [19]. Thanks go to Patrick Beeson, Dieter Fox, Dirk Hähnel, Mike Bosse, John Leonard, Cyrill Stachniss for providing this data. The data for the Freiburg University Hospital was provided by Bastian Steder, Rainer Kümmerle, Christian Dornhege, Michael Ruhnke, Cyrill Stachniss, Giorgio Grisetti, and Alexander Kleiner.

## REFERENCES

[1] E. Olson, "M3RSM: Many-to-many multi-resolution scan matching," in *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, June 2015.

[2] K. Konolige, G. Grisetti, R. Kümmerle, W. Burgard, B. Limketkai, and R. Vincent, "Sparse pose adjustment for 2D mapping," in *IROS*, Taipei, Taiwan, 10/2010 2010.

[3] F. Lu and E. Milios, "Globally consistent range scan alignment for environment mapping," *Autonomous robots*, vol. 4, no. 4, pp. 333–349, 1997.

[4] F. Martín, R. Triebel, L. Moreno, and R. Siegwart, "Two different tools for three-dimensional mapping: DE-based scan matching and feature-based loop detection," *Robotica*, vol. 32, no. 01, pp. 19–41, 2014.

[5] S. Kohlbrecher, J. Meyer, O. von Stryk, and U. Klingauf, "A flexible and scalable SLAM system with full 3D motion estimation," in *Proc. IEEE International Symposium on Safety, Security and Rescue Robotics (SSRR)*. IEEE, November 2011.

[6] M. Himstedt, J. Frost, S. Hellbach, H.-J. Böhme, and E. Maehle, "Large scale place recognition in 2D LIDAR scans using geometrical landmark relations," in *Intelligent Robots and Systems (IROS 2014), 2014 IEEE/RSJ International Conference on*. IEEE, 2014, pp. 5030–5035.

[7] K. Granström, T. B. Schön, J. I. Nieto, and F. T. Ramos, "Learning to close loops from range data," *The International Journal of Robotics Research*, vol. 30, no. 14, pp. 1728–1754, 2011.

[8] G. Grisetti, C. Stachniss, and W. Burgard, "Improving grid-based SLAM with Rao-Blackwellized particle filters by adaptive proposals and selective resampling," in *Robotics and Automation, 2005. ICRA 2005. Proceedings of the 2005 IEEE International Conference on*. IEEE, 2005, pp. 2432–2437.

[9] G. D. Tipaldi, M. Braun, and K. O. Arras, "FLIRT: Interest regions for 2D range data with applications to robot navigation," in *Experimental Robotics*. Springer, 2014, pp. 695–710.

[10] J. Strom and E. Olson, “Occupancy grid rasterization in large environments for teams of robots,” in *Intelligent Robots and Systems (IROS), 2011 IEEE/RSJ International Conference on*. IEEE, 2011, pp. 4271–4276.

[11] R. Kümmerle, G. Grisetti, H. Strasdat, K. Konolige, and W. Burgard, “g2o: A general framework for graph optimization,” in *Robotics and Automation (ICRA), 2011 IEEE International Conference on*. IEEE, 2011, pp. 3607–3613.

[12] L. Carlone, R. Aragues, J. A. Castellanos, and B. Bona, “A fast and accurate approximation for planar pose graph optimization,” *The International Journal of Robotics Research*, pp. 965–987, 2014.

[13] M. Bosse and R. Zlot, “Map matching and data association for large-scale two-dimensional laser scan-based SLAM,” *The International Journal of Robotics Research*, vol. 27, no. 6, pp. 667–691, 2008.

[14] S. Agarwal, K. Mierle, and Others, “Ceres solver,” http://ceres-solver.org.

[15] E. B. Olson, “Real-time correlative scan matching,” in *Robotics and Automation, 2009. ICRA’09. IEEE International Conference on*. IEEE, 2009, pp. 4387–4393.

[16] P. Agarwal, G. D. Tipaldi, L. Spinello, C. Stachniss, and W. Burgard, “Robust map optimization using dynamic covariance scaling,” in *Robotics and Automation (ICRA), 2013 IEEE International Conference on*. IEEE, 2013, pp. 62–69.

[17] A. H. Land and A. G. Doig, “An automatic method of solving discrete programming problems,” *Econometrica*, vol. 28, no. 3, pp. 497–520, 1960.

[18] J. Clausen, “Branch and bound algorithms-principles and examples,” *Department of Computer Science, University of Copenhagen*, pp. 1–30, 1999.

[19] A. Howard and N. Roy, “The robotics data set repository (Radish),” 2003. [Online]. Available: http://radish.sourceforge.net/

[20] K. Konolige, J. Augenbraun, N. Donaldson, C. Fiebig, and P. Shah, “A low-cost laser distance sensor,” in *Robotics and Automation, 2008. ICRA 2008. IEEE International Conference on*. IEEE, 2008, pp. 3002–3008.

[21] R. Kümmerle, B. Steder, C. Dornhege, M. Ruhnke, G. Grisetti, C. Stachniss, and A. Kleiner, “On measuring the accuracy of SLAM algorithms,” *Autonomous Robots*, vol. 27, no. 4, pp. 387–407, 2009.