Journal of Computing and Information Technology - CIT 17, 2009, 1, 51–65
doi:10.2498/cit.1001174

Parallel and Cached Scan Matching for Robotic 3D Mapping

Andreas Nüchter

Institute of Computer Science, University of Osnabrück, Germany

Intelligent autonomous acting of mobile robots in unstructured environments requires 3D maps. Since manual mapping is a tedious job, automatization of this job is necessary. Automatic, consistent volumetric modeling of environments requires a solution to the simultaneous localization and map building problem (SLAM problem). In 3D this task is computationally expensive, since the environments are sampled with many data points with state of the art sensing technology. In addition, the solution space grows exponentially with the additional degrees of freedom needed to represent the robot pose. Mapping environments in 3D must regard six degrees of freedom to characterize the robot pose. This paper summarizes our 6D SLAM algorithm and presents novel algorithmic and technical means to reduce computation time, i.e., the data structure cached *k*-d tree and parallelization. The availability of multi-core processors as well as efficient programming schemes as OpenMP permit the parallel execution of robotics tasks.

Keywords: 3D scan matching, iterative closest point algorithm, graphSLAM, cached *k*-d tree search, parallel *k*-d tree search, simultaneous localization and mapping, openMP

1. Introduction

Methods for solving the SLAM problem are a key scientific issue in mobile robotics research. SLAM solutions are important in providing mobile systems with the ability to operate with real autonomy. Many mobile robots are nowadays equipped with a 3D laser scanner to gather 3D range information about the environment. Multiple 3D scans are necessary to digitalize environments without occlusions. To create a correct and consistent model, the scans have to be merged into a single coordinate system. The initial registration is usually done with the well-known Iterative Closest Points (ICP) algorithm [7]. In all sequential strategies, where each scan is matched to some previous one, small errors add up to global inconsistencies. These errors are due to imprecise measurements as well as small registration errors, which can never be avoided. SLAM algorithms that use information about closed loops help diminish these effects. So, Lu and Milios proposed a probabilistic scan matching algorithm for solving the simultaneous localization and mapping (LUM) [19]. In recent work, these algorithms are applied to 3D laser scan mapping [25, 8]. Figure 1 gives an example of a 3D map generated by a mobile robot.

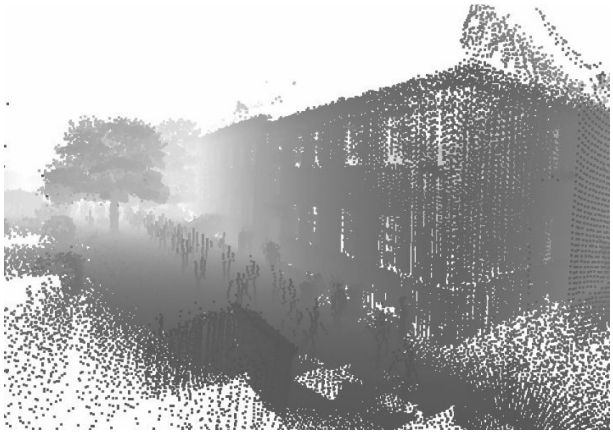


Figure 1. 3D map acquired by a mobile robot.

3D scan matching approaches to SLAM tend to be computationally expensive due to several reasons: First, the amount of data: A 3D laser range finder scans the environment with a large number of samples. Second, the additional three DoF result in an exponentially larger solution space, i.e., the solution is computationally more complex. New algorithms as well as new hardware developments help reduce these

computational costs. In a popular formulation of Moore's law, one could state that the number of transistors on integrated circuits doubles every 18 months. This is emphasized by the observation of the appearance of dual-core and quad-core CPUs in the consumer market. These chips multiply the number of computing units whereas the increase of the clock rate of the chips seems to stop due to thermal issues.

This paper presents a variation of 3D scan matching algorithms, namely cached k-d tree search and the parallel Iterative Closest Points (pICP) and the parallel Lu / Milios SLAM algorithm (pLUM). The algorithms have been optimized for execution on a shared memory machine with p processors, i.e., for quad and dual-core processors. The application to robotics is obvious: Several mobile robots are already controlled by dual-core notebooks. OpenMP is used to program the dual-core processors in a multi threaded fashion. To this end, this paper focuses on the implementation details of scan matching for our SLAM front-end.

The paper is structured as follows: After a brief description of the state of the art, we describe our 3D scan matching algorithm and our SLAM approach (Section 2 and 3). The parallelization of these two algorithms is introduced in Section 4 and 5. In Section 6 the experiments and results on various data sets are presented, starting with comments on the load balancing issue. Section 7 concludes the paper.

1.1. Current Trends in Processor Technology

In April of 2005, Intel announced the Intel Pentium Processor Extreme Edition, featuring an Intel dual-core processor. An Intel dual-core processor-based PC enables a higher throughput and simultaneous computing using a multi-core architecture. Intel dual-core CPUs supporting Hyper-threading Technology can process four software threads simultaneously by using more efficiently the resources that otherwise may sit idle [1]. Since multi-core processors represent a major evolution in computing technology, Intel's competitors have dual and quad-core processors, too. All these machines have a shared memory model.

Parallelization is an important optimization issue, since the increase of the clock rate has nearly stopped. Therefore, it is not possible to wait until computers have gotten fast enough to run a single thread algorithm in the desired time.

State of the art in programming multiple instruction multiple data paths (MIMD) processors such as dual and quad-core computers is OpenMP. Programs written in C and Fortran are divided into single threaded and parallel regions using OpenMP using compiler directives, which are handled by the preprocessor. Compared to other underlying parallelization schemes like `pthreads` or the message passing system (MPI), the underlying philosophy of OpenMP is to make, small changes to existing single threaded programs in order to yield parallel executable code, e.g., when parallelizing a `for` loop. Mathematical computations are especially in the focus of parallelization. In addition to compiling into parallel threads, instruction level parallelism (single instruction multiple data paths, SIMD) is employed by current compilers.

1.2. 3D Metric Robotic Mapping – State of the Art

Metrical maps represent explicit distances in the environment. These maps are either 2D, usually an upright projection, or 3D, i.e., a volumetric environment map. State of the art for 2D metric maps are probabilistic methods, where the robot has a probabilistic motion and perception model. SLAM in well-defined, planar indoor environments is considered solved, a survey of these techniques is presented by Thrun in [26]. Furthermore, SLAM approaches can be classified by the number of DoF of the robot pose. A 3D pose estimate contains the (x, y)-coordinate and a rotation θ, whereas a 6D pose estimate considers all degrees of freedom a rigid mobile robot can have, i.e., the (x, y, z)-coordinate and the roll, yaw and pitch angles. This emerging research topic is called 6D SLAM, In previous work, we presented the mobile robot Kurt3D that uses a tiltable 2D laser range finder [24] in a stop-scan-match-go-process to create a 3D map of the environment by merging several 3D scans into one coordinate system [22, 25]. Here,

online map generation, i.e., the map is available right after the robot run without extra map computing time, was possible, through using pairwise scan matching with an ICP algorithm. The speed-ups have been realized using data reduction and approximate k-d tree search. Similar experiments have been made by Newman et al. [20]. A recent trend in laser based 6D SLAM is to overcome stop-and-go fashion of scan acquisition by rotating or pitching the scanner while moving [9, 28, 31].

Another trend in SLAM research is to apply probabilistic methods to 3D mapping. Katz et al. use a probabilistic notion of ICP scan matching [16]. Weingarten et al. [30] and Cole et al. [9] apply extended Kalman filter to the mapping problem. We extend this state of the art by a GraphSLAM method. A similar approach was used in [27]. However, their algorithm is not practical due to the reported computational requirements. Furthermore, Frese presented an extension of his treemap SLAM algorithm to six degrees of freedom, which, however, covers the least-square estimation core and no actual scan-data processing [10].

2. The ICP Algorithm

The ICP Algorithm was developed by Besl and McKay [7] and is usually used to register two given 3D point sets in a common coordinate system. The algorithm calculates the registration iteratively. In each iteration step, the algorithm selects the closest points as correspondences and calculates the transformation, i.e., rotation and translation ($\mathbf{R}$, $\mathbf{t}$), for minimizing the equation

$$E(\mathbf{R},\mathbf{t}) = \sum_{i=1}^{N_m}\sum_{j=1}^{N_d} w_{i,j}\|\mathbf{m}_i - (\mathbf{R}\mathbf{d}_j + \mathbf{t})^2, \quad (1)$$

where N_m and N_d, are the number of points in the model set M and data set D, respectively, and w_{ji} are the weights for a point match. The weights are assigned as follows: $w_{ji} = 1$, if $\mathbf{m}_i$ is the closest point to $\mathbf{d}_j$, $w_{ji} = 0$ otherwise. Eq. (1) is reduced to

$$E(\mathbf{R},\mathbf{t}) \propto \frac{1}{N}\sum_{i=1}^{N}\|\mathbf{m}_i - (\mathbf{R}\mathbf{d}_i + \mathbf{t})^2, \quad (2)$$

with $N = \sum_{i=1}^{N_m}\sum_{j=1}^{N_d} w_{i,j}$, since the correspondence matrix can be represented by a vector v containing the closest point pairs, i.e., $v = ((\mathbf{d}_1, \mathbf{m}_{f(\mathbf{d}_1)}), (\mathbf{d}_2, \mathbf{m}_{f(\mathbf{d}_2)}), \ldots, (\mathbf{d}_{N_d}, \mathbf{m}_{f(\mathbf{d}_{N_d})}))$, with $f(x)$ the search function returning the closest point. The assumption is that in the last iteration step the point correspondences, thus the vector of point pairs, are correct.

Besl and McKay show that the iteration terminates in a minimum [7]. Note: Normally, implementations of ICP would use a maximal distance for closest points to partially handle overlapping point sets. In this case, the proof in [7] does no longer hold, since the overlap and number of points as well as the value of $E(\mathbf{R},\mathbf{t})$ might increase after applying a transformation.

Four methods are available to calculate the transformation in each ICP iteration: A SVD-based method of Arun et al. [3], a quaternion method of Horn [14], an algorithm using orthonormal matrices of Horn et al. [15] and a calculation based on dual quaternions of Walker et al. [29]. These algorithms show similar performance and stability concerning noisy data [18]. The difficulty of the minimization problem is to enforce the orthonormality of matrix $\mathbf{R}$.

Next, we give a brief overview of the SVD-based algorithm. The first step of the computation is to decouple the calculation of the rotation $\mathbf{R}$ from the translation $\mathbf{t}$ using the centroids of the points belonging to the matching, i.e., for all points in vector v:

$$\mathbf{c}_m = \frac{1}{N}\sum_{i=1}^{N}\mathbf{m}_i, \qquad \mathbf{c}_d = \frac{1}{N}\sum_{i=1}^{N}\mathbf{d}_j \quad (3)$$

and

$$M' = \{\mathbf{m}'_i = \mathbf{m}_i - \mathbf{c}_m\}_{1,\ldots,N}, \quad (4)$$

$$D' = \{\mathbf{d}'_i = \mathbf{d}_i - \mathbf{c}_d\}_{1,\ldots,N}. \quad (5)$$

After replacing (3), (4) and (5) in the error function, $E(\mathbf{R},\mathbf{t})$ Eq. (2) becomes:

$$E(\mathbf{R},\mathbf{t}) \propto \frac{1}{N}\sum_{i=1}^{N} \|\mathbf{m}'_i - \mathbf{R}\mathbf{d}'_i - \underbrace{(\mathbf{t} - \mathbf{c}_m + \mathbf{R}\mathbf{c}_d)}_{=\tilde{\mathbf{t}}}\|^2$$

$$= \frac{1}{N}\sum_{i=1}^{N} \|\mathbf{m}'_i - \mathbf{R}\mathbf{d}'_i\|^2 \quad (6a)$$

$$- \frac{2}{N}\tilde{\mathbf{t}} \cdot \sum_{i=1}^{N} \left(\mathbf{m}'_i - \mathbf{R}\mathbf{d}'_i\right) \quad (6b)$$

$$+ \frac{1}{N}\sum_{i=1}^{N} \|\tilde{\mathbf{t}}\|^2. \quad (6c)$$

In order to minimize the sum above, all terms have to be minimized. The second sum (6b) is zero, since all values refer to centroid. The third part (6c) has its minimum for $\tilde{\mathbf{t}} = \mathbf{0}$ or

$$\mathbf{t} = \mathbf{c}_m - \mathbf{R}\mathbf{c}_d. \quad (7)$$

Therefore the algorithm has to minimize only the first term, and the error function is expressed in terms of the rotation only:

$$E(\mathbf{R},\mathbf{t}) \propto \sum_{i=1}^{N} \|\mathbf{m}'_i - \mathbf{R}\mathbf{d}'_i\|^2. \quad (8)$$

Theorem: The optimal rotation is calculated by $\mathbf{R} = \mathbf{V}\mathbf{U}^T$. Hereby, the matrices $\mathbf{V}$ and $\mathbf{U}$ are derived from the singular value decomposition $\mathbf{H} = \mathbf{U}\Lambda\mathbf{V}^T$ of a correlation matrix $\mathbf{H}$. This 3×3 matrix $\mathbf{H}$ is given by

$$\mathbf{H} = \sum_{i=1}^{N} \mathbf{m}'^T_i \mathbf{d}'_i = \sum_{i=1}^{N} (\mathbf{m}_i - \mathbf{c}_m)(\mathbf{d}_i - \mathbf{c}_d)^T \quad (9)$$
$$= \begin{pmatrix} S_{xx} & S_{xy} & S_{xz} \\ S_{yx} & S_{yy} & S_{yz} \\ S_{zx} & S_{zy} & S_{zz} \end{pmatrix},$$

with $S_{xx} = \sum_{i=1}^{N} m'_{ix} d'_{ix}$, $S_{xy} = \sum_{i=1}^{N} m'_{ix} d'_{iy}$, ... The analogous algorithm is derived directly from this theorem; the proof is given in [3].

Alternatively to minimization via SVD, one can use quaternions to represent the rotation. The quaternion is the eigenvector with the maximal eigenvalue of the 4×4 cross-covariance matrix (*) given in Figure 2. Finding the vector requires solving the characteristic polynomial of degree 4, that can be computed by Ferrari's method. The cross-covariance matrix is calculated in terms of $S_{xx}, S_{xy}, \ldots$, as given above.

The next Section recapitulates existing methods for closest point search and presents a novel method applicable for ICP. It combines k-d trees with caching.

2.1. Closest Point Search with k-d Trees

k-d trees are a generalization of binary search trees. Every node represents a partition of a point set to the two successor nodes. The root represents the whole point cloud and the leaves provide a complete disjunct partition of the points. These leaves are called buckets (cf. Figure 3). Furthermore, every node contains the limits of the represented point set.

2.1.1. Searching k-d Trees

k-d trees are searched recursively. A given 3D point needs to be compared with the separating plane in order to decide on which side the search must continue. This procedure is executed until the leaves are reached. There, the algorithm has to evaluate all bucket points. However, the closest point may be in a different bucket, iff the distance to the limits is smaller than the one to the closest point in the bucket. In this case, backtracking has to be performed. Figure 3 shows a backtracking case, where the algorithms have to go back to the root. The test is known as ball-within-bounds test [6, 11, 12].

$$\mathbf{N} = \begin{pmatrix} (S_{xx}+S_{yy}+S_{zz}) & (S_{yz}+S_{zy}) & (S_{zx}+S_{xz}) & (S_{xy}+S_{yx}) \\ (S_{yz}+S_{zy}) & (S_{xx}-S_{yy}-S_{zz}) & (S_{xy}+S_{yx}) & (S_{zx}+S_{xz}) \\ (S_{zx}+S_{xz}) & (S_{xy}+S_{yx}) & (-S_{xx}+S_{yy}-S_{zz}) & (S_{yz}+S_{zy}) \\ (S_{xy}+S_{yx}) & (S_{yz}+S_{zy}) & (S_{zx}+S_{xz}) & (-S_{xx}-S_{yy}+S_{zz}) \end{pmatrix} \quad (*)$$

Figure 2. Computation of the cross-covariance matrix for the quaternion-based solution.

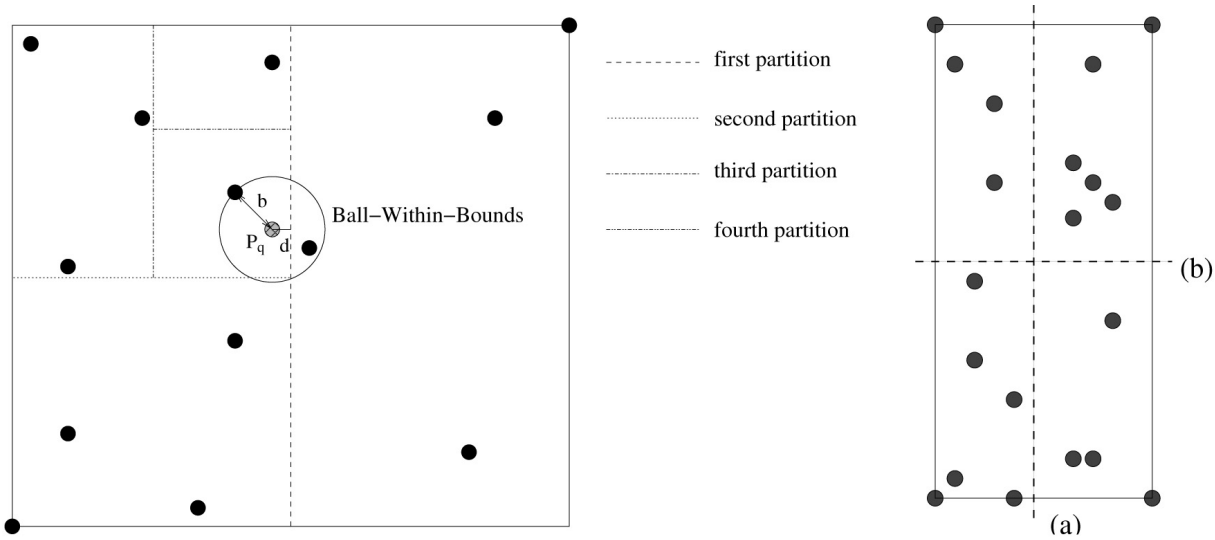


Figure 3. Left: Recursive construction of a k-d tree. If the query consists of point p_q, k-d tree search has to backtrack to the tree root to find the closest point. Right: Partitioning of a point cloud. Using the cut (b) rather than (a) results in a more compact partition and a smaller probability of backtracking [11].

2.1.2. The Optimized k-d Tree

The objective of optimizing k-d trees is to reduce the expected number of visited leaves. Three parameters are adjustable, namely, the direction and position of the split axis as well as the maximal number of points in the buckets. Splitting the point set at the *median* ensures that every k-d tree entry has the same probability [11]. The median can be found in linear time, thus the time complexity for constructing the tree is not affected. Furthermore, the split axis should be oriented *perpendicular* to the longest axis to minimize the amount of backtracking (see Figure 3, right). Friedman and associates prove that a bucket size of 1 is optimal [11]. Nevertheless, in practice it turned out that a slightly larger bucket size is faster [13].

2.1.3. Approximate k-d Tree Search

S. Arya and D. Mount introduce the following notion for approximating the nearest neighbor in k-d trees [4]: Given an $\epsilon > 0$, then the point $p \in D$ is the $(1+\epsilon)$-approximate nearest neighbor of the point p_q, iff

$$\|p - p_q\| \leq (1+\epsilon)\|p^* - p_q\|,$$

where p^* denotes the true nearest neighbor, i.e., p has a maximal distance of ϵ to the true nearest neighbor. Using this notation, the algorithm records the closest point p in every step. The search terminates if the distance to the unanalyzed leaves is larger than

$$\|p_q - p\|/(1+\epsilon).$$

Figure 4 (left) shows an example where the gray cell need not to be analyzed, since the point p satisfies the approximation criterion.

2.1.4. Approximate Box Decomposition Trees

Arya et al. have presented an algorithm for approximating the nearest neighbor search and proved its optimality [5]. They use a balanced box decomposition tree (bd-tree) as their primary data structure. This tree combines two important properties of geometric data structures: First, as in the k-d tree case, the set of points is exponentially reduced. Second, the aspect ratio of the tree edges is bounded by a constant. Not even the optimized k-d tree is able to make this assurance, but quadtrees show this characteristic [5]. The actual box decomposition search tree is composed of splits and shrinks. Figure 4 (c) shows the general structure.

The search procedure of bd-trees is similar to the one of approximate k-d trees. The approximate search is discontinued (cf. Figure 4) if the distance to the unanalyzed leaves is larger than

$$\|p_q - p\|/(1+\epsilon).$$

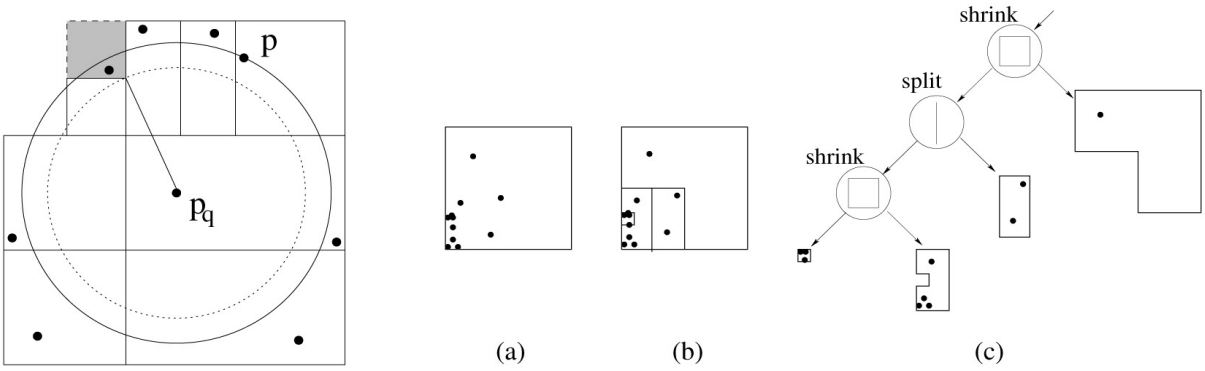


Figure 4. Left: The $(1+\epsilon)$-approximate nearest neighbor. The solid circle denotes the ϵ environment of p_q. The search algorithm need not analyze the gray cell, since p satisfies the approximation criterion. Middle and right: (a) Given point set. (b) decomposition into buckets. (c) Tree layout. Figure adapted from [4, 5].

2.2. Cached *k*-d Tree Search

k-d trees with caching contain, in addition to the limits of the represented point set and to the two child node pointers, one pointer to the predecessor node. The root node contains a null pointer. During the recursive construction of the tree, this information is available and no extra computations are required.

For the ICP algorithm, we distinguish between the first and the following iterations: In the first iteration, a normal *k*-d tree search is used to compute the closest points. However, the return function of the tree is altered, such that, in addition to the closest point, the pointer to the leaf containing the closest point is returned and stored in the vector of point pairs. This supplementary information forms the cache for future look-ups. Note: The cache here is a main memory section storing intermediate results and is not the hardware, i.e., on die cache.

In the following iterations, these stored pointers are used to start the search. If the query point is located in the bucket, the bucket is searched and the ball-within-bounds test is applied. Backtracking is started, iff the ball lies not completely within the bucket. If the query point is not located within the bucket, then backtracking is started, too. Since the search is started in the leaf node, explicit backtracking through the tree has to be implemented using the pointers to the predecessor nodes (see Figure 5). Algorithm 1 summarizes the ICP with cached *k*-d tree search.

Performance of cached *k*-d tree search. The proposed ICP variant uses exact closest point search. In contrast to the previously discussed approximate *k*-d tree search for ICP algorithms [12, 21], registration inaccuracies or errors due to approximation cannot occur.

Algorithm 1 ICP with cached k-d tree search

```
1:  for i = 0 to maxIterations do
2:    if i == 0 then
3:      for all d_j ∈ D do
4:        search k-d tree of set M top down for
          point d_j
5:        v_j = (d_j, m_{f(d_j)}, ptr_to_bucket(m_{f(d_j)}))
6:      end for
7:    else
8:      for all d_j ∈ D do
9:        search k-d tree of set M bottom up for
          point d_j using ptr_to_bucket(m_{f(d_j)})
10:       v_j = (d_j, m_{f(d_j)}, ptr_to_bucket m_{f(d_j)})
11:     end for
12:   end if
13:   calculate transformation (R, t) that minimizes
      the error function Eq. (2)
14:   apply transformation on data set D
15: end for
```

Friedman et al. prove that searching for closest points using *k*-d trees needs logarithmic time [11], i.e., the amount of backtracking is independent of the number of stored points in the tree. Since the ICP algorithm iterates the closest point search, the performance derives to $\mathcal{O}(IN_d \log N_m)$, with I the number of iterations. Note: Brute-force ICP algorithms have a performance of $\mathcal{O}(IN_dN_m)$.

The proposed cached *k*-d tree search needs $\mathcal{O}((I + \log N_m)N_d)$ time in the best case. This performance is reached if constant time is needed

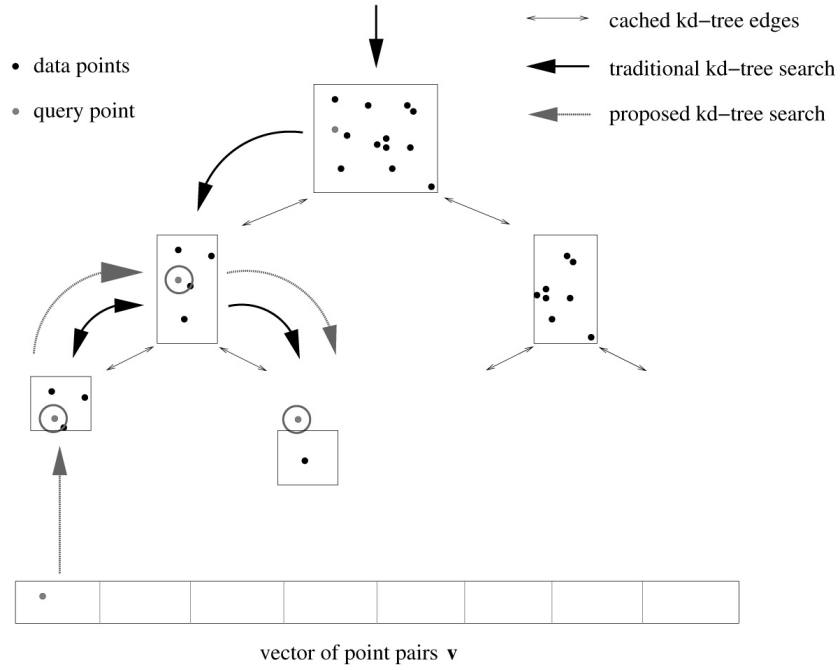


Figure 5. Schematic description of the proposed search method: Instead of closest point searching from the root of the tree to the leaves that contain the data points, a pointer to the leaves is cached. In the second and following ICP iteration, the tree is searched backwards.

for backtracking, resulting in $N_d \log N_m$ time for constructing the tree, and $I \cdot N_d$ for searching in case no backtracking is necessary. Obviously, the backtracking time depends on the computed ICP transformation $(\mathbf{R}, \mathbf{t})$. For small transformations the time is nearly constant.

Cached k-d tree search needs $\mathcal{O}(N_d)$ extra memory for the vector v, i.e., for storing the pointers to the tree leaves. Furthermore, additional $\mathcal{O}(N_m)$ memory is needed for storing the backwards pointers in the k-d tree.

3. Lu / Milios Style GraphSLAM

To solve SLAM, a 6D graph optimization algorithm for global relaxation based on the method of Lu and Milios [19] is employed, namely Lu and Milios style SLAM (LUM). Details of the 6D optimization, i.e., how the matrices have to be filled, can be found in [8].

Given a network with $n+1$ nodes $X_0, ..., X_n$ representing the 6D poses $V_0, ..., V_n$ in Euler angles, and the directed edges $D_{i,j}$, we aim to estimate all poses optimally to build a consistent map of the environment. For simplicity, we make the approximation that the measurement equation is linear, i.e.,

$$D_{i,j} = X_i - X_j.$$

Since the optimization is done in an iterative fashion, problems due to linearization do not occur. An error function is formed such that minimization results in improved pose estimations:

$$W = \sum_{(i,j)} (D_{i,j} - \bar{D}_{i,j})^T C_{i,j}^{-1} (D_{i,j} - \bar{D}_{i,j}). \quad (10)$$

where $\bar{D}_{i,j} = D_{i,j} + \Delta D_{i,j}$ models random Gaussian noise added to the unknown exact pose $D_{i,j}$. The covariance matrices $C_{i,j}$ describing the pose relations in the network are computed based on the paired points of the scan matching. The error function Eq. (10) has a quadratic form and is therefore solved in closed form by Cholesky decomposition in the order of $\mathcal{O}(\backslash\backslash)$ for n graph edges ($n \ll N$, where N is the number of points per 3D scan). The algorithm optimizes Eq. (10) gradually, by iterating the following five steps [8]:

1. Compute the point correspondences (n closest points) for any link (i,j) in the given graph.

2. Calculate the measurement vector $\bar{D}_{ij}$ and its covariance C_{ij}.
3. From all $\bar{D}_{ij}$ and C_{ij} form a linear system $\mathbf{GX} = \mathbf{B}$.
4. Solve for $\mathbf{X}$
5. Update the poses and their covariances.

4. The Parallel ICP Algorithm

The basic work for parallelization of the ICP algorithm (pICP) was done by by Langis et al. [17]. pICP showed compelling results on a cluster with p processors for registration of large data sets. The basic idea is to divide the data set D into p parts and to send these parts together with the whole model set M to the child processes that compute concurrently the expensive closest point queries. Afterwards, these points correspondences are transmitted back to the parent that uses it for computing the transformation by minimizing Eq. (2). Then this transformation is sent to the childs, which transform the data set D. The process is iterated until the minimum of Eq. (2) is reached.

The parallelization scheme for ICP uses a lot of bandwidth for transmitting corresponding points. Thus, Langis et al. [17] proposed several enhancements to the parallel method. One of these improvements is avoiding the transfer of the corresponding points. The computation of the correlation matrix $\mathbf{t}$ in Eq. (9) is parallelized and partially executed by the child processes, i.e., Eq. (9) is transformed as follows

$$\mathbf{H} = \sum_{i=1}^{N} (\mathbf{m}_i - \mathbf{c}_m)(\mathbf{d}_i - \mathbf{c}_d)^T$$

$$= \sum_{i=1}^{p} \sum_{j=1}^{N_i} (\mathbf{m}_{i,j} - \mathbf{c}_m)(\mathbf{d}_{i,j} - \mathbf{c}_d)^T, \qquad (11)$$

(12)

where N_i denotes the number of points of child i, $\mathbf{c}_{m_i}, \mathbf{c}_{d_i}$ the centroid of points of child i, and $(\mathbf{m}_{i,j}, \mathbf{d}_{i,j})$ the jth corresponding point of child i. Furthermore, for Eq. (11) holds

$$\mathbf{H} = \sum_{i=1}^{p} \sum_{j=1}^{N_i} \Big(\big(\mathbf{m}_{i,j} - \mathbf{c}_{m_i} + (\mathbf{c}_{m_i} - \mathbf{c}_m)\big) \big(\mathbf{d}_{i,j} - \mathbf{c}_{d_i} + (\mathbf{c}_{d_i} - \mathbf{c}_d)\big)^T \Big)$$

$$= \sum_{i=1}^{p} \big(\mathbf{H}_i + N_i(\mathbf{c}_{m_i} - \mathbf{c}_m)(\mathbf{c}_{d_i} - \mathbf{c}_d)^T\big) \quad (13)$$

$$\text{where,} \quad \mathbf{H}_i = \sum_{j=1}^{N_i} (\mathbf{m}_{i,j} - \mathbf{c}_{m_i})(\mathbf{d}_{i,j} - \mathbf{c}_{d_i})^T \qquad (14)$$

Therefore, after the point correspondences are parallelly computed Eq. (13) and (14) form the algorithm for computing the correlation matrix $\mathbf{t}$ in a parallel fashion. The centroids $\mathbf{c}_m$ and $\mathbf{c}_d$ (cf. Eq.(3)) are also computed from these intermediate results, i.e.,

$$\mathbf{c}_m = \frac{1}{N} \sum_{i=1}^{p} N_i \mathbf{c}_{m_i}, \qquad \mathbf{c}_d = \frac{1}{N} \sum_{i=1}^{p} N_i \mathbf{c}_{d_j}$$

Finally, after the parent has collected the vectors $\mathbf{c}_{m_i}$, $\mathbf{c}_{d_i}$ and the matrix $\mathbf{t}_i$ from the childs, it is possible to compute the transformation $(\mathbf{R}, \mathbf{t})$ to align the point sets.

4.1. Parallelization of k-d tree Search

This paper considers an OpenMP implementation for the pICP algorithm, that is a shared memory architecture with p processors or threads (symmetric multiprocessing). Therefore, there is no need to create p copies of the model set M. However the search is executed in parallel, using parallel k-d tree search.

4.1.1. Parallel Construction of k-d Trees

Since the k-d tree partitions the point set in two disjunct subsets, the construction is easily partitioned by executing the two recursive cases in parallel.

4.1.2. Parallel Search of k-d Trees

In the pICP algorithm several search requests have to be handled by the k-d tree at the same time. Efficient implementations of k-d tree search algorithms use static, i.e., global variables to save the current closest point. This current point is first derived by depth first search

and then updated in the backtracking phase. When executing the search with parallelism p copies of this point must be created.

4.1.3. Implementation Details

Most high performance processors insert a hardware cache buffer between slow memory and the high speed registers of the CPU. If several threads use unique data elements on the same hardware cache line for read and write, then so-called false sharing might occur. If one of the threads writes to a cache line, the same cache line referenced by the second thread is invalidated of the cache. Any new references to data in this cache line by the second thread results in a cache miss and the data has to be loaded again from memory. To avoid false sharings, we pad each thread's data element to ensure that elements owned by different threads all lie on separate cache lines as follows:

```
class KDParams {
public:
  /** pointer to the closest point.
    *  size = 4 bytes of 32 bit machines
    */
  double *closest;
  /** distance to the closest point.
    * size = 8 bytes
    */
  double closest_d2;
  /** pointer to the point,
    * size = 4 bytes of 32 bit machines
    */
  double *p;
  /** expand to 128 bytes to avoid
    * false-sharing, 16 bytes from above
    * + 28*4 bytes = 128 bytes */
  int padding[28];
};
```

In our search tree, these padded parameters are included using memory alignment:

```
class KDtree {
  // [snip]
public:
   __declspec (align(16)) \
      static KDParams params
      [MAX_OPENMP_NUM_THREADS];
};
```

4.1.4. Parallel Search of Cached k-d Trees

Searching cached k-d trees in a parallel fashion is accomplished by placing the `for`-statements of algorithm 1 in OpenMP directives. The cache is currently a section of the main memory, thus all threads are allowed to access the cached data via shared memory.

5. The Parallel Lu / Milios Style GraphSLAM

There are several steps of the Lu and Milios-based GraphSLAM algorithm that might be executed in a parallel fashion, resulting in the pLUM algorithm. The point correspondences for any link (i, j) in the given graph are computed in parallel (step 1) as well as the computation of the measurement vector D_{ij} (step 2) and the formation of the linear system (step 3). In principle, also the last step, i.e., solving the linear system of equations, can be executed in a parallel fashion by the parallel Cholesky decomposition [23]. However, since we use a sparse Choleskey decomposition that runs in linear time, no additional speedup is needed in this step.

Two possible strategies exist for parallelizing step 1: First, for all links the computation of the correspondence search proceed as the unimproved pICP case in Section 4. The drawback of this strategy is that the closest point pairs have to be transmitted back to the parent thread. The more advanced strategy is to compute for the n given links, the correspondences on the p processors. In this strategy several k-d trees of different data sets have to be constructed, maintained and searched in a parallel fashion. However, it turns out that the parallelization scheme in subsection 4.1.3 also works in this pLUM variant. Since the global variables of the parallel k-d tree depend on the thread number, several data sets are processed at the same time. The search function of every k-d tree associated with a data set stores its local values separately.

6. Experiments and Results

6.1. Evaluation of Cached k-d Tree Search

The proposed method has been evaluated with three data sets from different domains. The computation was done on an Intel Core 2 Duo with 1.83 GHz running Linux OS, with the same compiler options, i.e., with Intel's `icc -O2`. Since k-d tree search and cached k-d tree search are very similar, most parts of the code were identical in the comparison experiments.

In all tests, ICP with cached k-d tree search outperformed ICP with conventional k-d tree search. Three detailed analyses are provided:

1. The performance of the cached k-d tree search depending on a change of the bucket size was tested: For small bucket sizes, the speed-up is larger (Figure 6, top). This behavior originates from the increasing time needed to search larger buckets.
2. The search time per iteration was recorded during the experiments (Figure 6, middle). For the first iteration the search times are equal, since cached k-d tree search uses conventional k-d tree search to create the cache. In the following iterations, the search time drops significantly and remains nearly constant. The conventional k-d tree search increases in speed, too. Here, the amount of backtracking is reduced due to the fact that the calculated transformations $(\mathbf{R}, \mathbf{t})$ are getting smaller.
3. The number of points to register influences the search time. With increasing number of points, the positive effect of caching algorithms becomes more and more significant (Figure 6, bottom).

Additional results on cached k-d tree search are reported in the following subsection in combination with parallelization.

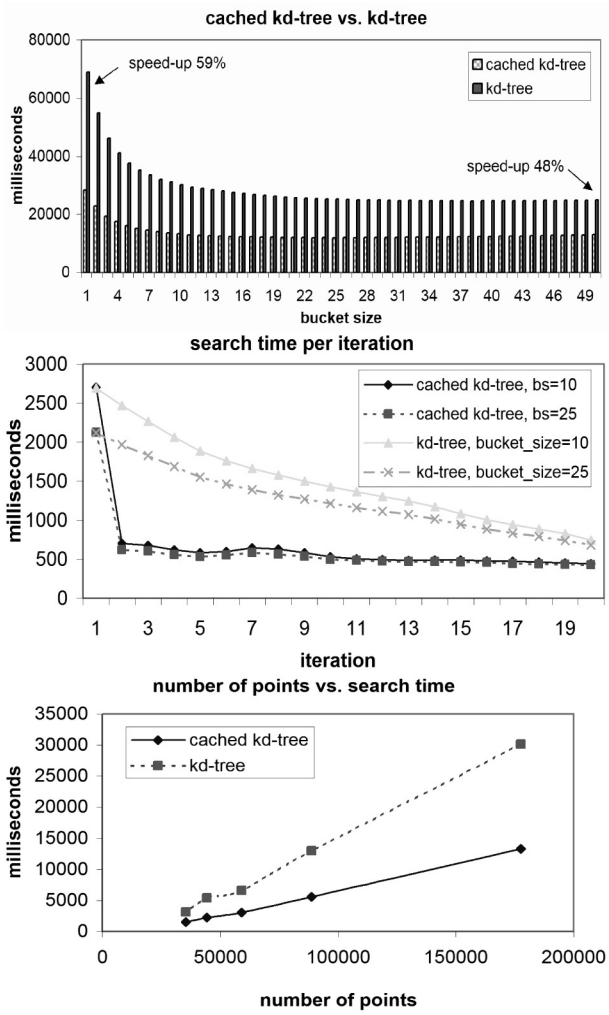


Figure 6. Detailed results for the data set "cluttered indoor environment". Top: Search time vs. bucket size. Middle: Search time per iteration for bucket sizes 10 and 25. Bottom: Search time depending on the number of points.

6.2. Evaluation of the Parallelization

Load Balancing. The computations carried out in a parallel fashion have to be scheduled to the p processors. The goal of balancing the load for the processors is to maximize the gained speed-up aiming to reach the optimal runtime of t_s/p, with t_s the time for executing scan matching on a single processors.

The scheduling of parallel point searching for pICP is done by OpenMP altering the `for` statement. Every thread determines its thread number and uses this information for the parallel k-d tree search. Prior splitting of the data set avoids the determination of the thread number for every point. Splitting the data set has to be done in a randomized fashion to ensure load balancing. It turned out that this requires more time than the determination of the thread number in every iteration. Similar arguments apply for pLUM.

Matching Laser Scans Acquired with Kurt3D. In this experiment we use the exploration robot Kurt3D. Figure 7 shows the robot in a natural outdoor environment. The 3D laser range finder [24] is built on the basis of a SICK 2D range finder by extension with a mount and a small servomotor. Resolution can be adjusted at the expense of scanning time; scanning is done in a stop-scan-go fashion. All computations have been carried out with Kurt3D that is equipped with a Panasonic Toughbook CF-74 with an Intel Core Duo-T2400, 1.83 GHz.

We made two experiments with Kurt3D. First we drove a closed loop in our office environment and acquired 11 3D scan with 85000 data points each. Figure 8 shows the office scene.

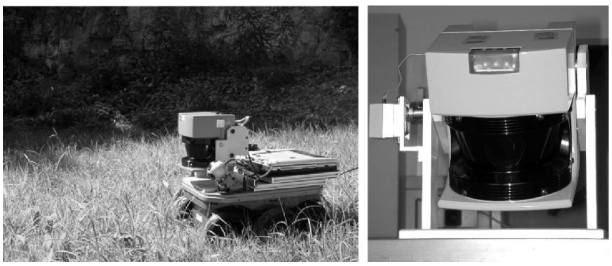


Figure 7. Left: Kurt3D. Right: The 3D laser scanner is built of 2D laser scanner and a servo motor step-rotating the scanner.

Table 1 shows the results. The overall running time was reduced to 76.2%, i.e., by a factor of 1.31. Using four instead of two threads yields a slight improvement, probably due to improved scheduling issues. Using more than four threads did not lead to additional advances.

The second experiment was performed in Dagstuhl castle, where we acquired a 3D model of interior of the new building wing (cf. Figure 9) containing 83 3D scans with the same resolution as in the previous experiment. The overall time spent by the 3D mapping algorithms, i.e., ICP and LUM was 112141 ms using single thread execution, 78141 ms using two threads and 70218 ms using four threads.

Matching of Continously Acquired 3D Scans. Wulf et al. presented a rotating 3D scanner in [31]. In the following experiment we processed 468 3D scans containing approximately 20000 3D points per scans. These scans where matched with ICP and since there were several loops we applied LUM after each loop detection. Loop detection is done using a simple distance criterion: If two estimated robot poses V_i and V_j are close enough, i.e., their distance falls below a threshold (here: 5 meters), then we assume that these scans overlap and are matchable. The overall time spent by the 3D mapping algorithms was reduced from 4526 sec. to 3420 sec. using four threads, i.e., a speed-up of 1.32 was obtained. Figure 10 shows the final map (top view) and two detailed views.

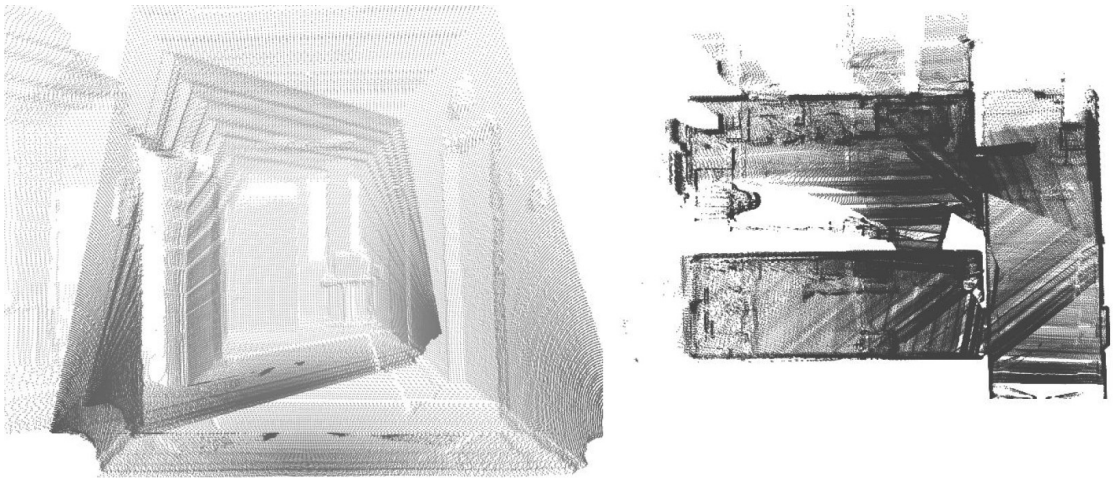


Figure 8. Left: 3D scans of an office environment. Right: Top view of the map representing one small closed loop.

algorithm	**one thread**		**two threads**		**four threads**	
	without cache	with cache	without cache	with cache	without cache	with cache
ICP	12688	10945	9821	9222	9750	9081
LUM	703	692	610	550	453	412
total	13391	11637	10431	9772	10203	9493

Table 1. Running times in milliseconds for single-thread computation vs. multi-thread computation. The first rows represent the speed-up for pure ICP and LUM computations on an Intel Core Duo-T2400.

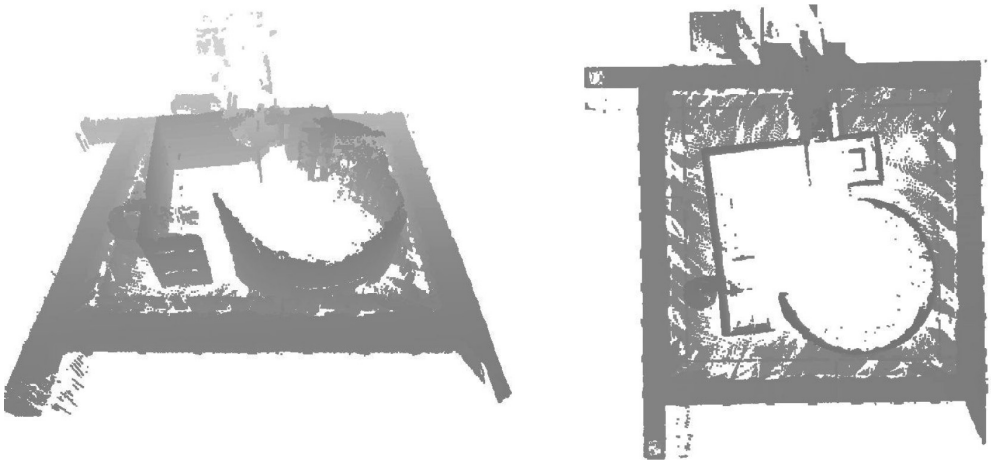


Figure 9. Left: 3D view of the new wing of Dagstuhl castle. Right: Top view.

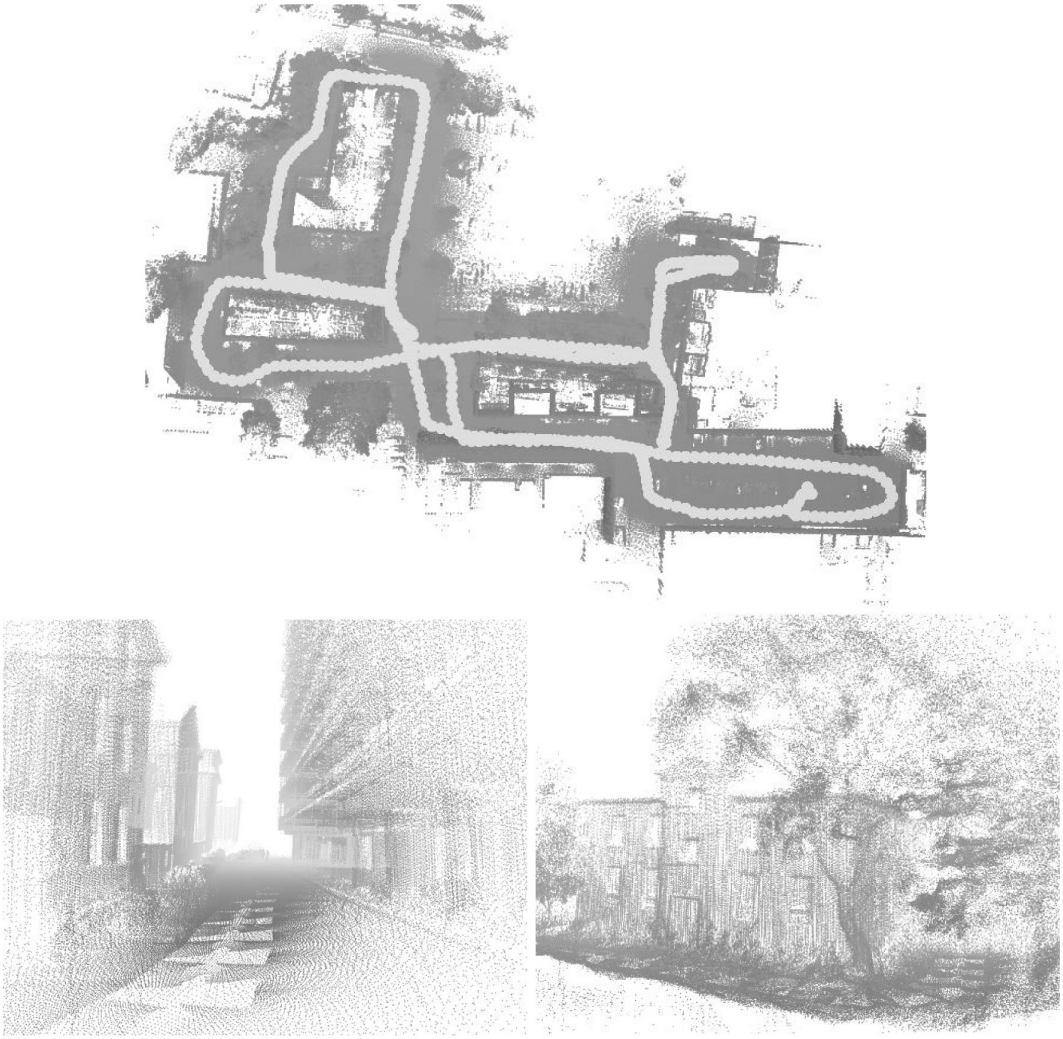
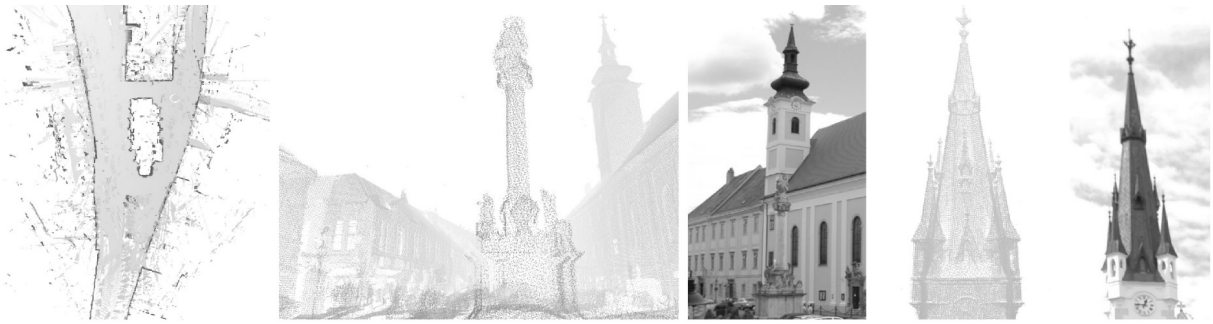


Figure 10. Top: Trajectory and map of the campus of Leibniz Universität Hannover. Bottom: Two detailed 3D views. Bottom left: modelled skyscraper. Bottom right: building with a tree in front. An additional illustration of this scene is given in Figure 1.

number of 3D scans	one thread		two threads		four threads	
	without cache	with cache	without cache	with cache	without cache	with cache
2 (ICP)	20750	12651	10985	8597	10938	8555
3 (ICP)	41984	30122	21750	16788	21750	15991
6 (ICP)	134031	100523	77390	70974	77047	70087
11 (ICP)	369828	258300	218125	198556	210515	198547
11 (LUM)	794110	729199	690023	672912	678111	652245
total	1163938	987499	908148	871468	888626	850792

Table 2. Running times in milliseconds for single thread computation vs. multi-thread computation. The first rows represent the speed-up for pure ICP and LUM computations.



(a) (b) (c) (d) (e)

Figure 11. The main square in Horn (Austria). (a) 3D map in top view. (b) Monument in the center of the main square. (c) Corresponding photo (right part). (d) Church spire. (e) Photo of St. Georg church. Data provided by courtesy of RIEGL LMS GmbH [2].

Experiments with a High Resolution Data Set. In the last experiment, we tested the proposed parallelization on high resolution 3D scans provided by RIEGL Laser Measurement Systems GmbH [2]. For mapping we use 11 3D scans containing over 300000 data point each, covering a very large area. Table 2 shows the performance on this data set, Figure 11 shows the final map, two detailed views with photos of the scene. The most gain is achieved by parallelization of ICP, up to a factor of 1.91. The total speed-up on this data set is 1.28. Besides processing these scans on a dual-core T2400, we tried our algorithms on Osnabrück's compute server with 4 dual-core Itanium-II processors and achieved significant speed-ups, up to a factor of 3.4. Measurements are not available since the server is operated in multi-user mode.

7. Summary and Outlook

This paper has presented an approach to parallelize two well know mapping algorithms, namely ICP and LUM. Since 2D mapping is a special case of 3D mapping, the presented algorithms can handle 2D laser range data as well, but processing 3D data imposes a greater need for efficiency. The focus of the parallelization was to keep up with current hardware developments given by the introduction of dual and multi-core CPUs. The achieved speed-ups vary between the algorithms, for ICP a maximum speed-up of factor 1.92 can be reported, which is close to optimal on a dual-core CPU. The average speed-up is lower (approximately 1.25). This difference is due to algorithm parts that cannot be parallelized and due to data managing processes.

In future work, we will concentrate on parallelizing probabilistic robotic algorithms to exploit current computing hardware. Particle filters seem to be good candidates for parallelization. The overall goal is to combine the reliability of probabilistic algorithms, like particle filters with the precision of deterministic approaches like scan matching.

Acknowledgment

The author would like to acknowledge Nikolaus Studnicka (RIEGL Laser Measurement Systems GmbH, Horn) and Oliver Wulf, Bernardo Wagner (Leibniz Universität Hannover) for providing the data sets. Further thanks to Dorit Borrmann, Jan Elseberg for implementing the Lu / Milios style GraphSLAM with 6 DoF [8], to Joachim Hertzberg and Kai Lingemann for supporting this work.

References

[1] `http://www.intel.com/technology/computing/dual-core/`, Intel Dual-Core Processors, 2007.

[2] `http://www.riegl.co.at/`, Riegl Laserscanner, 2007.

[3] K. S. ARUN, T. S. HUANG, AND S. D. BLOSTEIN. Least square fitting of two 3-d point sets. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 9(5):698 – 700, 1987.

[4] S. ARYA AND D. M. MOUNT. Approximate nearest neighbor queries in fixed dimensions. In *Proceedings of the 4th ACM-SIAM Symposium on Discrete Algorithms*, pages 271 – 280, 1993.

[5] S. ARYA, D. M. MOUNT, N. S. NETANYAHU, R. SILVERMAN, AND A. Y. WU. An Optimal Algorithm for Approximate Nearest Neighbor Searching in Fixed Dimensions. *Journal of the ACM*, 45:891 – 923, 1998.

[6] J. L. BENTLEY. Multidimensional binary search trees used for associative searching. *Communications of the ACM*, 18(9):509 – 517, September 1975.

[7] P. BESL AND N. MCKY. A method for Registration of 3–D Shapes. *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)*, 14(2):239 – 256, February 1992.

[8] D. BORRMANN, J. ELSEBERG, K. LINGEMANN, A. NÜCHTER, AND J. HERTZBERG. Globally consistent 3d mapping with scan matching. *Journal Robotics and Autonomous Systems*, 56(2), 2008.

[9] D. M. COLE AND P. M. NEWMAN. Using Laser Range Data for 3D SLAM in Outdoor Environments. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA '06)*, Orlando, Florida, U.S.A., May 2006.

[10] U. FRESE. Efficient 6-DOF SLAM with Treemap as a Generic Backend. In *Proc. of the IEEE Internation Conference on Robotics and Automation (ICRA '07)*, Rome, Italy, April 2007.

[11] J. H. FRIEDMAN, J. L. BENTLEY, AND R. A. FINKEL. An algorithm for finding best matches in logarithmic expected time. *ACM Transaction on Mathematical Software*, 3(3):209 – 226, September 1977.

[12] M. GREENSPAN AND M. YURICK. Approximate K-D Tree Search for Efficient ICP. In *Proc. of the 4th IEEE Int. Conf. on Recent Advances in 3D Digital Imaging and Modeling (3DIM '03)*, pages 442 – 448, Banff, Canada, October 2003.

[13] M. A. GREENSPAN, G. GODIN, AND J. TALBOT. Acceleration of Binning Nearest Neighbor Methods. In *Proc. of the Conference Vision Interface*, 2000.

[14] B. K. P. HORN. Closed–form solution of absolute orientation using unit quaternions. *Journal of the Optical Society of America A*, 4(4):629 – 642, April 1987.

[15] B. K. P. HORN, H. M. HILDEN, AND SH. NEGAHDARIPOUR. Closed–form solution of absolute orientation using orthonormal matrices. *Journal of the Optical Society of America A*, 5(7):1127 – 1135, July 1988.

[16] R. KATZ, N. MELKUMYAN, J. GUIVANT, T. BAILEY, J. NIETO, AND E. NEBOT. Integrated Sensing Framework for 3D Mapping in Outdoor Navigation. In *Proc. of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS '06)*, Beijing, China, October 2006.

[17] C. LANGIS, M. GREENSPAN, AND G. GODIN. The parallel iterative closest point algorithm. In *Proceedings of the 3rd IEEE International Conference on Recent Advances in 3D Digital Imaging and Modeling (3DIM '01)*, pages 195 – 204, Quebec City, Canada, June 2001.

[18] A. LORUSSO, D. EGGERT, AND R. FISHER. A Comparison of Four Algorithms for Estimating 3-D Rigid Transformations. In *Proc. of the 4th British Machine Vision Conference (BMVC '95)*, pages 237 – 246, Birmingham, Sept. 1995.

[19] F. LU AND E. MILIOS. Globally Consistent Range Scan Alignment for Environment Mapping. *Autonomous Robots*, 4(4):333 – 349, October 1997.

[20] P. M. NEWMAN, D. M. COLE, AND K. HO. Outdoor SLAM using Visual Appearance and Laser Ranging. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA '06)*, Barcelona, Spain, April 2005.

[21] A. NÜCHTER, K. LINGEMANN, J. HERTZBERG, AND H. SURMANN. 6D SLAM with Approximate Data Associoation. In *Proc. of the 12th IEEE Int. Conf. on Advanced Robotics (ICAR '05)*, pages 242 – 249, Seattle, U.S.A., July 2005.

[22] A. NÜCHTER, K. LINGEMANN, J. HERTZBERG, H. SURMANN, K. PERVÖLZ, M. HENNIG, K. R. TIRUCHINAPALLI, R. WORST, AND T. CHRISTALLER. Mapping of Rescue Environments with Kurt3D. In *Proc. of the IEEE Int. Workshop on Rescue Robotics (SSRR '05)*, pages 158 – 163, 2005.

[23] E. E. SANTOS AND P.-Y. CHU. Efficient and optimal parallel algorithms for Cholesky decomposition. *Journal of Mathematical Modelling and Algorithms*, 2(3):217–234, September 2003.

[24] H. SURMANN, K. LINGEMANN, A. NÜCHTER, AND J. HERTZBERG. A 3D laser range finder for autonomous mobile robots. In *Proc. of the 32nd Int. Symposium on Robotics (ISR '01)*, pages 153 – 158, Seoul, Korea, April 2001.

[25] H. SURMANN, A. NÜCHTER, K. LINGEMANN, AND J. HERTZBERG. 6D SLAM A Preliminary Report on Closing the Loop in Six Dimensions. In *Proc. of the 5th IFAC Symp. on Intelligent Autonomous Vehicles (IAV '04)*, Portugal, 2004.

[26] S. THRUN. ROBOTIC MAPPING: A SURVEY. In G. Lakemeyer and B. Nebel, editors, *Exploring Artificial Intelligence in the New Millenium*. Morgan Kaufmann, 2002.

[27] R. TRIEBEL AND W. BURGARD. Improving Simultaneous Localization and Mapping in 3D Using Global Constraints. In *Proceedings of the National Conference on Artificial Intelligence (AAAI '05)*, 2005.

[28] R. TRIEBEL, P. PFAFF, AND W. BURGARD. Multi-level surface maps for outdoor terrain mapping and loop closing. In *Proc. of the IEEE/RSJ Int. Conference on Intelligent Robots and Systems (IROS '06)*, Beijing, China, October 2006.

[29] M. W. WALKER, L. SHAO, AND R. A. VOLZ. Estimating 3-d location parameters using dual number quaternions. *CVGIP: Image Understanding*, 54:358 – 367, November 1991.

[30] J. WEINGARTEN AND R. SIEGWART. EKF-based 3D SLAM for structured environment reconstruction. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS '05)*, pages 2089 – 2094, Edmonton, Alberta Canada, August 2005.

[31] O. WULF, K. O. ARRAS, H. I. CHRISTENSEN, AND B. A. WAGNER. 2D Mapping of Cluttered Indoor Environments by Means of 3D Perception. In *Proc. of the IEEE Int. Conf. on Robotics and Automation (ICRA '04)*, pages 4204 – 4209, New Orleans, USA, April 2004.

Received: November, 2007
Revised: March, 2008
Accepted: May, 2008

Contact address:
Andreas Nüchter
Institute of Computer Science
University of Osnabrück
Germany
e-mail: nuechter@informatik.uni-osnabrueck.de

ANDREAS NÜCHTER is a research associate at the University of Osnabrück. His past affiliations were with the Fraunhofer Institute for Autonomous Intelligent Systems (AIS, Sankt Augustin) and the University of Bonn, from which he received the diploma degree in computer science in 2002 (best paper award from the German Society of Informatics (GI) for his thesis). He holds a doctoral degree from the University of Bonn. His research interests include reliable robot control, 3D environment mapping, 3D vision, and laser scanning technologies, resulting in fast 3D scan matching algorithms that enable robots to map their environment in 3D using 6 degrees of freedom poses. The capabilities of these robotic SLAM approaches were demonstrated at RoboCup Rescue competitions, ELROB and several other events. He is a member of the GI and the IEEE.