2009 IEEE International Conference on Robotics and Automation
Kobe International Conference Center
Kobe, Japan, May 12-17, 2009

Comparison of Surface Normal Estimation Methods for Range Sensing Applications

Klaas Klasing Daniel Althoff Dirk Wollherr Martin Buss
Institute of Automatic Control Engineering
Technische Universität München
80290 Munich, Germany
{kk,da,dw,mb}@tum.de

Abstract— **As mobile robotics is gradually moving towards a level of semantic environment understanding, robust 3D object recognition plays an increasingly important role. One of the most crucial prerequisites for object recognition is a set of fast algorithms for geometry segmentation and extraction, which in turn rely on surface normal vectors as a fundamental feature. Although there exists a plethora of different approaches for estimating normal vectors from 3D point clouds, it is largely unclear which methods are preferable for online processing on a mobile robot. This paper presents a detailed analysis and comparison of existing methods for surface normal estimation with a special emphasis on the trade-off between quality and speed. The study sheds light on the computational complexity as well as the qualitative differences between methods and provides guidelines on choosing the 'right' algorithm for the robotics practitioner. The robustness of the methods with respect to noise and neighborhood size is analyzed. All algorithms are benchmarked with simulated as well as real 3D laser data obtained from a mobile robot.**

I. Introduction

Mobile robotics has grown of age in the past decade. While fundamental challenges such as 2D SLAM, navigation and obstacle avoidance have been successfully overcome for structured indoor environments, more advanced capabilities such as 3D object recognition and interpretation are still in their childhood. One of the most important features for geometry reconstruction and interpretation of 3D range data is the surface normal vector. However, since it cannot be measured directly, it must be estimated for every point of a (noisy) point cloud. Many researchers have addressed this problem in the context of computational geometry, where the goal is to find methods that deliver the highest quality possible, usually regardless of computation time because the data is processed offline. In this paper, representative methods for normal vector estimation are analyzed and compared with respect to their performance in terms of accuracy *and* speed. The motivation behind this is that for a mobile robot which is to recognize objects in an online fashion, the generation of surface normals is merely a preprocessing step that must be carried out extremely fast.

The methods presented in this paper are applicable to 3D point clouds as sampled from real-world surfaces by a range sensor. The experimental results are based on the laser range data obtained by the 3D scanning setup described in [1]. However, the variation of the results with respect to different depth noise is examined, so that the analysis also applies for point clouds obtained by other sensors, e.g. stereo cameras.

The remainder of this paper is structured as follows: Section II provides a problem statement and addresses the problem of local neighborhood. Section III presents the chosen methods for normal vector estimation. In Section IV the simulated and real data sets as well as as the benchmark criteria are presented. Results are presented in Section V and discussed in Section VI.

II. Surface Normal Estimation

A. Problem Statement

3D point clouds obtained from range sensors represent a noisy sampling of surfaces that exist in the real world. The explicit information about orientation and curvature of these surface is lost in the sampling process. It resides implicitly in the relationship between a sampled point and points in its neighborhood. Normal vector estimation seeks to restore this information by constructing a set of vectors that are orthogonal to the tangential plane of each surface point they touch. The estimated normal vector is a so-called local descriptor. The majority of methods try to estimate a normal vector for every point in the point cloud. This is a reasonable procedure especially if the resulting normal vector is integrated into a feature space in which every data item corresponds to one point in the point cloud.

We consider a set of n points $P = \{\boldsymbol{p}_1, \boldsymbol{p}_2, ..., \boldsymbol{p}_n\}, \boldsymbol{p}_i \in \mathbb{R}^3$ which shall be represented by a data matrix

$$\boldsymbol{P} = [\boldsymbol{p}_1, \ \cdots \ , \ \boldsymbol{p}_n]^T,$$

where $\boldsymbol{p}_i = [p_{ix}, p_{iy}, p_{iz}]^T$ represent the 3D coordinates of the measured points. For every point $\boldsymbol{p}_i$ we would like to estimate a normal vector $\boldsymbol{n}_i = [n_{ix}, n_{iy}, n_{iz}]^T$ from a set of k points in its neighborhood $Q_i = \{\boldsymbol{q}_{i1}, \boldsymbol{q}_{i2}, ..., \boldsymbol{q}_{ik}\}, \boldsymbol{q}_{ij} \in P, \boldsymbol{q}_{ij} \neq \boldsymbol{p}_i$. In the following we will refer to

$$\boldsymbol{Q}_i = [\boldsymbol{q}_{i1}, \ \cdots \ , \ \boldsymbol{q}_{ik}]^T \text{ and } \boldsymbol{Q}_i^{+} = [\boldsymbol{p}_i, \ \boldsymbol{q}_{i1}, \ \cdots \ , \ \boldsymbol{q}_{ik}]^T,$$

where $\boldsymbol{Q}_i$ is the neighbor matrix and $\boldsymbol{Q}_i^{+}$ is the augmented neighbor matrix containing all neighbors plus the point $\boldsymbol{p}_i$ itself.

B. Nearest Neighbors

Identifying a suitable set of neighboring points is a non-trivial task. The most natural data structure that captures the notion of neighborhood is an undirected graph $\boldsymbol{G} = (\boldsymbol{V}, \boldsymbol{E})$, in which every vertex $\boldsymbol{v}_i \in \boldsymbol{V}$ corresponds to a point $\boldsymbol{p}_i \in \boldsymbol{P}$ and is connected to neighbors via edges $\boldsymbol{e}_{ij} \in \boldsymbol{E}$.

There are essentially only two types of graph that are used for normal vector estimation:

- The *k nearest neighbor* (kNN) graph connects every vertex to its k nearest neighbors, where 'nearness' is measured by a Euclidean distance metric $\rho(\boldsymbol{v}_i, \boldsymbol{v}_j) = \left\|\boldsymbol{p}_i - \boldsymbol{p}_j\right\|_2$ for the scope of this paper. The number of neighbors is fixed and this may lead to overlapping triangles especially at the rim of a point cloud.
- The *Delaunay tessellation* (DT) graph connects simplices of points, such that the circumsphere of each resulting simplex does not contain any of the other points. In 3D this yields a tetrahedral mesh, in 2D we obtain a triangular mesh. Each vertex has a variable number of neighbors and there are never any overlapping triangles.

The methods presented in the following are benchmarked on both types of graph. Implementation-specific details are explained in Subsection IV-D.

III. Examined Methods

Existing normal estimation methods can be divided into *optimization-based* methods and *averaging* methods. In the following the methods are briefly described.

A. Optimization-based Methods

Many existing methods calculate the normal vector by solving the optimization problem

$$\min_{\boldsymbol{n}_i} J(\boldsymbol{p}_i, \boldsymbol{Q}_i, \boldsymbol{n}_i),$$

where $J(\boldsymbol{p}_i, \boldsymbol{Q}_i, \boldsymbol{n}_i)$ is a cost functional penalizing certain criteria, e.g. the distance of points to a local plane or the angle between tangential vectors and the normal vector. Figure 1 (a) and (b) visualize these two approaches. As a matter of fact, many seemingly very different methods differ only in the cost functional they optimize. Furthermore the computational performance depends to a large degree on the optimization procedure that is used. In case J can be stated as a linear problem in matrix-vector notation, the minimizer can be expressed directly as the result of a *singular value decomposition* (SVD) or a *principal component analysis* (PCA). In case of a nonlinear dependence for which an analytic solution becomes infeasible, gradient techniques must be employed. All optimization-based methods examined in this paper pose a convex linear optimization problem, so that gradient techniques and their related problems are not examined.

The optimization-based methods can be divided into linear and quadratic fitting approaches. This paper considers three linear and two quadratic approaches. It is shown that many existing methods belong to these basic categories or to variations of them.

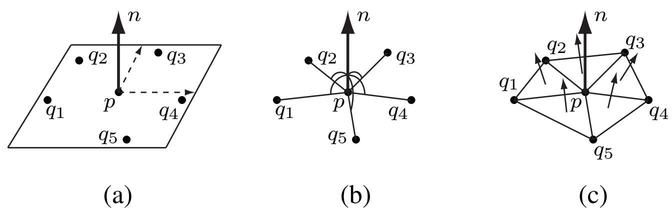

Fig. 1. Different approaches for estimating normal vectors: (a) A plane is fitted to p and its neighbors. (b) The angle between the normal vector and the tangential vectors is maximized. (c) The normal vectors of triangles formed with pairs of neighbors are averaged.

1) PlaneSVD: A classical method [2], [3] is to fit a local plane $S_i = n_{ix}x + n_{iy}y + n_{iz}z + d$ to the points in $\boldsymbol{Q}_i^+$, i.e. solve

$$\min_{\boldsymbol{b}_i} \left\| \begin{bmatrix} \boldsymbol{Q}_i^+ & \boldsymbol{1}_{k+1} \end{bmatrix} \boldsymbol{b}_i \right\|_2, \tag{1}$$

where $\boldsymbol{1}_m$ represents an $m \times 1$ vector of ones and $\boldsymbol{b}_i = [\boldsymbol{n}_i^T, d]^T$. Just as any of the following methods, this optimization problem can be solved by computing the SVD ($\boldsymbol{U\Sigma V}^T$) of $\boldsymbol{Q}_i^+$. The minimizer $\hat{\boldsymbol{b}}_i$ is the vector in $\boldsymbol{V}$ that corresponds to the smallest singular value in Σ, see [4], and thus the normal vector $\boldsymbol{n}_i$ is obtained directly.

2) PlanePCA: Instead of minimizing the fitting error, one can minimize the variance by removing the empirical mean from the data matrix $\boldsymbol{Q}_i^+$ and then performing a SVD on the modified data matrix [5]:

$$\min_{\boldsymbol{n}_i} \left\| \left[\boldsymbol{Q}_i^+ - \bar{\boldsymbol{Q}}_i^+ \right] \boldsymbol{n}_i \right\|_2, \tag{2}$$

where $\bar{\boldsymbol{Q}}_i^+ = \boldsymbol{1_{k+1}} \bar{\boldsymbol{q}}_i^{+T}$ is a matrix containing the mean vector $\bar{\boldsymbol{q}}_i^+ = \frac{1}{k+1}(\boldsymbol{p}_i + \sum_{j=1}^{k} \boldsymbol{q}_{ij})$ in every row. This is equivalent to performing a PCA of the original matrix $\boldsymbol{Q}_i^+$ and choosing the principal component with the smallest covariance, which is why this method is dubbed *PlanePCA*.

Interestingly enough, this formulation is equivalent to the *Maximum Likelihood* estimation of the local plane described in [6] and [7]. If the points $\boldsymbol{p}_i$ are regarded as measurements perturbed by independent Gaussian noise $N(0, \sigma)$ around their true position $\hat{\boldsymbol{p}}_i = [\hat{p}_{ix}, \hat{p}_{iy}, \hat{p}_{iz}]$ in the x, y, and z direction, then maximizing the likelihood

$$\mathcal{P} = \prod_{\boldsymbol{p}_j \in \boldsymbol{Q}_i^+} \frac{1}{2\pi\sigma^2} e^{-\frac{[(p_{jx} - \hat{p}_{jx})^2 + (p_{jy} - \hat{p}_{jy})^2 + (p_{jz} - \hat{p}_{jz})^2]}{2\pi\sigma^2}} \tag{3}$$

of the plane fit given the measurements is equivalent to minimizing

$$J = \sum_{\boldsymbol{p}_j \in \boldsymbol{Q}_i^+} \frac{(\boldsymbol{n}_i \boldsymbol{p}_j)^2}{\|\boldsymbol{n}_i\|_2^2}, \quad \text{s.t. } \|\boldsymbol{n}_i\|_2 = 1 \tag{4}$$

as is shown in [7]. In [6] $\boldsymbol{n}_i$ is then obtained as the eigenvector corresponding to the largest eigenvalue of

$$G = \frac{1}{k+1} \sum_{\boldsymbol{q}_i^+ \in \boldsymbol{Q}_i^+} (\boldsymbol{q}_i^+ - \bar{\boldsymbol{q}}_i^+)(\boldsymbol{q}_i^+ - \bar{\boldsymbol{q}}_i^+)^T, \tag{5}$$

which is, however, equivalent to the minimizer found by a SVD applied to (2), see [8]. Thus a PCA performed on the neighborhood $\boldsymbol{Q}_i^+$ is equivalent to a maximum likelihood

TABLE I
OPTIMIZATION-BASED METHODS

	#	Method	$J(\boldsymbol{p}_i, \boldsymbol{Q}_i, \boldsymbol{n}_i)$	Matrix Size
LINEAR	1	*PlaneSVD*	$\left\| \left[\boldsymbol{Q}_i^+, \mathbf{1}_{k+1}\right] \left[\boldsymbol{n}_i^T, d\right]^T \right\|_2$	$(k+1) \times 4$
	2	*PlanePCA (ML)*	$\left\| \left[\boldsymbol{Q}_i^+ - \mathbf{1}_{k+1} \bar{\boldsymbol{q}}_i^{+T}\right] \boldsymbol{n}_i \right\|_2$	$(k+1) \times 3$
	3	*VectorSVD*	$\left\| \left[\boldsymbol{Q}_i - \mathbf{1}_k \boldsymbol{p}_i^T\right] \boldsymbol{n}_i \right\|_2$	$k \times 3$
QUADR.	4	*QuadSVD*	$\left\| \left[\boldsymbol{R}_i, \mathbf{1}_k\right] \boldsymbol{c}_i \right\|_2$	$k \times 10$
	5	*QuadTransSVD*	$\left\| \left[\boldsymbol{R}_i', \mathbf{1}_k\right] \boldsymbol{c}_i - \boldsymbol{q}_{iz}' \right\|_2$	$k \times 6$

estimation of a local plane fitted in $\boldsymbol{Q}_i^+$.

3) VectorSVD: An equally straight-forward alternative to fitting a local plane S into $\boldsymbol{Q}_i^+$ is to maximize the angle (minimize the inner product) between the tangential vectors from $\boldsymbol{p}_i$ to $\boldsymbol{q}_{ij}$ and the normal vector $\boldsymbol{n}_i$. A moment's thought should convince the reader that this is equivalent to fitting a local plane that is fixed in $\boldsymbol{p}_i$, or – in other words – shifting the origin to $\boldsymbol{p}_i$ and then fitting a plane $S_i = \boldsymbol{n}_{ix}x + \boldsymbol{n}_{iy}y + \boldsymbol{n}_{iz}z$ to the points in $\boldsymbol{Q}_i$:

$$\min_{\boldsymbol{n}_i} \left\| \left[\boldsymbol{Q}_i - \mathbf{1}_k \boldsymbol{p}_i^T\right] \boldsymbol{n}_i \right\|_2 \tag{6}$$

In analogy to the *PlanePCA* method it is tempting to define a *VectorPCA* method, that minimizes the variance of the inner product defined in *VectorSVD*. In fact this is the approach presented in [9]. However, the results produced by this method are almost identical to the ones obtained from *PlanePCA*, because their data matrices differ only by $\boldsymbol{p}_i$. When the centroid of $\boldsymbol{Q}_i$ is equal to $\boldsymbol{p}_i$ the two methods are identical. *VectorPCA* is therefore not considered in the benchmark.

4) QuadSVD: Some methods approximate not only the orientation of the tangent plane but also the curvature [10], [11] by fitting a quadric surface of the form $S = c_1x^2 + c_2y^2 + c_3z^2 + c_4xy + c_5xz + c_6yz + c_7x + c_8y + c_9z + c_{10}$ to the set of neighbors. The optimization problem is then

$$\min_{\boldsymbol{c}_i} \left\| \left[\boldsymbol{R}_i, \mathbf{1}_k\right] \boldsymbol{c}_i \right\|_2, \tag{7}$$

where $\boldsymbol{R}_i$ contains the linear and quadratic data items $[q_{ix}^2, q_{iy}^2, q_{iz}^2, q_{ix}q_{iy}, q_{ix}q_{iz}, q_{iy}q_{iz}, q_{ix}, q_{iy}, q_{iz}]$ in each row and $\boldsymbol{c}_i = [c_{i1}, \cdots, c_{i10}]$ is the coefficient vector, which is obtained by a SVD of $\boldsymbol{R}_i$. $\boldsymbol{n}_i$ is then obtained directly from the coefficients, see [12] for details.

A major drawback of this method is that, depending on the neighborhood strategy, $\boldsymbol{Q}_i$ may not contain ten or more neighbors, which makes fitting S to $\boldsymbol{Q}_i$ an ill-posed problem. The next method represents a partial remedy of this problem.

5) QuadTransSVD: A common procedure is to transform the points to a coordinate system in which some dependence $z = f(x, y)$ can be exploited to reduce the number of coefficients to be estimated. In [13] a suitable coordinate system is identified by fitting a local plane to $\boldsymbol{Q}_i$ (with *VectorSVD*) and using the resulting normal vector as the new z-axis of a cartesian coordinate system with origin $\boldsymbol{p}_i$. After transforming $\boldsymbol{Q}_i$ to this system, the quadric surface $S = c_1x^2 + c_2y^2 + c_3xy + c_4x + c_5y + c_6 - z$ is fitted to the set of transformed points $\boldsymbol{Q}_i'$. This is accomplished by solving

$$\min_{\boldsymbol{c}_i} \left\| \left[\boldsymbol{R}_i', \mathbf{1}_k\right] \boldsymbol{c}_i - \boldsymbol{q}_{iz}' \right\|_2, \tag{8}$$

where $\boldsymbol{R}_i'$ contains the linear and quadratic data items $[q_{ix}'^2, q_{iy}'^2, q_{ix}'q_{iy}', q_{ix}', q_{iy}']$ in each row, $\boldsymbol{c}_i = [c_{i1}, \cdots, c_{i6}]$ is the coefficient vector and $\boldsymbol{q}_{iz}'$ is the vector of transformed z-coordinates. The solution is again found via the SVD ($U\Sigma V^T$) of $\boldsymbol{R}_i'$ by computing[1]

$$\boldsymbol{c}_i = \sum_{j=1}^{r} \frac{u_j^T \boldsymbol{q}_{iz}'}{\sigma_j} \boldsymbol{v}_j,$$

where $r = \operatorname{rank}(\boldsymbol{R}_i') = 3$, see [4]. With the coefficient vector $\boldsymbol{c}_i$ at hand, the normal vector in the transformed coordinate system is obtained by the cross product

$$\boldsymbol{n}_i' = [1, 0, c_{i4}]^T \times [0, 1, c_{i5}]^T$$

and $\boldsymbol{n}_i$ is finally obtained by transforming $\boldsymbol{n}_i'$ back to the original coordinate system.

B. Averaging Methods

An attractive alternative to finding the normal vector by optimization is to calculate it as the weighted average of the normal vectors of the triangles formed by p_i and pairs of its neighbors, see Figure 1 (c). In theory there are $\binom{k}{2} = \frac{1}{2}k(k-1)$ such triangles, however, usually one only steps through k pairs of consecutively stored neighbors $\boldsymbol{q}_{i,j}$ and $\boldsymbol{q}_{i,j+1}$, assuming that the neighbors are stored in some reasonable order. The DT graph provides an angular ordering in a local plane, but we shall see that even the kNN graph, where no specific ordering is obeyed, yields good results if the number of neighbors is sufficient. The basic averaging method [14] is[2]

$$\boldsymbol{n}_i = \frac{1}{k} \sum_{j=1}^{k} w_j \frac{\left([\boldsymbol{q}_{i,j} - \boldsymbol{p}_i] \times [\boldsymbol{q}_{i,j+1} - \boldsymbol{p}_i]\right)}{\left|[\boldsymbol{q}_{i,j} - \boldsymbol{p}_i] \times [\boldsymbol{q}_{i,j+1} - \boldsymbol{p}_i]\right|}, \tag{9}$$

where $w_j = 1$. There exist numerous variations of this scheme, which are all based on using a weighted average ($w_j \neq 1$) instead of the basic average and include among others *angle-weighted*, *area-weighted*, *centroid-weighted* and *gravitational-weighted* methods. A good summary of these methods can be found in [15]. Due to space constraints and due to the similar performance of many variants only two representative methods are considered in the following.

1) AreaWeighted: This method represents the fastest algorithm of the averaging methods. The normal vector of each triangle is weighted by the magnitude of its area, which is in turn equal to half the magnitude of the normal vector, thus

$$w_j = \frac{1}{2} \left|[\boldsymbol{q}_{i,j} - \boldsymbol{p}_i] \times [\boldsymbol{q}_{i,j+1} - \boldsymbol{p}_i]\right|, \tag{10}$$

[1]This procedure is equivalent to but faster than the explicit computation of the pseudo-inverse.

[2]For notational simplicity assume here that the last-plus-one index maps onto the first neighbor again, i.e. $k + 1 := 1$.

TABLE II

AVERAGING METHODS

#	Method	Weighting Factor
6	*AreaWeighted*	$\frac{1}{2}\left\|[\boldsymbol{q}_{i,j}-\boldsymbol{p}_i]\times[\boldsymbol{q}_{i,j+1}-\boldsymbol{p}_i]\right\|$
7	*AngleWeighted*	$\cos^{-1}\left(\frac{\langle \boldsymbol{q}_{i,j}-\boldsymbol{p}_i, \boldsymbol{q}_{i,j+1}-\boldsymbol{p}_i\rangle}{\|\boldsymbol{q}_{i,j}-\boldsymbol{p}_i\|\|\boldsymbol{q}_{i,j+1}-\boldsymbol{p}_i\|}\right)$

which cancels the normalization factor in (9); so the normal vector is simply the average of the (unnormalized) cross products of the adjacent triangles.

2) *AngleWeighted:* Some researchers use the angle between two consecutive tangential vectors instead of the triangle area [15]. The weight is then expressed as

$$w_j = \cos^{-1}\left(\frac{\langle \boldsymbol{q}_{i,j}-\boldsymbol{p}_i, \boldsymbol{q}_{i,j+1}-\boldsymbol{p}_i\rangle}{|\boldsymbol{q}_{i,j}-\boldsymbol{p}_i|\,|\boldsymbol{q}_{i,j+1}-\boldsymbol{p}_i|}\right), \tag{11}$$

which penalizes acute triangle 'shards'.

C. Computational Complexity

In terms of computational complexity the five optimization-based methods differ mostly in the size of the matrix that is to be decomposed by the SVD. The SVD has $O(d^3)$, where d is number of columns. Thus all linear methods should perform approximately the same, while *QuadSVD* should be substantially slower ($d = 10$). *QuadTransSVD* has $d = 6$, however, there is an extra SVD involved for finding the transformation plane and also the overhead of the two transformations contributes to the overall cost. For the two averaging methods no optimization is needed, thus they should perform substantially faster than the previous methods.

Table I summarizes the cost functionals and matrix sizes of the optimization-based methods and Table II lists the weighting functions of the averaging methods.

IV. BENCHMARK

A. Performance Index

Given the analytic comparison of the methods and their computational effort from the previous section the question is how they perform in terms of quality. Let us define the quality γ of an estimated normal vector $\boldsymbol{n}_i$ as the absolute value of its normalized inner product with the ground truth normal vector $\hat{\boldsymbol{n}}_i$:

$$\gamma_i := \gamma(\boldsymbol{n}_i, \hat{\boldsymbol{n}}_i) = \frac{|\langle \boldsymbol{n}_i, \hat{\boldsymbol{n}}_i\rangle|}{|\boldsymbol{n}_i|\,|\hat{\boldsymbol{n}}_i|} \tag{12}$$

The central question raised in this paper is whether the increased computational effort of some methods is justified by a higher overall quality $\gamma = \frac{1}{n}\sum_{i=1}^{n}\gamma_i$. We therefore define a *performance index*

$$\pi(\hat{t}) = \frac{1}{2}\gamma + \frac{1}{2}\left(\frac{\hat{t}}{t}\right), \tag{13}$$

where t is the computation time of the algorithm and $\hat{t}$ is the optimal time we wish for a surface normal estimation algorithm to have. The rationale behind this is that while quality

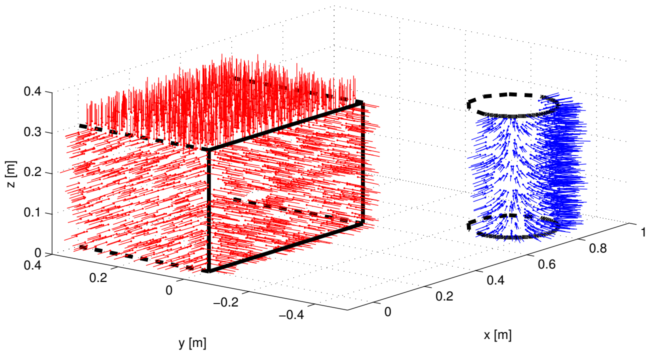

Fig. 2. Simulated noisy data and ground truth normal vectors of a box and a cylinder.

is naturally normalized by comparison with the ground truth, there exists no such normalization for computation time. This dilemma is overcome by letting the user define a reasonable lower bound $\hat{t}$. The ideal algorithm would yield $\gamma = 1$ in $t = \hat{t}$ time and receive a perfect performance index of $\pi(\hat{t}) = 1$. Of course, this performance index may be varied to include different weighting of the two criteria quality and time or a nonlinear penalty of higher computation times. However, for the scope of this paper the simple index from (13) shall suffice to produce a ranking of methods.

B. Scenario I: Simulated Point Cloud

The first scenario used for benchmarking is a simulated noisy point cloud covering the surface of a box and a cylinder shown in Figure 2. In indoor robotics an overwhelming portion of the measured data consists of such simple surfaces (walls, doors, cupboards, recycling bins etc). In the simulation the points were perturbed by Gaussian measurement noise $N(0, \sigma)$ around the true measurement value along the line of sight of the individual rays. This technique is suitable for simulating range data from both laser ranger finders as well as time-of-flight cameras [16]. All methods were benchmarked with different values of depth noise σ and different neighborhood sizes k. The ground truth normal vectors in this scenario were obtained effortlessly from the analytical description of the simulated surfaces.

C. Scenario II: Real Point Cloud

The second benchmark scenario consists of a point cloud obtained from a SICK LMS400 laser scanner mounted on the *Autonomous City Explorer* robot, see [17]. A description of the scanning setup can be found in [1]. To be able to reconstruct the ground truth normal vectors, three simple objects (two boxes and one cylindrical recycling bin) were placed in the scene, see Figure 3 (top), and their exact positions and orientation in the scene were measured. The resulting point cloud can be seen in Figure 3 (bottom).

D. Implementation Details

All algorithms were implemented in C and benchmarked on a Linux desktop computer with a 3.0GHz AMD Athlon XP processor and 2GB RAM. Nearest neighbors were computed using kd-trees from the ANN library[3] and Delaunay

[3]`http://www.cs.umd.edu/~mount/ANN/`

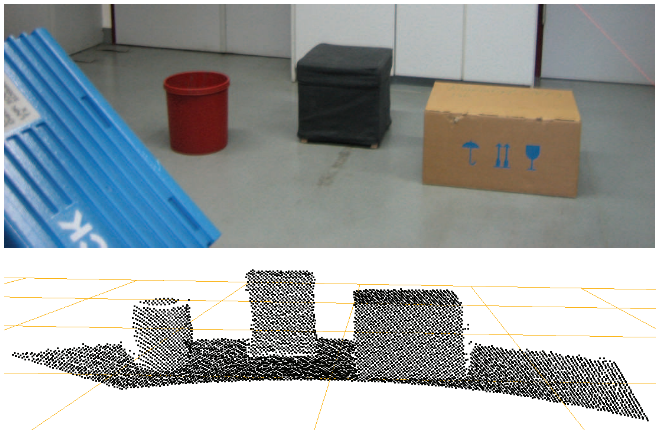
Fig. 3. Scanning setup: (top) real objects, (bottom) resulting point cloud

tessellations were generated with the Qhull package[4]. The singular value decompositions and matrix operations were carried out with the GNU Scientific Library[5] in conjunction with CBLAS.

Although the two types of neighborhood graph described in subsection II-B are relatively simple data structures, their construction is by no means simple. As was recently shown [18], for range sensing applications it is possible to maintain and continuously update a kNN graph. The 3D Delaunay tessellation is cumbersome because it yields a tetrahedral mesh which must be simplified to a triangular mesh that is aligned with the true surface. For this paper, the triangular surface mesh was obtained by projecting the neighborhood $\boldsymbol{Q}_i^+$ onto a local plane and calculating the 2D Delaunay tessellation in this plane. It should be noted that the choice of graph depends mostly on the user's needs and its efficient construction depends on the specific range sensing scenario. The present study aims to shed light on the efficiency of normal vector estimation itself, therefore the graph construction time is not included in the benchmarked computation time of the algorithms. All methods were benchmarked on both a kNN graph and a DT graph, except for the two quadratic optimization-based methods, which were benchmarked on a kNN graph only. This is because the DT graph yields a variable neighborhood size and does not always produce the required number of neighbors for the quadric fit.

V. Results

The results for Scenario I can be seen in Figure 4(a)-(d). The simulated data set consists of $14,200$ points. For the kNN benchmark, the neighborhood size was varied between $k = 10$ and $k = 50$, while σ, the measurement noise standard deviation, was kept constant at $\sigma = 15\text{mm}$[6]. The running times in Figure 4(b) hold no surprises except that the linear optimization-based methods outperform the averaging methods already for medium-sized neighborhoods. The overall quality γ of all methods depicted in Figure 4(a) is rather poor because of the strong noise σ. The presence of such noise clearly separates the methods from each other:

[4]`http://www.qhull.org/`
[5]`http://www.gnu.org/software/gsl/`
[6]This is the maximum statistical error of the SICK LMS400.

TABLE III
Results Scenario II: DT Graph

Method:	1	2	3	6	7
γ:	0.9413	0.9436	0.9328	0.9180	0.8840
t [msecs]:	75.65	58.44	62.00	32.64	63.24

PlaneSVD, *PlanePCA*, and *QuadTransSVD* take the lead with a strong increase in quality starting at medium-sized neighborhoods. The two averaging methods finish last with only slow improvement, while *QuadSVD* and *VectorSVD* show a mediocre performance. Figure 4(c) shows the resulting performance index $\pi(20\text{ms})$, meaning that we expect the ideal algorithm to compute the normal vectors in 20ms. It reflects the strength of *PlanePCA* and *PlaneSVD*, which receive high scores for both quality and running time on a kNN graph. Figure 4(d) shows the benchmark of the five applicable methods (all except the quadratic ones) for the DT graph. The variation of quality with the noise level clearly shows that all methods suffer from the locality of the DT graph. In all benchmark scenarios the number of neighbors in the DT graph never exceeded ten, whereas the kNN graph could compensate noise for neighborhoods of size ten and above.

Scenario II contains the real range scan consisting of $7,346$ points and with a much smaller noise level σ. The ranking of running times is the exact same as for the simulated data and is omitted due to space constraints; absolute running times were halved, which corresponds to the fact that there are half the number of points in the data set. Figure 4(e) shows that with smaller noise, *QuadTransSVD* performs even slightly better than *PlanePCA* and the two averaging methods show constant improvement of γ with increasing neighborhood size, albeit still at a poor performance level. The performance index $\pi(10\text{ms})$ in Figure 4(f) shows that *PlanePCA* still yields the best quality and speed for medium-sized neighborhoods for the real range data, this time sharing the lead with *VectorSVD*. Since the noise was constant (but unknown), the DT graph benchmark yielded only one set of values shown in Table III. The running times and qualities show that with low noise the *AreaWeighted* method becomes a favorable alternative on a DT graph.

VI. Conclusion

In this paper a detailed analysis and comparison of different methods for normal vector estimation has been presented. Numerous algorithms were shown to belong to one of two basic classes and representative approaches were benchmarked on simulated as well as real laser data obtained from a mobile robot. An important conclusion is that the choice of the 'best' algorithm depends mostly on the graph structure chosen by the user. If a triangular mesh such as the DT graph is available (and can be updated incrementally) then the *AreaWeighted* method yields the best speed at acceptable quality. In case a kNN graph is maintained and updated, *PlanePCA* is the universal method of choice because of its superior performance in terms both quality and speed.

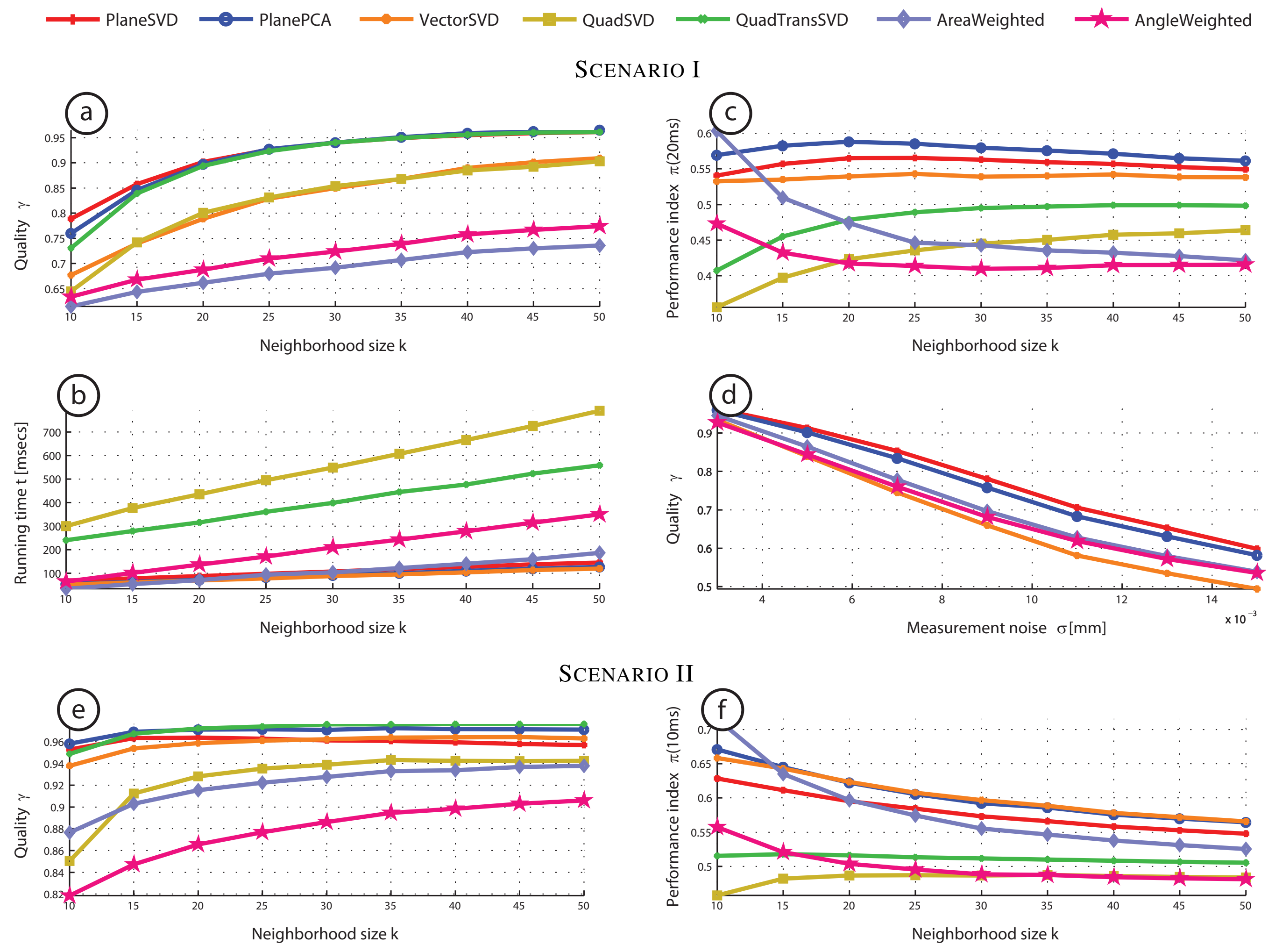

Fig. 4. Results for Scenario I (a)-(d) and Scenario II (e),(f)

ACKNOWLEDGMENTS

This work is supported in part within the DFG excellence initiative research cluster *Cognition for Technical Systems – CoTeSys*, see also `www.cotesys.org`.

REFERENCES

[1] K. Klasing, D. Wollherr, and M. Buss, "A clustering method for online segmentation of 3d laser data," in *Proc. ICRA*, 2008.

[2] R. Hoffman and A. K. Jain, "Segmentation and classification of range images," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 9, no. 5, pp. 608–620, 1987.

[3] J. Huang and C.-H. Menq, "Automatic data segmentation for geometric feature extraction from unorganized 3-d coordinate points," *IEEE Trans. Robot. Automat.*, vol. 17, pp. 268–279, 2001.

[4] G. H. Golub and C. F. V. Loan, *Matrix computations (3rd ed.)*. Baltimore, MD, USA: Johns Hopkins University Press, 1996.

[5] H. Hoppe, T. DeRose, T. Duchamp, J. McDonald, and W. Stuetzle, "Surface reconstruction from unorganized points," *Computer Graphics*, vol. 26, no. 2, pp. 71–78, 1992.

[6] C. Wang, H. Tanahashi, H. Hirayu, Y. Niwa, and K. Yamamoto, "Comparison of local plane fitting methods for range data," in *CVPR (1)*, pp. 663–669, 2001.

[7] K. Kanatani, *Statistical Optimization for Geometric Computation: Theory and Practice*. New York, NY, USA: Elsevier, 1996.

[8] E. Simoncelli, "Least squares optimization." Lecture Notes, http://www.cns.nyu.edu/˜eero/teaching.html, July 2003.

[9] M. Gopi, S. Krishnan, and C. T. Silva, "Surface reconstruction based on lower dimensional localized delaunay triangulation," *Comput. Graph. Forum*, vol. 19, no. 3, 2000.

[10] M. J. Milroy, C. Bradley, and G. W. Vickers, "Segmentation of a wrap-around model using an active contour," *Computer-Aided Design*, vol. 29, no. 4, pp. 299–320, 1997.

[11] M. Yang and E. Lee, "Segmentation of measured point data using a parametric quadric surface approximation," *Computer-Aided Design*, vol. 31, no. 7, pp. 449–457, 1999.

[12] D. OuYang and H.-Y. Feng, "On the normal vector estimation for point cloud data from smooth surfaces," *Computer-Aided Design*, vol. 37, no. 10, pp. 1071–1079, 2005.

[13] M. Vanco, *A Direct Approach for the Segmentation of Unorganized Points and Recognition of Simple Algebraic Surfaces*. PhD thesis, University of Technology Chemnitz, 2003.

[14] H. Gouraud, "Continuous shading of curved surfaces," *IEEE Transactions on Computers*, vol. C-20, pp. 623–629, June 1971.

[15] S. Jin, R. R. Lewis, and D. West, "A comparison of algorithms for vertex normal computation," *The Visual Computer*, vol. 21, no. 1-2, pp. 71–82, 2005.

[16] R. Sagawa, T. Oishi, A. Nakazawa, R. Kurazume, and K. Ikeuchi, "Iterative refinement of range images with anisotropic error distribution," in *Proceedings of the IEEE International Conference on Intelligent Robots and Systems (IROS)*, pp. 79–85, 2002.

[17] G. Lidoris, K. Klasing, A. Bauer, T. Xu, K. Kühnlenz, D. Wollherr, and M. Buss, "The Autonomous City Explorer project: Aims and system overview," in *Proc. IROS*, (San Diego, CA), October 2007.

[18] K. Klasing, D. Wollherr, and M. Buss, "Realtime segmentation of range data using continuous nearest neighbors," in *Proc. ICRA*, 2009.