Towards Identification of Best Practice Algorithms in 3D Perception and Modeling

Sebastian Blumenthal and Erwin Prassler and Jan Fischer and Walter Nowak

***Abstract*— Robots need a representation of their environment to reason about and to interact with it. Different 3D perception and modeling approaches exist to create such a representation, but they are not yet easily comparable. This work tries to identify *best practice* algorithms in the domain of 3D perception and modeling with a focus on environment reconstruction for robotic applications. The goal is to have a collection of refactored algorithms that are easily measurable and comparable. The realization follows a methodology consisting of five steps. After a survey of relevant algorithms and libraries, common representations for the core data-types Cartesian point, Cartesian point cloud and triangle mesh are identified for use in harmonized interfaces. Atomic algorithms are encapsulated into four software components: the Octree component, the Iterative Closest Point component, the k-Nearest Neighbors search component and the Delaunay triangulation component. A sample experiment demonstrates how the component structure can be used to deduce *best practice*.**

I. INTRODUCTION

Autonomous robots are complex systems with different kinds of hardware and software components. A developer, who has to design a robot for a specific task, faces many problems: What is the right choice of sensors, actuators or the robot platform? Which algorithms are the most suitable for an application? Which of them need to be integrated and which need to be reimplemented? Nowadays, robots are often build from scratch because a well established *robot development process* is missing completely. A *robot development process* would significantly help the developer and speed up the development time. One important aspect is that the process should help to get access to *best practice* choices for the application.

In the context of this work an algorithm is characterized as *best practice* if it performs better than other algorithms, for a specific task. Depending on the task for a robotic system, the superior or best algorithm might not be the same. The basis for a comparison to identify *best practice* is deduced by *benchmarks*.

This work is biased towards 3D perception and modeling for mobile manipulation applications. It tries to close the gap between the real world and 3D environment models that are needed by mobile manipulation planning algorithms. Therefore in this paper the scope is limited to 3D environment reconstruction. Object detection techniques are not covered here.

The prevalent representation for 3D processing algorithms are Cartesian point clouds. As mobile manipulation planners typically need a triangle set representation of the environment, the processing stages for 3D perception and modeling aim to create the required triangle mesh of the environment with sensor data as input. This process has several stages [1], [2], [3]: *depth perception*, *filtering*, *registration*, *segmentation*, *mesh generation* and *visualization* (cf. Fig. 1).

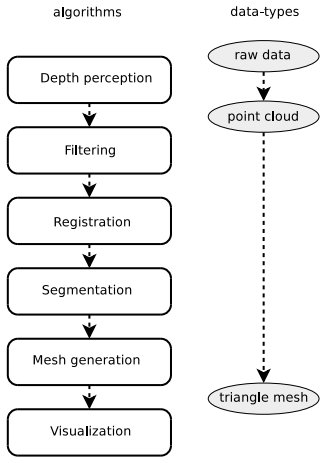

Fig. 1. Overview of 3D perception and modeling processing stages and involved data-types.

- *Depth perception* describes technology to measure distances in a 3D environment.
- A *filter* is an algorithm that is able to process a data stream, in this case point cloud data.
- *Registration*, also sometimes referred as *matching*, is the process of merging data from different viewpoints into one global, consistent coordinate frame.
- *Segmentation* means a spatial partitioning of point clouds into subsets that belong to different geometric objects.
- The goal of the *mesh generation* step is to transform a 3D point cloud into a surface mesh.
- *Visualization*, or *rendering*, is the process of displaying the 3D models. This process can be regarded as optional but it is useful for visually judging success or failure of an algorithm or to debug an application.

There is no strict order of these steps. For instance

The research leading to these results has received funding from the European Community's Seventh Framework Programme (FP7/2007-2013) under grant agreement no. FP7-ICT-231940-BRICS (Best Practice in Robotics).

Sebastian Blumenthal and Walter Nowak are with GPS GmbH, Nobelstr. 12, 70569 Stuttgart, Germany, {blumenthal, nowak}@gps-stuttgart.de

Erwin Prassler is with the University of Applied Sciences Bonn-Rhein-Sieg, Grantham-Allee 20, 53757 Sankt Augustin, Germany, erwin.prassler@h-brs.de

Jan Fischer is with the Fraunhofer IPA, Nobelstr. 12, 70569 Stuttgart, Germany, jan.fischer@ipa.fraunhofer.de

registration could be performed after mesh generation. Some steps can even be skipped completely, depending on the application.

For each processing stage, a variety of different algorithms and implementations exists. These are typically difficult to compare or exchange due to a multitude of reasons, including use of different data structures, incompatible interfaces, dependency on specific software environments and so on. In this paper possible solutions to improve this situation are presented. Building upon a more general methodology for determining best practice, existing algorithms covering the stages depicted in Fig. 1 are analyzed and refactored based on principles from component-based software engineering.

The remainder of this paper is organized as follows. Section II surveys the related work in the fields of robotic benchmarking, algorithms and open source libraries for 3D perception and modeling in the scope of environment reconstruction. Section III presents the approach to identify *best practice* and Section IV applies it to the field of 3D perception and modeling. The paper is closed with a conclusion in Section V.

II. STATE OF THE ART

This Section will start with current efforts in benchmarking of robots and why it is so difficult to compare different robotic systems, followed by algorithms for 3D perception and modeling and a list of publicly available libraries that at least partially implement the presented algorithms.

A. Benchmarking in robotics

Benchmarking in robotics does not have a well established methodology for experiments yet, therefore related work is paying more and more attention to this issue [4]. Benchmarking in robotics can be roughly categorized into a *top-down* and a *bottom-up* perspective [5]. In *top-down* approaches the robot is investigated as a whole system or a "black box". Examples are robot competitions like RoboCup[1], the DARPA Grand challenges and the URBAN challenge[2]. The advantage is the easier evaluation whether a robot can achieve a given task. A more fine-grained benchmarking is achieved in a *bottom-up* fashion. Here each single sub-entity is individually evaluated. Sub-entities can be algorithms, devices, atomic components or complex/composite components. A special problem and open issue is the fact that no common software interfaces are yet available for these entities [6]. Most benchmarking efforts have been concentrating on the evaluation of a robotic system as a whole, rather than on a component level.

The definition of performance metrics for benchmarking is a difficult problem. [6], [7], [8] started to develop performance metrics for navigation applications. Another difficulty arises from the fact that algorithms often exploit certain assumptions that are hidden and not made explicit to the reader, for example *tuning* parameters that are not mentioned. An issue often neglected is the problem of ground truth which is only partially solved (for example in simulation) [9].

[1] `http://www.robocup.org/`
[2] `http://www.darpa.mil/grandchallenge/index.asp`

B. Algorithms for 3D perception and modeling

The focus of the following survey is on the registration and the mesh generation process. Filtering and segmentation methods are also briefly discussed.

Filtering: For 3D perception and modeling, there are three major filtering techniques for point clouds: *noise reduction*, *size reduction* and *normal estimation*. A noise reduction filter tries to cope with noisy measurements from a depth perception device. [10] reduce "salt and pepper noise" by a median filter. [11] use smoothing techniques for point clouds based on estimated normals and a robust hyperplane projection. Among other methods, the *Octree* decomposition is a popular method to downsample point cloud data [12]. Filters for *normal estimation* calculate normals for the points in a point cloud, such that the normals represent the plane normal of an underlying patch of the surface. Point normals are required for various algorithms like registration methods [13], segmentation [3] and mesh generation algorithms [14], [15] [16]. Most approaches calculate the k-Nearest Neighborhood of a query point and fit some geometric patch to approximate the local surface [17], [18], [19], [3]

Registration: Global strategies for registration involve genetic algorithms like [20], or evolutionary computation approaches [21]. As these registration methods are computationally expensive they are uncommon for applications in the robotics domain, where (near) real-time capabilities are important. A recent development *HSM3D* [22] uses the Hough transform for global registration. As comprehensive experiments have not been performed yet, it remains unclear how competitive this solution is compared to other existing approaches. Therefore, most efforts have been spent on *local* registration techniques.

The most prominent *local* registration algorithm is the *Iterative Closest Point (ICP)* by [23] and [24]. Since then various improvements have been made that mostly address the *correspondence problem* of finding corresponding point pairs in two point clouds and the *rigid transformation estimation problem*. Approaches to increase *robustness* for the correspondence problem typically enhance the spatial information of a point by additional information like intensity [25], color [26] or point normals [13]. The calculation of distinctive features in point clouds to deliver an approximate initial alignment is applied by [27] and [28]. Recently the *Point Feature Histograms (PFH)* [29] has been developed, with an improved version called *Fast Point Feature Histograms (FPFH)* [30]. [31] improves robustness for the correspondence problem, by computing a rough initial transformation guess with an *Extended Gaussian Image (EGI)* and a rotational Fourier function.

Various improvements that address *computational complexity* for the ICP have been proposed by [32]. A widely used solution to reduce complexity in k-Nearest Neighbors problems is the usage of *k-d trees* [12]. Approximated versions

of *k-d trees* have been developed by [33], [34] and [35]. The concept of *hierarchical k-means* is used by [36] in their vocabulary tree approach. [37] present a *Morton* ordering based search which can be parallelized well on multi-core and multi-processor hardware. Along with the improvements for computational complexity, the computation of the *rigid transformation estimation* is an important step for the ICP algorithm: As depicted by [12], four *closed-form* variants exist. Beside the closed form solutions, *approximated* estimation approaches exist as well, such as the *helical motion* based method proposed by [38].

A recent development for a local non-ICP based method is the probabilistic *Normal Distributions Transform (NDT)* [39]. A comparison of ICP and NDT [40] concludes that no algorithm clearly outperforms the other one.

Segmentation: Common segmentation criteria are *normals* of the points in a point cloud [1], as demonstrated in [3] in a kitchen-like environment. Model fitting is performed by [2], as they try to detect buildings in a reconstruction process for large environments. [41] use a sampling technique to find basic shapes like planes, cylinders, cones and tori in a point cloud. The segmentation algorithm of [42] first performs a decomposition of space into regular cubes, then planes are detected with the help of the RANSAC principle. Finally a refinement step with a region growing strategy is applied. Further segmentation methods are based on edges [43], curvature estimates [44] or smoothness constraints [45].

Mesh generation: Some algorithms need *estimated normals* in the point cloud, for example the *Poisson reconstruction* method by [14]. Many approaches for surface mesh generation are based on the *Delaunay triangulation*.

[46] contributes fundamentals for mesh generation algorithms by analyzing the properties for common geometric representations, including partitions with the Delaunay criterion. The author introduces the strategy to label tetrahedrons into *inside* and *outside* simplices. This categorization allows to deduce the surface of an object. The algorithms from the *CRUST* and *COCONE* family [47], [48], [49] rely on Delaunay triangulation, as well as the α-shapes algorithm by [50]. Other approaches that are independent of Delaunay triangulations have been proposed. [51] presents the growing based ball-pivoting algorithm. The *Frontal Algorithm* by [52] calculates special points and generates the mesh according to a rule set.

C. Open source libraries for 3D perception and modeling

This section surveys existing open source libraries that are related to the 3D perception and modeling domain. All libraries are written in C/C++.

Functionality for algorithms in the *Filtering*, *Registration* and *Segmentation* steps can be found in the 6DSLAM[3], the MRPT[4], the IVT[5], the Fair[6], packages from the ROS[7], in particular the PCL[8], the VTK[9], or the ITK[10] library. Libraries that have a focus on mesh generation are Meshlab[11], CGAL[12], Gmsh[13], Qhull[14], or OSG[15]. Most libraries offer capabilities for *Visualization*. Algorithms to solve the k-Nearest Neighbor search can be found in the ANN[16] the FLANN[17] or the STANN[18] library.

[3] http://slam6d.sourceforge.net/
[4] http://www.mrpt.org/
[5] http://ivt.sourceforge.net/
[6] http://sourceforge.net/projects/openvolksbot/
[7] http://www.ros.org/wiki/

III. CONCEPT

The process of identifying *best practice* algorithms can be categorized into different phases [53]. There are five steps: *Exploration*, *Harmonization*, *Refactoring*, *Integration* and *Evaluation*. The goal of this process is to have a framework of software components that allows to replace atomic elements to easily create a set of benchmarks that enables to compare and judge the algorithms.

A. Exploration

In the *Exploration* phase, knowledge about the algorithmic domain has to be acquired. A state-of-the-art literature **survey** as well as a software library survey has to be conducted.

B. Harmonization

The *Harmonization* phase tries to find harmonized data-types and interfaces for atomic elements. It can be further subdivided into the three following steps:

1) Identify **common data-types**: Identify the commonly used data-types, and analyze the commonalities of representations in existing libraries.
2) Identify **"black-boxes"**: This is the step of de-modularizing algorithms into atomic elements. The most common modules shall be found and represented as stand alone software components.
3) Identify **common interfaces**: The atomic elements identified in the previous step need to communicate to other atomic elements. To get access to a component, one has to use a set of interfaces.

All algorithms and data-types are designed with a potential user in mind, for example a mobile manipulation planning library that needs a 3D model of the environment to create paths or trajectories for the hardware platform. To foster openness, flexibility and reusability of the refactored algorithms, a *separation of concerns* is aspired with respect to the principles of *coordination*, *configuration*, *computation*, and *communication* [54], [55].

The process of modularization, decoupling and abstracting to more general interfaces certainly involves trade-offs between usability or generality and highly coupled or optimized

[8] http://www.ros.org/wiki/pcl
[9] http://ivt.sourceforge.net/
[10] http://www.itk.org/
[11] http://meshlab.sourceforge.net/
[12] http://www.cgal.org/
[13] http://www.geuz.org/gmsh/
[14] http://www.qhull.org/
[15] http://www.openscenegraph.org/projects/osg
[16] http://www.cs.umd.edu/~mount/ANN/
[17] http://people.cs.ubc.ca/~mariusm/index.php/FLANN/FLANN
[18] http://sites.google.com/a/compgeom.com/stann/

algorithms. But as the intention is to find *best practice* algorithms the decoupling has a higher priority over highly optimized algorithms. However, in a robot development process an optimization step might follow after the best practice algorithms have been determined for a specific application.

C. Refactoring

The *Refactoring* step can be characterized by **filling black boxes with life**. That means the desired behavior for an algorithm must be defined by implementing it. Code should be reused from existing libraries.

D. Integration

In preparation of benchmarking the implemented algorithms, they need to be **integrated into a test-bed**. This test-bed might be a real robot, a simulation framework or a software framework that can prepare data sets to be used as input for a benchmark.

E. Evaluation

The final step is the *Evaluation* or **benchmarking** of the refactored and harmonized algorithms. Each component implementation serves as a point of variation for the benchmarks. These benchmarks can be performed on a real robot, within a simulation framework or with recorded or artificial data sets.

IV. RESULTS

The methodology proposed in Section III has been applied to the 3D Perception and Modeling domain with a focus on environment reconstruction. In the following results for each phase are discussed. A summary of the results gathered in the *Exploration* phase has been already presented in Section II-B and Section II-C.

A. Harmonization of data-types

The three predominant data-types are Cartesian points, Cartesian point clouds, and triangle meshes. An in-depth analysis of existing open source libraries (cf. Section II-C) revealed the following commonalities.

Cartesian point: Most libraries use a representation with simple x, y, and z variables as coordinates. Basic matrix operation functionality is commonly used among the investigated libraries, as the Cartesian point also serves as vector of dimensionality three. Thus a harmonized point should support simple vector algebra as addition, subtraction and multiplication with a scalar. Streaming support is not a commonly available feature. Nevertheless it can be convenient to have, as it allows to easily dump the output for debugging or logging activities, and makes conversions to other point representations simpler. A harmonized data-type must preserve flexibility to future extension. Possible extensions of a point could be color information, estimated normals, weights, probabilities, feature vector descriptors for distinctive properties or flags if a point is valid or not. It is possible to implement such extensions via class inheritance, but with growing requirements the inheritance hierarchy would have a combinatorial explosion of possible combinations. To overcome this problem we propose to use the *decorator* software pattern [56].

Cartesian point cloud: The harmonized Cartesian point cloud will essentially consist of a vector of points as this is the most common way, among the investigated libraries, to represent it. Supported operations should include streaming capabilities similar to the Cartesian point requirements, homogeneous transformations and simple manipulation operations such as adding new points.

Triangle meshes: The analysis of the libraries revealed two common ways to represent triangle meshes: The first variant uses an *implicit* representation with a vector of vertices and a vector of indices. Three consecutive indices, referencing the vertices vector, form a triangle. The advantage is the memory efficient storage, as vertices do not need to be inserted multiple times into the mesh if one vertex belongs to several triangles. The disadvantage is that both vectors have to be carefully maintained while adding or removing triangles. Furthermore it is less flexible for future extension, because a triangle might have additional information like color, a validity flag, a probability or a texture reference.

The *explicit* version uses a vector of triangles, where each triangle consists of three vertices. This representation is more flexible in the sense that a basic triangle class can be extended or *decorated* in future developments, similar to the Cartesian point. On the other hand it might be less memory efficient. A harmonized triangle mesh should support both representations. This can be achieved by an abstract class allowing access with a common interface, so a potential user can choose which implementation fits most to an application or a problem. However that does not necessarily mean both representations are always fully exchangeable. A common functionality is the streaming support which allows to easily read and write data to standard output, files or other implementations of triangle meshes. As an additional feature a harmonized triangle mesh should support homogeneous coordinates transformations similar to the Cartesian point clouds.

B. Harmonization and refactoring of common algorithms

For 3D perception and modeling in an environment reconstruction context, at least the following atomic components can be identified: The Octree component, the Iterative Closest Point component, the k-Nearest Neighbors search component and the Delaunay triangulation component (cf. Fig. 2).

This list does not claim to be complete, but these are the most common elements as deduced from the *Exploration* phase.

1) The Octree component: The Octree algorithm is a popular method to reduce point clouds. It is used for voxel representations or for surface mesh generation approaches. The Octree can be regarded as a common atomic element for 3D perception and modeling applications.

The Octree component has two different roles: as *reduction filter*, and as structured *partition* of the space into cubes. To account for both roles, two separate provided interfaces

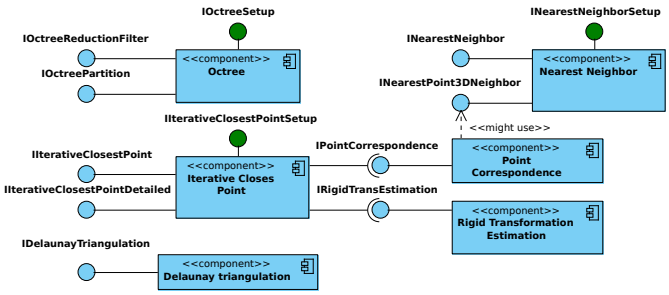

Fig. 2. Overview of atomic software components for 3D perception and modeling. Configuration interfaces are marked with green color.

are offered for each functionality. The first functional interface provides capabilities to reduce point clouds. The other functional interface provides functionality to partition a point cloud into a set of smaller point clouds. A third provided interface is offered in order to decouple the configuration from the functional interfaces. For the Octree only the parameter *voxel size* needs to be configured. The Octree component does not depend on other modules and thus has no required interfaces.

2) The Iterative Closest Point component: A common method to register multiple point clouds into one common coordinate frame is the Iterative Closest Point (ICP) algorithm.

The ICP algorithm has two major sub-elements: a step that establishes *point correspondences* and a step that *estimates rigid transformations*. Both steps can be solved by various approaches. To be able to exchange atomic parts, both steps are encapsulated as subcomponents. These subcomponents are addressed by two required interfaces.

The ICP component offers the matching functionality in two provided interfaces. The first, minimal interface needs to accept two point clouds: *model* and *data*, and calculates the translation and rotation that needs to be applied to the data so that it is aligned to the model. The rotation and translation can be summarized in a homogeneous transformation matrix. The second interface reveals more internal details to the user of this component. This interface has getter and setter methods for the *data* and the *model* point cloud and a method that invokes only one iteration of the ICP and returns the error. It allows to define new termination criteria or it enables a system scheduler to invoke this component iteratively according to a scheduling policy. This second interface is *stateful* that means the results rely on previous states. This implies a *contract* on the interface: to correctly use this interface first set *data* and *model*, then invoke the next iteration, as often as desired.

The first provided interface is *stateless*, that means the behavior is always the same, while the behavior of the second interface depends on the history of preceding events [57]. Both interface types are clearly separated. There is an additional interface that fulfills the *configuration* role. It allows to manipulate the convergence threshold, the maximum number of iterations and to configure the required subcomponents.

The Point Correspondence subcomponent: The component for establishing the point-to-point correspondences needs two point clouds as input data and returns a list of corresponding points. The provided interface has just one method that allows to calculate point-to-point correspondences. The component does not need to be configured, as it has no parameters, nor does it need a required interface. As internal realization the k-Nearest Neighbor search component could be used. This is left to the implementation.

The Rigid Transformation Estimation subcomponent: The second required interface for the ICP algorithm is provided by the Rigid Transformation Estimation component. It provides one interface that has a list of point correspondences as input and a homogeneous transformation as output parameter, in order to store the resulting transformation. The component has no required interfaces and no parameters that need to be configured.

3) The k-Nearest Neighbor search component: Nearest Neighbor search operations are for example used by *registration* or *normal estimation filtering* algorithms, thus it can be seen as a common atomic element for 3D perception and modeling.

The component will account for two different roles. The first one is a generic use case that might want to apply the component in a completely different context than robotics, and the second use case applies Cartesian points with dimension three. The more general interface allows to set a multidimensional vector *data*. The query consists of a data vector, as well as the parameter k, and it will return a vector of indices to the k nearest neighbors. The specific interface for 3D perception and modeling domain uses data that is defined by a point cloud instead of multidimensional data vector. The query consists of Cartesian point, rather than data vector of variable length.

Both interfaces are *stateful*, as most implementations first create an appropriate search structure, like for example a search tree. Search queries are then accelerated by using that structure. Whenever in the k-Nearest Neighbor interfaces the *data* is set, these search structures are created. The configuration interface allows to get the *dimension* and to set and get an optional parameter *maximalDistance* for the maximal allowed distance to regard an element as neighbor.

4) The Delaunay Triangulation component: The Delaunay Triangulation algorithm is commonly used as atomic element among the mesh generation algorithms. For example the algorithms of the *CRUST* and *COCONE* family or the α-shapes method, depend on this triangulation (cf. Section II-B).

The primary role of the Delaunay Triangulation component is to create a triangulation from a point cloud. The result of a 3D triangulation is a set of tetrahedrons. All triangulations obey the *Delaunay property*, and do not need any further parameters. That is why there is no configuration interface for this component.

C. Framework integration

The implementations of the proposed data-types and components have been integrated into a library called *BRICS_3D*. It supports loading datasets like depth images or point cloud data and realizes visualization capabilities with the OSG library. The BRICS_3D library will be published as an open source library[19], such that the components can be reused in robotic applications as well as new custom defined benchmarks can be performed.

D. Evaluation

This section presents a sample benchmark of the ICP component. It has been selected because it is a popular core component for environment reconstruction applications. This benchmark serves as a representative sample to illustrate the concept of benchmarking on a component-based level. The used performance metric is the computational time which can bee seen as the *cost* of the computation. This metric is only one among others like *utility* or *reliability*, but further experiments are beyond the scope of this paper.

Four implementations of the point correspondence components are used. All are based on the k-Nearest Neighbors component. The first uses a *k-d tree* [12], the second uses the *ANN* [33], the third the *FLANN* [35] and the last the *STANN* [37] library. The rigid transformation estimation implementation comprise an *SVD* based, a Quaternion (*QUAT*) based, a helical motion estimation (*HELIX*) and a linear approximation (*APX*) based solution [12].

The benchmarks have been performed with the *Stanford Bunny*[20] data sets *bun000.ply* and *bun045.ply*. The parameters of the ICP are set as follows: convergence threshold which defines the minimum convergence slope is 0.00001, the maximal point-to-point distance is 50 and the maximal amount of iterations is set to 100. The benchmarks are performed on an off-the-shelf laptop with 2GHz dual core processor and 3GB memory. The operating system is an Ubuntu 9.10 with Kernel version 2.6.31-20. The source code is compiled with the gcc compiler version 4.4.1 with debug flags enabled. Every matching process was repeated 10 times. All algorithm combinations converge after 20 to 30 iterations and result in a remaining RMS error of approximately 0.002. The mean values for the computational time are depicted in Fig. 3.

Independent of the used rigid transformation estimation algorithms, the k-d tree and the ANN point-to-point correspondence implementation outperforms the other algorithms. The STANN implementation is by far the slowest approach. As the used test-bed offers only limited parallel computing hardware, the STANN library was not able to demonstrate its full potential.

The selection of the rigid transformation estimation has only a minor influence on the timing behavior, whereas the choice of the point correspondences algorithm has a major impact.

[19] `http://www.best-of-robotics.org/`
[20] `http://graphics.stanford.edu/data/3Dscanrep/`

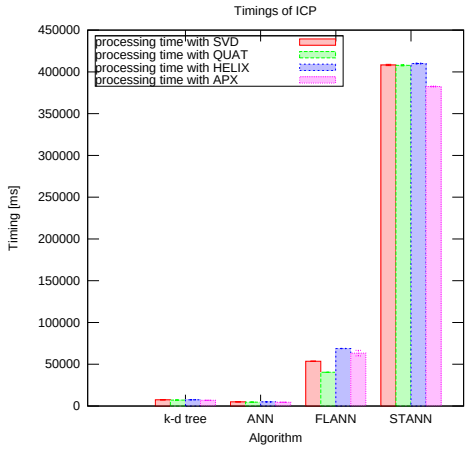

Fig. 3. Benchmark results for the Iterative Closest Point algorithm

The benchmark results are only valid for the used test-bed and the used test data. Furthermore, the effects of wrapping and adopting the implementations are neglected, still it shows that it is possible to get access to best practice algorithms by decomposition into atomic components and benchmarking those.

V. CONCLUSION

This paper has presented a methodology to identify *best practice* algorithms in robotics. This methodology has been applied to the domain of 3D perceptions and modeling with a focus on environment reconstruction. One important aspect was the application of software engineering principles, in particular *software components*, to refactor existing algorithms into common atomic and reusable elements. These elements can be benchmarked to deduce the *best practice* algorithms for a specific task with implementations provided by the BRICS_3D open source library. An open issue that remains for future work is the implementation of more components for 3D perception and modeling. This work has covered only a subset of all available algorithms. Currently the *segmentation* stage as well as object detection techniques are not yet addressed. Normal estimation, noise reduction, alternative registration methods like NDT or HSM3D and mesh generation with the α-shapes algorithm are subject for future implementation.

The ability to make *best practice* choices of algorithms for a specific robotic task, in early stages of a robot development process, in combination with reuse of software components, reduces the efforts to develop software for robotics applications.

VI. ACKNOWLEDGMENTS

The authors would like to thank Davide Brugali, Luca Gherardi and Herman Bruyninckx for iteratively reviewing the source code and providing design guidelines. Thanks to Alexey Zakharov for his valuable comments.

REFERENCES

[1] G. Hirzinger, T. Bodenmüller, H. Hirschmüller, R. Liu, W. Sepp, M. Suppa, T. Abmayr, and B. Strackenbrock, "Photo-realistic 3d modelling-from robotics perception towards cultural heritage," *Recording, Modeling and Visualization of Cultural Heritage*, p. 361, 2006.

[2] S. You, J. Hu, U. Neumann, and P. Fox, "Urban site modeling from lidar," *Lecture Notes in Computer Science*, pp. 579–588, 2003.

[3] R. Rusu, Z. Marton, N. Blodow, M. Dolha, and M. Beetz, "Towards 3d point cloud based object maps for household environments," *Robotics and Autonomous Systems*, vol. 56, no. 11, pp. 927–941, 2008.

[4] F. P. Bonsignorio, J. Hallam, and A. P. del Pobil, "Good experimental methodologies in robotics: State of the art and perspectives," in *Proc. of the Workshop on Performance Evaluation and Benchmarking for Intelligent Robots and Systems, IEEE/RSJ Int. Conf. on Intelligent Robots and Systems*, 2007.

[5] I. Ranó and J. Minguez, "Steps towards the automatic evaluation of robot obstacle avoidance algorithms," in *Proc. of Workshop of Benchmarking in Robotics, in the IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)*, 2006.

[6] D. Calisi, L. Iocchi, and D. Nardi, "A unified benchmark framework for autonomous Mobile robots and Vehicles Motion Algorithms (MoVeMA benchmarks)," *University of Rome*, 2008.

[7] F. Amigoni, M. Reggiani, and V. Schiaffonati, "An insightful comparison between experiments in mobile robotics and in science," *Autonomous Robots*, pp. 1–13, 2009.

[8] N. Munoz, J. Valencia, and N. Londoño, "Evaluation of navigation of an autonomous mobile robot," in *Proc. of Int. Workshop on Performance Metrics for Intelligent Systems Workshop (PerMIS)*, 2007, pp. 15–21.

[9] F. Bonsignorio, J. Hallam, and A. del Pobil, "Defining the Requisites of a Replicable Robotics Experiment," in *Workshop on good experimantal methodology in robotics, RSS*, 2009.

[10] A. Nüchter, K. Lingemann, J. Hertzberg, and H. Surmann, "6d slam-3d mapping outdoor environments," *Journal of Field Robotics*, vol. 24, 2007.

[11] B. Mederos, L. Velho, and L. De Figueiredo, "Smooth surface reconstruction from noisy clouds," *Journal of the Brazilian Computing Society*, vol. 1, 2004.

[12] A. Nüchter, *3D Robotic Mapping The Simultaneous Localization and Mapping Problem with Six Degrees of Freedom*. Springer, 2008.

[13] M. Alexa, J. Behr, D. Cohen-Or, S. Fleishman, D. Levin, and C. Silva, "Computing and Rendering Point set surfaces," in *Proceedings of the Conf. on Visualization'01*. IEEE Computer Society Washington, DC, USA, 2003, pp. 21–28.

[14] M. Kazhdan, M. Bolitho, and H. Hoppe, "Poisson surface reconstruction," in *Proceedings of the fourth Eurographics symposium on Geometry processing*. Eurographics Association, 2006, p. 70.

[15] J. Carr, R. Beatson, B. McCallum, W. Fright, T. McLennan, and T. Mitchell, "Smooth surface reconstruction from noisy range data," *ACM GRAPHITE*, vol. 3, pp. 119–126, 2003.

[16] J. Carr, R. Beatson, J. Cherrie, T. Mitchell, W. Fright, B. McCallum, and T. Evans, "Reconstruction and representation of 3d objects with radial basis functions," in *Proceedings of the 28th annual Conf. on Computer graphics and interactive techniques*. ACM New York, NY, USA, 2001, pp. 67–76.

[17] M. Pauly, M. Gross, and L. Kobbelt, "Efficient simplification of point-sampled surfaces," in *IEEE visualization*, 2002, pp. 163–170.

[18] H. Hoppe, T. DeRose, T. Duchamp, J. McDonald, and W. Stuetzle, "Surface reconstruction from unorganized points," *Computer Graphics-New York-Association for Computing Machinery-*, vol. 26, pp. 71–71, 1992.

[19] N. Mitra and A. Nguyen, "Estimating surface normals in noisy point cloud data," in *Proceedings of the nineteenth annual symposium on Computational geometry*. ACM New York, USA, 2003, pp. 322–328.

[20] L. Silva, O. Bellon, and K. Boyer, "Precision range image registration using a robust surface interpenetration measure and enhanced genetic algorithms," *IEEE transactions on pattern analysis and machine intelligence*, pp. 762–776, 2005.

[21] O. Cordón, S. Damas, and J. Santamaría, "A fast and accurate approach for 3D image registration using the scatter search evolutionary algorithm," *Pattern Recognition Letters*, vol. 27, no. 11, pp. 1191–1200, 2006.

[22] S. Carpin and A. Censi, "An experimental assessment of the HSM3D algorithm for sparse and colored data," in *Int. Conf. on Intelligent Robots and Systems (IROS)*, 2009.

[23] P. Besl and H. McKay, "A method for registration of 3-d shapes," *IEEE Transactions on pattern analysis and machine intelligence*, vol. 14, no. 2, pp. 239–256, 1992.

[24] Z. Zhang, "Iterative point matching for registration of free-form curves," *Int. Journal of Computer Vision*, vol. 13, no. 2, pp. 119 – 152, 1992.

[25] D. Akca, "Matching of 3D surfaces and their intensities," *ISPRS Journal of Photogrammetry and Remote Sensing*, vol. 62, no. 2, pp. 112–121, 2007.

[26] A. Johnson and S. Kang, "Registration and integration of textured 3D data," *Image and Vision Computing*, vol. 17, no. 2, pp. 135–147, 1999.

[27] N. Gelfand, N. Mitra, L. Guibas, and H. Pottmann, "Robust global registration," in *Symposium on Geometry Processing*, vol. 2, no. 3, 2005, p. 5.

[28] G. Sharp, S. Lee, and D. Wehe, "ICP registration using invariant features," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, pp. 90–102, 2002.

[29] R. Rusu, N. Blodow, Z. Marton, and M. Beetz, "Aligning point cloud views using persistent feature histograms," in *Proceedings of the 21st IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS), Nice, France*, 2008.

[30] R. Rusu, N. Blodow, and M. Beetz, "Fast Point Feature Histograms (FPFH) for 3D Registration," in *Proceedings of the IEEE Int. Conf. on Robotics and Automation (ICRA), Kobe, Japan*, 2009.

[31] A. Makadia, A. Patterson, and K. Daniilidis, "Fully automatic registration of 3D point clouds," in *CVPR*, vol. 6, 2006, pp. 1297–1304.

[32] S. Rusinkiewicz and M. Levoy, "Efficient variants of the icp algorithm," in *Proc. 3DIM*, 2001, pp. 145–152.

[33] S. Arya, D. Mount, N. Netanyahu, R. Silverman, and A. Wu, "An optimal algorithm for approximate nearest neighbor searching fixed dimensions," *Journal of the ACM (JACM)*, vol. 45, no. 6, pp. 891–923, 1998.

[34] C. Silpa-Anan and R. Hartley, "Optimised KD-trees for fast image descriptor matching," in *Proc. CVPR*, 2008, pp. 1–8.

[35] M. Muja and D. Lowe, "Fast approximate nearest neighbors with automatic algorithm configuration," in *International Conference on Computer Vision Theory and Applications (VISAPP'09)*, 2009.

[36] D. Nister and H. Stewenius, "Scalable recognition with a vocabulary tree," in *Proc. CVPR*, vol. 5, 2006.

[37] M. Connor and P. Kumar, "Fast construction of k-nearest neighbor graphs for point clouds," in *IEEE Transactions on Visualization and Computer Graphics*, September 2009.

[38] H. Pottmann, S. Leopoldseder, and M. Hofer, "Registration without ICP," *Computer Vision and Image Understanding*, vol. 95, no. 1, pp. 54–71, 2004.

[39] M. Magnusson, A. Lilienthal, and T. Duckett, "Scan registration for autonomous mining vehicles using 3D-NDT," *Journal of Field Robotics*, vol. 24, no. 10, pp. 803–827, 2007.

[40] M. Magnusson, A. Nüchter, C. Lörken, A. Lilienthal, and J. Hertzberg, "Evaluation of 3D Registration Reliability and Speed - A Comparison of ICP and NDT," in *Proceedings of the IEEE Int. Conf. on Robotics and Automation (ICRA), Kobe, Japan*, 2009, pp. 3907–3912.

[41] R. Schnabel, R. Wahl, and R. Klein, "Efficient RANSAC for point-cloud shape detection," in *Computer Graphics Forum*, vol. 26, no. 2, 2007, pp. 214–226.

[42] J. Weingarten, G. Gruener, and R. Siegwart, "A fast and robust 3D feature extraction algorithm for structured environment reconstruction," in *Int. Conf. on Advanced Robotics, Coimbra, Portugal*, 2003.

[43] A. Sappa and M. Devy, "Fast range image segmentation by an edge detection strategy," in *Proceedings of Third International Conference on 3-D Digital Imaging and Modeling*, vol. 3, 2001.

[44] P. Besl and R. Jain, "Segmentation through variable-order surface fitting," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 10, no. 2, pp. 167–192, 1988.

[45] T. Rabbani, F. van den Heuvel, and G. Vosselmann, "Segmentation of point clouds using smoothness constraint," *International Archives of Photogrammetry, Remote Sensing and Spatial Information Sciences*, vol. 36, no. 5, pp. 248–253, 2006.

[46] J. Boissonnat, "Geometric structures for three-dimensional shape representation," *ACM Transactions on Graphics (TOG)*, vol. 3, no. 4, pp. 266–286, 1984.

[47] N. Amenta, S. Choi, and R. Kolluri, "The power crust, unions of balls, and the medial axis transform," *Computational Geometry: Theory and Applications*, vol. 19, no. 2-3, pp. 127–153, 2001.

[48] T. Dey and S. Goswami, "Tight cocone: a water-tight surface reconstructor," *Journal of Computing and Information Science in Engineering*, vol. 3, p. 302, 2003.

[49] R. Kolluri, J. Shewchuk, and J. O'Brien, "Spectral surface reconstruction from noisy point clouds," in *Proceedings of the 2004 Eurographics/ACM SIGGRAPH symposium on Geometry processing*. ACM New York, USA, 2004, pp. 11–21.

[50] H. Edelsbrunner and E. Mücke, "Three-dimensional alpha shapes," in *Proceedings of the 1992 workshop on Volume visualization*. ACM, 1992, p. 82.

[51] F. Bernardini, J. Mittleman, H. Rushmeier, C. Silva, G. Taubin, *et al.*, "The ball-pivoting algorithm for surface reconstruction," *IEEE Transactions on Visualization and Computer Graphics*, vol. 5, no. 4, pp. 349–359, 1999.

[52] J. Schöberl, "NETGEN An advancing front 2D/3D-mesh generator based on abstract rules," *Computing and Visualization in Science*, vol. 1, no. 1, pp. 41–52, 1997.

[53] E. Milke, S. Christen, E. Prassler, and W. Nowak, "Towards Harmonization and Refactoring of Mobile Manipulation Algorithms," in *Proc. Int. Conf. on Advanced Robotics (ICAR)*, June 2009.

[54] L. Andrade, J. Fiadeiro, J. Gouveia, and G. Koutsoukos, "Separating computation, coordination and configuration," *Software Focus*, vol. 14, no. 5, pp. 353–369, 2002.

[55] M. Radestock and S. Eisenbach, "Coordinating components in middleware systems," *Software Focus*, vol. 15, no. 13, pp. 1205–1231, 2003.

[56] G. Erich, H. Richard, J. Ralph, and V. John, *Design patterns: elements of reusable object-oriented software*. Addison Wesley Publishing Company, 1995.

[57] D. Brugali and P. Scandurra, "Component-based Robotic Engineering Part I: Reusable building blocks," *IEEE Robotics and Automation Magazine*, December 2009.