BOR3D: A Use-Case-Oriented Software Framework for 3-D Object Recognition

Martin Bertsche, Tobias Fromm, and Wolfgang Ertel
Institute of Artificial Intelligence
Ravensburg-Weingarten University of Applied Sciences
88250 Weingarten, Germany
{martin.bertsche, fromm, ertel}@hs-weingarten.de

***Abstract*—3-D object recognition has become a major research topic. New low-cost sensors hit the market making 3-D vision affordable for anyone. On the software side, promising open-source tools and libraries have prospered recently. The most important one, regarding 3-D data processing, undoubtedly is the Point Cloud Library (PCL). From an integrator's point of view, the main benchmarks applied to such a library are the amount of use cases that can be implemented and the effort that is entailed. We have noticed that the scientific community pays little attention to these needs. We hope to change this situation by proposing a framework that is primarily inspired by the PCL. Clearly separating the roles of its users, it leaves integration to the integrators and algorithms to the specialists. This way we hope to provide a means for development teams to participate in the recent advances even if they do not have a special focus on machine vision.**

I. Introduction

3-D imaging has become easy to use and affordable in the last few years. This is due to the development of 3-D sensors for applications in the gaming industry. As a result further advancements within the field of 3-D object recognition are flourishing. Research is driven forward by a large community of scientists and software developers.

However, for sophisticated applications, appropriate software development tools are required. Therefore, a software framework is needed that permits the integration of the most recent developments in a reusable and robust fashion. One cannot afford to lose sound solutions due to minor software issues.

Within the 3-D data processing field the *Point Cloud Library (PCL)* [1] represents an effort to achieve these goals. It is an open-source collection of libraries addressing the most prominent areas in the field. It is well on the way in becoming a virtual standard in point cloud processing by minimizing incompatibilities and conversion issues.

The PCL provides an excellent basis for further research and development. However, certain deficiencies remain considering user-friendliness and the reusability of high-level solutions. Presently, inexperienced users looking for quick solutions to their object recognition problem are faced with difficulties. In-depth knowledge is required to apply the provided algorithms.

For 2-D object recognition the MOPED [2] framework is rather specialized in its field of application. MOPED fulfills the requirements of user-friendliness by not requiring detailed knowledge of the underlying algorithms.

Knowing the versatility of the PCL as well as the usability of MOPED, a new framework was designed for 3-D object recognition uniting ease of use and flexibility. Another layer of abstraction was added which accommodates a high-level, user-friendly interface not only for the PCL, but for other libraries as well. This new framework is named *BOR3D* (pronounced: "bored"), an acronym for "Boilerplate Object Recognition 3-D".

In this paper, we will apply the term *use case* to a certain goal within the scope of 3-D object recognition. This for example could be the recognition of a number of pre-trained, non-occluded objects placed on a table. In Section IV this exact use case is implemented and described in detail as a showcase example.

The remaining paper is structured as follows: First, we refer to previous work and projects with a similar methodology, but aiming on different applications. Section III then reveals the big picture as well as details of our approach. In Section IV we will apply this to a concrete use case, while finally, in Section V, we will give future prospects and possibilities of BOR3D.

II. Related Work

As indicated in the introduction, this work is primarily inspired by the Point Cloud Library. It serves as a basis providing algorithms and basic data structures. The PCL's goal is to offer state-of-the-art 3-D data processing algorithms. In the past collaboration between developers has been difficult due to software incompatibilities. Researchers are now able to publish their findings in software using the PCL as a common denominator. It delivers a fair amount of standardization by introducing useful interfaces. A primary development philosophy that the PCL developers are following is "write once, parametrize everywhere"[1] It means that processing procedures are developed only once for a given basic task. Adapting the algorithm to specific environment should only be a matter of parametrization. BOR3D is fully adopting this principle.

While the PCL is well endowed with the latest algorithms, there is little guidance on how to combine and integrate them. The main reason for this of course lies within the PCL's

[1] Source: http://www.pointclouds.org/assets/rss2011/00_introduction.pdf, page: 42

978-1-4673-0856-4/12/$31.00 ©2012 IEEE

definition. Being a library instead of a framework it does not impose any limitations on the way applications are designed.

MOPED is an open-source project, described by the authors as a *real-time object recognition and pose estimation system*. It can be considered a framework as it provides abstraction from the algorithmic level. Thus, it pursues goals very similar to ours with the difference in focus on 2-D object recognition.

Many papers use the term ”framework” for a general approach. In this paper we denote that ”framework” always refers to a software framework.

III. Approach

We start off with a top-down approach: The general task performed when recognizing objects is to match input data to model data. The simple perspective entails two separate tasks: One is *training*, which means to generate model data and to store it in a database. The other is to match the input data acquired by a sensor with the trained data. We call this *recognition*.

At this point all the inherent challenges are neglected. We introduce *methods* as a place holder for the actual implementation. Our framework needs to know a method to perform training and recognition. The easiest way to picture the semantics is the *Strategy* [3] design pattern. It is assumed that for an object recognition problem there exist multiple solutions. Based on a-priori knowledge the user chooses the method which best suits the given problem.

A. User-Level Abstraction

The two pieces of self-explanatory C++ code below are to illustrate our vision of a user-level interface provided by our framework. It is designed to fulfil the following requirements: (1) Using BOR3D only requires information about the given use case such as the data which is generated by the sensors. (2) Embedding object recognition in existing software must have a small code footprint. (3) Providing parameters for the processes needs to be done in a standardized fashion.

Example: Training

```
Training<MethodXY> training;
training.configure("training.cfg");
training.setObjectName("Object1");
training.newModel();
```

Example: Recognition

```
Recognition<MethodXY> recognition;
recognition.configure("recognition.cfg");
Results results = recognition.recognize();
```

We are aiming at a steep learning curve for beginners. Choosing the correct method for training and recognition can be challenging when faced with a wide range of them. Therefore, the names of the methods must clearly indicate their purposes.

The methods defined for the two processes need to be compatible since we intend to use the data generated during training for the recognition process. A simple solution is to define a single method for model creation and recognition. However we have seen that the training data of a given method can often be used for multiple recognition approaches. Ignoring these cases would impair the flexibility of our framework. Therefore, a notion of compatibility is needed to find valid combinations. Later on in this section we will see that matching method pairs are easily discovered by comparing certain attributes.

Theoretically, given a selection of use cases and corresponding methods, we can now apply this framework. In order to implement solutions for actual applications we need to take into account that a user's task will always be different from the one a method was designed for. A method designer has to make provisions by exposing a set of parameters for fine-tuning purposes. By doing so we are in violation of our predominant principle, to hide from the user as much of the implementation's details as possible.

However, up to now all object recognition algorithms rely on assumptions regarding the data acquired. These assumptions are encoded in the algorithms themselves as well as in the parameters used for each processing step. The only way to shape the parametrization in a user friendly way is to provide adequate documentation. This documentation has to follow the framework's level of abstraction by describing the parameters' influence on the processing result, or its dependence on input data characteristics. Only then are users able to deduce the appropriate settings without knowing the underlying algorithms.

This level of abstraction which we call the *user level* is designed to be a beginner's entry point to 3-D object recognition providing an end-user programming interface that is fast and easy to use. The guidelines we set for the methods' interface definition and documentation encourage the implementation of reusable and flexible code.

B. Configuration-Level Abstraction

Key to the straight-forward programming interface as proposed in Subsection III-A is the *method* place holder. We use the notion of an underlying method to mask out the know-how needed to implement software for object recognition use cases. In this subsection we will continue this practice on another yet lower level of abstraction that we call the *configuration level*.

A survey of recent 3-D object recognition approaches has led us to a generalized view on the necessary steps involved. As depicted in Figure 1 for both training and recognition we isolated four general tasks that need to be performed. The task of a user working on this level is to test different methods for acquisition, preprocessing, Modelling, Description, Storage and Matching for a given use case in order to create or optimize a process. Therefore we require rapid exchangeability of processing pipeline elements.

1) Acquisition: Acquiring 3-D data is the first step and common to both processes. For a user-friendly framework it is vital to offer a generalized interface for a wide variety of acquisition devices.

2) Digression: PFH and VFH: For the generalized processes' remaining discussion we will take *Point Feature Histogram (PFH)* [4] and *Viewpoint Feature Histogram (VFH)*

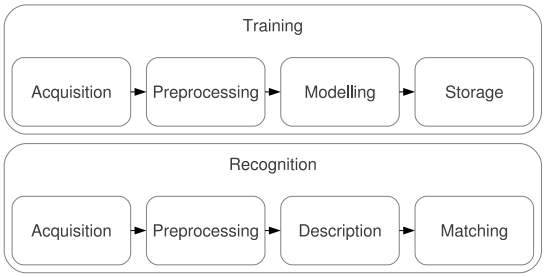

Fig. 1 – Generalized training and recognition processes.

[5] as examples for object recognition approaches to make our classification of the tasks more tangible.

PFH describes, as its name implies, features of a point inside a point cloud. Given its neighborhood, which is defined by a sphere or a maximum number of neighboring points, a histogram is computed which encodes local features of the neighborhood. Scanning an object from multiple view points during training and computing the PFH descriptors allows us to piece together the resulting point clouds to form a single 3-D model. This process is called *registration*. We can then use the PFH descriptors we already know and assign them to points of the registered model or just compute new ones from the model cloud.

For recognition using PFH we scan the current scene and calculate its point descriptors. These can then be matched with the descriptors of all objects in our training database. After filtering the results considering the matches' quality and cumulation we receive potential correspondences between points in our scan and points of our models. These correspondences are used to align the models with the scan. For that we use *Iterative Closest Point (ICP)* [6], the same technique that is used for registration. Thus we are combining object recognition and 6-DOF pose estimation.

Using VFH we can take a different approach that does not rely on surface alignment. It is actually based on a computationally faster modification of PFH called *Fast Point Feature Histogram (FPFH)* [7]. During training the local FPFH descriptors of the object's segment are encoded in a high-dimensional global descriptor. VFH also encodes information about the point from which the object was viewed. This is why during training we are able to store the viewpoint along with the segment's descriptor. This is to be repeated for many different viewpoints per object.

Recognizing objects with VFH is basically the same process as training. Here we need segmentation which is described in Subsection III-B3. For all resulting segments the VFH descriptor is computed and matched with the descriptors in our training database. Filtering the match results with regard to their quality we are able to retrieve not only the object but also viewpoint information, permitting us to do 6-DOF pose estimation.

3) Preprocessing: Both approaches discussed in Subsection III-B2 require *segmented* point clouds for training, meaning that the actual training data is a set of contiguous clouds that *only* contain points belonging to the object being trained. The reason for this is that the model data created during training should not be polluted by information that does not describe the object. For the recognition process the requirements of the two examples are different.

VFH is relying on segmentation to provide candidate clouds that can be compared to the database. Segmentation for recognition however will be different to the segmentation needed for training. In most cases there will not be a set-up of the scene that is as well-defined as it is possible for training environments. Segmentation for recognition purposes has to cope with the fact that there can be multiple objects in the scene and also with other problems like occlusion. At the moment, to the authors' best knowledge there are no segmentation algorithms able to solve this problem generally. However, there are many solutions for special cases like objects on a tabletop or objects on a shelf. Therefore we state that the choice of the segmentation algorithm is highly dependent on the use case at hand.

In contrary to VFH, PFH can be applied to a point cloud of the entire scene and does not require prior segmentation. The reason for that has been provided when discussing the method in Subsection III-B2. This does *not* imply that preprocessing can be omitted completely for PFH.

Our preprocessing step is a place holder for multiple steps performed on raw data. It accounts for the fact that most object recognition approaches are either unable to use sensor output directly or exhibit poor results concerning recognition quality and processing time when used with such data. We explained that choosing the correct preprocessing algorithm is important for training and even more so for some recognition methods. Our framework must therefore provide the capability to exchange preprocessing algorithms easily for testing and evaluation purposes.

4) Description and Modeling: As seen in Subsection III-B2, PFH and VFH are very dissimilar approaches to object recognition regarding the computations involved. However, the data generated by both methods can be considered similar. For that reason we are able to develop the simple and extendible data layout depicted in Figure 2 that is able to support current object recognition techniques.

During training, segments are created that contain a small fraction of the amount of data captured by the sensor, ensuring that the amount of memory necessary for storing them is minimal. Storage efficiency is further increased by creating a

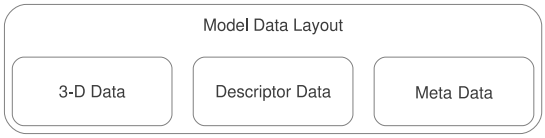

Fig. 2 – Generalized data layout.

registered[2] model instead of saving each single shot. However, since keeping the framework flexible is one of our primary goals, we define that the 3-D portion of our data layout is allowed to contain multiple point clouds.

3-D data is also the key to model compatibility since it can be used to compute the various higher-level descriptors such as PFH and VFH but also *Signature of Histograms of Orientation (SHOT)* [8], FPFH or *Spin Images* [9] - just to name a few. Of course training will usually be performed for a certain method and therefore for a certain descriptor. So in order to avoid recomputing higher-level descriptors whenever starting a recognition process, we define the *descriptor data* field. It is to contain the specialized descriptors used by the method chosen for training.

The question whether a training method is compatible with a recognition method is obviously answered here. If the data description method used during training does not match the description method used for the current scene, then a comparison will be meaningless or even impossible in the subsequent matching step. We define that the method chosen for recognition always supersedes the method chosen for training. This way we can define two cases in which the database and the recognition algorithm are compatible: The first case is when the same description method was chosen for both training and recognition. If this is not the case, the database must contain 3-D data which allows the framework to recompute compatible model descriptions. We do not demand but rather encourage the availability of 3-D data in the database since this can consume high amounts of memory.

For VFH we explained that its abilities can only be fully used when providing view point information in addition to a segment's descriptor. For such cases we chose to include a meta data field in our layout. Meta data does not only come in handy for VFH. A major reason for object recognition is our desire to have robots manipulate or operate objects. Let the object be a coffee maker for instance: If we want the robot to serve us coffee we need it to be capable of pressing the buttons. These however are not necessarily detectable with the hardware being used. In this case their positions relative to an object coordinate system can be stored in a meta data object.

The description and modelling tasks of training and recognition generally use the data provided by the respective preprocessing step to create a higher level description of the data. We already mentioned many of those methods throughout this paper. These descriptions have a major advantage over plain 3-D data: They are easily compared to another description of the same type. All of these methods have their advantages and drawbacks especially when considered for different object recognition use cases. By providing a generalized data layout and defining *modeling* and *description* as place holders we equip the user with the ability to quickly exchange the data description method used for both of the processes.

[2]The process of piecing together shots taken from multiple view points is called registration.

5) Matching and Storage: Making the training data persistent is what we define as the storage task. However, discussing the pros and cons of various data formats is not the goal of this paper. We simply state that the framework is to support multiple, preferably human-readable formats to persistently store training results. We define this to be the task of an entity we call the *database*.

Matching is the effort of comparing the descriptions stored in the database to a description created from new sensor data. In the context of our framework we define this to be a task of the database as well. It is fed with the current scene description as a query and we expect it to return corresponding matches as results. The *matching* place holder stands for the method how matching is performed. For 3-D data we would probably use an octree data structure to perform nearest neighbour searches. We can use the similar k-d tree to do the same with k-dimensional descriptions. However, we can also think of a support vector machine that is trained with classes that represent our models. As for all steps introduced in this section we demand that the choice of the actual implementation is open to configuration.

6) Configuration: Having defined the general steps we deem necessary for object recognition, we are able to explain what a *method* actually is. A method is a configuration entity for either training, recognition or both. It determines the implementation used for each step. So it is only a continuation of the *Strategy* design pattern, mentioned in Section III-A.

The added control given to users at the configuration level of course comes at a cost. In contrast to the user level we demand that the developer knows about the algorithms he configures for each step. For instance it will not make much sense to assign a preprocessing step not performing segmentation to a recognition process if the following step is computing a VFH descriptor. We intend to help developers during configuration with documentation similar to the one offered at the user level.

C. Developer-Level Abstraction

On the developer level actual implementations of all the processing steps are provided. For this level we require a clean and simple set of rules on how to wrap an algorithm for BOR3D. Only so our framework will be able to attract scientists who wish to publish their latest findings. However it must be emphasized that a good portion of the expert's work is in providing comprehensible documentation.

IV. A Use Case Solved

We will now show the tabletop use case mentioned in the introduction to describe the current state of our framework's implementation. First we briefly explain how our implementation is subdivided with respect to the steps introduced in Section III-B. The steps necessary to provide the user-level API will be described thereafter.

A. Tabletop: Processing Pipeline

We implement the tabletop use case using the VFH descriptor and a Kinect controller. As stated in the introduction we

do not allow occlusion between objects on the table which simplifies the segmentation task. Our processing pipeline is best described graphically as it is done in Figure 3.

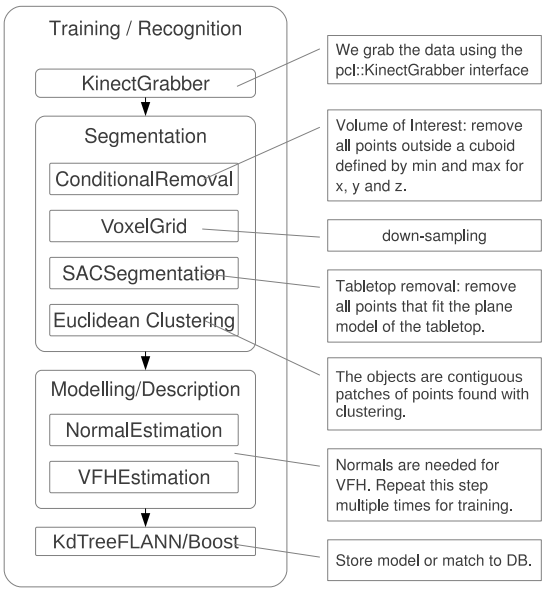

Fig. 3 – Tabletop object recognition processing pipeline.

B. Framework Implementation

We still need to solve the puzzle how a simple line like "`Training<Tabletop> training;`" is able to spawn a processing pipeline like the one shown in Figure 3. The `Training<MethodT>` and `Recognition<MethodT>` class templates we have seen in the examples of Section III-A represent the two basic processes entailed in object recognition. We also established that the *method* here represented by the `MethodT` template parameter is nothing more than a configuration entity. In C++ types (or classes) are used as template parameters. Pipeline configuration in our framework is done at compile-time. The whole pipeline set-up is determined by data types. Why this is necessary can be seen when considering the flexibility we demand of our framework. Looking at the different descriptors we know from Section III-B2, we see that different data types are needed to represent the respective data descriptions. Clearly the descriptor decision does not only impact the computation but also the type of data stored in our models and database. Another reason for type based configuration is the fact that some sensors provide more than just 3-D data: They can also assign colors to the points or do some preprocessing on their own like estimating normals. Different data types are needed to use this additional information inside our framework.

1) Compile-Time Configuration: C++ allows the definition of nested types. That means that inside a class the developer is able to define another class or redefine an existing type with another name using the `typedef` statement. This is especially interesting when defining template classes. Using the `typedef` statement they are able to publish their parameter's types with predefined names. In the case of training we defined that the "value" of `MethodT` can be accessed with `Training<MethodT>::MethodType`. Defining the name `MethodType` to access the template parameter is like defining an interface.

The second property of the C++ language we use is template specialization. For different template parameters a function or class be completely reimplemented to exhibit a different behavior.

With these possibilities in mind we define that the `MethodT` parameter is no actual type that will ever be instantiated by the C++ compiler. It is rather an aggregation of several types that influence different parts of the behavior of the training and recognition processes. This is achieved by defining the interfaces that are used for data exchange within training and recognition as templates as well. Due to the dependencies that exist between the steps we defined in Section III-B, the actual `MethodT` is holding types that serve configuration purposes themselves. The current configuration interface is best described considering Figure 4. This way we

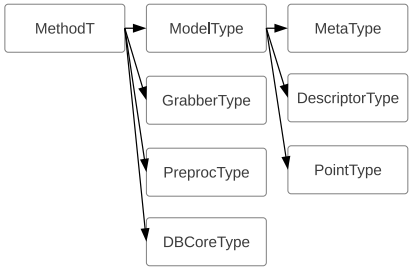

Fig. 4 – Type based configuration.

can see that the grabber used determines the point type used by the processes while the type of model chooses the descriptor and meta data types.

2) Training and Recognition: For training and recognition our framework defines the class templates we already know from the examples. They are the manifestation of the generalized processes. Training and recognition use special interface templates that we designed for each processing step. These interfaces define how data is passed on between the single steps in a way that does not depend on the data type. We achieve this using containers which are templates as well. The standard container for data exchange we use is STL's `std::vector<T>`. It can easily be serialized using BOOST's serialization library and it is quickly converted to PCL's `pcl::PointCloud<PointT>` container. In future we also want the container type to be configurable in order to minimize conversion overhead.

3) Database: In addition to training and recognition we defined another class only visible to the user because of the training file generated by training. The database is the one entity that forms the connection between training and recognition. The `ModelDB` class template is its representation in our framework. It uses the `DBCoreType` parameter to

choose its matching method. Compatibility between a training file and the current recognition method is determined by the `ModelType` parameter.

4) Tabletop Implementation: What remains to be done in order to have our framework perform the processes described in Figure 3 is to provide implementations for the interface templates needed by training and recognition. This is the work that is done at the developer level. We wrap the PCL's Kinect grabber to fulfil our framework's `GrabberBase` interface. The same has to be done for segmentation. Since for VFH description and modeling are the same, we are able to use only one wrapper. The training class template is taking care of the repetition needed to form a model. After completion we move on to the configuration level. Here we create a new *method* by either explicitly instantiating the `Method` template class with the types of our wrappers or by publishing a `typedef` with the same definition. Choosing the typedef way enables us to rename the unreadable template instantiation to "Tabletop". This method can then be used to configure `Training<MethodT>` and `Recognition<MethodT>`.

V. Conclusion and future work

In this paper we introduced the software framework *BOR3D* which is capable of combining different object recognition techniques. At the same time it increases the ease of use for developers working on use cases in real-world scenarios. The separation of user level, configuration level and developer level implementation assigns a defined role to every type of user. It relieves beginners of having to dig deep into the underlying techniques although they only wish to use a boilerplate solution. Implementation and configuration is reserved for users whose primary focus is machine vision.

Looking deeper into the more specialized abstraction levels, there is also a lot of potential for future use. Like in MOPED, the choice between a multitude of algorithms for many different purposes enables the user-level developer to focus on setting parameters for the techniques he wishes to use, not having to worry about their implementation instead. The flexibility of our framework as well as its abstraction capabilities conceal developer-level and even configuration-level implementation issues from the user's eyes. However, these algorithms need to be integrated first, either in the shape of complete libraries or standalone. Eventually, this framework's power will increase crucially with the amount of methods and algorithms included.

We believe that the basic ideas of our framework do not only apply to 3-D object recognition but also to further signal processing disciplines in which active research is conducted. Being seen as a 2-D sister project of the PCL, OpenCV [10] incorporates tools and algorithms pursuing an objective similar to the PCL's. Thus, an important step towards the inclusion of object recognition using 2-D images is the integration of OpenCV into our proposed framework which increases its current capabilities many times over. Additionally, one can also think of using mutually complementing libraries or even switching between some of them which provide similar functionality, like the matrix tools in OpenCV, Armadillo [11] and Eigen [12].

BOR3D's flexibility is tightly connected to the amount and variety of use cases included. A very demonstrative example whose integration we plan to tackle in the near future is a highly cluttered scene with a lot of objects being placed in a shelf like in the *Cluster Recognition and 6DOF Pose Estimation using VFH descriptors* tutorial included in the PCL [13]. The tabletop scenario described herein only serves as a starting point.

The framework will be used in term projects of undergraduate and graduate students at Ravensburg-Weingarten University of Applied Sciences starting in March 2012. We are currently working on a questionnaire for the users to evaluate BOR3D regarding our requirements.

BOR3D is available on SourceForge (http://sourceforge.net/projects/bor3d).

Acknowledgments

The authors would like to thank the members of the Institute of Artificial Intelligence for fertile discussions. This work was conducted within the Collaborative Center for Applied Research on Service Robotics (ZAFH Servicerobotik) and supported by research grants of the state of Baden-Württemberg and the European Union.

References

[1] R. B. Rusu and S. Cousins, "3D is here: Point Cloud Library (PCL)," in *IEEE International Conference on Robotics and Automation (ICRA)*, Shanghai, China, May 9-13 2011.

[2] A. Collet, M. Martinez, and S. S. Srinivasa, "The MOPED framework: Object Recognition and Pose Estimation for Manipulation," *The International Journal of Robotics Research*, 2011.

[3] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, *Design Patterns – Elements of Reusable Object-Oriented Software*, 1st ed. Amsterdam: Addison-Wesley Longman, 1995.

[4] R. B. Rusu, N. Blodow, Z. C. Marton, and M. Beetz, "Aligning Point Cloud Views using Persistent Feature Histograms," in *Proceedings of the 21st IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), Nice, France, September 22-26*, 2008.

[5] R. B. Rusu, G. Bradski, R. Thibaux, and J. Hsu, "Fast 3D Recognition and Pose Using the Viewpoint Feature Histogram," in *Proceedings of the 23rd IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Taipei, Taiwan, 2010.

[6] P. J. Besl and N. D. McKay, "A method for registration of 3-d shapes," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 14, pp. 239–256, February 1992.

[7] R. B. Rusu, N. Blodow, and M. Beetz, "Fast point feature histograms (fpfh) for 3d registration," in *The IEEE International Conference on Robotics and Automation (ICRA)*, Kobe, Japan, 05/2009 2009. [Online]. Available: http://files.rbrusu.com/publications/Rusu09ICRA.pdf

[8] F. Tombari, S. Salti, and L. Di Stefano, "Unique signatures of histograms for local surface description," in *Proceedings of the 11th European conference on computer vision conference on Computer vision: Part III*, ser. ECCV'10. Berlin, Heidelberg: Springer-Verlag, 2010, pp. 356–369. [Online]. Available: http://dl.acm.org/citation.cfm?id=1927006.1927035

[9] A. E. Johnson, "Spin-images: A representation for 3-d surface matching," Tech. Rep., 1997.

[10] G. Bradski, "The OpenCV Library," *Dr. Dobb's Journal of Software Tools*, 2000.

[11] C. Sanderson, "Armadillo library," http://arma.sourceforge.net, 2011.

[12] G. Guennebaud, B. Jacob *et al.*, "Eigen v3," http://eigen.tuxfamily.org, 2010.

[13] R. B. Rusu, "Point Cloud Library: Cluster Recognition and 6DOF Pose Estimation using VFH descriptors," http://pointclouds.org/documentation/tutorials/vfh_recognition.php, 2011.