

Occluded Object Geometry Estimation

by

Matthew S. Roscoe,

Kenneth B. Kent and Paul G. Plöger

TR14-228

October 11th 2013

Faculty of Computer Science
University of New Brunswick
Fredericton, NB, E3B 5A3
Canada

Phone: (506) 453-4566

Fax: (506) 453-3566

Email: fcs@unb.ca

<http://www.cs.unb.ca>

Abstract

Object classification systems are becoming more pervasive with the wide availability of low cost RGBD sensors. These sensors are making 3D information available to a large audience, but do so at the cost of precision and accuracy. In order to classify the information obtained from these sensors these deficiencies need to be accounted for. Objects that are of most interest to robotics are typically located in cluttered environments. They suffer from occlusion caused by perspective or other objects being located between the sensor and the target object. This project investigates reconstructing the surface of the object that cannot be fully seen due to occlusion or sensor deficiencies.

Table of Contents

| | |
|--------------------------------------|-----------|
| Abstract | i |
| Table of Contents | iv |
| List of Figures | v |
| Glossary | vi |
| 1 Introduction | 1 |
| 1.1 Occlusion | 3 |
| 1.1.1 Visual Occlusions | 4 |
| 1.1.2 Physical Occlusions | 6 |
| 1.2 Detailed Paper Outline | 7 |
| 2 Related Work | 8 |
| 2.1 Point Cloud Methods | 8 |
| 2.1.1 Inpainting Methods | 9 |
| 2.2 Mesh Based Methods | 10 |
| 3 System Design | 13 |

| | | |
|----------|---------------------------------------|-----------|
| 3.1 | Overview | 13 |
| 3.2 | Scenario Descriptions | 14 |
| 3.2.1 | RoboCup@Home | 14 |
| 3.2.2 | RoboCup@Work | 15 |
| 3.2.3 | Assumptions | 16 |
| 3.3 | System Overview | 17 |
| 3.3.1 | Data Capture | 18 |
| 3.3.2 | Point Cloud Processing | 18 |
| 3.3.3 | Mesh Processing | 19 |
| 4 | Data Preparation | 20 |
| 4.1 | Data Capture | 20 |
| 4.2 | Downsampling | 21 |
| 4.3 | Noise Reduction | 22 |
| 4.4 | Object Candidate Extraction | 23 |
| 4.5 | Smoothing & Upscaling | 25 |
| 4.6 | Mesh Creation | 26 |
| 5 | Occlusion Repair | 29 |
| 5.1 | Occlusion Identification | 29 |
| 5.1.1 | PCL 3D Edge Detection | 30 |
| 5.1.2 | VTK Hole Detection | 31 |
| 5.1.3 | Further Discussion | 32 |
| 5.2 | Occlusion Repair | 33 |

| | | |
|----------|---|-----------|
| 5.2.1 | Mesh “Shrink Wrapping” | 33 |
| 5.2.2 | VTK HoleFillerFilter | 35 |
| 5.3 | Surface Smoothing | 37 |
| 6 | Results, Conclusions & Future Work | 38 |
| 6.1 | Overall Results | 38 |
| 6.1.1 | Object Candidate Creation | 39 |
| 6.1.1.1 | RoboCup Constraints | 39 |
| 6.1.2 | Occlusion Detection | 40 |
| 6.1.2.1 | NaN Clustering & Repair | 40 |
| 6.2 | Occlusion Repair | 42 |
| 6.3 | Conclusions | 44 |
| 6.4 | Future Work | 46 |
| 6.4.1 | Porting High Fidelity Methods | 46 |
| 6.4.1.1 | NaN Clustering & Region Growing | 47 |
| | References | 52 |

List of Figures

| | | |
|-----|--|----|
| 1.1 | Original 2D image no modifications made. | 1 |
| 1.2 | 2D scene capture segmented based on colour. | 1 |
| 1.3 | Side on view of a visual occlusion. | 4 |
| 1.4 | Birds eye view of a visual occlusion. | 4 |
| 1.5 | Side view of a physical occlusion. | 6 |
| 1.6 | Birds eye view of a physical occlusion. | 6 |
| 4.1 | Initial point cloud to be converted. | 26 |
| 4.2 | Common output from the Poisson mesh construction method. | 26 |
| 4.3 | Best meshing obtained from Kinect input. | 28 |
| 5.1 | Occluding Edges in Green and Occluded Edges in Red [1]. | 30 |
| 5.2 | Sphere being wrapped around example points [2]. | 34 |
| 5.3 | Typical Shrink Wrapping Results. | 34 |
| 5.4 | Mesh with incorrectly repaired occlusions. | 36 |
| 6.1 | Typical object candidate data capture with desired areas for NaN Clustering highlighted in Red. | 41 |

Glossary

2D 2-Dimensional. 1, 2

3D 3-Dimensional. 2, 21

AFM Advancing Front Method. 11, 35

centroid is a point in 3D space, which represents the center of mass (gravity) of a 3-Dimensional region. In this case the region will be a grouping of points from a point cloud. . 21

matte is a surface whose material reflects at most 10% of the initial light source. For our purposes this referees to reflecting the pattern being emitted for detection by a PrimeSense[®] device. . 16

mesh is a a collection of nodes, which are connected by edges in order to create faces. These faces will be combined to create a solid surface which will provide a 3D representation of the object that was initially just made up of the nodes. . 8, 17–19, 25–27, 33, 38, 42

mobile robot is an automated machine that is capable of movement in any given environment. 4

NaN Not A Number. 41

object feature is an isolated part of an image or RGB-D image that can later be used in order to identify similar objects that the system will encounter. . 15

occlusion is defined in Section 1.1. . 19

orbiting is to move around an object in a curved path, usually in a circular or elliptical manner. . 6

PCL Point Cloud Library. 17, 18, 20–22, 24, 25, 40–42

physical interaction is when components of two different objects come into contact with one another. This can happen either through surface contact or surface intersection. 4, 6

point cloud point clouds are a data type that stores information. Normally the exterior of a surface in a coordinate system. At a minimum they must store the points x, y, z coordinates but can additionally store information such as colour Red-Green-Blue (RGB) and an alpha value for the associated colour. . vi, 8–10, 17–20, 22, 23, 25, 27, 33, 40–42

RGB Red-Green-Blue. vii, 1, 9, 10, 41, 42

RGB-D Red-Green-Blue-Depth. 2, 3, 9, 15, 17, 30, 33, 38, 46, 47

ROS Robot Operating System. 17, 18

VOXEL Volumetric Pixel. 21

VTK Visualization Tool Kit. 17, 44

Chapter 1

Introduction

In order for robotic systems to interact with the environment they operate in, they need to be able to perceive and understand this environment. Initially this was done using standard monocular, color RGB cameras. This class of sensors provides a standard 2-Dimensional (2D) image, which computers then try to process to determine what was in the environment.

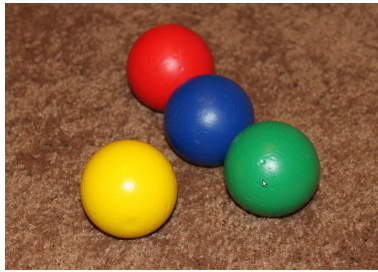


Figure 1.1: Original 2D image no modifications made.

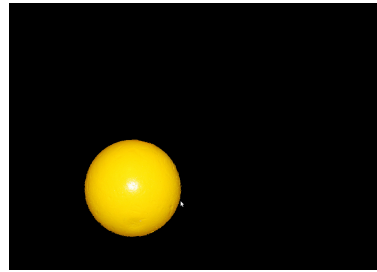


Figure 1.2: 2D scene capture segmented based on colour.

The image segmentation technique is applied to a 2D image, in order to split it into smaller, more meaningful portions (segments). Breaking the image into smaller groupings one is better able to understand what the robot is observing. The segmentation process can be done based on colour, shape, size, or any number of criteria.

As can be seen from the segmentation in Figure 1.2, it is possible find the *yellow* object as desired. However there are remaining questions about the object that was found. Due to a lack of depth information there is no way to tell from the raw data if the object is spherical or circular in nature. Questions also remain about where the object is located, and about its size. This information could be obtained with more processing on the image and making several assumptions. This information can be obtained without processing by using a 3-Dimensional (3D) sensors.

In the last few years the advent of Red-Green-Blue-Depth (RGB-D) sensors¹ has allowed robotics to move from low-cost 2D sensing to low-cost 3D sensing. This allows robotic systems to explore their environments using more detailed information. Instead of segmentation being performed based on 2D shape, color, etc...the system can now operate using 3D information from the environment.

¹Some example sensors would be the Microsoft Kinect[3], ASUS Xtion Pro[4] and the PrimeSense[®] Carmine and Capri sensors [5]

This information provides the system with the ability to not only precisely know the size and shape of the object, but its location and orientation with respect to the sensor. From this additional information, it makes previous issues of occlusion more evident. In order to take advantage of the shape information it must be corrected in order to allow for a more complete image of what is being seen.

This project takes the raw input data from a RGB-D sensor and corrects it so that it can be more reliable during the classification process. The system will attempt to remove impurities in the surfaces such as noise or missing information. This project will present a reconstruction process that by using only the raw input data, is able to construct a representation that has filled both occlusions and data impurities.

1.1 Occlusion

The Merriam-Webster dictionary describes occlusion as “to close up or block off” [6]. For this project focus lies in the “blocking off” portion of this definition is the most applicable to our task. The project will be looking at scenes or scenarios where one object has blocked off another. Occlusions are grouped into two categories:

1. Visual Occlusion.

2. Physical Occlusion.

1.1.1 Visual Occlusions

This is the simplest form of occlusion to deal with when working with mobile robots. This type of occlusion is where one or more objects are *visually* blocking the robot from being able to see another object. The key aspect to visual occlusion is that the objects, which are occluding the target do not have any physical interaction with the desired object.²



Figure 1.3: Side on view of a visual occlusion.



Figure 1.4: Birds eye view of a visual occlusion.

The simplest way to deal with this class of occlusion is to simply move around the subject until the system can fully view the target object. As can be seen from Figure 1.4 the system could easily rotate the camera angle in Figure 1.3 counter clockwise to clear the occlusion. This is a relatively simple task for a mobile robot and allows for the resolution of the visual occlusion. For more

²From now on the desired object will be referred to as the *target object*.

information about motion planning from occlusion please see the following resources [7, 8, 9]

1.1.2 Physical Occlusions

This type of occlusion is what this project focuses on. Like visual occlusions there is an object located between the camera and the target object. The difference with this class of occlusion is that there is physical interaction between the target object and the occlusion causing object(s).



Figure 1.5: Side view of a physical occlusion.



Figure 1.6: Birds eye view of a physical occlusion.

The most notable characteristic of this class of occlusion is that the system cannot remove the occlusion by orbiting the target object in any direction. The occlusion could be described as a “part” of the target object itself. The technique for solving this class of occlusion is primarily inspired by the way human beings attempt to deal with occlusions.

1.2 Detailed Paper Outline

The second chapter of this paper discusses relevant work to the desired direction of this paper. Chapter three discusses the relevant scenarios that define this work, as well as a broad overview of the proposed pipeline. Chapters four and five discuss the processing steps in the pipeline in detail. Chapter six discusses the overall evaluation of the pipeline along with final discussions as well as future directions that this work could follow.

Chapter 2

Related Work

This chapter presents work that is related to the goal of this project (filling in visual occlusion). It presents an assessment of the viability of a variety of techniques to guide the decision making process. There are two main types of related work that this project focused on: point cloud based methods and mesh based methods. This chapter examines various methods from both categories and presents the argument for working with a mesh based approach.

2.1 Point Cloud Methods

Point clouds are a data type containing x, y, z information that is stored in a coordinate system. The points are normally placed in the coordinate system in relation to the position of the sensor. Most of the structure a

person sees when looking at a point cloud is being inferred by them without them knowing. Most industrial applications requiring 3D information if it is initially collected as a point cloud, it will be converted to a mesh for further processing.

2.1.1 Inpainting Methods

Image Inpainting [10] is a well studied problem in the field of computer vision. It commonly refers to the process of identifying missing data in an RGB image and using the information around it to hypothesize about what colors should fit in the missing data. While using this process with RGB information is common, doing so with the newer RGB-D information is not.

In a paper by Sharf et al. [11] a method was proposed that focused on hole filling. This method works from a coarse grain to a fine grain method in which they would fill in the rough surface geometry and then detailed information about the infilled area would be copied from another part of the object. Their final processing step is to “wrap” the generated data so that it would match the surrounding area. One of the biggest problems this paper faced was the difficulty of defining the boundary of a hole in point cloud data and they were using dense data samples. To fit the requirements of run time on robotics, the running time needs to be kept low. This leads to less densely populated point clouds making the hole identification problem more difficult.

Doria et al. [12] build from the previously mentioned work to combine both infilling for the RGB information as well as using depth gradients to correct for missing depth information. While their results are impressive they are relying on information that is present in the scene in order to fill in the missing data. While this is not a major drawback for their methods as they have large point clouds and their processes are not time sensitive they can afford to keep large amounts of data. Robotics application would require significantly smaller data sizes. This leads to objects or working areas of $1000 - 10000pts$. Given that with a hole size of $14000pts$ they had a running time of around $23s$ [12] the running times increase almost exponentially from there with their most tested being $45000pts$ with a running time of $2m19s$ [12]. While the results are for the most part highly accurate the running times and requirement for a large amount of input data means that this method is not applicable to this project.

2.2 Mesh Based Methods

As mentioned in section 2.1.1 point clouds are normally post processed into a mesh based data structure for future processing. The reason for this is that a mesh has more structure to it. They typically consist of nodes (the points from the point cloud) and edges (links between the nodes) and finally faces/polygons (links between various edges). These three components are

combined to create a surface which is a closed form representation of the item scanned. There are many different ways to represent the faces / polygons but in the end it is a solid (closed) mesh that represents the object being searched for. Thanks to the solid nature of the mesh there is a great deal more information about the object and the holes it contains.

Peter Liepa [13] presented a method for filling unstructured triangular meshes. This method worked by identifying the hole that was present in the mesh. Then triangulating [14] the detected hole(s). Refining the mesh to better represent the object or portion of the object that is being dealt with, and Finally fairing (smoothing) [15] the mesh (minimizing the fairness function). The biggest problem with this approach is the $O(n^3)$ running time on the triangulation of the hole(s). In general though their algorithm tends to run in terms of seconds instead of minutes as long as the number of edges that make up the occlusion (hole) is below 100.

Taking a different approach Zhao et al. [16] applies the Advancing Front Method (AFM) [17]. This method instead of trying to connect multiple points often on opposite ends of the hole (triangulating) works its way from an initial edge across the hole. Once they have this initial mesh they refine it using the Poisson equation to refine the initial mesh and ensure that it fits in the hole it was meant to fill. While this method also suffers from the requirement of needing the hole to be a part of a closed mesh, the running

times are much more acceptable. For generating 578 new vertices it takes a total of $174ms^1$, and for 181 new vertices it takes $18ms$. These run times are still outside of our desired range of $\leq 33ms$ but much more acceptable than a runtime in the range of seconds to minutes.

The running times and the robustness to noise and missing data make the second presented approach the desired methodology. For more information about the various methods currently available please consult the following articles: [18]

¹The runnings times are broken into two processing parts: time spent solving harmonic equations and time spent solving Poisson equations

Chapter 3

System Design

3.1 Overview

This chapter presents and discusses the operating scenarios that drove the development of the occlusion repair system. The system requirements that were derived from these scenarios are presented as well as any assumptions that were made. Finally a high level overview of the occlusion repair system will be presented.

3.2 Scenario Descriptions

The proposed system is implemented with use in the RoboCup(@Home & @Work) competitions. The main systems come from the *b-it-bots* teams [19]. This system focuses on a deployment using either the Care-O-bot 3[®] [20] or the KUKA[®] youBot robotics platform [21]. The PrimeSense[®] device will be placed at 1.30m (@Home) or 0.70m (@Work) in the vertical axis, depending on the competition the system is being used in. The placement of the sensors on the robotics platforms is meant to allow for the maximum effective use of the sensor, based on the system common operating environment.

The major difference between the two competitions from the view of this research is size and location of the target objects. The differences in setting and object attributes leads to a wide variety of scenarios that the system must be able to perform in. The specific requirements for each competition will be presented in the following sections.

3.2.1 RoboCup@Home

The RoboCup@Home competition takes place in a home or apartment based environment [22]. With the objects being fairly large¹ it is easier to get more reliable data on the overall object shape as the object size will often be outside of the PrimeSense[®] devices error range[5]. The “rough” object shape is

¹See Figure 3.1 from the RoboCup@Home rulebook [22]

easier to obtain, allowing the system to focus on the smaller, more difficult to recreate details.

Unlike the *@Work* competitions the tasks facing the robot require the robot to not only be able to recognize an object but to detect detailed object features. These features allow the system to determine the best manner to *interact* with the object being recognized.

3.2.2 RoboCup@Work

At a high level the *@Work* competitions present a simpler problem at the moment: simply having to recognize “what” an object is so that it is possible to grasp (manipulate) the object at a later time. Unlike the *@Home* competition however the *RoboCup@Work* competitions brings an added level of complexity to the data gathering and processing problem.

The majority of the objects are much smaller than those that would be found in the *RoboCup@Home* league, with a minimum size of $2cm^3$ [23]. This presents an added challenge as the resolution of the current generation of RGB-D sensors have error ranges that with our set up would produce errors of up to $5mm$ (in the x,y or z direction) or up to 25% deviation from its true position.

3.2.3 Assumptions

Several assumptions about the placement and attributes of the objects were made in order to simplify the operations being performed.

1. Object sizes will be at least $2cm^3$ [23].
2. All objects will be located on planar surfaces such as a table or a RoboCup@Work service area.
3. All objects will be non-transparent, as transparent objects return corrupted data when using PrimeSense[®] based sensors [24].
4. All objects will have closed surfaces. They will be closed either through interaction with a surface plane or through the natural shape of the object.
5. All objects will have a matte surface finish. This is done to reduce the amount of corrupt data due to the pattern being reflected elsewhere in the visible scene.
6. To get the best results from the PrimeSense[®] devices target objects will be located between $0.80m$ and $1.3m$ from the sensor.

3.3 System Overview

The occlusion repair system is broken down into three. The system is designed to take in information from low cost RGB-D sensors and then to correct for errors that the sensor introduced as well as any occlusions present in the scene. The repair process for occlusion is broken down into 3 distinct phases. Each phase is associated with the software library that is being used for the processing.

1. Robot Operating System (ROS)
 - (a) Data Capture
2. Point Cloud Library (PCL)
 - (a) Scene pre-processing
 - (b) Object candidate Identification
 - (c) Object candidate processing
 - (d) Conversion from point cloud to mesh
3. Visualization Tool Kit (VTK)
 - (a) Identify occlusion
 - (b) “In-Fill” occlusion
 - (c) Smooth & Adjust in-fill

3.3.1 Data Capture

The system utilizes ROS in order to create a multipurpose input layer. This part of the system is meant to be easily replaced by any program which can output a PCL point cloud. For the purposes of the paper the Microsoft Kinect[3] was used for development and testing purposes.

3.3.2 Point Cloud Processing

This segment of the systems processing refers to any work being done with the point cloud data within PCL. This processing phase will look at the following:

1. Removing outliers.
2. Smoothing the data.
3. Plane extraction.
4. Point clustering (for object candidate extraction).
5. Upsampling object candidates.
6. Conversion from point cloud to a mesh

These processes will be discussed in full detail in Chapter 4.

3.3.3 Mesh Processing

This processing phase will work with mesh data structures which are nodes (much like the points in the point cloud) connected to each other through edges in a triangular fashion. These meshes are created in the previous processing phase in a method called triangulation [14]. Once the mesh is obtained the goal is to determine if any occlusions exist. If any are detected the system in-fills the occlusion and follows up with smoothing and adjusting based on known object geometries.

Chapter 4

Data Preparation

This chapter covers the process of capturing environment information and the various steps that are performed during the initial processing steps.

4.1 Data Capture

The data collection process is defined by the following steps requiring a PCL point cloud. This point cloud can be of any type supported by PCL as long as it contains at a minimum position information (x, y, z) .

The demo system was developed so that the data capture step lasted for a few seconds. This allows for the collection of several point clouds which are then integrated into a single point cloud through the process of registration [25]. This allows for the system to gather more information about the environment

through movement in order to address visual occlusion. For more about these solution methods see the section on visual occlusion repair 1.1.1.

4.2 Downsampling

Working with the amount of data that is obtained from the data capture process leads to long running times. The goal of this system is that it should be usable during *RoboCup* competitions. With most competitions running for a maximum of 5 minutes the shorter the processing time the more usable the system is. This must be balanced with the amount of data that is used to represent the information that was captured from the environment. Table 4.2 demonstrates the trade off between these two factors.

| Voxel Size | Points Removed | Running times | Processing FPS | System FPS |
|------------|----------------|---------------|----------------|------------|
| 0.001m | 0 | 0.010s | 80 | 3.2 |
| 0.002m | 5250 | 0.013s | 76 | 3.9 |
| 0.003m | 38000 | 0.013s | 76 | 5.9 |
| 0.005m | 72000 | 0.009s | 111 | 11.3 |

This project performs downsampling using the PCL voxel grid filter[26]. This filter creates a Volumetric Pixel (VOXEL) that encloses all of the points in that 3D space. The filter then determines the centroid of the points enclosed by the voxel. This centroid is used as the representative point for everything

enclosed by the voxel. This allows for the proper representation of the surface that was obtained while reducing the overall number of points which increases the running time of the system. See the highlighted region of Table 4.2 for the voxel size that this project uses.

4.3 Noise Reduction

After downsampling the data set there are still outliers or noise present in the data set. PrimeSense[®] devices have error rates that range from $1mm$ at $0.80m$ up to $3cm$ at $2m$ [27], and this leads to a fair amount of information that is wrong due to improper sensor measurements. PrimeSense[®] devices also can recognize the pattern it is emitting in places where it is not really found. All of these factors together can result in point clouds which contain a large amount of noise. In order to reduce this the PCL Statistical Outliers Removal Filter[28] is used.

This filter measures the mean distance between points using the K-nearest neighbors method. Once it has this distance it can determine what “should” be present in the scene and what *might* not belong based on a points distance to its *k-nearest neighbors*. If the point falls outside of 1 standard deviation¹ of the mean for that region the point(s) will be removed. Table 4.3 show the number of points removed by varying the mean and standard deviation. The

¹The Statistical Outliers Removal Filter assumes that all points are distributed in a Gaussian manner and this is used to determine the mean and standard deviation for the data set.

highlighted entry in this table represents the final values that were used during testing for this project.

| Mean K | Points Removed | Running times | Processing FPS | System FPS |
|--------|----------------|---------------|----------------|------------|
| 10 | 2200 | 0.09s | 11 | 9.5 |
| 25 | 6000 | 0.175s | 5.7 | 5.5 |
| 50 | 6000 | 0.32s | 3.1 | 2.9 |

4.4 Object Candidate Extraction

At this point in the point cloud processing phase, a more efficient representation of our data through both Downsampling and Noise Reduction has been obtained. The next step is to obtain the object candidates for further processing. One of the assumptions of this project is that all objects will be located on planar surfaces (tables, shelves, work spaces, etc. . .). Initially all planar surfaces in the captured environment information must be identified. This is done using the RANSAC[29] algorithm. The way that the data is captured the following assumptions can be made which allow the determination of which plane is the “operating surface”:

1. The normal for the “operating surface” will be orthogonal to floor on which the robot is standing. That is it should be to aligned to within $\pm 5^\circ$ of the $z - axis$ of the scene.
2. There may be multiple planes which satisfy the initial assumption in

the captured scene. To account for this the assumption is also made that the “operating surface” will also be the largest plane in the environment. This assumption can be safely made due to the competitions setup. Before this system is used, the robot will attempt to best align with the surface it wants to search for objects. This allows for the safe assumption that the largest plane will be the “operating surface” and all others *most likely* belong to objects in the scene.

Once the “operating surface” has been determined object candidates can be identified. From the assumptions before it can be assumed that all objects exist above the “operating surface”. Next the system will remove anything from the scene that is not both above this surface and within the boundaries of this surface. Once all of the extraneous information is removed the operating surface is also removed. What is left are points which could possibly represent the desired objects.

Now the system will use the PCL Euclidean Cluster Extraction filter[30] to determine if any of the remaining points in the scene represent objects. To do this there are three parameters which can be set:

1. **Cluster Tolerance** - this value determines how far away from any other point(s) a point within a cluster can be and still remain a part of the cluster being created.
2. **Min Cluster Size** - the minimum number of points that must be

present for that region to qualify as an object candidate.

3. **Max Cluster Size** - the maximum number of points that can be in a cluster. Any more than this number and the system probably found an environment element which is *not* an object but rather a larger environment fixture.

4.5 Smoothing & Upscaling

This is the final point cloud processing phase before transformation from a point cloud data type to a mesh data type. This process is designed to take an object candidate and to smooth the points in the point cloud that are representing the candidate object. This process is based around the PCL tutorial on *smoothing and normal estimation* [31]. This allows for the correction of minor defects that occurred due to the registration in the data capture section 4.1. This is accomplished using a PCL internal implementation of the Moving Least Squares algorithm[32]. The implementation of this algorithm within PCL means that the previously downsampled point cloud will be upscaled, so that the points that are passed to the mesh creation better represent the overall shape of the object (based on the information which is present from the data capture process).

4.6 Mesh Creation

Now that the system has a smoothed and upscaled version of the object candidates it is able to convert them into a mesh data type. Initially the Poisson method for creating a mesh was investigated. This method presented various problems due to the amount of noise that can still be present in the data. This resulted in “bubbles” being present in the constructed mesh. The Poisson method also enforces the concept of a closed mesh so it would always attempt to close any open areas that can be found between edges. This tended to lead to results which did not resemble the initial object at all.

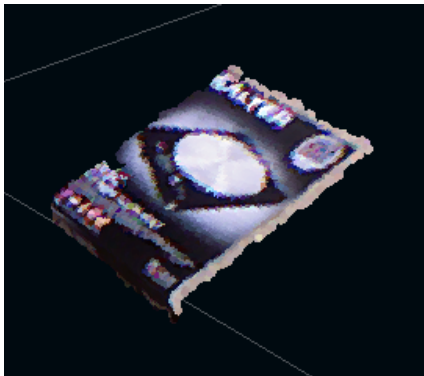


Figure 4.1: Initial point cloud to be converted.

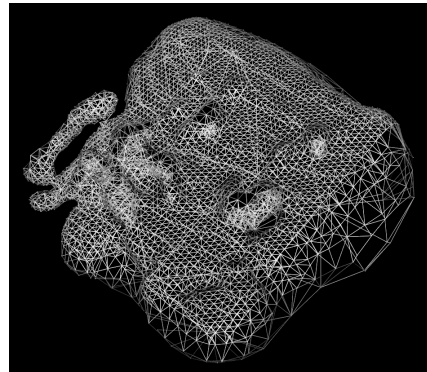


Figure 4.2: Common output from the Poisson mesh construction method.

As can be seen from Figure 4.1 and 4.2 this representation of the initial point cloud is not visually anything close to what is desired (a smooth plane(s)).

The project turned to Grid Projection methods for the point cloud to mesh conversion.

This is done using the PCL GridProjection system which is an implementation of “Polygonizing Extremal Surfaces with Manifold Guarantees” [33].

This method provides us with the following parameters:

1. Resolution - This sets the size of the grid cell telling the system how big the cell should be in meters.
2. Padding Size - The amount of space around the current cell that is being examined that should be included. This is so that if there are no data points inside the cell it will still be processed. This will possibly close holes in the mesh which are smaller than the passed in padding size.
3. Nearest Neighbor Number - The number of nearest neighbors that the system is looking for in relation to the current point.
4. Max Binary Search Level - The max depth the search function can iterate to.

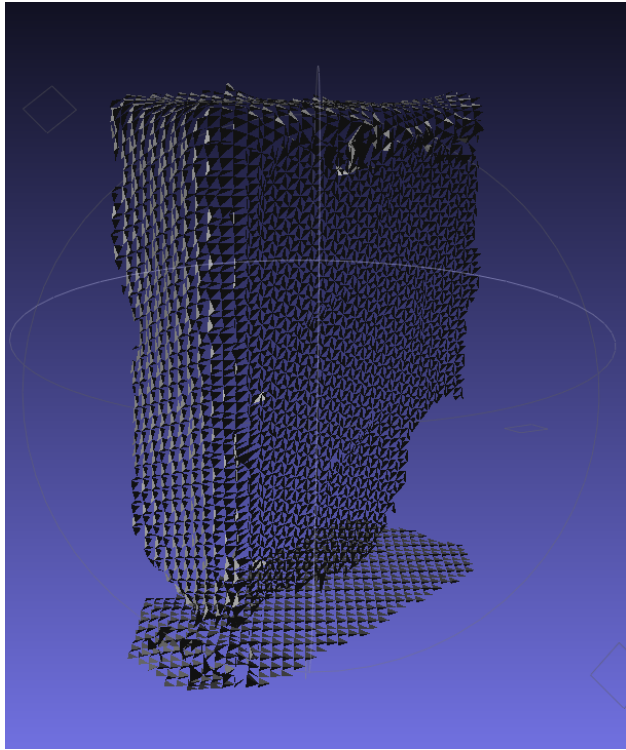


Figure 4.3: Best meshing obtained from Kinect input.

Chapter 5

Occlusion Repair

This chapter focuses on the methods used for performing the occlusion repair.

Each section presents and discusses the following processing steps:

1. Occlusion Identification
2. Occlusion Repair
3. Smoothing

5.1 Occlusion Identification

Before the information from the previous steps can be used, occlusions must be detected so they can be repaired. An occlusion often leaves target candidate object with a “hole” or area of missing information. The first step to repairing the occlusion is to identify the area with the missing information.

5.1.1 PCL 3D Edge Detection

During the 2012 Google Summer of Code, Changyun Choi investigated identifying 3D edges from RGB-D data sets. This work allows for the user to detect the following types of edges: boundary, occluding, occluded, and high curvature edges. The two that are of the most interest to this project are the occluding and occluded edges. The occluding edges are edges that are causing information to be missing elsewhere in the image due to the position of the recording device. The occluded edges are the ones that surround the “shadow” that the occluding object is casting onto other surfaces.

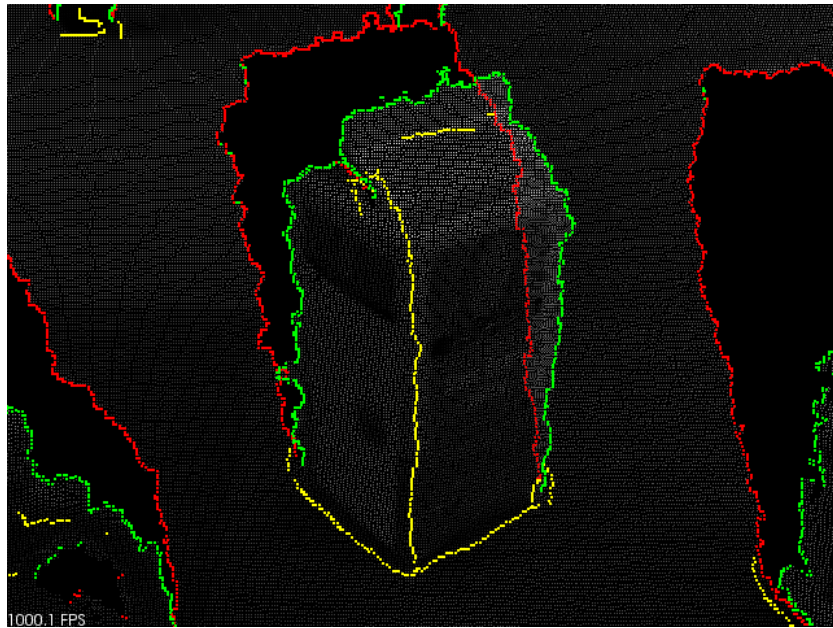


Figure 5.1: Occluding Edges in Green and Occluded Edges in Red [1].

As can be seen from Figure 5.1 in order for this method to work it cannot be feed the candidate object that is being worked with at but rather it requires

the full scene. The other problem is that there is no way to tell if these occlusions extend to another object. If the boundary of an object is found, and the full scene is observed it could be determined that it is “occluded”. If the object is examined without the environment, the same edge cause by the occlusion would be seen as a boundary edge rather than an edge with information missing due to occlusion.

It is very difficult to determine if an edge is actually a part of the object candidate or if the object candidate information is missing due to occlusion. No libraries were available and the implementation of such a method was outside of the scope of the project due to time restrictions.

5.1.2 VTK Hole Detection

The VTK library provides a FeatureEdge [34] function which takes in a VTK mesh and searches it for holes in the mesh structure. It starts by looking for an edge of a cell in a the mesh. Then by following the boundary in order to “close” the edges it has been following to obtain a loop it can determine as a “hole” that needs to be filled.

This functionality was available in a library, which allowed for the integration into the system. It also functioned well when being used with test set meshes such as the “Stanford bunny” [35] when information is removed from the test set meshes. This function however was only able to determine when there where holes that were not against the “outside” of an object. That means it

was only able to identify holes that were surrounded by other information making it impossible to detect occlusions along the edge of the object as well.

The project encountered a problem when trying to use this method with the project produced meshes. This came from the fact that the meshes were not smooth in nature. They contained a great deal of variance as well as jagged edges protruding from almost all parts of the object. This caused either too many “holes” to be detected or none at all as it could never find a “complete” area of the mesh to investigate. It also suffered from the previously mentioned drawback that this filter (nor any other in the VTK library) did not have the ability to determine if an edge was the proper “object edge” or if it was missing information due to occlusion.

5.1.3 Further Discussion

This portion of the project proved to be the most difficult and the sticking point for the project. Given the state of the information the system was being provided with and the requirements of the system there was no reliable way to detect occlusions. This is due to the fact that it requires almost all scene information to determine if an object has an edge which is caused by the natural contours of the object. The alternative is that the edge of the object being examined is caused due to the shadow of another object which is occluding our view of the object in question. As will be discussed in Section 6.4 there are some interesting approaches that if the investigation

of the possibilities without time restraints to determine if this problem can be solved with current generation RGB-D sensors.

5.2 Occlusion Repair

This section covers methods, that are available in order to close the hole in the mesh that were obtained in the previous processing step. Due to the fact that the project was unable to identify a point cloud based method the remaining methodologies will focus around working with mesh data structures.

5.2.1 Mesh “Shrink Wrapping”

This method of closing the hole doesn’t deal directly with the hole that was found but rather the whole object. This method assumes that the object candidate was reduced to at least the boundaries of the object. This method then produces a sphere that is larger than the object that should be wrapped. Then a VTK filter which takes the sphere and “shrink wraps” it to create a surface around the initial points that make up the objects.

The problem with this method is that it does not preserve fine details of the object candidate. It also does not repair just the hole but it creates a water tight mesh around the cloud which can obscure the initial shape depending on the state of the object candidate passed in.

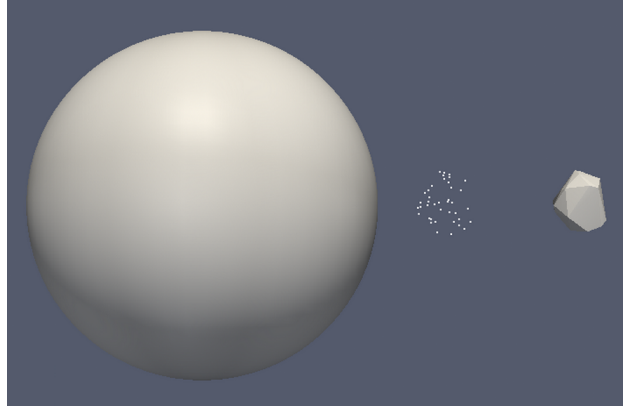


Figure 5.2: Sphere being wrapped around example points [2].

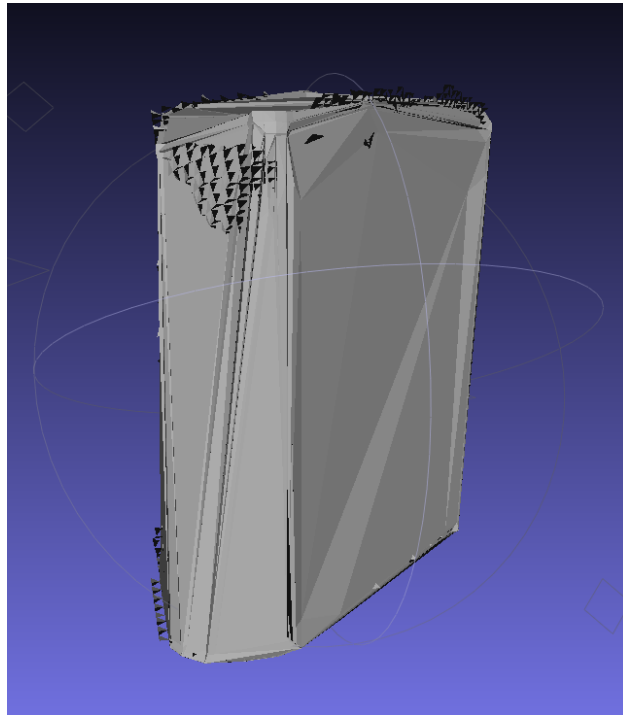


Figure 5.3: Typical Shrink Wrapping Results.

5.2.2 VTK HoleFillerFilter

The VTK library provides a Hole Filling filter [36]. This filter takes the output from the hole detecting filter and will apply the AFM method discussed in Section 2.2. As discussed in Section 5.1.2 the results from the previous processing steps do not provide a solid mesh with regular or smooth surfaces. This method was found to work well and repair holes that were found in meshes that were in the test data set. When applied to the meshes that this project produced the filter was unable to function.

A specific reasoning for this was not able to be directly determined. This was due to not being able to fully investigate the VTK filter. The working assumption was that, much like the Hole Detection Filter (Section 5.1.2), this filter requires a regular mesh. As such it is not able to work with the hole that it should obtain from the Occlusion Detection step. Without this information it is unable to close a hole which cannot be found.

There were a few occasions where a hole was detected and patched but when this happened the irregularity of the surround information caused for the “repaired” hole to be filled with “garbage” information which only made the mesh look more irregular. See Figure 5.4 for an example.

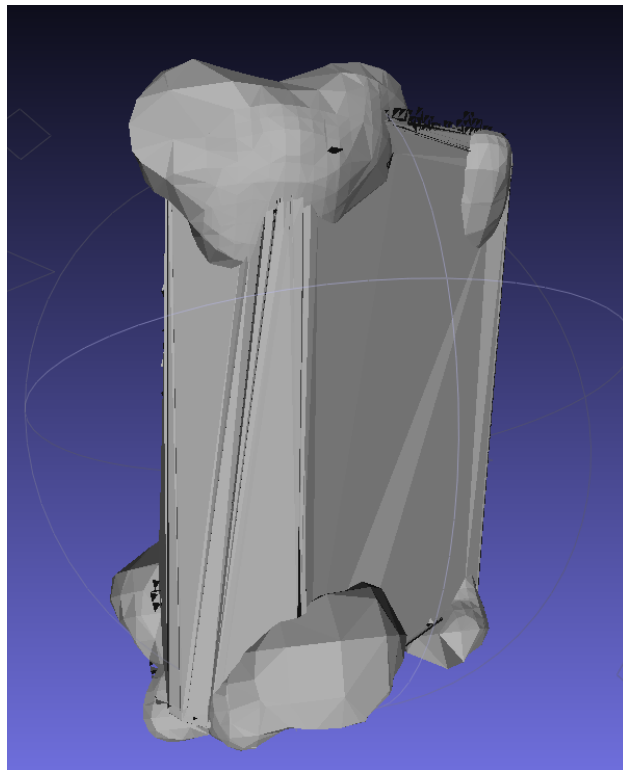


Figure 5.4: Mesh with incorrectly repaired occlusions.

5.3 Surface Smoothing

The final processes step when it comes to the occlusion repair is surface smoothing. It was designed to take the infilled patch and to smooth it and to better match the surface around the outside of the newly patched portion of the object candidate. This method, much like in the two previous sections, works well when applied to test data sets. It again failed when working with the meshes that were output by the occlusion repair system.

Chapter 6

Results, Conclusions & Future Work

In the previous chapter the methods and output obtained from each step in the process were presented. This chapter discusses the output in more detail as well as discussing results that could have been obtained.

6.1 Overall Results

The goal of this project was to investigate an occlusion repair system for incoming RGB-D information. As the project progressed and as can be seen in Chapter 2, the methods to perform this task are not yet present with the current generation of RGB-D sensors. This project based its approach on the mesh based methods presented in Sections 5.1.2, 5.2.1.

The end result of this project was that using currently available sensors and libraries, reliable results were *not* able to be obtained from the tested methods.

6.1.1 Object Candidate Creation

Object candidate creation creates the most difficulties. These methods did function but that they did not provide precise results. The first problem began with the sensors being used. The Microsoft Kinect, comes in its accuracy that does not allow for smooth surfaces in the first place. While all of the point cloud processing methods work well with this data. Faults can be seen once the point cloud information is converted to the mesh data structure that is where the faults with the point clouds can be seen.

6.1.1.1 RoboCup Constraints

One of the major contributors to the issues that the project faced. The use case for this project called for the software being usable for RoboCup competitions. This means that it would need to be processed fast enough that it would allow for the system to be able to use the results, while still being able to accomplish the rest of the task specification. As presented in Section 3.2 this means that the system would need to run between 3 – 5 seconds. This allows the system enough time to perform the rest of its tasks and to still remain competitive time wise.

The problem this presented is that the normal PCL filters can take a few seconds to run on a full sized point cloud (370,000 points for the Kinect). To compensate for this the point cloud is down-sampled in order to allow for faster processing later on. This lowers the number of points and generalizes their positions for later processing. As mentioned in Chapter 2 there are promising methods for working with point clouds but they require both very dense and accurate point clouds.

6.1.2 Occlusion Detection

The primary problem with detecting occlusions comes in that they are rarely surrounded on all sides by information. In the context of the project they are often edges or chunks of the object which become missing. This caused an unsurmountable amount of trouble for this project. While “holes” or patches of missing information in the object candidates are able to be identified. No reliable methods for determining if an edge was a true edge of the object or an occlusion where found. The identification of missing information inside the object was able to be obtained only if the holes were large enough for them to show up when the mesh was created.

6.1.2.1 NaN Clustering & Repair

Though a discussion with members working directly on the PCL library a possible solution to this problem was conceived but to late in the project to

be implemented and tested. The following is an outline of a new possible solution to the occlusion identification problem.

The PCL point cloud is a data structure that contains both position information $\langle x, y, z \rangle$ and color $\langle r, g, b \rangle$ information about the point which is being stored at the $\langle x, y, z \rangle$ coordinate. When there is nothing (missing information) but an RGB value, the PCL library refers to “Not A Number (NaN)” values. These values are still part of the point cloud but are values where there is nothing but RGB information.

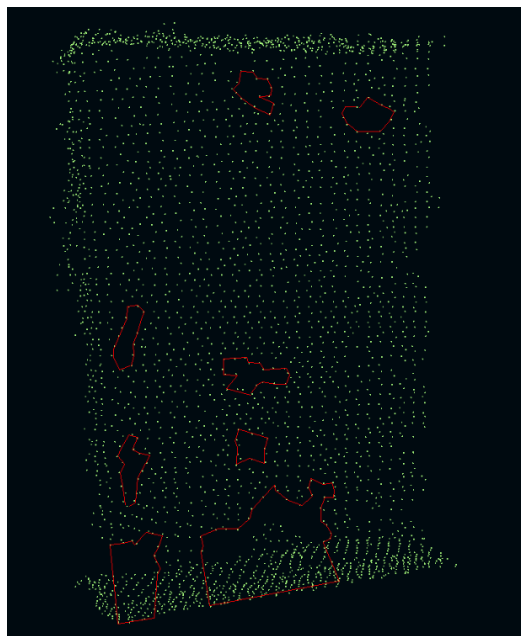


Figure 6.1: Typical object candidate data capture with desired areas for NaN Clustering highlighted in Red.

This provides the possibility to perform clustering based on NaN values. Once the clusters are found, they could use region growing to slowly close

the boundaries of the hole. This would allow for the repair of holes that very commonly show up in point cloud data. An example image of the type of holes and the regions that could be closed is provided in Figure 6.1.

The final step would be a smoothing process which would take the newly repaired surface and attempt to smooth the surface of the object to provide more reliable results for object identification. This would also be a desired filter or processing step in the PCL library as was discovered in the later portion this research project.

In order to ensure that a process such as this would be reliable the sensor would require similar sized RGB and depth sensors. This would ensure that each point in the depth image is mapped to only one point in the RGB image making this processing more reliable.

6.2 Occlusion Repair

Due to the obtained results when converting from point clouds to a mesh, data structure this section was unable to be thoroughly tested. The previous step (occlusion detection) is crucial so that the system knows what it needs to repair. A new method for performing this task was presented in Section 6.1.2.1, which could allow for the robust detection of occluded portions present in the objects. From that point the porting of either the high fidelity

methods or possibly using region growing segmentation to fill the holes.

6.3 Conclusions

The initial starting point of the project, investigated the use of a Growing Neural Gas Library[37]. The state of the library and how integrated it was to other software, made it not feasible for use with this project. The amount of time this investigation took limited the amount of time that remained to complete the project.

The project was unable to focus on the implementation of new methods but rather the integration of existing methods due to time constraints. As noted in Section 5.1 there are no currently available methods for working with point clouds. This limited the scope of the work to investigating using mesh based methodologies.

Most of the research performed in this area was funded by companies like Siemens Co or the Chinese Government. When contacting the authors for possible implementation details or available binaries, the project was informed that they were held as protected material and could not be shared. As such this limited the project to using the freely available libraries such as the VTK library [38].

As discussed in Section 6.2 there is no reliable way to repair the occlusions that were found during earlier processing steps. The objects that were returned bore little resemblance to the input objects. They also tended to ob-

scure any fine details that classifiers normally use in order to identify objects.

The created test system was able to function from the beginning to the end of the pipeline taken in raw data from a Microsoft Kinect and processing it to remove any occlusions that where present. It was not able to do this in a reliable manner which output objects that more closely resembled the object than the initial point cloud. It did however introduce a variety of methods that could be used to solve this problem and a very intriguing future topics that follow from this research.

6.4 Future Work

This section presents two possible future directions for this work. One examines taking existing methods for more complex and reliable sensors and porting it the more widely available RGB-D sensors. The other direction, looks at creating a new processing filter within PCL that leverages the merging of depth and color information to identify and repair occluded holes in data.

6.4.1 Porting High Fidelity Methods

This project, while showing that currently existing libraries are unable to perform the desired task showed a very interesting and in demand problem: The smoothing and repair of low fidelity point cloud data sets.

As discussed in Section 2.1 there are methods, that can perform the task of occlusion repair but they have some caveats. Normally they operate on full point clouds and those point clouds are made using sensors such as the Velodyne High Definition LiDAR [39], creating very dense point clouds (1.3 million points vs 370 thousand for the Kinect). They also have a much higher accuracy rate of $1.3cm$ under perfect conditions [40] when used with a range up to 25m). This increased amount of data and accuracy allows for better mapping of the environment and more data to work with. These conditions are required for the methods in the previous sections to work properly.

This leaves the question of how do you port these model to low accuracy and low density point clouds such as those obtained from the current generation of RGB-D sensors.

6.4.1.1 NaN Clustering & Region Growing

In Section 6.1.2.1 the concept for a new PCL filter is presented. This could be combined with furthering the PCL 3D Edge Detection functions to estimate possibly occluded edges. Combining these two methods it may be possible to provide a solution to the occlusion problem using the current generation of RGB-D sensors.

Bibliography

- [1] PCL Developers Blog. [Online]. Available: <http://www.pointclouds.org/blog/gsoc12/cchoi/index.php>
- [2] Kitware. ConvexHullShrinkWrap. [Online]. Available: <http://www.vtk.org/Wiki/VTK/Examples/Cxx/PolyData/ConvexHullShrinkWrap>
- [3] M. Research. (2012, May) Microsoft Kinect. [Online]. Available: <http://msdn.microsoft.com/en-us/library/hh855347.aspx>
- [4] ASUS. ASUS Xtion Pro. [Online]. Available: http://www.asus.com/Multimedia/Motion_Sensor/Xtion_PRO/
- [5] Primesense. Primesense 3D Sensors. [Online]. Available: <http://www.primesense.com/wp-content/uploads/2012/12/PrimeSenses.3DsensorsWeb.pdf>
- [6] Merriam-Webster. Occluded. [Online]. Available: <http://www.merriam-webster.com/dictionary/occluded>
- [7] X. Zhou, B. He, and Y. F. Li, "A Novel View Planning Method for Automatic Reconstruction of Unknown 3-D Objects Based on the Limit Visual Surface," in *2009 International Conference on Image and Graphics (ICIG)*. IEEE, Jan. 2009, pp. 301–306.

- [8] B. He, X. Zhou, and Y. F. Li, "A View Planning Method for Automatic 3D Modeling based on the Trend Surface and Limit Region," in *Instrumentation and Measurement Technology Conference, 2009. I2MTC '09. IEEE*, 2009.
- [9] W. R. Scott, G. Roth, and J.-F. Rivest, "View planning for automated three-dimensional object reconstruction and inspection," *Computing Surveys (CSUR)*, vol. 35, no. 1, Mar. 2003.
- [10] M. Bertalmio, G. Sapiro, V. Caselles, and C. Ballester, "Image inpainting," in *SIGGRAPH '00: Proceedings of the 27th annual conference on Computer graphics and interactive techniques*. ACM Press/Addison-Wesley Publishing Co. Request Permissions, Jul. 2000.
- [11] A. Sharf, M. Alexa, and D. Cohen-Or, "Context-based surface completion," *SIGGRAPH '04: SIGGRAPH 2004 Papers*, Aug. 2004.
- [12] D. Doria and R. J. Radke, "Filling large holes in LiDAR data by inpainting depth gradients," in *2012 IEEE Computer Society Conference on Computer Vision and Pattern Recognition Workshops (CVPR Workshops)*. IEEE, pp. 65–72.
- [13] P. Liepa, "Filling holes in meshes," in *SGP '03: Proceedings of the 2003 Eurographics/ACM SIGGRAPH symposium on Geometry processing*. Eurographics Association, Jun. 2003.
- [14] M. Bern and D. Eppstein, "Mesh generation and optimal triangulation," *Computing in Euclidean geometry*, 1992.
- [15] R. Schneider and L. Kobbelt, "Mesh fairing based on an intrinsic PDE approach," *Computer-Aided Design*, vol. 33, no. 11, pp. 767–777, Sep. 2001.
- [16] W. Zhao, S. Gao, and H. Lin, "A Robust Hole-Filling Algorithm for Triangular Mesh," in *Computer-Aided Design and Computer Graphics, 2007 10th IEEE International Conference on*, 2007, p. 22.

- [17] P. L. George and E. Seveno, “The advancing front mesh generation method revisited,” . . . *Journal for Numerical Methods in . . .*, 1994.
- [18] T. Ju, “Fixing geometric errors on polygonal models: a survey,” *Journal of Computer Science and Technology*, 2009.
- [19] BRSU University. b-it-bots team website. [Online]. Available: <http://b-it-bots.de/Home.html>
- [20] Fraunhofer. Care-O-bot 3. [Online]. Available: <http://www.care-o-bot.de/en/care-o-bot-3.html>
- [21] Locomotec. KUKA youBot. [Online]. Available: <http://youbot-store.com/>
- [22] Open Source Robotics Foundation, “RoboCup@Home Rules & Regulations,” pp. 1–67, Sep. 2013.
- [23] roboCup. RoboCup@Work. [Online]. Available: <http://www.robocupatwork.org/>
- [24] S. Albrecht and S. Marsland, “Seeing the Unseen: Simple Reconstruction of Transparent Objects from Point Cloud Data,” *workshops.acin.tuwien.ac.at*.
- [25] PCL. **Iterative Closest Point**. [Online]. Available: http://docs.pointclouds.org/trunk/classpcl_1_1_iterative_closest_point.html
- [26] ——. Downsampling a PointCloud using a VoxelGrid Filter. [Online]. Available: http://pointclouds.org/documentation/tutorials/voxel_grid.php#voxelgrid
- [27] ROS. (2011, Jun.) Kinect Accuracy. [Online]. Available: http://www.ros.org/wiki/openni_kinect/kinect_accuracy
- [28] PCL. Removing outliers using a StatisticalOutlierRemoval filter. [Online]. Available: http://pointclouds.org/documentation/tutorials/statistical_outlier.php#statistical-outlier-removal

- [29] M. Fischler and R. C. Bolles, “Random Sample Consensus, A Paradigm for Model Fitting with Applications to Image Analysis and Automated Cartography,” *Communications of the ACM*, 1981.
- [30] PCL. Euclidean Cluster Extraction. [Online]. Available: http://pointclouds.org/documentation/tutorials/cluster_extraction.php#cluster-extraction
- [31] ——. Smoothing and normal estimation based on polynomial reconstruction. [Online]. Available: <http://pointclouds.org/documentation/tutorials/resampling.php#moving-least-squares>
- [32] P. Lancaster and K. Salkauskas, “Surfaces generated by moving least squares methods,” *Mathematics of computation*, 1981.
- [33] R. Li, L. Liu, L. Phan, S. Abeysinghe, C. Grimm, and T. Ju, “Polygonizing extremal surfaces with manifold guarantees,” in *SPM '10: Proceedings of the 14th ACM Symposium on Solid and Physical Modeling*. ACM Request Permissions, Sep. 2010.
- [34] Kitware. vtkFeatureEdges Class Reference. [Online]. Available: <http://www.vtk.org/doc/nightly/html/classvtkFeatureEdges.html>
- [35] M. Levoy. The Stanford 3D Scanning Repository. [Online]. Available: <http://graphics.stanford.edu/data/3Dscanrep/>
- [36] Kitware. vtkFillHolesFilter Class Reference. [Online]. Available: <http://www.vtk.org/doc/nightly/html/classvtkFillHolesFilter.html>
- [37] C. A. Mueller, N. Hochgeschwender, and P. Ploger. (2011, Jan.) Surface Reconstruction with Growing Neural Gas. [Online]. Available: http://www.b-it-bots.de/Publications_files/SurfaceReconstructionWithGrowingNeuralGas.pdf
- [38] Kitware, “VTK - The Visualization Toolkit,” *vtk.org*.
- [39] Velodyne Lidar. [Online]. Available: <http://velodynelidar.com/lidar/lidar.aspx>

- [40] C. Glennie and D. D. Lichti, “Static Calibration and Analysis of the Velodyne HDL-64E S2 for High Accuracy Mobile Scanning,” *Remote Sensing*, vol. 2, no. 6, pp. 1610–1624, Jun. 2010.