UNIVERSITÀ DI PARMA

Dottorato di Ricerca in Tecnologie dell'Informazione

XXX Ciclo

Advances in Robot Attention for Next-Best View Planning

Coordinatore:

Chiar.mo Prof. Marco Locatelli

Tutor:

Chiar.mo Prof. Jacopo Aleotti

Dottorando: *Riccardo Monica*

Anni 2014/2017

Abstract

This dissertation presents advancements in next-best view algorithms applied to 3D reconstruction. 3D reconstruction has been a topic of great interest in recent years, due to the diffusion of cheap depth sensors such as the Microsoft Kinect. Algorithms such as KinectFusion have been developed to merge multiple views from these sensors. The experimental setup used in this thesis involves a Kinect sensor mounted on a robot arm. The robot can move the sensor in order to reconstruct the objects in a tabletop scenario. Next-best view algorithms compute the optimal sensor placement to observe the unknown space. Such algorithms are computationally expensive, since they need to simulate the sensor from each candidate pose. In this thesis, the concept of attention is applied to next-best view, in order to reduce the computation time and concentrate robot exploration towards a goal. According to attention, a vision system assigns different importance to various parts of a scene. Resources can then be focused on the most important parts of the scene. In the first approach presented in this dissertation, attention is driven by the user, who moves objects in the tabletop scenario. Then, the robot updates the 3D representation, by focusing views in the areas where changes have happened. In the second approach, the robot is attracted by the objects already present in the scene, as they are discovered during exploration. A segmentation algorithm is used in order to partition the scene. A saliency value is assigned to each segment, and the most salient one is selected. In both approaches, experiments were carried out in order to highlight the reduced computation time and the goal-oriented behavior of the robot. Further work in this thesis proposes advancements towards the use of the ElasticFusion reconstruction algorithm. Unlike KinectFusion, ElasticFusion operates on a surfel-based 3D representation. Surfels are designed for computer graphics, and they are processed faster on GPU than KinectFusion volumetric representation. A color and position enhancement filter is proposed, to be run alongside ElasticFusion 3D reconstruction, in order to obtain a better segmentation.

Table of Contents

Introduction **1**

1 State of the Art **7**

1.1 Next best view planning 7

1.2 Active robot exploration 10

1.3 Robot attention 12

1.4 Human task segmentation 14

1.5 RGB-D enhancement 16

1.6 Point cloud segmentation 17

1.7 Annotation interfaces 18

2 User-driven spatial attention **21**

2.1 Method overview 23

2.2 Robot attention approach 25

2.2.1 The KinectFusion algorithm 27

2.2.2 KinFu Large Scale 28

2.2.3 Next-best view approach 31

2.2.4 Determination of regions of interest 34

2.3 Software architecture 39

2.3.1 Extension of KinFu Large Scale for robot attention 39

2.3.2 Extension of KinFu Large Scale for next-best view 41

2.3.3 System integration in ROS 43

2.4 Results . . . 45
2.4.1 Evaluation of Kinect sensor tracking solutions . . . 45
2.4.2 GMM-AS algorithm evaluation . . . 46
2.4.3 Evaluation of the robot attention system . . . 49
2.4.4 Evaluation of the developed KinFu LS extensions . . . 56
2.5 Discussion . . . 58

3 Object-driven spatial attention 61
3.1 Proposed method for next-best view planning . . . 62
3.1.1 Contour extraction from TSDF volume . . . 65
3.1.2 Saliency of point cloud segments . . . 70
3.1.3 Kinect V2 depth image pre-processing . . . 74
3.2 Experimental evaluation . . . 79
3.2.1 Robot setup and experimental procedure . . . 79
3.2.2 Experiments . . . 80
3.2.3 Evaluation of depth image pre-processing . . . 88
3.3 Discussion . . . 91

4 Extension towards a surfel-based next-best view planner 93
4.1 Surfel segmentation enhancement . . . 95
4.1.1 Method . . . 96
4.1.2 Data processing pipeline . . . 102
4.1.3 Results . . . 104
4.1.4 Discussion . . . 110
4.2 Point cloud annotation tool . . . 112
4.2.1 Method . . . 114
4.2.2 User interface . . . 120
4.2.3 Results . . . 123
4.2.4 Discussion . . . 129

5 Conclusions 131

Bibliography 135

Acknowledgments 149

List of Figures

1 The robot system. 2

1.1 The reconstruction system used by Kriegel. A laser scanner, an ASUS Xtion depth camera and a stereo camera pair were mounted on the robot arm. On the right, Kriegel's boundary-based view generation method computed the average normal in the red area. Image source: [1]. 9

1.2 The segmentation-based exploration system by Xu et al. Image source: [2]. 11

1.3 The saliency method by Schauerte et al. From left to right: the scene, the image-based saliency map (bottom-up saliency) and the gesture-based saliency map (top-down saliency). Images source: [3] © 2010 IEEE. 13

1.4 Trajectories (left) and the corresponding GMM (right), as obtained by Sang Hyoung Lee et al. Image source: [4] © 2012 IEEE. 15

1.5 RGB image (left), noisy depth image (center) and enhanced depth image using temporal mode filter (right). Image source: [5] © 2012 IEEE. 16

1.6 Segmentation examples using Supervoxel-LCCP. Image source: [6] © 2014 IEEE. 18

1.7 Point cloud annotation process (left to right) using min-cut algorithm and constraints. Green dots are placed to define the foreground, red dots to define the background. Image source: [7] © 2009 IEEE. . . . 20

2.1 Experimental setup (left). Example of KinectFusion output (top right). Motion planning environment (bottom right). 22
2.2 The general flowchart of the system. 24
2.3 An example of 2D TSDF volume. The red line represents the object surface. The yellow area represents the negative values inside the object, while the cyan area the positive ones. 29
2.4 A 2D representation of the KinFu LS reference frames. Unit vectors for $\{O\}$, $\{E\}$ and $\{H\}$ are shorter, since they use expanded (scaled) coordinates. 30
2.5 The sphere sampled by NBV algorithm around the POI with radius d. 32
2.6 The trajectory recorder and *POI* detection as functional blocks. . . . 36
2.7 An example trajectory (left image) obtained from the placing task of an object in point A. The superimposed GMM (right image). Only the blue "thick" Gaussian correctly determines a *POI*. 37
2.8 An example trajectory (left image) obtained from the task of picking and placing an object from A to B. The superimposed GMM (right image). Only the blue and the yellow "thick" Gaussians correctly determine two *POI*s. 38
2.9 Relations among the most important classes of KinFu LS. 40
2.10 The multithread shifting procedure. 43
2.11 The two environments used for testing KinFu tracking accuracy. . . 46
2.12 KinFu egomotion tracking error in the first scenario of Figure 2.11. . 47
2.13 A trajectory representing an object placement task to location A (left) and the superimposed GMM (right). 50
2.14 A trajectory representing a pick-and-place task from location A to B (left) and the superimposed GMM (right). 50
2.15 The environment before (left), during (center) and after (right) the user task. First experiment (top row) and second experiment (bottom row). 51

2.16 The environment 3D representation with user hand trajectory (left), GMM (center) and generated spherical regions of uncertainty (right) for the first (top row) and second experiment (bottom row) in black color. 53
2.17 KFOVP snapshots in experiment 1. From left to right: OpenRAVE planning, robot NBV configuration, KinFu synthetic depth map and reconstruction in the ternary representation (unknown voxels are displayed in black), for each view pose (top to bottom). 54
2.18 KFOVP snapshots in experiment 2. Robot NBV configuration (top row), KinFu synthetic depth map (middle row) and the reconstruction in the ternary representation (bottom row, unknown voxels are displayed in black) for each view pose (left to right). 54
2.19 Final environment representation for KFOM and KFOVP in the two experiments. 55
2.20 The number of unknown voxels inside each uncertainty region, after user task (U) and at each robot NBV iteration. 56
2.21 Environment (left), ternary representation (center) and POI (right), for the experiment described in section 2.4.4. 57

3.1 Pipeline of the view planning algorithm. The grey background highlights the intermediate phase 63
3.2 Flowchart of viewpoint generation (line 2 in Algorithm 4). Each candidate view direction generates 40 view poses for the sensor. 65
3.3 A simplified 2D example of the contour extraction algorithm using Von Neumann neighborhood (4-connected) and Moore neighborhood (8-connected). In the previous view the sensor observed the object from the right side. A computed contour cell is marked with the cross. The thicker square highlights the Moore neighborhood of the contour cell. The green segment represents a frontier. Known and occupied cells are displayed in red, known and empty cells are in white, unknown cells are in dark grey. 67

3.4 Generation of the next potential viewpoint for (a) a rounded object (b) an object with sharp edges. Viewpoints B (computed from the surface normal) and C (computed from the frontier normal) are very similar in case (a), but not in case (b). Only in case (b) viewpoint C allows the observation of the unknown volume behind the sharp edge. 68

3.5 Left: a jug observed from the current sensor viewpoint. Center: the 3D mesh reconstructed by KinFu. Right: the volumetric representation (rotated view), with occupied (white) and unknown (black) voxels. 71

3.6 Left: contour voxels (black) and the contour points (red). Right: contour points with normals. A contour point represent a group of similar contour voxels. 71

3.7 Proposed procedure for point cloud segmentation and computation of the saliency value of each segment. 71

3.8 Example of point cloud segmentation and saliency evaluation. Brighter segments have higher saliency value. (1): a picture of the scenario, (2): saliency evaluated by segment roundness alone, (3): saliency evaluated by segment isolation alone, (4): saliency evaluated by both segment roundness and isolation according to equation 3.12. The segment isolation factor reduces the saliency of the noisy segments inside the red ellipse. 73

3.9 Saliency computed after the initial scan in experiment 2 described in Section 3.2.2, using $B_{th} \in \{0.005, 0.01, 0.02, 0.05, 0.1(m)\}$ (from left to right). 73

3.10 Top left: a scene as seen from the sensor. Top right: the image from the depth camera. Lower left: the point cloud acquired by the sensor, filtered by the Freenect2 driver. Lower right: the point cloud filtered by the proposed method: both shadow points and veil points are correctly removed. 75

3.11 The Kinect V2 sensor with IR camera, RGB camera and IR emitters. 75

3.12 The Kinect V2 (on the left) observes a scene composed by an object (in the center) and a background plane (on the right). The object partially occludes the background plane. Three kinds of occlusions are possible: camera only (yellow), IR emitter only (blue), both (red). . 76

3.13 Illustration of the horizontal angles α and β of the camera and IR emitter with respect to the observed points. 77

3.14 The experimental setup (left). Motion planning environment based on Moveit! (top right). Screenshot of KinFu output during the initial scan phase (bottom right). 79

3.15 The experimental scenarios used for the evaluation. 80

3.16 Candidate viewpoints (represented by arrows) for the proposed approach (experiment 1, third NBV). Left: candidate viewpoints of all segments. Right: candidate viewpoints for the most salient segment only. 82

3.17 The graphs show the number of unknown voxels near the objects in the scene for the first five next-best views. 85

3.18 Images of experiment 1 using the proposed method (left to right). Top: saliency map of point cloud segments; middle: 3D volumetric representation; bottom: planned robot next-best views. 86

3.19 Images of experiment 1 using the standard NBV approach. Top: 3D volumetric representation; bottom: planned robot next-best views. . 86

3.20 Images of experiment 3 using the proposed method (left to right). Top: saliency map of point cloud segments; middle: 3D volumetric representation; bottom: planned robot next-best views. 87

3.21 Images of experiment 3 using the standard NBV approach. Top: 3D volumetric representation; bottom: planned robot next-best views. . 87

3.22 3D volumetric representation of the environment in the four experiments after five next-best views: proposed method (top), standard approach (bottom). 88

3.23 From left to right: the scenario used for testing the proposed depth image pre-processing filter, image preprocessed by the Freenect2 bilateral filter only, image preprocessed by the Freenect2 bilateral and edge-aware filters, image preprocessed by the Freenect2 bilateral filter and the proposed filter. The image is displayed in color although the algorithm operates on the depth map only. Outliers points are displayed in red. 89

3.24 The simulated environment, with object size 4 cm (left) and 16 cm (right). White: area illuminated by the emitter only. Grey: area illuminated and properly acquired by the camera. Red: shadow visible by the camera. The vertical blue band in the right image is a region of space that is neither illuminated by the emitter nor observed by the camera. 90

4.1 The reconstruction with mean color (top left) is more blurred than the output of the proposed filter (top right). As such, segmentation with the proposed filter (bottom right) is better than without using it (bottom left). 96

4.2 Flowchart of the method for surfel segmentation enhancement. . . . 99

4.3 The experimental pipeline. 102

4.4 The five scenarios (left) and their ground truth segmentation (right). 106

4.5 Close-up of scenario 1, reconstruction and segmentation using Flood Fill. With the enhanced method, the segmentation of the small tools on the panels is more complete. 107

4.6 Close-up of scenario 2, reconstruction and segmentation using Flood Fill. In the enhanced case, a lower number of single-surfel segments are produced along image borders, since the color gradient is sharper. 107

4.7 Close-up of scenario 4, reconstruction and segmentation using SV-LCCP. The difference between segmentations is less evident. 107

4.8 Close-up of scenario 5, reconstruction and segmentation using Flood Fill. In the enhanced case, the segmentation of the two magazines is complete, and details (e.g. the heading) have been properly detected. 108
4.9 Detail of scenario 1, with position not enhanced (left) and enhanced (right). 108
4.10 Precision/Recall curves for Flood Fill, for the five scenarios. 109
4.11 Precision/Recall curves for Supervoxel-LCCP, for the five scenarios. 109
4.12 The RViz-based interface of the annotation tool. 113
4.13 Annotation example on the neighborhood graph. The virtual point is displayed in light gray and it is connected to all the control points (dashed arcs). 114
4.14 Example of point cloud labeling from control point selection and computation of the shortest-path tree. Colors correspond to global labels L_i. 116
4.15 Point cloud viewer (left and middle). Labels only (right image). . . . 120
4.16 Rectangular selection: before selection (left), during selection (center), after selection (right). During selection, a yellow rectangle is displayed on the area being selected. 121
4.17 The annotation panel. 122
4.18 Two test scenarios (right and left column) for the user study. Point cloud (first row), ground truth annotations for Control Points selection (second row) and Rectangular Selection (third row). 123
4.19 Top row: the user first places green control points on the background (middle image) and then red points on the object (right image). Bottom row: the user first places red control points (middle image) on the object and then green control points on the background (right image). The resulting annotations are equivalent. 124
4.20 Stray points (highlighted by circles) behind the small horse object (top right image) and on the white box (bottom right image). 127
4.21 A cork jug seen from the inside (left). RS annotation (middle) and CP annotation (right). Missing (black) points appear in the RS case. 128

List of Tables

2.1 3D object reconstruction errors. 46

2.2 Evaluation of GMM-AS and ZVC-like ($\Lambda = 2$, $POITh = 2.5$, $B = 4$, $ZVCTh = 1.72$). 47

2.3 Evaluation of GMM-AS and ZVC-like at different speed multipliers, on the whole dataset. 48

2.4 Evaluation of GMM-AS and ZVC-like at various noise standard deviation values. 48

2.5 The saliency computed for each Gaussian in Fig. 2.13. 49

2.6 Saliency computed for each Gaussian in Fig. 2.14. 49

2.7 Reconstruction error, completeness, number of views and execution time for KFOM and KFOVP. 52

2.8 Time (seconds) of the most relevant algorithm phases, averaged for each experiment. 52

2.9 Execution times for one phase of NBV-GPU, NBV-CPU and NBV-CPU-step (times in seconds). 52

2.10 Comparison of overhead in standard KinFu LS and NBV-GPU (times in ms). 57

3.1 Average total time (seconds) and standard deviation over the four experiments for each phase 81

3.2 Saliency values and number of view poses for the point cloud segments in Fig. 3.16 (in descending order of saliency) up to the first segment not belonging to the objects (part of the supporting table) . 83
3.3 Marks showing NBVs pointing towards the objects (✓) or not (×), for all the experiments 84
3.4 Number of measurements and false measurements produced by each algorithm. 89
3.5 False discovery rate of shadow points 90

4.1 Dataset statistics 105
4.2 Fixed Parameters 110
4.3 Minimum Bidirectional Consistency Error 111
4.4 Parameters used in the experiments. 122
4.5 Average task completion time, number of Undo operations per minute, and number of annotation errors for scenario 1. 125
4.6 Average task completion time, number of Undo operations per minute, and number of annotation errors for scenario 2. 126
4.7 Questionnaire results, showing subjective evaluation by the users. . 128
4.8 Execution times. 129

Introduction

Overview

This thesis presents advancements on the topic of next-best view (*NBV*) algorithms, applied to 3D reconstruction using a robot arm. In particular, the thesis proposes the application of the spatial attention concept to NBV. An attention-based system focuses on interesting regions of the environment. Advantages include faster computation and accurate exploration of the interesting regions, to the detriment of others.

3D reconstruction has been a topic of great interest in recent years after the diffusion on the market of cheap RGB-D sensors, like the Microsoft Kinect. RGB-D sensors produce color and depth images, i.e. images in which each pixel represents the distance of the point from the sensor. These sensors have a high framerate and are able to acquire a large amount of data. However, they have low accuracy and limited field of view. Algorithms have been developed to merge multiple views of these sensors into a single 3D reconstruction.

The first well-known solution has been the KinectFusion algorithm [8], which merges the views in a volumetric representation. Other algorithms, like ElasticFusion [9], operate on a surfel-based representation. These algorithms are designed to perform 3D reconstruction while a user moves the sensor around in the environment. Such 3D reconstruction algorithms are able to estimate the sensor motion from the sensor data itself, without the need for an external tracking source.

These algorithms may be decomposed into two broad phases, which are run iteratively for each sensor frame. In the *tracking* phase, the current data acquired by the

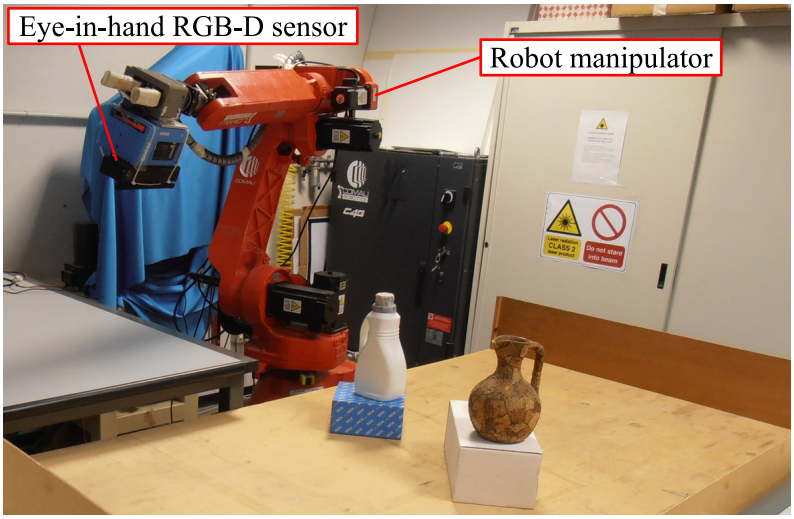


Figure 1: The robot system.

sensor is compared with the (partial) 3D reconstruction from the previous iteration, and the sensor pose is estimated. In the *integration* phase, the new data is merged with the 3D reconstruction. The tracking phase is often simplified by the hypothesis that the sensor did not move excessively since the last acquisition. This assumption only holds if enough frames are processed per time unit, so real-time constraints are often present.

A 3D representation updated in real-time may be useful for robot planning and navigation. When the sensor is mounted on the robot itself, the robot can perform exploration and mapping tasks. As the robot moves around the environment, information is accumulated in the 3D representation. The experimental setup used in most parts of this thesis involves a robot manipulator (Fig. 1) with a sensor in eye-in-hand configuration. The robot is a Comau SMART SiX manipulator, with six degrees of freedom. The robot has a Kinect sensor mounted on the wrist, and is able to move the sensor around the environment. Objects to be reconstructed are placed on the table in front of the robot.

The movement performed by the robot while exploring and reconstructing the en-

vironment may be optimized, in order to maximize the acquired information. The optimization is performed online, exploiting the previously acquired information. This problem pertains to the domain of active perception. Active perception problems appears whenever a robot is able to actively move one of its sensors.

A common approach to solve active exploration and 3D reconstruction problems is the use of a next-best view algorithm. A NBV algorithm estimates the "best" configuration in which a sensor should be placed to observe the environment, according to some metric and/or heuristic. NBV algorithms operate on a partially-unknown environment representation. In other words, the result is the best solution as far as it is currently known. When the sensor is moved in the selected configuration, the knowledge of the environment is updated, and the following NBV is computed taking into account the new information.

The result of a NBV iteration depends on the previously acquired data, which in turn depends on all previous NBV iterations. Therefore, a complete NBV application necessarily depends on the actual state of the real environment (or a simulation), which makes the task challenging. Moreover, sensor placement may be slow or expensive, and computation of the NBV must be executed online while the robot is in a ready state, waiting for the next pose.

NBV approaches are usually composed of two phases. In the *view generation* phase, a discrete set of candidate viewpoints are generated. In the *viewpoint evaluation* phase, viewpoints are evaluated and the most promising one is selected. In traditional NBV approaches [10], viewpoints are generated by sampling configurations constrained on a sphere centered around the object. Then, a score is computed for every viewpoint, in order to find the best candidate. Both phases may benefit from a heuristic which is able to reduce the number of viewpoints generated or evaluated. In this thesis, attention has been investigated to this purpose.

Attention is a concept originated in psychology, by observing the way in which humans (and other animals) are able to focus on specific input stimuli at the expense of others. A common explanation for this behavior is the limited availability of sensing and processing resources. The human eye is able to see with maximum acuity only in a very small part of the retina, the fovea, only about 5° wide. As such, a hu-

man being can only perceive a very small region of space with maximum accuracy. The human brain behaves similarly: processing resources can be allocated (voluntarily or not) on specific parts of the environemt.

In computer vision, attention is commonly simulated by assigning a numeric value (called *saliency*) to parts of the scene. The saliency value is computed by a heuristic which assigns high values to highly-informative regions, i.e. parts where more detail is available. Image processing can then focus only on those highly-informative regions. Often, the attentional behavior leads to a reinforcing loop, where new information is discovered in the region, thus further increasing its saliency.

Contributions

The goal of this thesis is the application of spatial attention to enhance NBV algorithms. Spatial attention is applied to 3D volumes, for which a saliency value is computed. Such concept of spatial attention lends itself to the enhancement of NBV approaches. A NBV algorithm exploits the attention system to generate viewpoints oriented towards the most salient regions. Moreover, viewpoint evaluation time is reduced by excluding viewpoints without enough saliency.

Two approaches which exploit attention for Next Best View have been proposed in this thesis. In the first approach (Chapter 2) the attention of the robot is attracted by the user actions. The environment (i.e. the table top, with objects) is initially completely known. The user performs actions which may cause changes, e.g. the movement of an object. A NBV cycle is started to explore the regions where changes are likely to have occurred, in order to restore an updated representation of the environment.

For this approach, two novel methods have been developed. Firstly, a user hand trajectory analysis approach has been developed to compute the salient regions of space. Trajectory analysis is performed by fitting a Gaussian Mixture Model (*GMM*) on the trajectory points. The GMM reduces the trajectory into a set of Gaussians. Regions where the user performed manipulation actions are represented by round and thick Gaussians in the model. Conversely, less relevant regions such as the movement

between manipulation actions are represented by long and thin Gaussians. Based on this observation, heuristics are applied to compute a spatial saliency value from the model. Secondly, extensive modifications have been applied to KinectFusion in order to integrate next-best view into it. KinectFusion performs ray casting as part of its tracking phase, in order to project the current 3D representation and compare it with the current sensor input. The same ray casting procedure was exploited for NBV evaluation. Therefore, ray casting was performed on GPU, with greatly increased efficiency.

In the second approach (Chapter 3), an exploration system is proposed, in which the environment is initially unknown. The robot attention is attracted by the objects as they are discovered. This attention approach leads to the evaluation of a lower number of poses, i.e. only poses pointing towards the objects. A novel method to generate view poses from the 3D reconstruction itself has been developed, exploiting frontiers between known and unknown space. Therefore, poses are not constrained on spheres, and POIs are not needed. Moreover, a spatial attention heuristic based on segmentation has been proposed. The 3D representation is partitioned into segments. Each segment is assigned a saliency value, based on a heuristic which favors segments belonging to objects. In the experimental setup, the Kinect sensor was replaced with a Kinect V2. A novel shadow removal filter has been developed in order to remove artifacts produced by this sensor, which is based on time-of-flight technology.

In Chapter 4, further improvements are shown, for the surfel-based 3D representation of ElasticFusion [9]. In this thesis, a preliminary investigation of a NBV algorithm based on ElasticFusion surfel-based 3D representation has also been investigated, and a few contributions are proposed towards this goal. It has been observed that the averaging process in ElasticFusion produces blur around color discontinuities, which can hamper segmentation. Therefore, an improvement for segmentation has been proposed based on a mode filter. The mode filter keeps track of multiple possible positions and colors during ElasticFusion 3D reconstruction. The loss of information due to the averaging process is reduced. To efficiently generate a ground truth for segmentation, an innovative annotation tool was also developed. The annotation tool is based on the selection of sparse control points on the point cloud.

The remaining points acquire the label of the nearest control point, according to a shortest-path tree segmentation algorithm.

Thesis structure

This thesis is organized as follows. Chapter 1 analyzes the state of the art for the topics involved in the thesis. In Chapter 2, the devised user-driven spatial attention system [11, 12] is described. The object-driven approach [13] is described in Chapter 3. Improvements towards the use of a surfel-based representation [14][15] are presented in Chapter 4. Experiments and results are reported at the end of the respective chapters, with a brief discussion. Final remarks, conclusions and future work are proposed in Chapter 5.

Chapter 1

State of the Art

1.1 Next best view planning

The first solution to the next-best view problem was presented in 1985 by Connolly [10]. In this approach, the volumetric 3D representation, based on an octree, encoded empty, occupied and unknown space. Viewpoints were sampled on a sphere centered on a target point, corresponding to the center of the object to be reconstructed. The view direction was oriented towards the target point. From each view direction, a depth camera was simulated through ray casting. A score function counted the number of unknown voxels visible from each candidate pose. The pose with the highest score was chosen as the next-best view.

Connolly's work introduced the seminal concepts for next-best view computation. A *ternary 3D representation* stores the current state of 3D reconstruction process. Unlike a standard binary representation, a ternary representation keeps track of occupied space, empty space and unknown space, which should be explored. A portion of the volume is marked as empty if the sensor observed through it. During the *view generation* phase, the candidate view poses are generated. Using a *sensor model*, a virtual sensor is simulated in each candidate pose. The *pose evaluation* phase computes a score for each candidate, and finds the best one.

The problem of choosing the target point in Connolly's system was solved by

Banta et al. [16]. Target points were generated as the centroids of unknown volume clusters.

Pito [17] proposed a next-best view system using a turntable which rotated a single object with respect to a fixed range camera. In this configuration, a turntable provided only one degree of freedom. Moreover, Pito's experiments were limited to single objects. However, an alternative pose generation method was introduced, based on the incomplete borders of the object, to detect the regions where further exploration is possible. Moreover, a polygon mesh was used as 3D representation, instead of an octree. In this case, an explicit representation of the unknown space is not needed, because edges of the mesh were blindly explored, regardless of the expected information gain. A ternary mesh representation was introduced in [18]. In this case, the mesh surface was labeled as "measured" between occupied and empty space, or "void", between empty and unknown space. The next-best view algorithm attempted to maximize the "void" surface visible from the chosen candidate pose.

The view direction for maximum quality should always be perpendicular to the unexplored surface of the object. Since the unexplored surface is unknown, this constraint on pose generation is an open issue of next-best view algorithms. Several approaches have been developed to estimate the unknown surface by extrapolation. Whaite et al. [19] approximated the objects in the scene with simple parametric models based on superquadrics. The shape of the unexplored parts was approximated in [20] by closing the partial mesh, in order to estimate the unknown volume. In [21], the object was approximated by B-Splines, and the next-best view metric was based on entropy. In [22], the shape of the object was inferred by fitting a local spheroid on the object edges. Approximation-based approaches were only useful for simple objects, which can be extrapolated from the trend of the known surface.

Several approaches for full six-degrees-of-freedom planning around a single object have been proposed. In [23], a next-best view algorithm was integrated with a robot arm planner, in order to obtain collision free sensor poses. The next-best view algorithm was adapted in [24] to reduce the registration error between the data acquired from inaccurate view poses. In [25], the observations were directed towards the object using a visual servoing approach. Unlike the solutions proposed in this

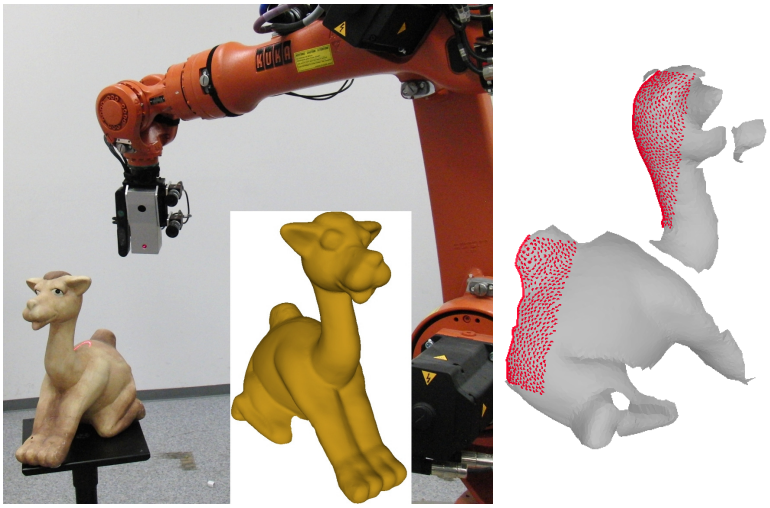


Figure 1.1: The reconstruction system used by Kriegel. A laser scanner, an ASUS Xtion depth camera and a stereo camera pair were mounted on the robot arm. On the right, Kriegel's boundary-based view generation method computed the average normal in the red area. Image source: [1].

thesis, these approaches considered a single object in the environment.

Constraint-based next-best view were proposed in [26], which performed mesh-based 3D reconstruction on a turntable. The space in which candidate view poses were sampled is limited by occlusion constraints, i.e. volumes where the target point was not visible.

In [27], a candidate view score function was modified to include the distance traveled. Also, the expected quality of the candidate view was estimated from the normal of nearby occupied voxels. The proposed approach was agnostic with respect to the actual technical solution for sensor movement. It was evaluated only in simulation.

A probabilistic framework for next-best view planning in cluttered environments was presented in [28]. The rays during ray casting did not stop at the first intercepted unknown voxel. Instead, the probability of an unknown object increased the longer the ray could travel in the unknown space without entering empty or occupied space.

In [29] optimizations of the next-best view algorithm for autonomous UAV explo-

ration were proposed. It is one of the few examples where next-best view was computed on the GPU. Another optimization on GPU is proposed in this thesis (Chapter 2), but for the general-purpose reconstruction algorithm KinectFusion.

Kriegel et al. [30] presented an autonomous reconstruction system for tabletop scenes that supports next-best view planning and object recognition. He investigated and combined multiple sensors, depth cameras and laser scanners, as shown in Fig. 1.1. In [31], Kriegel also investigated the generation of the next-best scan path for a laser scanner. In [32] a non model-based approach was introduced for next-best view using the boundaries of the scan. The approach estimated the surface trend of the unknown area beyond the boundaries. A limitation of the use of boundaries is that they work differently for square and rounded objects. Indeed, the trend beyond the corner of a square object seen from the front can't be estimated. A special case was added by Kriegel to solve this issue. An alternative solution without special cases is shown in Chapter 3.

1.2 Active robot exploration

Robot exploration is a natural application of next-best view algorithms. In an exploration scenario, a robot starts with a partial representation of the environment and needs to expand it. The problem may be solved using a next-best view cycle, where the robot iteratively computes the best viewing pose. This approach to exploration is used in Chapter 3 of this thesis.

The automated 3D reconstruction system by Kriegel [1] has already been reported in previous Section 1.1. However, unlike Kriegel's work, robotic applications do not usually focus on the improvement of the next-best view algorithm itself. Instead, they use it as a building block for more complex systems.

Next-best view has been applied to exploration systems aimed at object recognition. In [33], the robot classified the objects in the scene against a fixed model database. The classification procedure produced recognition hypotheses, which were then verified by further observations from other viewpoints. Similarly, [34] distinguished between similar objects by focusing the camera on details like text and bar-

Figure 1.2: The segmentation-based exploration system by Xu et al. Image source: [2].

code labels. In [35], the object 3D models were analyzed in advance, in order to find the best viewpoints for classification. During exploration, the view pose were scored according to their discriminatory value, i.e. the probability that decisive details could be acquired from that view pose.

Several applications of exploration for robot grasping have been proposed. In [36], a humanoid robot held an object in one hand. The robot reconstructed the object by rotating it. Next-best view was performed to find the best rotation and to minimize occlusion caused by the robot hand. In [37], the system was extended with dual-arm grasping. The 3D reconstruction could not be completed with only one hand, because part of the object was occluded. Therefore, the object was grasped from another side by the other hand and the reconstruction process continued. In [38], a PR2 robot used KinectFusion and next-best view to explore a tabletop environment, and found object handles to be grasped. In [39] a multi-scale variant of KinectFusion was developed, to maintain a fine representation for object grasping and a coarse representation for room navigation.

In recent years, few works have investigated the use of next-best view in conjunction with segmentation. These works are based on the same assumption presented in Chapter 3, i.e. the segmentation may provide cues for robot exploration. In [40] an

active object recognition system was proposed for a mobile robot. Box-like objects were segmented. A feature-based model was used to compute the next-best view by predicting both visibility and likelihood of feature matching. The robot had only two degrees of freedom, the position around the table and the distance from it. The mobile robot was not autonomous, but it was placed on the floor as dictated by the next-best view algorithm. In [2] (Fig. 1.2) a graph cut object segmentation was performed on an initial robot scan. The 3D representation was obtained through Kinect V1 and KinectFusion. Then, the PR2 robot combined next-best push planning and next-best view planning to perform proactive exploration of ambiguously segmented regions. Unlike the exploration system presented in this thesis, next-best view planning was performed only on pushed objects.

Other exploration strategies have also been proposed where the robot interacts with the environment by pushing objects. In [41], the robot autonomously moved the objects to improve segmentation. A probabilistic model was used to detect regions where segmentation is ambiguous. However, next-best view planning was not used and the sensor was moved on a fixed trajectory. A similar approach was proposed by [42]. Whenever the robot pushes an object, a probabilistic model was used to detect the region of the scene that changed according to the robot motion. That region likely belonged to the pushed object.

1.3 Robot attention

In computer vision, the main goal of attention is the assignment of a saliency value to each image pixel. Sensor movement, when possible, is usually accomplished with a simple pan-tilt head. Traditionally, the computation of saliency derives from two approaches. In the *bottom-up* approach, saliency is a function of local properties of the image itself, i.e. of local features or information density. For the *top-down* approach, external cues are exploited to compute saliency, e.g. the color of the object that the system is trying to locate. In many current approaches, *bottom-up* and *top-down* saliency are combined together.

In [43], attention was attracted to local features in the environment. These fea-

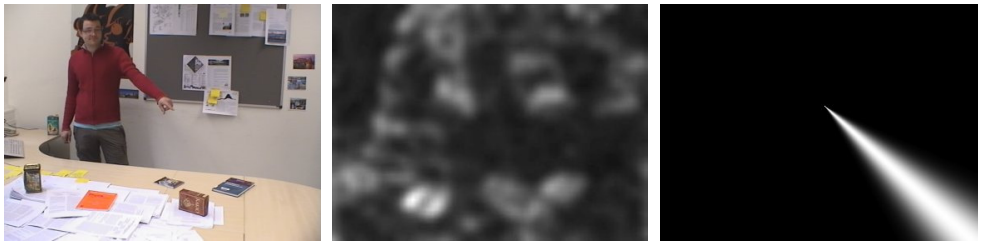


Figure 1.3: The saliency method by Schauerte et al. From left to right: the scene, the image-based saliency map (bottom-up saliency) and the gesture-based saliency map (top-down saliency). Images source: [3] © 2010 IEEE.

tures were classified by a neural network. Over time, the neural network learned to detect features that mark salient objects. A pan-tilt head moved the sensor to focus on the most salient region. A blob-based approach was proposed in [44]. The image was segmented into blobs using color discontinuities. Saliency was attributed to blobs using the difference between the color of the blob and the average color of the surroundings.

Novelty detection, i.e. the localization of regions of space where changes have occurred, may be used to attract the attention of a robot. In Chapter 2 of this thesis, novelty detection is triggered by user actions. Many approaches detected differences between multiple scans of the same region by a robot. Drews et al. [45] introduced a method for novelty detection from laser scan data. The laser scanner was mounted on a mobile robots which explored repeatedly the environment, in order to construct and maintain a consistent 3D representation. A Gaussian Mixture model was fit onto the 3D point cloud. The models obtained by subsequent explorations were compared and significant changes in the Gaussian configuration were detected.

Many works addressed the novelty detection problem in conjunction with segmentation. However, these approaches aimed at segmentation enhancement using novelty detection as a clue. In this thesis, segmentation was used to enhance attention, as shown in Chapter 3. Moreover, active exploration using next-best view planning was not considered. In the work of Alimi et al. [46], scene 3D reconstruction was repeated multiple times. Differences in the scene were detected and were assumed to

be linked to objects which were moved around. These differences were used to obtain a 3D representation of the moved objects. In [47] a similar method was used in order to train a segmentation algorithm. A similar assumption about moved objects was used by Herbst et al. [48, 49]. In [48], the assumption was applied to algorithm for object discovery from multi-scene Markov random field analysis. In [49], a similar method was presented for online 3D object segmentation and mapping.

A few works have investigated saliency or novelty detection based on user actions, as proposed in Chapter 2 of this thesis. Schauerte et al. [3] developed a method to detect the object pointed at by the user. Saliency computation was assisted by analyzing the directions of the pointing gesture (Fig. 1.3). Similarly, in [50] the focus of attention was estimated by gaze direction and exploited in a human-robot interaction task. Petsch et al. [51] proposed a framework for detection of unexpected (surprising) manipulation events. Manipulation events were represented by a graph of positions and connecting gestures. When a movement performed by the user is not present in the graph, the "surprise" may mark the discovery of a new gesture. In [52] a weakly supervised method identified daily actions from video sequences. The actions were differentiated according to the changes produced in the enviroment.

1.4 Human task segmentation

In Chapter 2, Section 2.2.4, a method for trajectory analysis is proposed. The method fits a Gaussian Mixture Model (*GMM*) on the user hand trajectory. Then, a saliency value is computed for each multivariate Gaussian, in order to estimate the positions in which the user has changed the environment. The use of a GMM to segment a trajectory was first proposed by [4]. Automatic segmentation of full-body motion trajectories was achieved by fitting the model through Expectation Maximization (Fig. 1.4). Given multiple recordings of the motion, regression was performed to estimate a continuous trajectory reproducible by a robot. The approach was only intended for task segmentation and trajectory simplification. It was not designed to detect meaningful and salient manipulation tasks.

Some traditional approaches for human task segmentation were based on hand

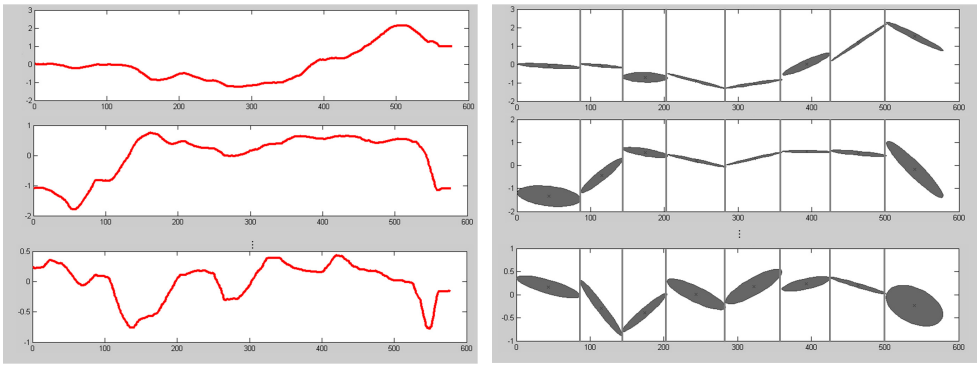


Figure 1.4: Trajectories (left) and the corresponding GMM (right), as obtained by Sang Hyoung Lee et al. Image source: [4] © 2012 IEEE.

velocity analysis. A similar method is compared in Section 2.4.2 with the GMM-based method proposed in this thesis. In [53] manipulation tasks were segmented into pre-grasp, grasp and manipulation phases. Hand speed and fingertip polygon sweep volume were monitored, and the trajectory was segmented at local minima. Yeasin et al. [54] used a binocular vision system and feature tracking to learn skills from multiple human demonstrations. Demonstrations were performed by the user, while the hand and four fingers were tracked. Then, a robot manipulator repeated the user actions. Trajectories were decomposed into simple motion primitives at significant speed variations. In [55], trajectory analysis was performed based on velocity features. Such velocity features were used to train a Hidden Markov Model. The Zero Velocity Crossing (*ZVC*) method was employed to detect regions where velocity is low or reverses direction.

Faria et al. [56, 57] proposed an approach to segment and classify the action phases of a grasping task. The approach exploited hand orientation, trajectory curvature and human gaze direction. In [58], the hand trajectory was split by analyzing the video of the movement. Each video frame was segmented using watershed algorithm. Frame segmentations were compared between subsequent frames. When the neighborhood relation between the image segments changed, the start of a new trajectory segment was marked.

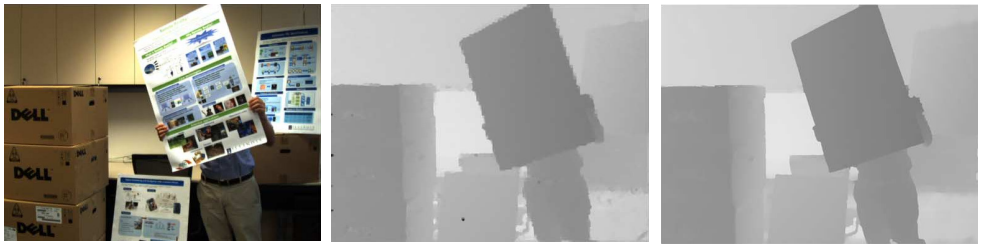


Figure 1.5: RGB image (left), noisy depth image (center) and enhanced depth image using temporal mode filter (right). Image source: [5] © 2012 IEEE.

1.5 RGB-D enhancement

In Chapter 4, Section 4.1, an enhancement algorithm for surfel-based 3D reconstruction is presented. The algorithm improves the reconstruction process to produce sharper discontinuities in the final 3D representation. Multiple hypotheses are tracked and a mode filter choses the most frequent at the end of reconstruction. A guided filter [59] is used for depth enhancement. The sharper discontinuities are then exploited for segmentation. Online 3D reconstruction enhancement to improve segmentation was not considered before.

Most enhancement algorithm operate on single RGB-D images. Depth enhancement is usually guided by the RGB image, which has lower noise. Joint bilateral filter has seen widespread use for depth enhancement and upsampling, due to its edge-preserving properties. In [60], joint bilateral filter was introduced to upsample a low-resolution image using an high resolution guidance image. Joint bilateral filter was modified in [61] to deal with the effect of occlusions and misalignments in RGB-D cameras. In [62], joint bilateral filter was combined with a depth-only bilateral filter to reduce noise in flat areas. In this thesis, a guided filter has been used. The guided filter has edge-preserving properties similar to the joint bilateral filter, but its computation is faster and (according to [59]) it produces less artifacts. Other works have exploited the guided filter to enhance depth in RGB-D images [63, 64]. Mode-based filters are known for their edge-preserving properties [65], but they have been rarely used.

Unlike single-image filtering, temporal filtering exploits a number of previous sensor frames in order to improve the current image. Temporal filtering may be applied whenever an image stream is available. RGB-D video temporal enhancement has received some attention in recent years, mostly in relation to 3D television. In [66] a temporal median filter was used for depth filtering and hole filling for a Kinect depth camera. In [5] a mode filter was used to enhance temporally adjacent depth frames (Fig. 1.5), but no underlying 3D model was present. Image segmentation has been used to improve RGB-D frames, but the use of RGB-D temporal enhancement for object segmentation (i.e. the opposite) has not been considered before. In [67], RGB image segmentation was used to correct discontinuities in the depth map. In [68] inconsistencies were fixed by comparing multiple input images. RGB-D frames were segmented into uniform patches. Patches in subsequent input images were associated using variational inference and inconsistencies were removed.

1.6 Point cloud segmentation

Point cloud segmentation is the process of partitioning a point cloud into subparts, called segments. A good segmentation algorithm produces segments which have some meaning, e.g. whose points belong to a single real-world object. Point cloud segmentation has been exploited in this thesis, in Chapter 3 and Section 4.1.

The 3D reconstruction algorithm proposed in Section 4.1 improves the segmentation of the final, unorganized point cloud. However, most works segment single frames of a RGB-D sensor. Segmentation of single frames takes advantage of the neighborhood relation between image pixels in order to detect discontinuities, as in [69]. In [70], segmentation in real-time during surfel reconstruction itself was accomplished by propagating the labels of single-frame segmentation. In [71], a method to segment an unorganized point cloud was proposed. The points neighborhood relations were computed by K-nearest-neighborhood search. Then, the cloud was segmented along normal discontinuities. Segmentation of partially reconstructed point clouds was studied in [72].

In Chapter 3 the Supervoxel-LCCP [73, 6] (Fig. 1.6) algorithm is used as part

Figure 1.6: Segmentation examples using Supervoxel-LCCP. Image source: [6] © 2014 IEEE.

of the saliency computation process. Moreover, in Section 4.1 the same algorithm is used to evaluate the 3D reconstruction enhancement algorithm. The Supervoxel-LCCP segmentation algorithm is aimed at unorganized point cloud segmentation. It has been employed in many applications since its inclusion into the Point Cloud Library. The algorithm initially over-segments the point cloud and builds patches called *supervoxels*. Then, nearby supervoxels are merged in order to obtain convex segments. In [74], Supervoxel segmentation was used as a pre-processing step to compute saliency in a 3D model. In [75], Supervoxels were merged to infer scene semantics.

1.7 Annotation interfaces

In order to obtain a ground truth for the segmentation algorithm presented in Chapter 4, a novel annotation tool was developed (Section 4.2) during this thesis. The tool offers assisted annotation. In this section, the state of the art of assisted annotation is briefly discussed.

The annotation of organized point clouds, such as those obtained from single frames of range cameras, may ignore the 3D nature of the data, since the point cloud

can be projected back onto the sensor image frame. Annotation of RGB-D images was considered in the work by Russell et al. [76], which extends the LabelMe online annotation tool [77]. Annotation was performed on the range image using a polyline selection technique. In [78], annotation of RGB-D images was assisted by inferring the 3D structure of the scene through structural and geometric priors learned from previous annotation sessions. In [79] an interactive semantic modeling approach for indoor scenes was proposed for RGB-D images. The approach proposes a segmentation to the user, who can correct it by painting strokes. In [80] a 3D reconstruction and labeling tool was proposed where annotated labels were propagated from one frame to another.

The annotation tool proposed in this thesis supports unorganized point clouds. Unorganized point cloud annotation poses additional challenges, since there is no obvious mapping between the 2D space of a computer screen and 3D space of a point cloud. Some attempts have been made to extend selection on a 2D image to 3D space. Several selection strategies were proposed in literature for discrete objects as discussed by Bacim et al. [81]. In [82] two spatial, structure-aware selection techniques were developed. These techniques only required users to draw a lasso, i.e. a loop around the 2D projections of important parts of the 3D point cloud. While effective, such selection algorithm was quite complex and did not work in real-time.

Many authors investigated the use of 3D interfaces for selection. 3D interfaces make use of a 3D "cursor" which moves in space, instead of a 2D mouse pointer. Coffey et al. [83] proposed a virtual reality system exploiting a touch interface with support to selection of data subsets. As part of the 3DUI contest, several solutions like [84, 85] were proposed, based on virtual reality and hand gestures. In [86], a preselection phase allowed easier navigation in the 3D environment. Then, a 3D pointer was implemented to perform actual selection. A disadvantage of 3D user interfaces is that they often require non-standard 3D hardware, which may require additional training. Moreover, advanced 3D interfaces like virtual reality may be tiring for the user.

Machine learning techniques for large scale assisted annotation have also been investigated by Boyko et al. in [87, 88]. The approach assumed that the point cloud

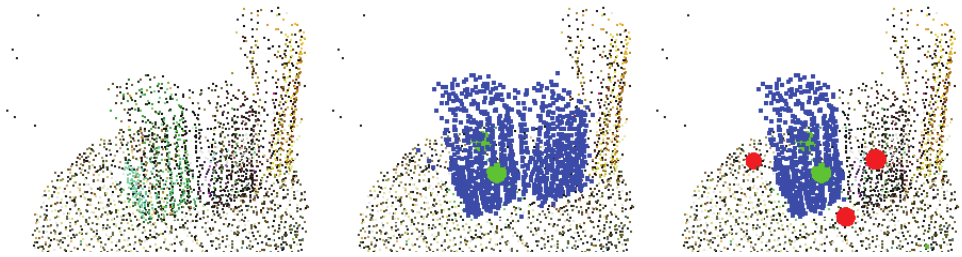


Figure 1.7: Point cloud annotation process (left to right) using min-cut algorithm and constraints. Green dots are placed to define the foreground, red dots to define the background. Image source: [7] © 2009 IEEE.

is composed by many similar objects whose shape could be learned by the system.

An approach similar to the one presented in this thesis was proposed in [7], where "constraints" (points or planes) were selected on the point cloud (Fig. 1.7). Constraints were used to perform a min-cut algorithm on the point cloud, in order to separate a foreground object from the background. Likewise, in [89] a min-cut algorithm exploiting user painted strokes was introduced. However, while effective to segment a single object from the background, the min-cut algorithm is not suitable to segment an arbitrary number of objects in a scene.

Chapter 2

User-driven spatial attention

In this chapter, the first approach for the application of spatial attention to next-best view is presented [11, 12]. A robot manipulator actively updates the 3D representation of the scene only where a change is detected in the environment. Changes in the environment are caused by user manipulation activities, which draw the attention of the robot. The robot can not observe at once the whole environment because of occlusions and kinematic constraints. Hence, the robot moves the sensor on its end-effector to complete and update the representation of the environment. The robot focuses the viewpoint of the eye-in-hand sensor towards the regions where user actions are more likely to have produced changes. This robot system is an example of active perception, since it closely links perception and motion planning.

The attention approach proposed in this chapter is developed for the 6-DoF robot manipulator already presented in the introduction. The robot arm operates in a tabletop scenario. The robot is equipped with a Kinect V1 range sensor (Fig. 2.1). Relevant changes in the environment are caused by manipulation actions of a person interacting with objects, which may include moving or removing an object in the scene or placing new objects. Potential changes are detected by analyzing the motion of the user hand by motion capture. The 3D environment representation is updated by the robot system only when one or more relevant changes are detected.

To merge information acquired by the range sensor from different views a mod-

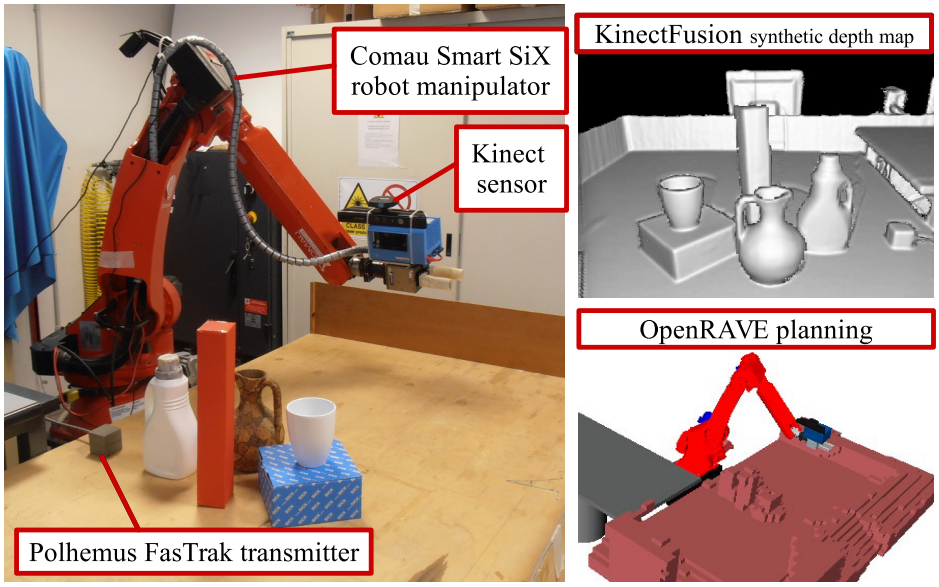


Figure 2.1: Experimental setup (left). Example of KinectFusion output (top right). Motion planning environment (bottom right).

ified version of KinFu was developed. The KinFu Large Scale (KinFu LS) project [90] is an open source implementation of KinectFusion [8] based on the PCL library [91]. The environment is modeled as a volumetric 3D voxel grid using a truncated signed distance function. Each voxel is labeled as unknown, empty or occupied. When salient user activities are detected, points of interests are computed from the estimated regions of the environment where something has possibly changed. Then, a next-best view (*NBV*) algorithm is executed iteratively to plan the best pose where the sensor must be placed to observe the regions of interest. Multiple views of the robot may be required in order to achieve a complete update of each region of interest. The robot exploration phase of each region of interest continues until information gain is negligible.

The main advancement proposed in this chapter is the application of attention to next-best view planning. A novelty detection system is developed, which mantains and updated 3D representation by analyzing user activities. Moreover, next-best view

planning is applied to KinectFusion 3D reconstruction. Two strategies for next-best view have been evaluated: keeping the KinFu active during the robot motion between consecutive views and turning it on only at the planned viewpoints. Finally, an algorithm is proposed which segments the human hand trajectory using a Gaussian Mixture Model (*GMM*) to detect where salient user actions occur. The proposed algorithm has been evaluated on a dataset of user activities and compared to a zero-velocity crossings (ZVC) approach.

As further contribution, the original KinFu algorithm has been adapted to the task proposed here. The adaptation has been published as open-source at `https://github.com/RMonica/ros_kinfu`. The next-best view algorithm is executed directly on the GPU, which required modifications to KinFu ray casting and shifting procedure. Moreover, KinFu has been modified to improve accuracy by exploiting the accurate kinematics of the robot, as in [39], instead of performing egomotion estimation. Experiments are reported to compare the GPU-based next-best view algorithm to the same algorithm executed on the CPU. As expected, the GPU-based next-best view is faster.

The chapter is organized as follows. Section 2.1 provides an overview of the proposed approach. KinectFusion is introduced in Section 2.2. Section 2.2 also describes the NBV algorithm developed on the GPU, how KinFu was adapted, and the algorithm for determination of the regions of interest. In section 2.3 the software architecture based on the PCL library is discussed as well as the integration with the ROS middleware. Section 2.4 illustrates the experimental results, which are briefly discussed in Section 2.5.

2.1 Method overview

The proposed attention-based approach enables a robot manipulator to update the 3D representation of the environment each time a change is detected. The update procedure consists of taking new observations using the eye-in-hand sensor in the regions where a user performed a task. The general behavior of the system is outlined in Figure 2.2. First, an initialization phase occurs, where the robot acquires data

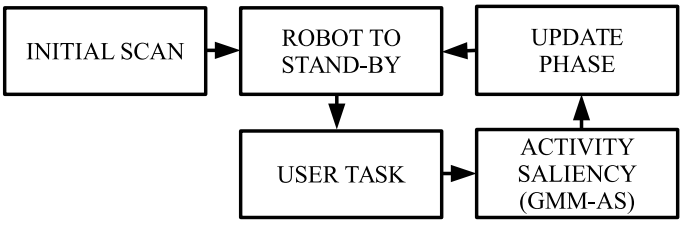


Figure 2.2: The general flowchart of the system.

along a predefined scan trajectory (which is assumed not to collide with any object) using KinectFusion. 3D data acquired after the initial scan only serves as a starting representation of the environment and may include large unseen areas due to object occlusions. Then, the robot moves to a "stand-by" configuration where it waits for a user to perform a task.

The user can interact with the environment by performing object manipulation tasks. Each user task may involve one or more sequential actions, each of which can be performed in different regions of the environment. For example, a pick-and-place task requires to direct the attention of the robot to both the initial and final configurations of the object. The trajectory of the user hand during a task is acquired by a motion capture sensor (described in Section 2.4) and it is analyzed by the GMM-AS algorithm, detailed in Section 2.2.4. GMM-AS generates a set of Points of Interest (*POI*). Each *POI* represents the approximate location of a region of interest where a change in the environment might have occurred.

Then, all the regions of interest are cleared with unknown voxels and the robot starts an update phase (Section 2.2) to explore all the now unknown regions of interest. After the update phase, the robot returns back to the "stand-by" configuration. The 3D representation of the environment is computed by the KinectFusion algorithm. The PCL KinFu LS implementation of KinectFusion was adopted and extended. An overview of the algorithm is reported in Section 2.2.1, while the implementation in Section 2.3.

The robot update phase is driven by a NBV algorithm [16, 10], detailed in Section 2.2.3. Given a ternary volumetric representation (occupied, empty, unknown) of the

environment, the NBV algorithm simulates the Kinect range sensor and produces a sequence of viewpoints ranked by their highest theoretical information gain. Collision free robot movements are planned using the OpenRAVE engine [92]. Occupied and unknown voxels are considered as obstacles in the motion planner. To speedup the planning phase, the representation is conservatively subsampled to voxels of size 4 cm.

2.2 Robot attention approach

The update phase of the environment 3D representation, detailed in Algorithm 1, starts when a set of Points Of Interest is generated (to be explained in Section 2.2.4). Each *POI* is characterized by a center point and a surrounding spherical shape $S(POI)$, describing the region of interest that it generates. First, all the regions of interest are filled with unknown voxels (line 2) which will be considered as obstacles while planning robot movements. Then, the first *POI* of the sequence is selected and viewpoints are computed to explore its region of interest $S(POI)$ by applying the NBV algorithm (line 5). The NBV algorithm scores viewpoints maximizing the estimated information gain. Information gain of each viewpoint $Gain[I]$ is computed as the number of unknown visible voxels. NBV evaluation is limited to the shape $S(POI)$ of the current *POI*, since other *POIs* will be better served by subsequent observations, centered on them.

All the viewpoints are then attempted one after the other, starting from the most promising in decreasing order. If one viewpoint predicts an information gain less than a given threshold $S(POI)$ is considered completely reconstructed and the next *POI*, if any, is selected (line 7). Viewpoints are re-computed after each successful observation (line 14). Once a valid viewpoint is found, a trajectory for the robot is planned (line 10). A viewpoint is skipped either when unreachable, due to robot kinematics, or when already attempted for the current *POI*. All information acquired by KinectFusion is used to update the environment representation, thus it may happen that a region of interest $S(POI_i)$ is partially observed during the exploration of another POI_j. This case is implicitly handled by the proposed algorithm.

Algorithm 1 Environment representation update phase

Require: *WS*: 3D volumetric environment representation;
Require: *POIs*: Point of Interest array;
Ensure: *WS*: The updated representation;

```
 1: for each POI in POIs do
 2:    ForgetAround(WS,POI);
 3: end for
 4: for each POI in POIs do
 5:    (Viewpoints,Gain) ← NextBestView(WS,POI);
 6:    for I from 1 to size(Viewpoints) do
 7:       if Gain[I] < IncTh then
 8:          break;
 9:       end if
10:       (Ok,Traj) ← PlanRobotTo(Viewpoints[I]);
11:       if Ok then
12:          MoveRobotAlong(Traj);
13:          PerformRobotOscillation();
14:          (Viewpoints,Gain) ← NextBestView(WS,POI);
15:       end if
16:    end for
17: end for
```

When a viewpoint configuration is reached, the robot moves so that the Kinect sensor tilts slightly (± 5 degrees) around the sensor horizontal axis (line 13), allowing KinFu to work properly on a continuous input stream of depth images from that viewpoint. The tilting motion is modeled in the NBV algorithm by configuring the simulated sensor with a wider vertical field of view than the real Kinect sensor. Two variations of the NBV exploration strategy have been evaluated. In the first, called "KinFu On ViewPoint" (KFOVP), KinFu is allowed to acquire new data only from the observation viewpoints as described above. In the second, called "KinFu On Motion" (KFOM), KinFu is also kept active during the motion of the robot between two consecutive viewpoints. KFOM should require a lower number of views to complete the environment update phase. Both KFOVP and KFOM use KinFu output as the

3D volumetric reconstruction of the scene. KinFu is not reset after each viewpoint observation.

2.2.1 The KinectFusion algorithm

KinectFusion is an iterative algorithm for real-time tracking of a moving depth camera and 3D fusion of the environment observations in a volumetric data structure. Software implementations exploit highly parallel GPU techniques (CUDA). The 3D environment representation is accessible at any time, provided a fast access to the GPU memory is available. KinectFusion represents the environment as an implicit surface model using a truncated signed distance function (TSDF). TSDF is a clamped function $R^3 \rightarrow R$ which maps the 3D coordinates (x, y, z) to the distance from the nearest surface, negative inside objects and positive outside. KinectFusion samples a TSDF volume in a regular grid of voxels with fixed size. Each voxel contains two values: a TSDF sampled value v and a weight w that counts the number of times the voxel has been observed.

Each KinectFusion iteration consists of four steps. *Ray casting*: A synthetic depth map (point cloud) is generated from the current TSDF volume, as seen by a virtual sensor placed in the last known sensor position. *Depth map conversion*: The latest measured depth image is converted into an organized point cloud. *Camera tracking*: The point cloud is aligned with the synthetic depth map using a modified point-to-plane ICP (Iterative Closest Point) algorithm and then the motion of the sensor is estimated from the ICP transformation. *Volumetric integration*: The point cloud is converted to global coordinates and merged with the TSDF volume. *Ray casting* and *volumetric integration* steps are described next to explain how KinectFusion was adapted to work with the NBV algorithm.

In the *ray casting* step, each ray of the sensor is sampled with constant step and traversed until it exits the TSDF volume or a zero crossing of the TSDF is found. When a zero crossing of the TSDF is found a new point is saved in the point cloud using a trilinear interpolation of the TSDF values of neighboring voxels. Since KinectFusion is executed on the GPU, each ray is evaluated in parallel.

During the *volumetric integration* phase, the latest sensor pose and the corre-

sponding observed points are converted in global coordinates. Then, the voxels intersected by the view ray are updated up to the observed point as follows. The weight value is used to average the new value with past values and it is increased at each new voxel observation. The weight value is capped to a maximum value M_w: thus, after some observations, the update procedure becomes a running average. The update equations of the TSDF and weight values of each intersected voxel can be written as follows:

$$v_{SDF} = \| p - o \| - \| c - o \| \tag{2.1}$$

$$v' = \frac{v \cdot w + clamp\,(v_{SDF}/M_v, -1, 1)}{w+1} \tag{2.2}$$

$$w' = min\,(w+1, M_w) \tag{2.3}$$

where p is the point observed by the sensor from position o and c is the position of the voxel in the TSDF volume, along the view ray $\overline{op}$. The temporary value v_{SDF} represents the Signed Distance Function, not yet truncated. The v_{SDF} value is normalized by the maximum value M_v and clamped between -1 and 1 to obtain the updated TSDF v'. Empty voxels far outside the surface have a TSDF value equal to 1, empty voxels near the surface have a TSDF value between 1 and 0, while occupied voxels inside the surface have a TSDF value below 0 (Figure 2.3). Voxels never observed have a 0 weight and their TSDF value remains at its default 0 value. Since the view ray stops at the observed point, voxels inside objects are never updated, even though from update equation 2.2 the TSDF value should be set to -1.

2.2.2 KinFu Large Scale

A limitation of the KinectFusion algorithm is that it can neither acquire data outside the TSDF volume, nor use data outside the TSDF volume for sensor tracking. Besides, the TSDF volume can not be expanded indefinitely due to limited GPU memory. Improved versions of KinectFusion have been developed based on the concept of downloading part of the 3D representation on the CPU memory, thus freeing space on the GPU for data processing [93]. This operation is called *shifting*.

PCL KinFu LS uses a cyclical buffer to obtain a faster shifting procedure, without

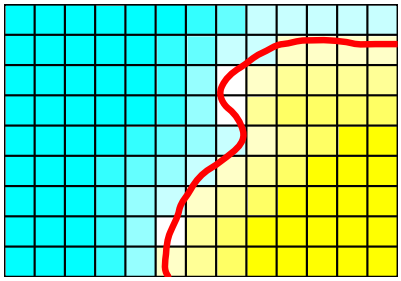


Figure 2.3: An example of 2D TSDF volume. The red line represents the object surface. The yellow area represents the negative values inside the object, while the cyan area the positive ones.

the need of memory deallocation and reallocation. The TSDF volume is allocated as a 3D matrix of fixed size. Whenever shifting occurs, the origin of the TSDF volume is translated, so it can represent a different area of the workspace. Slices of the TSDF volume are downloaded from the GPU memory to the CPU RAM. The now free slices on the GPU are then refilled with existing data (if any) from the CPU, or reset to unknown state ($v = 0$, $w = 0$). The new TSDF volume is partially overlapped with the old one, hence some data is kept on the GPU memory. Since the TSDF volume is organized as a 3D cyclical buffer only the origin is shifted and no data needs to be moved on the GPU. Downloaded slices would need a large amount of CPU memory, if they were saved as a voxel grid. The solution proposed by KinFu LS is to convert slices to a point cloud. Each point is defined by the 3D global coordinates of the voxel and the TSDF value. The weight value is lost in the process. Moreover, to reduce the size of the point cloud, all points with TSDF value $v = 1$ or never observed ($w = 0$) are removed from the representation.

KinFu LS introduces a number of translated and scaled reference frames. Reference frame $\{W\}$ is the world frame in real metric coordinates. The expanded frame $\{E\}$ is the origin in expanded coordinates, scaled with respect to world coordinates so that each TSDF voxel has edge 1, i.e. a point ${}^{E}p$ in frame $\{E\}$ can be expressed as:

$$ {}^{E}p = \frac{{}^{W}p}{e} \tag{2.4} $$

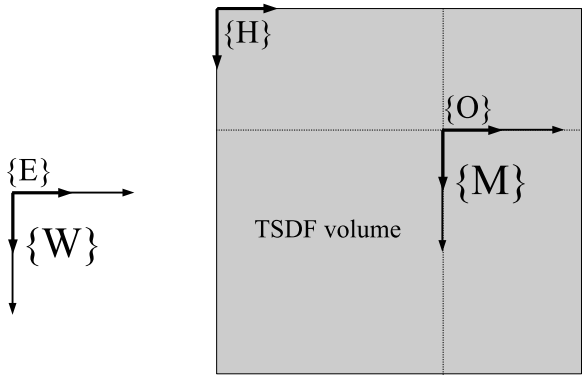


Figure 2.4: A 2D representation of the KinFu LS reference frames. Unit vectors for $\{O\}$, $\{E\}$ and $\{H\}$ are shorter, since they use expanded (scaled) coordinates.

where e is the voxel size. The TSDF shift frame $\{H\}$ is centered at the voxel of index $(0,0,0)$ in the TSDF volume in expanded coordinates. The TSDF frame $\{O\}$ is centered at the voxel of the TSDF volume that determines the shifting operation, in expanded coordinates. The TSDF frame $\{M\}$ has the same origin of $\{O\}$ but it is defined in real world metric coordinates. An overview of the reference frames is shown in Fig. 2.4.

Reference frames defined above are updated at each shifting operation. In the following, the notation ${}^{b}_{a}t$ indicates the translation vector of reference frame a with respect to frame b. Given that the TSDF volume is cyclical, a point ${}^{H}p$ in TSDF shift coordinates can be expressed in extended coordinates ${}^{E}p$ as:

$$ {}^{E}p = \left[\left({}^{H}p + {}^{O}_{H}t + S_V \right) \bmod S_V \right] + {}^{E}_{O}t \tag{2.5} $$

where S_V is an array which contains the size of the TSDF volume in voxels and *mod* is the component-wise modulo operator. S_V is added to $({}^{H}p + {}^{O}_{H}t)$ so that the result is always positive.

In the attention-based system a spherical region of interest $S(POI)$ of the TSDF volume has to be cleared if the following inequality holds:

$$ ||{}^{W}S_c - {}^{W}p|| < {}^{W}S_r \tag{2.6} $$

where ${}^{W}S_c$ is the sphere center and ${}^{W}S_r$ is the radius. TSDF voxels can be processed in parallel using a CUDA kernel. Each thread evaluates a different voxel, with index in TSDF shift coordinates ${}^{H}p$. To verify inequality 2.6, ${}^{H}p$ has to be converted to ${}^{W}p$ for each thread using equation 2.5 and then equation 2.4. However, it can be noticed that equation 2.4 can be applied to inequality 2.6 in advance, once for all on CPU. The CUDA kernel can then operate directly in the expanded coordinates $\{E\}$, and considering that

$$ {}^{O}S_c = {}^{E}S_c + {}^{O}_{E}t \tag{2.7} $$

inequality 2.6 can be rewritten as:

$$ ||{}^{O}S_c - {}^{O}p|| < {}^{O}S_r \tag{2.8} $$

Hence, a slightly simpler equation than 2.5 may be used to obtain the point to be evaluated for inclusion in $S(POI)$:

$$ {}^{O}p = \left({}^{H}p + {}^{O}_{H}t + S_V\right) \bmod S_V \tag{2.9} $$

If the voxel satisfies the inequality it is set to unknown.

2.2.3 Next-best view approach

This section describes the GPU-based NBV algorithm. The algorithm follows the two-step approach proposed in [16, 10]. In the first step, candidate viewpoints are sampled on a sphere with fixed radius centered on the current *POI*. Each sensor position is uniquely defined by assigning the radius (d), the *POI* and three attributes: longitude (ϕ), latitude (θ) and rotation around the sensor axis (ψ) as shown in Figure 2.5. Rotation ψ is mainly important to ease the search for a reachable pose of the robot manipulator. The three attributes are uniformly sampled in their ranges: $\phi \in [0^\circ, 360^\circ)$, $\theta \in [0^\circ, 90^\circ]$, $\psi \in [0^\circ, 360^\circ)$, to cover all the upper hemisphere. The sampling intervals are respectively 30°, 10° and 45°, resulting in a total of 960 sampled viewing poses. The radius d is fixed at 0.8 m having the Kinect sensor a minimum sensing distance of 0.5 m.

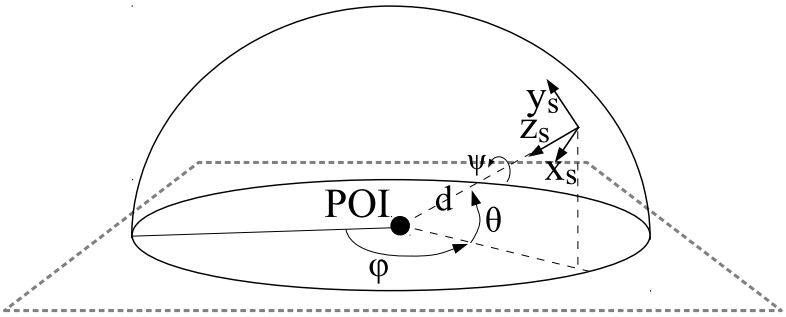


Figure 2.5: The sphere sampled by NBV algorithm around the POI with radius d.

In the second step, viewpoints are evaluated and sorted in decreasing order by a scoring function which counts the number of unknown visible voxels. The NBV algorithm must find where each viewing ray, passing through a pixel of the virtual sensor, intersects the first non-empty voxel in the environment representation. Thus, the second phase is the most time consuming. Unknown voxels are counted only if inside the current shape $S(POI)$. Therefore, a voxel v viewed by the simulated sensor of the NBV algorithm will contribute to the expected gain $Gain[I]$ of a viewpoint with index I as:

$$Gain[I] \leftarrow \begin{cases} Gain[I]+1 & \text{if } v = unkn. \wedge v \in S(POI) \\ Gain[I] & \text{otherwise} \end{cases} \tag{2.10}$$

The proposed algorithm, named NBV-GPU, exploits the TSDF volume itself. Indeed, it may be observed that the *ray casting* phase of KinFu can be adapted to obtain the observed TSDF voxels, provided the point of view of the virtual sensor can be arbitrarily set. The main advantage of this novel solution is that the GPU allows fast computation of the NBV at higher resolution than the CPU can achieve. Moreover, it does not require data to be extracted from the TSDF volume and converted on the CPU, and it uses exactly the same sensor model as KinFu.

In particular, the *ray casting* phase of KinFu has been modified to be executed on-demand by the NBV algorithm, from an arbitrary pose. When traversed by the

Algorithm 2 NextBestView($KinFu$,POI)

Require: *KinFu*: KinFu world representation;
Require: *POI*: Current Point of Interest;
Ensure: *Viewpoints*: Array of viewpoints;
Ensure: *Gains*: Scores assigned to each viewpoint;

```
1:  Viewpoints ← GenerateViewpointsAround(POI);
2:  Gains ← array[size(Viewpoints)];
3:  KinFu.ForceShiftToPoint(POI);
4:  for I from 1 to size(Viewpoints) do
5:     Gains[I] ← 0;
6:     Voxels ← KinFu.RayCast(Viewpoints[I]);
7:     for each Voxel in Voxels do
8:        if S(POI).Contains(Voxel) and Voxel.IsUnknown then
9:           Gains[I] ← Gains[I] + 1;
10:       end if
11:    end for
12: end for
13: OrderViewpointsAndGains(Viewpoints,Gains);
```

NBV algorithm rays must stop at zero crossings (default behavior), but also when an unknown voxel (with 0 weight) is intersected. An overview of the NBV procedure, called *NextBestView* in Algorithm 1, is shown in Algorithm 2. Initially, a shifting is forced to set the origin $\{M\}$ of the TSDF volume to the *POI* (line 3). This shift loads the region of interest on the TSDF volume. Unfortunately, as stated in Section 2.2.2, all shifting events, including those that happen when KinFu is active, cause the loss of both the weight values and the empty voxels with TSDF value $v = 1$. Hence, empty voxels far from the surface and the unknown voxels become impossible to distinguish. This information must be restored in the TSDF volume for the NBV to work properly. Keeping track of both unknown and empty voxels is achieved by using a custom octree data structure on CPU memory. This data structure was designed to efficiently store points observing that the number of unknown and empty voxels is much higher than the number of occupied ones. Details are provided in Section 2.3.2.

The NBV-GPU algorithm was compared to a standard algorithm where NBV is computed on the CPU (NBV-CPU). As explained above NBV-CPU requires volumetric data to be extracted from the GPU TSDF volume to the CPU. In addition, extracted data needs to be converted to a standard ternary representation (occupied, empty and unknown voxels). A conversion rule can be defined as follows:

$$\begin{cases} w = 0 & \rightarrow \textit{unknown voxel} \\ w > 0 & \begin{cases} v \leq 0 & \rightarrow \textit{occupied voxel} \\ v > 0 & \rightarrow \textit{empty voxel} \end{cases} \end{cases} \tag{2.11}$$

To speed-up ray casting NBV-CPU uses a standard PCL octree data structure, which is queried recursively to find the first intersection. In particular, NBV-CPU uses the *getIntersectedVoxelCenters* method of the *pcl::octree::OctreePointCloudSearch* class, configured to stop at the first non-empty octree voxel. This algorithm is optimized with respect to a raycasting algorithm in which the ray is followed through a voxel grid with a fixed step (named NBV-CPU-step in section 2.4.4), since the ray skips empty volumes thanks to the octree data structure.

2.2.4 Determination of regions of interest

Task trajectories are automatically recorded when the user hand is inside the bounding box of the workspace. The user hand is tracked by a motion capture sensor, described in section 2.4. When the hand exits the bounding box, the trajectory is analyzed. An overview of the procedure for trajectory acquisition, segmentation and *POIs* extraction is shown in Fig. 2.6.

The proposed algorithm for *POIs* extraction, named Gaussian Mixture Models Activity Saliency (GMM-AS), is reported in Algorithm 3. The hand trajectory is defined as $T = \{q_i\} = \{(x_i, y_i, z_i)\}$, a sequence of points q_i with $i \in 1 \ldots N$, where N is the length of the trajectory. Motion analysis for the determination of the *POIs* is executed in the augmented 4D space $S = \{(x_i, y_i, z_i, t_i)\}$ which includes time stamp information (lines 1 to 4). Including the time stamp helps the separation of points close in space but far in time. Indeed, self-intersections should not be encoded by the same Gaussian in the model.

Algorithm 3 GMM-AS POI detection

Require: T: the trajectory, sequence of points;
Ensure: $POIs$: the POI array;
$S \leftarrow$ empty set;
for i **from** 1 **to** size(T) **do**
  $S \leftarrow S \cup \{[x_i \quad y_i \quad z_i \quad t_i]\}$;
end for
$MinBIC \leftarrow +\infty$;
$GC \leftarrow 1$;
repeat
  $GMM \leftarrow$ ExpectationMaximization(S,GC);
  $BIC \leftarrow$ ComputeBIC(GMM,S);
  if $BIC < MinBIC$ **then**
    $(MinBIC, BestGMM, c^*) \leftarrow (BIC, GMM, GC)$;
  end if
  $GC \leftarrow GC + 1$;
until $BIC > MinBIC + BICTh$;
SortByTimestamp($BestGMM$);
for g **from** 1 **to** size($BestGMM$) **do**
  s_g = ComputeSaliency(G_g,c^*);
end for
$POIs \leftarrow$ empty array;
for g **from** 2 **to** size($BestGMM$)-1 **do**
  $th_g \leftarrow \frac{1}{2 \cdot \Lambda} \sum_{i=1}^{\Lambda} (s_{g+i} + s_{g-i})$;
  if $s_g > POITh \cdot th_g$ **then**
    $POIs \leftarrow POIs + \{[G_g.\mu_x \;\; G_g.\mu_y \;\; G_g.\mu_z]\}$;
  end if
end for

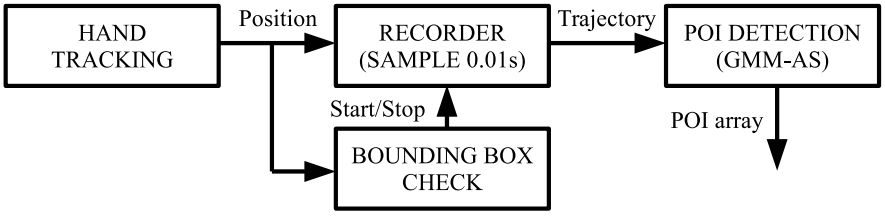


Figure 2.6: The trajectory recorder and *POI* detection as functional blocks.

GMM-AS fits a Gaussian Mixture Model $p(S) = \sum_{j=1}^{K} \pi_j \mathcal{N}(S|\mu_j, \Sigma_j)$ on the trajectory through Expectation Maximization (EM). EM is computed in function *ExpectationMaximization* (line 8), which takes as its arguments S and the number of Gaussians GC to be fitted, and it returns the GMM. The EM algorithm is initialized as in [4] by splitting the time span of the trajectory in equal segments and by assigning a Gaussian to each segment. The means and covariances of each Gaussian component are initialized using the sample mean and the sample covariances of the points which fall in the corresponding segment.

Since to apply the EM algorithm the number of Gaussians must be known, EM is executed multiple times with an increasing number of Gaussian components, starting from 1. For each GMM a BIC index [4] is computed by *ComputeBIC* (line 9). The lower the value of the BIC index the better the GMM fits the trajectory. It is assumed that the best fitting GMM model is in the first local minimum of the BIC index. The algorithm terminates when the current BIC exceeds the best found BIC by a small threshold $BICTh$ (lines 10 to 14). The Gaussians are ordered by increasing timestamp mean value in function *SortByTimestamp* (line 15).

It can be observed that, in the GMM, intermediate hand movements between manipulation actions (e.g. object transportation phases) are likely to be represented by long and narrow Gaussians, that exhibit a single dominant direction with high variance (Figs. 2.7 and 2.8). On the opposite, Gaussians with more balanced variances and a higher weight appear where the user performs a manipulation action. This can be explained by observing that user actions are usually performed at slow speed and involve changes of hand direction, hence more points are sampled without a domi-

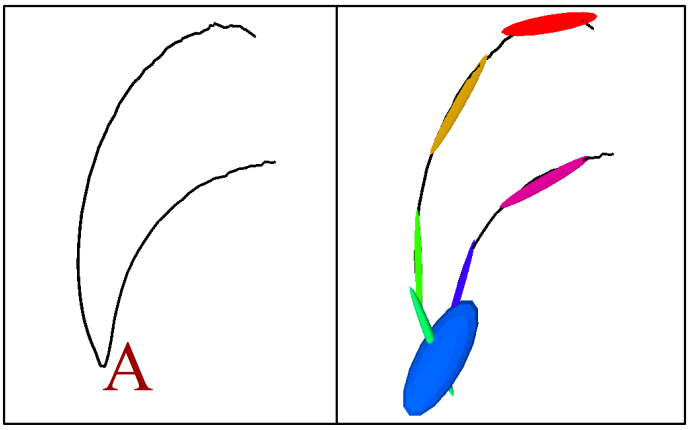


Figure 2.7: An example trajectory (left image) obtained from the placing task of an object in point A. The superimposed GMM (right image). Only the blue "thick" Gaussian correctly determines a *POI*.

nant direction. This facts can be exploited by evaluating each Gaussian G_g using a *ComputeSaliency* function (line 17) defined as follows:

$$\sigma_i = \sqrt{|\lambda_i|} \qquad i \in \{1,2,3\} \tag{2.12}$$

$$s = \frac{\sigma_1 \cdot \sigma_2 \cdot \sigma_3}{\sigma_1 + \sigma_2 + \sigma_3} \cdot \pi \cdot c^* \tag{2.13}$$

where λ_1, λ_2 and λ_3 are the eigenvalues of the Gaussian G covariance matrix, marginalized over Cartesian coordinates, s is the saliency, c^* is the Gaussian count and π is the weight (prior) of the Gaussian in the model. The saliency s_g of each Gaussian is compared to the average saliency of the neighbor Gaussians (lines 20-25). The first and last Gaussians are ignored, since they ofter suffer from boundary effects (line 20). If saliency is higher than the average saliency multiplied by $POITh$ (line 22), the spatial coordinates $(G_g.\mu_x,\ G_g.\mu_y,\ G_g.\mu_z)$ of the mean of the current Gaussian are added as a new *POI*. It must be noted that there is no reliable correlation between the shape (or size) of a Gaussian and the shape of the potential region of interest. Indeed, the shape of the Gaussian is mostly determined by the user's (local) trajectory and not by the shape of the object. Thus, the spherical region of interest $S(POI)$ is set

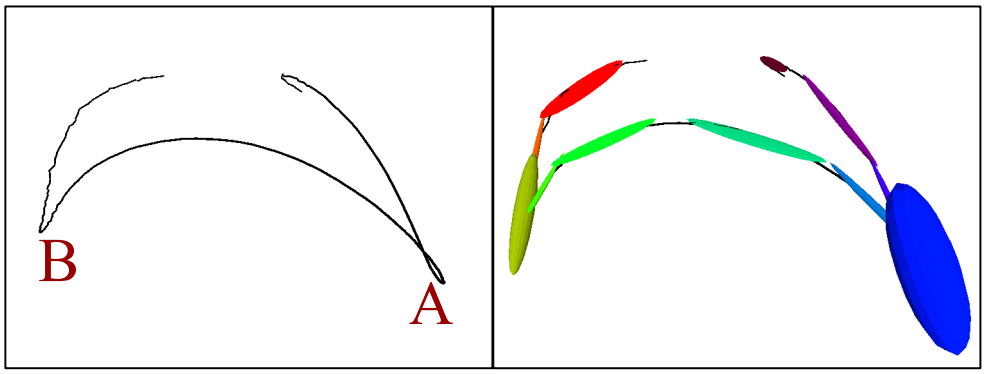


Figure 2.8: An example trajectory (left image) obtained from the task of picking and placing an object from A to B. The superimposed GMM (right image). Only the blue and the yellow "thick" Gaussians correctly determine two *POI*s.

with a fixed radius (${}^{W}S_r = 0.2$ m), larger than most common objects in manipulation scenarios. The user can, in principle, perform an arbitrary number of tasks. Each task can include an arbitrary number of actions. However, after each task the user must wait for the robot to complete the environment update phase before performing the following task. GMM-AS may generate multiple close *POIs* which, however, do not affect the performance of the system since these close *POIs* are simultaneously explored.

The GMM-AS algorithm was compared to an adaptive zero-velocity crossings (ZVC) algorithm, which is based the idea of looking for points of interest where slower hand movements appears. First, a mean filter with width $(2B+1)$ is applied to the trajectory to reduce noise. The average speed at index i is then computed as:

$$\bar{v}_i(r) = \frac{1}{\|R_i(r)\|} \sum_{j \in R_i(r)} \frac{||q_j - q_{j-1}||}{t_j - t_{j-1}} \tag{2.14}$$

where R_i is the set of the indices of the neighboring samples of q_i within a radius r, defined as:

$$R_i(r) = \{\ j\ \mid\ \forall k \in ([j,i] \cup [i,j]),\quad \|q_k - q_i\| < r\ \} \tag{2.15}$$

A slow hand movement is detected whenever the local speed is significantly lower

than the speed averaged on a wider range. In particular, the average speed is computed for two values of r: $r_1 = 10$ cm and $r_2 = 25$ cm. Segments of trajectory that satisfy $\bar{v}_i(r_1) \cdot ZVCTh < \bar{v}_i(r_2)$, where $ZVCTh$ is a constant multiplier, generate a *POI* at the average time index of the segment.

2.3 Software architecture

2.3.1 Extension of KinFu Large Scale for robot attention

KinFu LS defines *KinfuTracker* as the main class. *KinfuTracker* can be instantiated to a function object where the *operator()* method accepts a new depth map and executes one iteration of the algorithm. Two storage classes, *TsdfVolume* and *CyclicalBuffer* manage the world model. *TsdfVolume* acts as a proxy for the TSDF volume loaded on the GPU. *CyclicalBuffer* manages TSDF shifting and updates the reference frames. A nested class *WorldModel* manages the point cloud with intensity (*pcl::PointXYZI*) that contains the downloaded slices. The intensity value here represents the TSDF value v. Overall nesting relations are shown in Fig. 2.9.

To use KinFu as a persistent environment representation for the robot attention algorithm the following features were added. First, KinFu has to be interrupted and restarted on demand, for example when the robot reaches the stand-by configuration. Second, part of the environment has to be cleared, i.e. set to unknown to generate the regions of uncertainty of the *POIs*.

Interrupting KinFu can easily be achieved by not calling the *operator()* method as there is no active thread in the implementation. However, as the ICP algorithm works under the assumption of small sensor movements between consecutive readings, if a movement is too large egomotion estimation by ICP does not converge. This case happens if KinFu is suspended and resumed from another viewpoint. Therefore, to solve this issue, KinFu must be provided with the actual pose of the sensor. To this purpose a parameter, named *hint*, was added to *operator()*. The *hint* parameter accepts a struct of type *Hint*, defined as in Listing 2.1.

Three modes of operations are allowed: *HINT_TYPE_NONE* causes KinFu to use the default ICP egomotion tracking; *HINT_TYPE_FORCED* disables ICP and

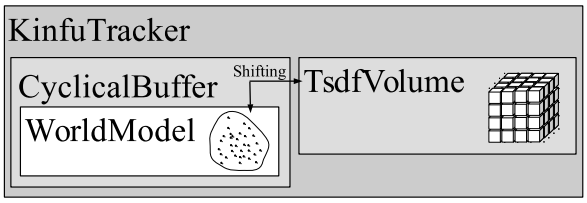


Figure 2.9: Relations among the most important classes of KinFu LS.

```
struct Hint
{
  enum Type
  {
    HINT_TYPE_NONE,
    HINT_TYPE_HINT,
    HINT_TYPE_FORCED
  };
  Type type;
  Eigen::Affine3f pose;
};
```

Listing 2.1: The *Hint* struct.

forces the use of the *pose* parameter; *HINT_TYPE_HINT* where ICP is enabled but it initialized with the *pose* parameter and not with the last sensor pose. In the proposed system, the pose of the sensor is computed from the robot kinematics. The third mode was added to allow an additional hybrid approach.

Clearing parts of the environment affects both *TsdfVolume* and *WorldModel* classes. The point cloud contained by the *WorldModel* class is in expanded coordinates $\{E\}$. Points not existing in this model are considered unknown when loaded into the TSDF volume. All the points in the *WorldModel* which verify inequality 2.6 (Section 2.2.2) in expanded coordinates are set to unknown. The sphere must also be cleared in the TSDF volume, accessed through the *TsdfVolume* class in reference frame $\{O\}$. This procedure is executed in parallel, on the GPU, for each voxel of the TSDF volume,

using equation 2.9 to clear voxels that satisfy inequality 2.8. If the voxel is inside the sphere, the voxel is set to unknown (TSDF value $v = 0$ and weight $w = 0$). A wrapper method was added to the main *KinfuTracker* class to convert the sphere to $\{E\}$ and $\{O\}$ coordinates and consistently clear both the *WorldModel* and the *TsdfVolume*.

2.3.2 Extension of KinFu Large Scale for next-best view

KinFu LS defines a *RayCaster* class that can be used to execute raycasting from an arbitrary pose in the TSDF volume. The *RayCaster* class can be configured with arbitrary intrinsics parameters: focal length and image size. The default configuration is changed, by setting a wider vertical field of view, when simulating the tilting motion of the real sensor described in Section 2.2. *RayCaster* uses a CUDA kernel to perform ray casting and to fill a vertex map with the voxels that are viewed by the virtual sensor. A boolean template parameter was added to enable the kernel to be used for the next-best view algorithm without affecting the original behavior.

For the next-best-view implementation rays need to be stopped at the first non-empty voxel. A knowledge map was added on the GPU, filled with values indicating if the ray intercepted an unknown voxel, an occupied voxel or exited the volume intercepting only empty voxels. Unknown voxels in the knowledge map are counted, to estimate the number of unknown voxels seen by the virtual sensor, only if the corresponding point in the vertex map is inside the shape $S(POI)$ (eq. 2.10).

As stated in Section 2.2.2, shifting loses distinction between unknown ($w = 0$) and empty ($v = 1$) voxels when downloading data from the GPU, as w is discarded in the *WorldModel*. For this reason, KinFu LS was modified in such a way that the distinction between unknown and empty voxels is properly saved and restored for the shifted slices. Adding additional information to the KinFu *WorldModel* would be extremely expensive in terms of memory occupation. Thus, a data structure was introduced to only distinguish empty and unknown voxels. This structure, named *BitmaskOctree*, is based on the *pcl::octree::OctreeBase* class. *BitmaskOctree* operates at lower $(1/8)$ resolution with respect to the TSDF volume. To match with the TSDF resolution, each octree leaf contains a 3D matrix *boost::bitmask* of $8 \cdot 8 \cdot 8 = 512$ bits. The compression is lossless in terms of resolution, however the actual value is bina-

rized to 0 (unknown) if $w = 0$ and 1 (known, i.e. empty or occupied) if $w > 0$. Octree leaves that are completely unknown are not stored in memory. The minimum amount of memory for a bitmask with a single point is 64 bytes. However, this event is very unlikely, since known voxels tend to form clusters and most of the bitmasks will be completely set. Indeed, leaves in the *BitmaskOctree* that are completely set are on average 69% of the total number of leaves.

```
class WeightCubeListener
{
  public:
  virtual void onReset() = 0;
  virtual void onClearSphere(...) = 0;
  virtual void onNewCube(...) = 0;
  virtual bool retrieveOldCube(...) = 0;
};
```

Listing 2.2: The *WeightCubeListener* interface. Parameters are omitted.

For the sake of generality *BitmaskOctree* was wrapped in a listener class. *CyclicalBuffer* was modified to accept a listener class which implements the *WeightCubeListener* interface, defined in listing 2.2. The *onReset* and *onClearSphere* methods are called when clearing the whole workspace or a spherical region of interest. When a shifting occurs, the method *onNewCube* is called to save TSDF weights before the shifting procedure. In addition, the *retrieveOldCube* method is polled during shifting and at each KinFu iteration until binarized weight information for the new TSDF volume origin is returned, extracted from the *BitmaskOctree*. The weight information is then merged with the TSDF volume in the GPU as follows. If the *BitmaskOctree* leaf is 1, the weight value in the TSDF volume is incremented by 1. Otherwise, the value on the TSDF volume is left unchanged. If incrementing the weight value results in a previously unknown TSDF voxel changing to known, the TSDF value is initialized to 1 (empty). The load operation may be executed at any time, amid KinFu iterations, as soon as the *retrieveOldCube* returns the weight information. Indeed, voxels changing from unknown to empty in the TSDF do not affect the correctness of the KinFu LS

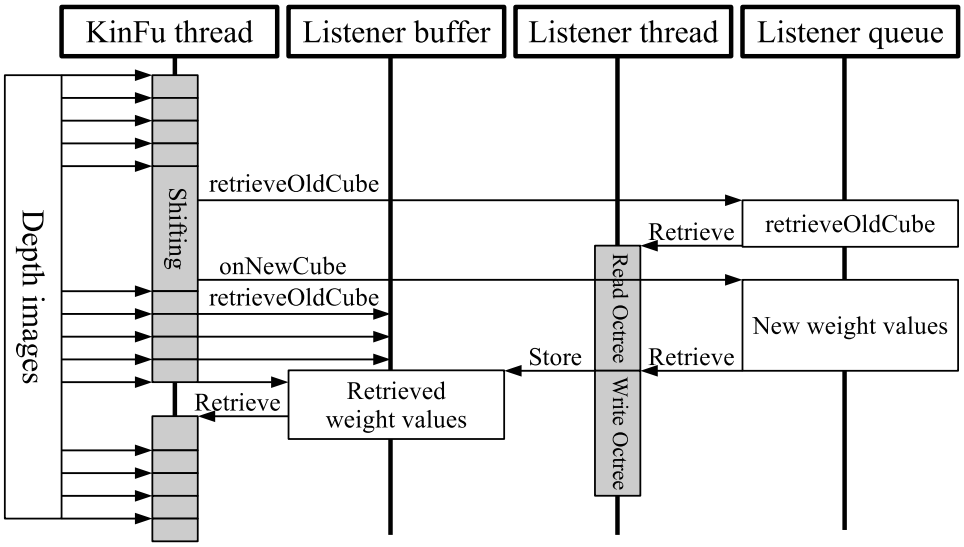


Figure 2.10: The multithread shifting procedure.

algorithm, because during standard execution the two types of voxel are treated in the same manner.

A multithread solution was implemented to execute asynchronously the setting and retrieval of values in the *BitmaskOctree*, without disrupting the KinFu execution, as shown in the sequence diagram in Fig. 2.10. The listener is an active object, with its own processing thread. At the first *retrieveOldCube* call the request is scheduled in a queue and, when the result is ready, it is returned during one of the subsequent polling calls to the method. Also, the *onNewCube* calls return immediately, while data is placed in the same queue for later processing.

2.3.3 System integration in ROS

KinFu LS was integrated into the ROS (Robot Operating System) framework and it is available for download (`https://github.com/RMonica/ros_kinfu`). Integration is based on the KinFuLS ROS wrapper [94], which publishes the synthetic depth map to a ROS topic and the current tracked sensor pose. The PCL KinFu LS

implementation is primarily designed to acquire data from the Kinect sensor, visualize it and save data to a file before termination. A post-processing phase is required to obtain a polygon mesh, by calling an executable which runs the Marching Cubes algorithm. In contrast to this standard behavior, as explained in Section 2.3.1 the attention-based system requires the use of KinFu as a persistent tool for environment representation.

In this work, the main *kinfu* node from [94] was extended to handle the modifications of KinFu LS. The extended *kinfu* node interacts with the ROS environment by accepting two types of messages: commands and requests. Commands affect the behavior of the system and do not require any further processing. Commands include *SUSPEND*, *RESUME*, *CLEAR_SPHERE*. Each command may also supply a hint or a forced hint, which act as defined in section 2.3.1. Also, a *SET_FORCED_TF_-FRAME* command binds the KinFu tracking to a reference frame using ros-tf. This feature is used to feed KinFu tracking with the robot odometry. Commands change asynchronously the state of a command subscriber, which applies the new configuration at the next iteration of the KinFu.

Request messages ask the *kinfu* node to publish a part of the environment. The part of the environment may be extracted in various formats, such as a point cloud, a polygon mesh or a voxel grid. Also, the projection of the environment on a virtual sensor (used for NBV) is executed through a request. Each request is served by a dedicated thread, different from the main KinFu thread. Requests copy data from the *WorldModel* and the *BitmaskOctree* after waiting for any KinFu pending operation in the *WeightCubeListener* queue or any *retrieveOldCube* unfinished calls. Each request thread must acquire an exclusive lock on KinFu to copy data. However, additional processing can be performed in parallel, such as execution of marching cubes to obtain the polygon mesh, or subsampling to reduce the size of the point cloud in the response.

2.4 Results

The experimental setup includes a small robot arm (Comau SMART SiX) with six degrees of freedom and a Kinect sensor mounted on the end-effector. The Kinect is calibrated with respect to the robot wrist. A Polhemus FasTrak is used for 3D hand tracking, calibrated with respect to the robot base frame. FasTrak is an electromagnetic device which tracks in real-time the position and orientation of a small receiver located on the wrist of the user. The advantage of using a magnetic motion capture device is that it does not suffer from occlusion problems. The overall configuration of the system is shown in Fig. 2.1. The software runs under the ROS framework on an Intel Core i7 4770 at 3.40 GHz, NVidia GeForce GTX 670.

2.4.1 Evaluation of Kinect sensor tracking solutions

Initial tests were performed to evaluate to what extent the internal drift of KinFu egomotion tracking may affect the robot attention system. In this section, KinFu egomotion tracking is compared with an alternative solution where KinFu is fed with the robot forward kinematics. Tests were performed by moving the robot arm along the trajectory used for the initial observation of the environment. Experiments were conducted in two scenarios with different objects as shown in Fig. 2.11. Tracking errors are computed as the difference between the robot forward kinematics (ground truth) and the KinFu egomotion estimation. Figure 2.12 shows the translation (Euclidean distance) and rotation (angle-axis representation) errors in the first scenario. The errors increase with time during robot motion. Egomotion tracking accumulates a large drift, as the translation error at the end is about 12 cm.

To further evaluate the accuracy of the two solutions three parameters were measured for each object and compared to the ground truth values: the object height and the lengths of the major and minor axis of the object projection on the horizontal table. Average errors for the parameters are reported in Table 2.1. The 3D reconstruction using the robot kinematics exhibits much higher accuracy than the one obtained using KinFu egomotion tracking. For this reason, during all the experiments of the attention-based system presented in Sections 2.4.3 and 2.4.4, the KinFu was fed with

Figure 2.11: The two environments used for testing KinFu tracking accuracy.

Table 2.1: 3D object reconstruction errors.

Scenario	Tracking method	Mean abs. (m)	St. dev. (m)
1	Egomotion	0.031	0.040
1	Robot kinematics	0.005	0.006
2	Egomotion	0.027	0.041
2	Robot kinematics	0.006	0.008

robot kinematics.

2.4.2 GMM-AS algorithm evaluation

The robustness of the GMM-AS algorithm was evaluated on a dataset of 330 tasks performed by four subjects. The dataset was published online at `https://github.com/RMonica/hand-polhemus-dataset`. The dataset contains 110 placement tasks of new objects in the environment, 110 object removal tasks from the environment and 110 pick-and-place tasks. Each recorded task trajectory was inspected manually and ground truth *POIs* were placed where actions actually occurred, i.e. at the initial location of the object for removal tasks, at the final location of the object for insertion tasks and at both the initial and final object locations for pick-and-place tasks. Object placement and removal tasks were evaluated together as they are characterized by a single action and they exhibit quite similar trajectories. Instead, pick-

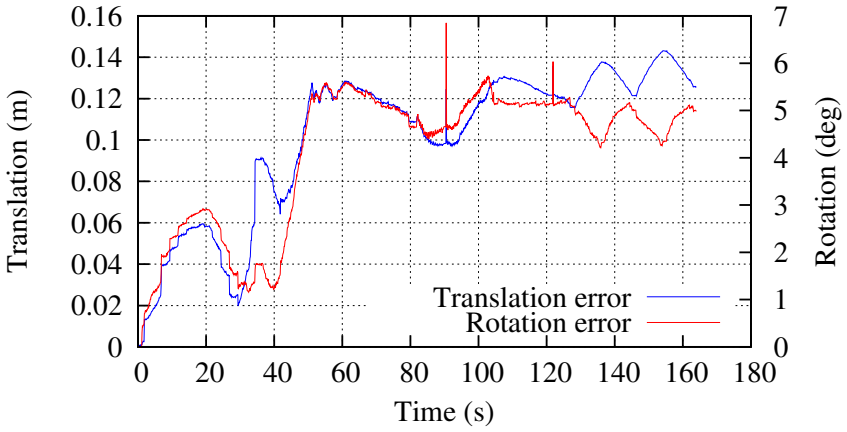


Figure 2.12: KinFu egomotion tracking error in the first scenario of Figure 2.11.

Table 2.2: Evaluation of GMM-AS and ZVC-like ($\Lambda = 2$, $POITh = 2.5$, $B = 4$, $ZVCTh = 1.72$).

Tasks	Algorithm	Precision	Recall
Placements and removals	ZVC-like	96.4%	97.3%
Placements and removals	GMM-AS	95.4%	95.9%
Pick-and-place	ZVC-like	94.5%	98.2%
Pick-and-place	GMM-AS	97.7%	97.3%

and-place tasks are characterized by two actions. Actions were counted as correctly detected if the distance to the true *POI* was within 20 cm (*POI* radius). False positives (FP) were counted when a *POI* was detected far from any ground truth *POIs*. False negatives (FN) were counted when the algorithm did not detect any *POI* near a ground truth *POI*. Tests were performed at different speeds of the task by undersampling ($\frac{1}{2}\times$, $\frac{1}{4}\times$) and oversampling ($2\times$, $4\times$) the trajectories. Additional tests were performed by adding to the trajectories white Gaussian noise.

The GMM-AS and the ZVC-like algorithms show comparable results on the dataset (Table 2.2). Also, ZVC-like shows a slightly better result than GMM-AS when the two algorithms are tested at different speeds (Table 2.3). However, to obtain

Table 2.3: Evaluation of GMM-AS and ZVC-like at different speed multipliers, on the whole dataset.

Speed multiplier	Algorithm	Precision	Recall
0.25	ZVC-like	96.6%	93.0%
0.25	GMM-AS	98.0%	86.1%
0.50	ZVC-like	95.7%	97.7%
0.50	GMM-AS	97.0%	95.9%
1.00	ZVC-like	95.5%	97.7%
1.00	GMM-AS	96.6%	96.6%
2.00	ZVC-like	95.7%	97.7%
2.00	GMM-AS	96.8%	92.5%
4.00	ZVC-like	96.4%	97.7%
4.00	GMM-AS	97.0%	80.7%

this result the noise-reduction window parameter B of ZVC must be at most 4. Such higher bound for B reduces the maximum noise that the ZVC-like algorithm is able to compensate. As seen in Table 2.4, GMM-AS is able to detect POIs even with 2.5 cm of standard deviation, while the ZVC-like algorithm fails.

The ZVC-like algorithm takes some milliseconds of computational time, while the GMM-AS takes a few (2$\sim$8) seconds. Nonetheless, this higher execution time does not affect the performance of the robot attention system which is mostly affected

Table 2.4: Evaluation of GMM-AS and ZVC-like at various noise standard deviation values.

Std. dev. (cm)	Algorithm	Precision	Recall
0.1	ZVC-like	95.4%	97.7%
0.1	GMM-AS	97.5%	98.4%
2.5	ZVC-like	95.1%	12.5%
2.5	GMM-AS	97.0%	80.2%
5.0	ZVC-like	97.5%	0.5%
5.0	GMM-AS	98.6%	56.8%

Table 2.5: The saliency computed for each Gaussian in Fig. 2.13.

Gaussian nr.	Saliency	Threshold	*POI*
1	0.005880	-	NO
2	0.001646	0.040424	NO
3	0.001352	0.031084	NO
4	0.073616	0.002673	YES
5	0.001749	0.038675	NO
6	0.002383	-	NO

Table 2.6: Saliency computed for each Gaussian in Fig. 2.14.

Gaussian nr.	Saliency	Threshold	*POI*
1	0.002341	-	NO
2	0.025118	0.002693	YES
3	0.001191	0.036059	NO
4	0.001855	0.035187	NO
5	0.066844	0.002574	YES
6	0.000679	0.035919	NO
7	0.003140	-	NO

by planning and by the slow velocity of the robot.

Figure 2.13 shows an example trajectory of an object placement task. The saliency assigned to each of the Gaussians is reported in Table 2.5. Only one Gaussian (the fourth) is classified as a salient action (placement) and generates a single *POI*. Another example, for a pick-and-place task, is shown in Fig. 2.14 and Table 2.6. In this case, only the second and the fifth Gaussians are detected as salient actions generating two *POIs*, which correspond to object picking and placing respectively.

2.4.3 Evaluation of the robot attention system

This section presents two complete experiments of the attention based system (Fig. 2.15) using NBV-GPU. Each experiment was performed two times using the same recorded user hand trajectory, the first one with KFOVP and the second with KFOM.

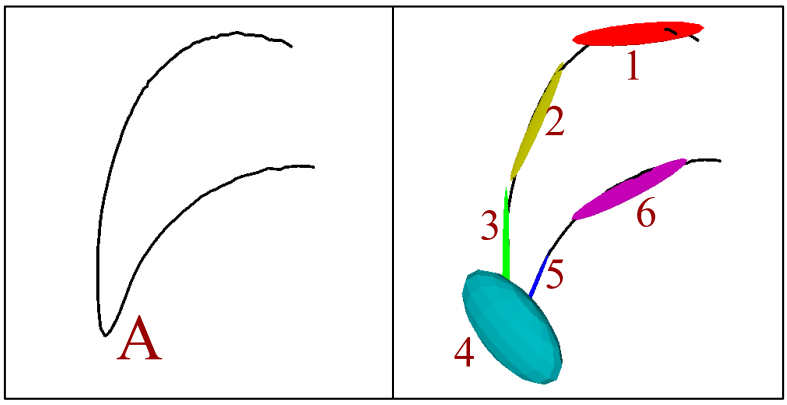


Figure 2.13: A trajectory representing an object placement task to location A (left) and the superimposed GMM (right).

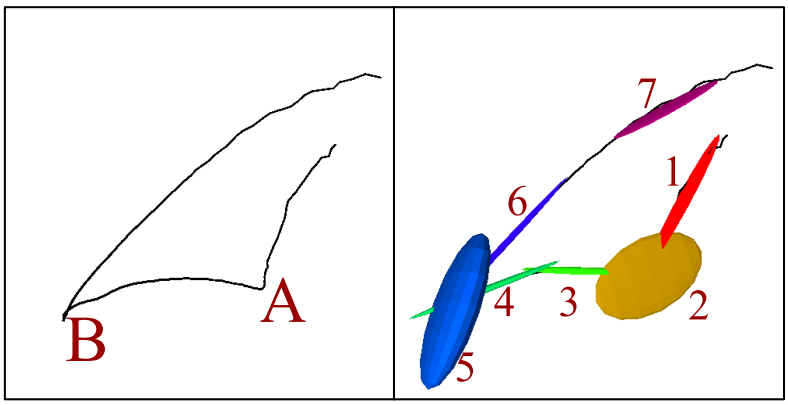


Figure 2.14: A trajectory representing a pick-and-place task from location A to B (left) and the superimposed GMM (right).

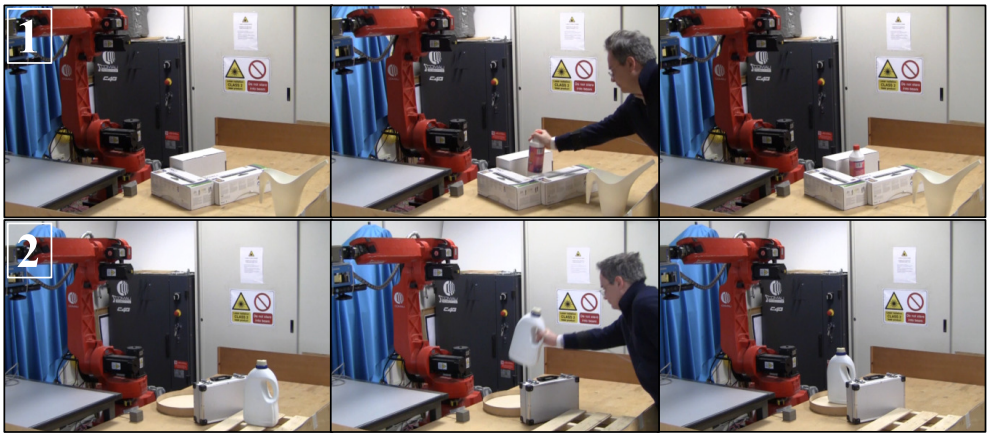


Figure 2.15: The environment before (left), during (center) and after (right) the user task. First experiment (top row) and second experiment (bottom row).

In the first experiment, the user placed a plastic bottle in the environment, among a group of boxes. In the second experiment, the user picked up a jug from a pallet and then placed the jug on a wooden disc after passing over a briefcase. The tasks generated *POIs* and corresponding regions of interest as shown in Fig. 2.16. GMM-AS generated a single correct *POI* in the first experiment. In the second experiment GMM-AS generated a total of three *POIs*, two of them almost superimposed where the jug is moved.

Figure 2.17 shows images of OpenRAVE planning, robot NBV configurations, KinFu synthetic depth maps and environment representations of KFOVP exploring the single *POI* in experiment 1. Figure 2.18 shows all the five views performed by KFOVP in experiment 2. The first view (first column) observes the empty space above the pallet where the jug is picked up (first POI). The other four views are needed to reconstruct the second and the third (almost superimposed) POIs where the jug is moved on the wooden disc.

Figure 2.19 illustrates the final result for the KFOVP experiments. The average accuracy of the reconstructed objects involved in the tasks was measured by using the same parameters introduced in Section 2.4.1. KFOM and KFOVP achieve the

Table 2.7: Reconstruction error, completeness, number of views and execution time for KFOM and KFOVP.

	Abs. error (m)	Compl.	Views	Time (s)
Experiment 1				
KFOVP	0.005	8	3	194
KFOM	0.008	9	2	191
Experiment 2				
KFOVP	0.005	10	5	402
KFOM	0.007	7	3	304

Table 2.8: Time (seconds) of the most relevant algorithm phases, averaged for each experiment.

Phase	Experiment 1		Experiment 2	
	KFOVP	KFOM	KFOVP	KFOM
Clear $S(POI)$	0.07	0.07	0.08	0.06
NBV-GPU	10.43	10.91	12.21	12.83
Robot movement	6.86	27.76	7.58	20.71

Table 2.9: Execution times for one phase of NBV-GPU, NBV-CPU and NBV-CPU-step (times in seconds).

	NBV-GPU	NBV-CPU	NBV-CPU-step
Execution time	10.6	211.5	619.6

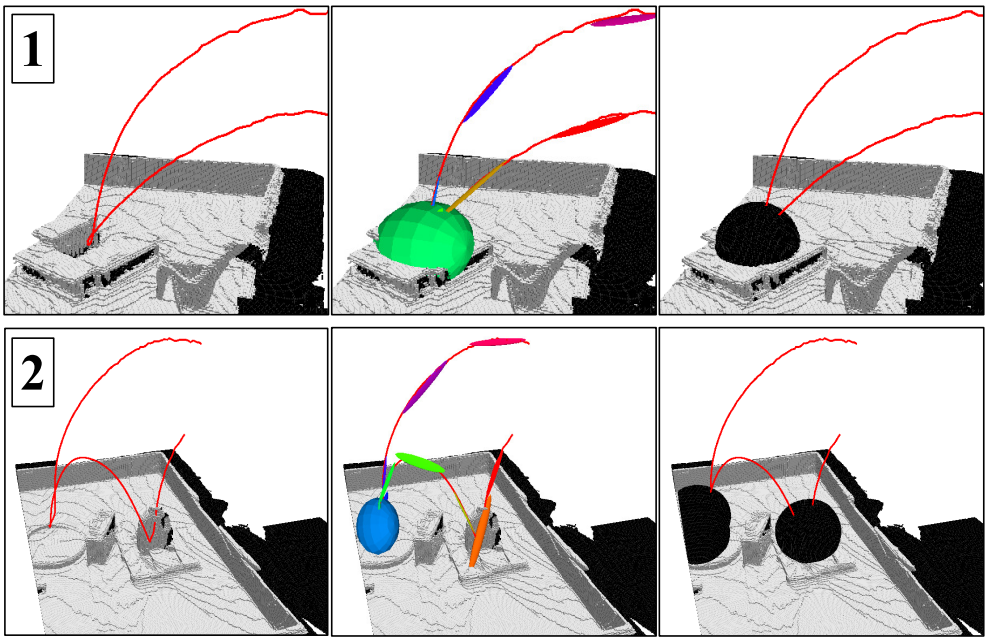


Figure 2.16: The environment 3D representation with user hand trajectory (left), GMM (center) and generated spherical regions of uncertainty (right) for the first (top row) and second experiment (bottom row) in black color.

same reconstruction accuracy as shown in Table 2.7. A qualitative score from 1 to 10 was given to evaluate the completeness of the reconstruction of each object. In the first experiment KFOM is not significantly faster than KFOVP, even if KFOM requires one less view. This can be explained by the slower robot movement during acquisition required by KFOM. The difference in total completion time is, however, higher in the second experiment where KFOVP requires two views more than KFOM. Table 2.8 shows the execution times of specific phases of the algorithm. As expected, the robot movement is slower in KFOM. In addition, for both KFOVP and KFOM the NBV-GPU algorithm is about 20% slower in the second experiment than in the first. This indicates that the computational time depends on the shape of the environment. In general, it can be concluded that there are no large differences between KFOM and KFOVP in terms of reconstruction accuracy and completeness. KFOM is slightly

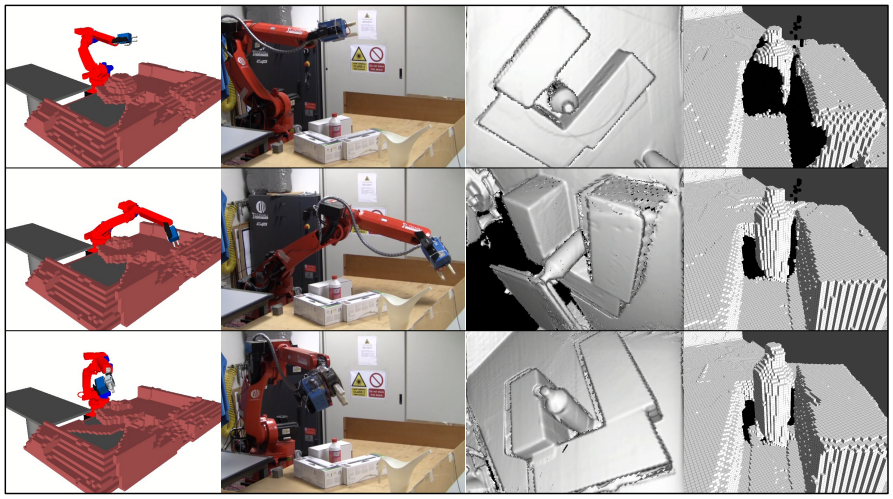


Figure 2.17: KFOVP snapshots in experiment 1. From left to right: OpenRAVE planning, robot NBV configuration, KinFu synthetic depth map and reconstruction in the ternary representation (unknown voxels are displayed in black), for each view pose (top to bottom).

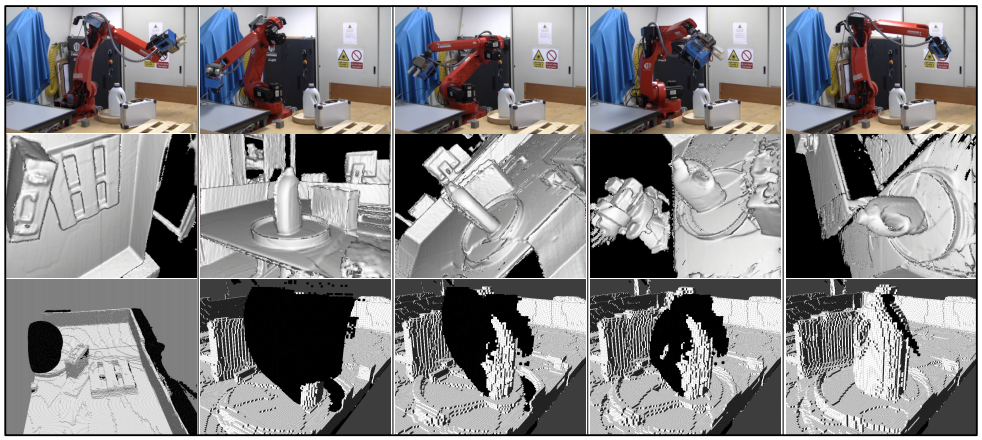


Figure 2.18: KFOVP snapshots in experiment 2. Robot NBV configuration (top row), KinFu synthetic depth map (middle row) and the reconstruction in the ternary representation (bottom row, unknown voxels are displayed in black) for each view pose (left to right).

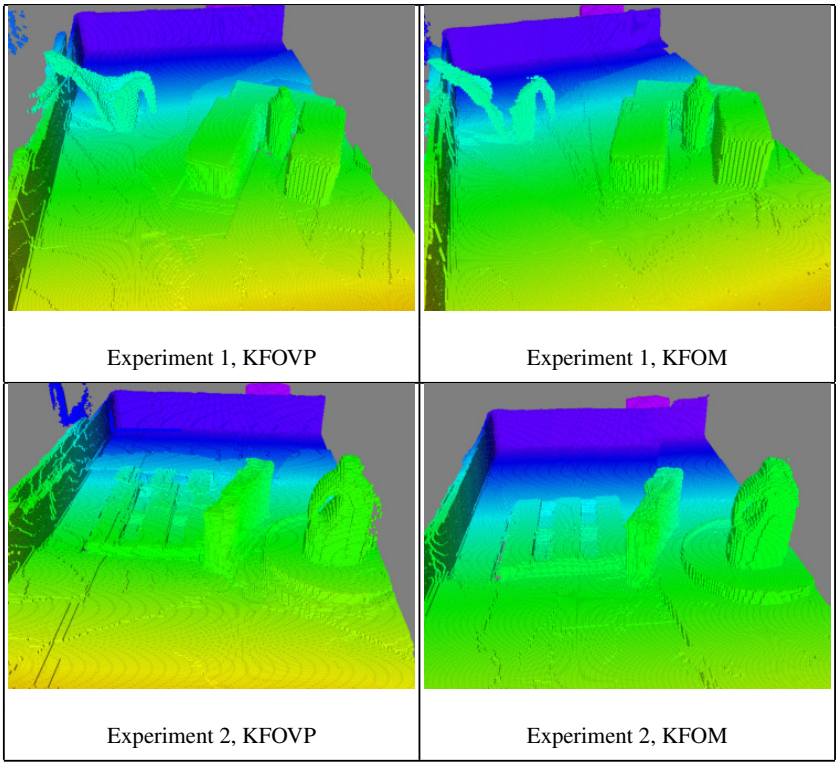


Figure 2.19: Final environment representation for KFOM and KFOVP in the two experiments.

faster than KFOVP when a large number of views is required.

Figure 2.20 shows the number of unknown voxels in the 3D representation, for each POI. The number of voxels decreases as the robot explores the environment and it never reaches 0 due to unknown voxels inside objects. In experiment 2, the number of unknown voxels in the first POI, above the pallet, decreases almost to 0 at the first view. Also, in experiment 2 it can be noticed that when using KFOM the robot observes voxels in the second and third *POI* simultaneously as it travels to the first view pose to observe the first *POI*, hence the number of unknown voxels of the second and third *POI* decreases even before NBV planning starts for these two *POIs*.

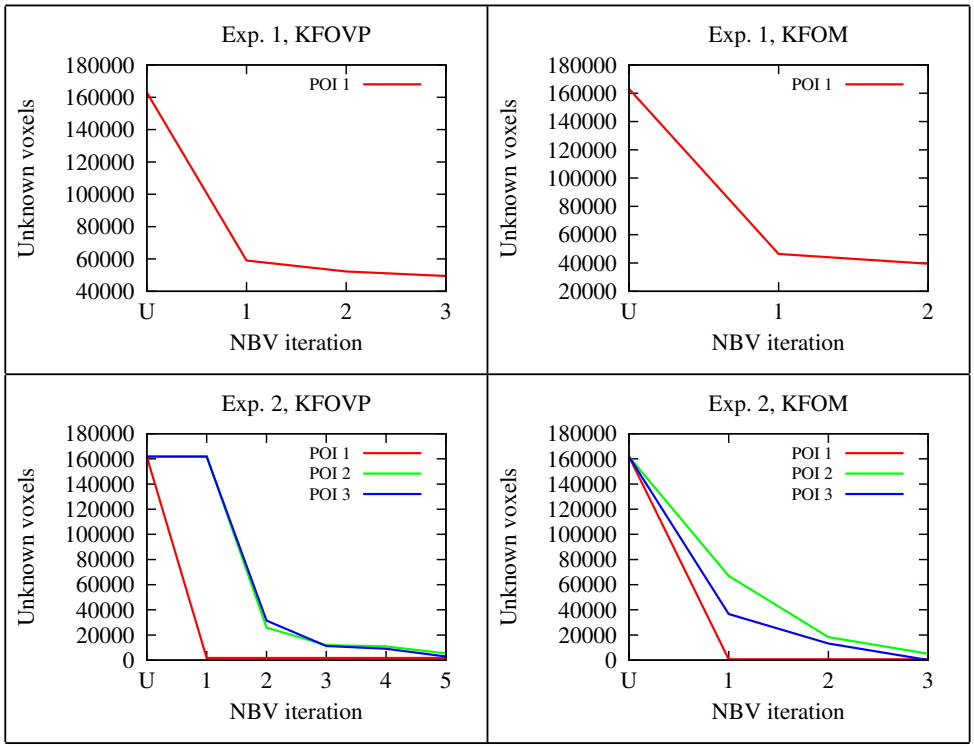


Figure 2.20: The number of unknown voxels inside each uncertainty region, after user task (U) and at each robot NBV iteration.

2.4.4 Evaluation of the developed KinFu LS extensions

A test scenario, shown in Figure 2.21, was set up to compare the performance of NBV-GPU and NBV-CPU. A single POI is detected when a plastic bottle is placed by the user on top of a box. The NBV-GPU algorithm takes about 10.6 seconds to compute the next best view, thus confirming the results found in Section 2.4.3. In contrast, when running the NBV-CPU algorithm at the same resolution the experiment was aborted due to excessive execution time. In Table 2.9 the execution times for the first NBV phase of the experiment are reported, for each NBV algorithm. The step used for the NBV-CPU-step algorithm was 0.3 cm, about half a voxel size.

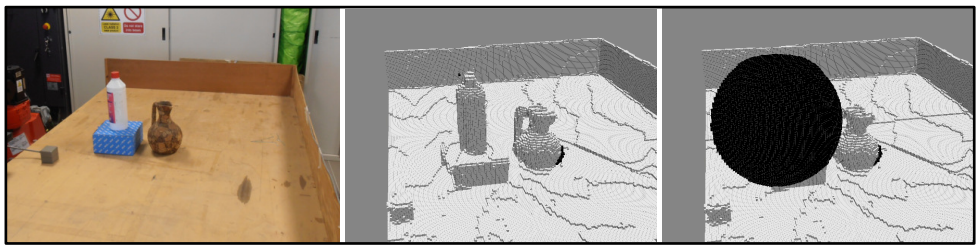


Figure 2.21: Environment (left), ternary representation (center) and POI (right), for the experiment described in section 2.4.4.

Table 2.10: Comparison of overhead in standard KinFu LS and NBV-GPU (times in ms).

Phase	Std KinFu LS	NBV-GPU
Shifting	830	836
weights download	0	283
read octree (parallel)	0	3396
weights upload	0	45
write octree (parallel)	0	1638
Total time	830	6232

These results confirm the advantage in terms of efficiency of using NBV-GPU, i.e. executing the ray casting phase on the GPU. In Table 2.10 the computational overhead introduced by NBV-GPU when downloading and uploading weight data between the *BitmaskOctree* and the TSDF volume is analyzed. A shifting operation takes about 6 seconds, compared to 800 ms of the standard KinFu LS implementation. However, thanks to the developed multithreading architecture, read and write operations on the octree run in parallel. Indeed, the actual overhead with respect to the standard KinFu LS implementation is only about 283 ms, which is due to the weight download phase, when KinFu must be locked exclusively. An additional exclusive lock must be acquired when uploading on the GPU the stored weight values from the CPU, however this phase requires only 45 ms.

2.5 Discussion

In this chapter, an attention strategy was applied to next-best view, in order to maintain an updated representation of a tabletop environment. The system presented here is user-driven, i.e. it is triggered by the user actions. Saliency is estimated using a GMM-based algorithm, and approximate locations of the salient user actions are computed. The next-best view planner focuses on those locations, thus exploiting the saliency information. The robot moves the range sensor to observe the locations, in order to update the 3D representation.

By focusing on the salient locations, the robot is able to update the representation without re-scanning the whole environment. The robot next-best views are focused on the relevant parts of the environment. Moreover, the tradeoff between two possible strategies has been evaluated. In the KFOVP strategy, the sensor acquires data only from the viewpoint estimated by the next-best view algorithm. In the KFOM strategy, data is acquired also during robot movement. It has been shown that the KFOM requires less views, but slower robot movement.

The system presents a number of novel research contributions. First of all, the error of KinFu egomotion tracking has been evaluated. The error has been shown to be too high for proper operation of the system. Therefore, the KinFu tracking was replaced with robot forward kinematics. Moreover, a GMM-AS algorithm is presented, which estimates the location of salient actions using only the user hand trajectory. The execution of next-best view on GPU, exploiting the very same ray casting mechanism of KinectFusion, allows for a much faster evaluation than the common CPU-based approach.

A few possible extensions of the system have not been investigated in this chapter. Human motion analysis may be enhanced by exploiting both skeletal tracking and 3D information from the Kinect sensor itself. Moreover, the difficult task of next-best trajectory, which plans optimal sensor motion trajectories instead of single poses, has not been investigated. The 3D representation is initialized by an exploration phase on a pre-defined trajectory, which is supposed to be collision-free and to produce a reasonably complete 3D reconstruction. Indeed, the proposed attention system cannot

be used as an exploration system, since it is triggered by the user actions. Instead, the object-driven attention system proposed in the next Chapter 3 is an exploration system, which exploits attention to produce a complete reconstruction of a static scene.

Chapter 3

Object-driven spatial attention

In this chapter, the concept of attention is applied to an exploration system for a static scene [13]. User activity is not expected in this scenario, but attention is driven by the objects detected in the scene.

The system is composed of the already mentioned 6-DoF robot manipulator and a Kinect V2 sensor in eye-in-hand configuration. The robot is tasked with the exploration of a tabletop scenario. The scene is initially unknown, and the robot moves the sensor along a short pre-defined trajectory on one side, in order to acquire data and initialize the representation. Then, the goal of the robot is to complete the 3D reconstruction of the tabletop scene.

The usual active perception behavior drives the robot by iteratively computing the next-best view and moving the robot to acquire new information. Next-best view algorithms attempt to maximize the information gain by exploring unknown or incomplete parts of the scene. They do so by maximizing of the area of unknown space visible from the next view pose. However, in this case the robot may prioritize large occluded areas that do not contain any interesting object. In particular, the robot may aim to explore the large unexplored volume at the edge of the scene, without completing already discovered objects.

To address this issue, in this chapter a novel approach for next-best view planning which exploits attention is proposed. A high saliency value is given to the incomplete

discovered objects in the scene, to prioritize them before looking for more information in the unexplored regions of the environment. The saliency is attributed without any prior knowledge about the object shape and position. Such non-model-based approach is achieved by applying a point cloud segmentation algorithm and then by assigning a saliency value to each segment. A heuristic method is developed to identify meaningful segments that belong to the objects. The point cloud segmentation method is based on Locally Convex Connected Patches (LCCP) [95], which is available within the Point Cloud Library. The approach is fully autonomous and it does not assume the existence of a dominant plane in the scene.

The development of the system presented in this chapter involves some additional improvements with respect to the state of the art. Firstly, the KinFu- and GPU-based next-best view system presented in the previous chapter is adapted to an exploration task. In particular, it is shown how candidate viewpoint directions can be extracted directly from the KinFu TSDF volume, using a local contour extraction algorithm. Moreover, version 2 of the Kinect sensor is used. This sensor is based on time-of-flight technology, unlike version 1 which is based on structured light triangulation principle. While more accurate, time-of-flight technology produces some undesired artifacts. In order to remove these artifacts, a novel procedure has been developed for Kinect V2 depth image pre-processing.

The chapter is organized into three sections. Section 3.1 describes the proposed active perception system. Section 3.2 illustrates the experiments and the results. Section 3.3 discusses the results and provides further insights.

3.1 Proposed method for next-best view planning

In traditional non-model-based approaches next-best view planning (*NBV*) is performed in two phases. In the first phase, candidate view poses are generated. In the second phase, all the poses are evaluated according to a score function to find the next-best view pose. The proposed pipeline to compute the NBV, illustrated in Fig. 3.1, differs from traditional approaches as it introduces an intermediate phase between viewpoint generation and evaluation. In the intermediate phase the input point

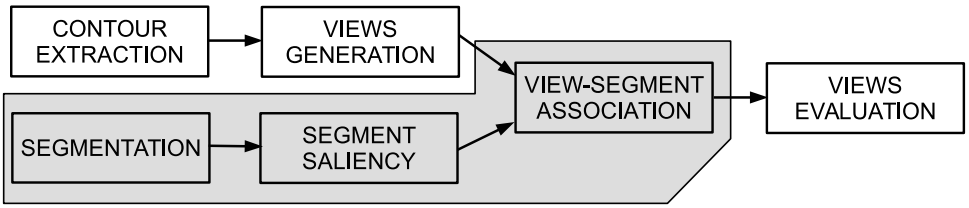


Figure 3.1: Pipeline of the view planning algorithm. The grey background highlights the intermediate phase

cloud is segmented into clusters and a saliency value is computed for each point cloud segment. The aim of the point cloud segmentation phase is to automatically detect segments that belong to the objects of the scene. In the evaluation phase potential view poses are associated to point cloud segments and the NBV is searched among view poses in decreasing order of segment saliency.

A more detailed overview of the proposed view planning pipeline is reported in Algorithm 4. The view generation phase is performed by a contour extraction algorithm (line 1), detailed in Section 3.1.1, which extracts contour points, i.e. points at the border of incomplete surfaces. Contour extraction also produces a view direction for each contour point. Then, from each view direction multiple view poses are generated, as shown in Fig. 3.2, mainly to increase the probability of finding a reachable pose for the robot manipulator. In particular, for each direction four additional view directions towards the same contour point are sampled within a small solid angle (15 degrees). To convert each view direction into a pose for the sensor, a distance from the contour point must be selected, compatible with the sensor minimum and maximum sensing distance. In theory, view poses may be generated by selecting multiple distances. However, for the experiments shown in next section 3.2 a fixed distance of 80 cm was adopted, which was empirically determined by evaluating the average maximum distance that the robot is able to reach from the objects in the current experimental setup. In addition, a rotation angle around the view direction must be chosen. Eight samples for each view direction are generated at 45 degrees intervals, starting from an arbitrary initial orientation.

Algorithm 4 View planning

Require: *WS*: 3D volumetric environment representation
Ensure: Next-best view

1: $Contour \leftarrow$ ContourExtraction(WS)
2: $\forall_{b \in Contour} \quad b.Viewpoints \leftarrow$ GetViewpoints(b)
3: $PointCloud \leftarrow$ ExtractSurfacePts(WS)
4: $Segments \leftarrow$ SegmentPointCloud($PointCloud$)
5: $Saliency \leftarrow$ SegmentSaliency($Segments$)
6: $Segments \leftarrow$ OrderBy($Segments$,$Saliency$)
7: **for** i **from** 1 **to** size($Segments$) **do**
8: $\quad SContour \leftarrow$ FindNear($Contour$,$Segments[i]$)
9: $\quad SViewpoints \leftarrow \bigcup_{b \in SContour} b.Viewpoints$
10: $\quad Scores \leftarrow$ EvaluateViewpoints($SViewpoints$)
11: $\quad SViewpoints \leftarrow$ OrderBy($SViewpoints$,$Scores$)
12: $\quad$ **for** j **from** 1 **to** size($SViewpoints$) **do**
13: $\quad\quad$ **if** $Scores[j] > ScoreTH$ **then**
14: $\quad\quad\quad$ **return** $SViewpoints[j]$
15: $\quad\quad$ **end if**
16: $\quad$ **end for**
17: **end for**
18: **return** {no suitable viewpoint found}

In line 3, a point cloud (*PointCloud*) is extracted from the TSDF volume using the marching cubes algorithm, already available in KinFu. Marching cubes generates a mesh from the isosurface between positive (empty) and negative (occupied) TSDF voxels. The vertices of the mesh define the point cloud. In the segmentation phase (line 4) the point cloud is segmented using the LCCP algorithm. Then, a saliency value is computed for each segment (line 5), as described in Section 3.1.2. Finally, the segments are ordered by decreasing saliency (line 6).

In the viewpoint evaluation phase (lines 7-17) view poses are associated to segments and are processed by decreasing segment saliency. In particular, all contour points close to the current segment are determined (line 8). Given the set P_C ($\equiv$ *PointCloud*) of all points in all segments, a contour point p is close to the current

Figure 3.2: Flowchart of viewpoint generation (line 2 in Algorithm 4). Each candidate view direction generates 40 view poses for the sensor.

segment S if the nearest point to p in P_C belongs to S. All view poses generated by the contour points of the current segment are then retrieved (line 9). View poses associated to a segment are evaluated by assigning a score proportional to the expected information gain, as in traditional NBV approaches. Indeed, the expected information gain of each view pose is given by the amount of unknown voxels visible from that pose. The amount of unknown voxel is computed as stated in previous Chapter 2. A voxel contributes to the score only if it falls inside a sphere with radius 20 cm larger than the bounding sphere of the segment.

View poses associated to the current point cloud segment are then ranked and processed in decreasing order of score. If the expected information gain of a view pose exceeds a threshold value (line 13) that pose is considered the NBV. Otherwise, if the expected information gains of all the view poses of the current segment are below the threshold, the algorithm moves to evaluate the view poses of the next most salient segment. In summary, the proposed procedure is aimed at giving priority to active exploration of salient segments of unknown objects, not fully reconstructed, rather than favoring viewpoints that blindly try to minimize the size of the unknown space.

3.1.1 Contour extraction from TSDF volume

As explained in Chapter 2, the TSDF volume is a volumetric representation of the environment used by the KinectFusion algorithm. The space is subdivided into a regular 3D grid of voxels and each voxel holds the sampled value $v(x,y,z)$ of the Truncated Signed Distance Function $R^3 \rightarrow R$, which describes the signed distance from the nearest surface, clamped between a minimum and a maximum value. The TSDF is positive in empty space and negative in occupied space. Each voxel also

contains a weight w, initialized to 0, that counts the number of times the voxel has been observed, up to a maximum amount. The TSDF value v and the weight can be used to distinguish between empty, occupied and unknown voxels as shown in Equation 2.11, which is recalled here for convenience:

$$\begin{cases} w = 0 & \rightarrow \textit{unknown voxel} \\ w > 0 & \begin{cases} v \leq 0 & \rightarrow \textit{occupied voxel} \\ v > 0 & \rightarrow \textit{empty voxel} \end{cases} \end{cases} \tag{3.1}$$

Rarely observed voxels have a low weight, while completely unknown voxels have 0 weight. In unexplored space, or deep inside the surface of objects, voxels are unknown.

In NBV planning a frontier is defined as the region between seen-empty voxels and unknown space. A frontier is a region that can be explored, since the viewing sensor might be placed in the empty space next to the frontier to observe the unknown space. Occupied voxels do not belong to the frontier, since the sensor can not see through them. However, occupied voxels lying next to a frontier have implications for NBV planning. Indeed, observation of the region of space in close proximity to occupied voxels next to a frontier can extend the perception of the surface of the object those occupied voxels belong to.

In this chapter, a contour is defined as the set of empty voxels that are near to occupied voxels next to a frontier, i.e. a contour consists of empty voxels that are near to both an occupied voxel and an unknown voxel. To exclude false positive known voxels from being processed, due to noise, a voxel is considered known if observed at least 5 times, i.e. $w \geq W_{th}$, where $W_{th} = 5$ is a lower bound threshold. Given the 6-connected neighborhood N_e^6 and the 18-connected neighborhood N_e^{18} of a voxel at position e, the voxel belongs to a contour if the following conditions hold:

$$\begin{cases} w(e) \geq W_{th} \wedge v(e) > 0 \\ \exists\, u \in N_e^6 \mid w(u) < W_{th} \\ \exists\, o \in N_e^{18} \mid w(o) \geq W_{th} \wedge v(o) \leq 0 \end{cases} \tag{3.2}$$

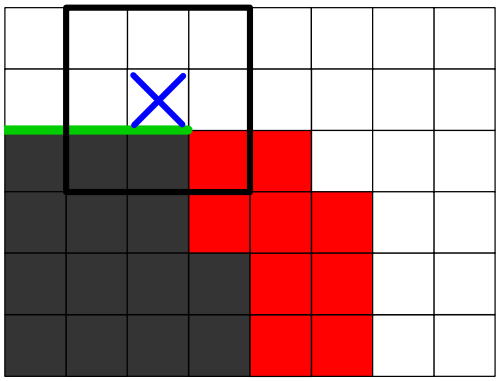
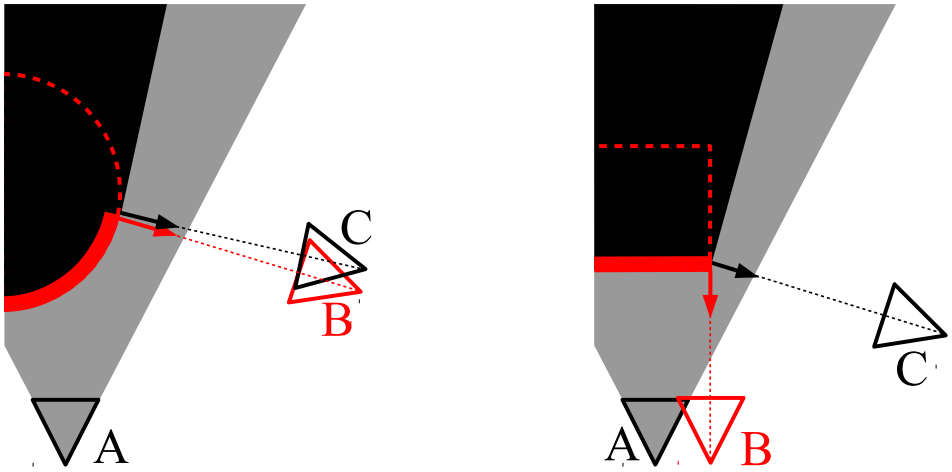


Figure 3.3: A simplified 2D example of the contour extraction algorithm using Von Neumann neighborhood (4-connected) and Moore neighborhood (8-connected). In the previous view the sensor observed the object from the right side. A computed contour cell is marked with the cross. The thicker square highlights the Moore neighborhood of the contour cell. The green segment represents a frontier. Known and occupied cells are displayed in red, known and empty cells are in white, unknown cells are in dark grey.

A simplified 2D example is shown in Fig. 3.3, using the Von Neumann neighborhood (4-connected) and the Moore neighborhood (8-connected) in place of the 6-connected neighborhood N_e^6 and the 18-connected neighborhood N_e^{18} used in the 3D case. In the previous view the sensor observed the object from the right side, thus the view was partially obstructed and the cells in the lower left part of the image are not observed and left unknown. The cross marks a computed contour cell.

Given the previous definitions a method to compute a potential view direction from each contour voxel is described next. For optimal observation, the sensor should observe the object perpendicularly to its surface. Thus, if the object were fully known, the opposite of the surface normal computed on the occupied voxel next to the contour voxel could be used as potential view direction. The normal to the surface can be computed from the TSDF volume as the gradient $\nabla v\,(x,y,z)$ of v.

Given a neighborhood N_e of a voxel at position e, the normal may be approxi-



(a) (b)

Figure 3.4: Generation of the next potential viewpoint for (a) a rounded object (b) an object with sharp edges. Viewpoints B (computed from the surface normal) and C (computed from the frontier normal) are very similar in case (a), but not in case (b). Only in case (b) viewpoint C allows the observation of the unknown volume behind the sharp edge.

mated as (normalization omitted):

$$n_e = \sum_{c \in N_e} v(c) \cdot \frac{c-e}{\|c-e\|} \tag{3.3}$$

which, for a 6-connected neighborhood, reduces to

$$n_e = \begin{bmatrix} v(x+1,y,z) - v(x-1,y,z) \\ v(x,y+1,z) - v(x,y-1,z) \\ v(x,y,z+1) - v(x,y,z-1) \end{bmatrix} \tag{3.4}$$

since $(c-e)/\|c-e\|$ are unary vectors of the coordinate system.

The limitation of this approach is shown in Figures 3.4a and 3.4b. In both examples the sensor takes a first observation from the bottom, at position A. The observed

volume is displayed in light grey. An object, marked with a dashed line, is partially observed in the red region. The volume behind the object remains unknown (black). The surface normal for the computed contour cell is displayed as a red arrow pointing outside the object. The generated potential viewing pose (B) from the surface normal is shown as the red triangle. In Fig. 3.4a for a rounded object the surface normal provides a good direction for the next view. However, for objects with sharp edges (like boxes), as illustrated in Fig 3.4b, the normal at the edge does not reflect the trend of the object in the unknown space behind the edge. Thus, the normal at the contour cell does not provide a suitable view direction since it does not allow the observation of the region of the object in the unknown space. Indeed, in this second example at location B the sensor can not acquire any new information, since the lower plane of the box has already been observed from the initial view.

To overcome this limitation, in this section a method is proposed, that computes the potential view directions using the normal to the frontier, i.e. the normal to the unknown volume. The normal to the frontier is indicated as view C. While in Fig. 3.4a for a rounded object the viewing pose is rather similar to the one computed by the surface normal, in Fig. 3.4b for a sharp edge view C provides a much better view direction to observe the object from the side.

The normal of the frontier may be approximated locally using the gradient of the weight function $\nabla w(x,y,z)$ which can be computed as

$$n_e = \sum_{c \in N_e^{26}} w'(c) \cdot \frac{c-e}{\|c-e\|} \tag{3.5}$$

where the 26-connected neighborhood of a voxel is used to reduce noise and sampling effects.

Since $w(c)$ is a positive integer value, equation 3.5 uses a modified weight function w' defined as

$$w'(c) = \begin{cases} -W_{th} & \text{if } c \text{ occupied} \\ \min\left(w(c) - W_{th}, W_{th}\right) & \text{otherwise} \end{cases} \tag{3.6}$$

For occupied voxels weight w is set to $-W_{th}$, since the normal should point away from them. Otherwise, w is first centered around 0 and then truncated to W_{th}.

In practice, after extraction of all the contour voxels with their view directions (line 1 in Algorithm 4), similar contour voxels are reduced into a contour point by a region growing algorithm. Two contour voxels at position e_1 and e_2, with view direction n_1 and n_2 are considered similar if

$$\begin{cases} \|e_1 - e_2\| < D_{th} \\ \|n_1 \cdot n_2\| < A_{th} \end{cases} \tag{3.7}$$

Each group of similar voxels is reduced to a single contour point with an associated view direction by averaging the positions and the view directions of the voxels.

Figures 3.5 and 3.6 show an example of contour extraction and viewpoint computation. In Fig. 3.5 the sensor observes a jug from the current NBV and a partial 3D representation is produced by KinFu. As shown by the ternary volumetric representation, voxels behind the object remain unknown. Contour voxels are extracted and clustered as illustrated in Fig. 3.6. The normal vectors point outwards towards the empty space. Thus, from that directions the robot may be able to observe the unknown space behind the object.

3.1.2 Saliency of point cloud segments

This section illustrates how the segmentation of the point cloud, extracted from the TSDF volume, is performed and how the saliency value of each segment is computed (lines 4-5 in Algorithm 4). The procedure is illustrated in Fig. 3.7. The point cloud is segmented by the LCCP [95] algorithm, available in the PCL library. LCCP partitions the input point cloud into a set of *Segments* (line 4 in Algorithm 4) by merging patches, called supervoxels, of an over-segmented point cloud. Supervoxels are generated by the a Voxel Cloud Connectivity Segmentation algorithm (VCCS) by [73].

VCCS requires knowledge about the normals to the point cloud. Such vertex normals can computed with minimum overhead by using the gradient of the TSDF volume, as shown by equation 3.3 in Section 3.1.1, using a 6-connected neighborhood which is is already available for marching cubes operations.

The saliency function is a heuristic model that should provide an objectness measure, i.e. it should provide higher values for segments that belong to real objects of the

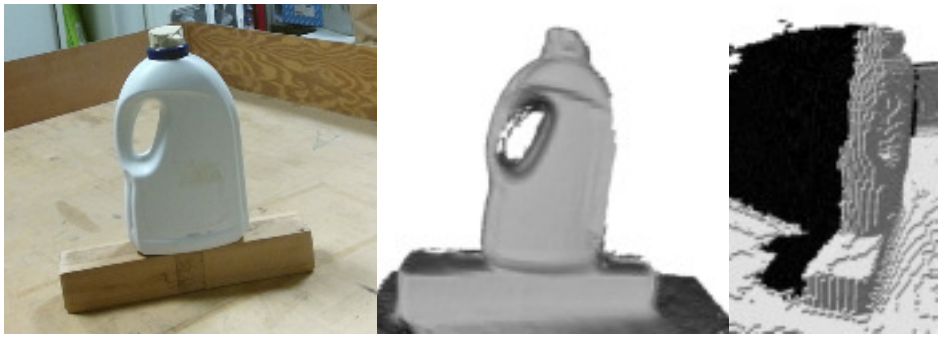


Figure 3.5: Left: a jug observed from the current sensor viewpoint. Center: the 3D mesh reconstructed by KinFu. Right: the volumetric representation (rotated view), with occupied (white) and unknown (black) voxels.

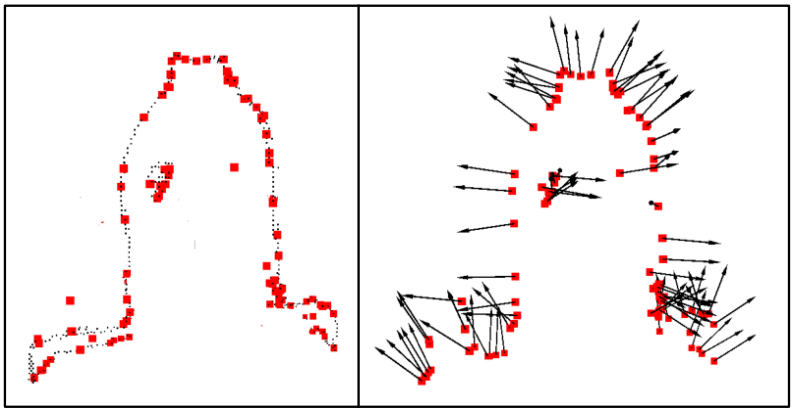


Figure 3.6: Left: contour voxels (black) and the contour points (red). Right: contour points with normals. A contour point represent a group of similar contour voxels.

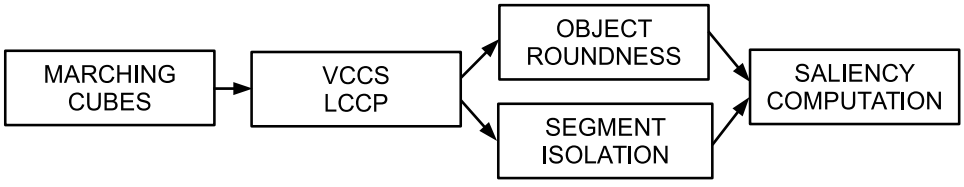


Figure 3.7: Proposed procedure for point cloud segmentation and computation of the saliency value of each segment.

scene. Here, the saliency of each segment is computed as a function of two features: the segment roundness and the degree of isolation.

The roundedness of a segment S is estimated as the ratio of the minimum and maximum sizes of the Oriented Bounding Box (OBB) of S. The sizes (d_1, d_2, d_3) of the OBB are defined in a local reference frame T_{OBB} centered at the mean point of the segment whose axes are given by the eigenvectors of the covariance matrix of the points (principal axes of inertia). More formally,

$$\begin{aligned} d_1 &= \max_{c \in S}(c'_x) - \min_{c \in S}(c'_x) \\ d_2 &= \max_{c \in S}(c'_y) - \min_{c \in S}(c'_y) \\ d_3 &= \max_{c \in S}(c'_z) - \min_{c \in S}(c'_z) \end{aligned} \tag{3.8}$$

where c is a point of S in the world reference frame and c' is the transformed point in the local reference frame

$$c' = T_{OBB}^{-1} \cdot c \tag{3.9}$$

The minimum and maximum sizes of the OBB of S are then

$$\begin{aligned} d_{max} &= \max_{i \in \{1,2,3\}}(d_i) \\ d_{min} &= \min_{i \in \{1,2,3\}}(d_i) \end{aligned} \tag{3.10}$$

The degree of isolation of a segment is defined as the fraction of points for which the distance to points belonging to other segments is at least B_{th}. Given a segment $S \in Segments$ and the set $\hat{S}$ of all the points not in S, the degree of isolation of S is given by

$$F(S) = \frac{\left\| \left\{ c \in S \mid \forall o \in \hat{S},\ |c-o| > B_{th} \right\} \right\|}{\|S\|} \tag{3.11}$$

where $\|S\|$ is the total number of points in S. Equation 3.11 can be efficiently computed using a KdTree radius search of size B_{th}. Feature F has three benefits. First, it is meant to reward isolated segments belonging to partially observed objects, since a large part of their boundary is not shared with any other segment. Second, this heuristic is helpful for noise rejection as noisy segments, not well separated from

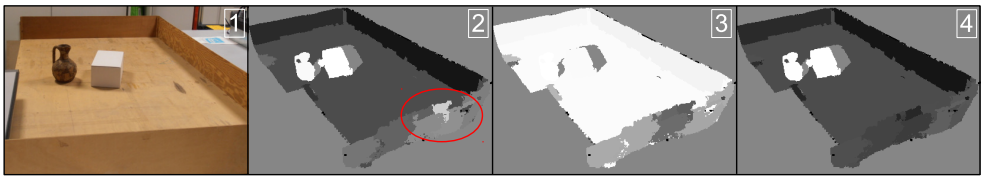


Figure 3.8: Example of point cloud segmentation and saliency evaluation. Brighter segments have higher saliency value. (1): a picture of the scenario, (2): saliency evaluated by segment roundness alone, (3): saliency evaluated by segment isolation alone, (4): saliency evaluated by both segment roundness and isolation according to equation 3.12. The segment isolation factor reduces the saliency of the noisy segments inside the red ellipse.

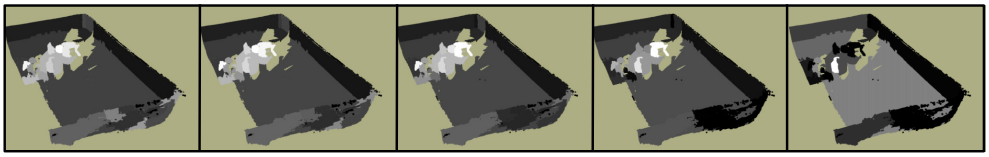


Figure 3.9: Saliency computed after the initial scan in experiment 2 described in Section 3.2.2, using $B_{th} \in \{0.005, 0.01, 0.02, 0.05, 0.1(m)\}$ (from left to right).

other segments, often have a large boundary. Third, equation 3.11 penalizes small segments.

Finally, the saliency value of a segment S is computed as

$$Saliency(S) = F(S) \cdot \frac{d_{min}}{d_{max}} \tag{3.12}$$

so that saliency is proportional to the degree of segment isolation and it grows the more the maximum and the minimum sizes of the OBB are similar. An example of a segmented point cloud with saliency values is shown in Fig. 3.8. It can be noted that the segment isolation factor reduces the saliency value of noisy segments (inside the red ellipse).

Figure 3.9 shows the effect of B_{th} on the saliency. As B_{th} increases, the saliency value of the noisy segments at the front decreases. However, when B_{th} is too high, all

the points of the small segments are rejected and, therefore, small objects assume a zero saliency value (black color). Hence, for the experimental evaluation reported in Section 3.2, the value $B_{th} = 0.02$ m was chosen.

3.1.3 Kinect V2 depth image pre-processing

This section describes a low-level pre-processing filter to improve the quality of Kinect V2 depth data. The Kinect2 driver (Freenect2) provides two pre-processing filters: a bilateral filter and an edge-aware filter. The proposed filter is executed at the end of the standard filtering pipeline in place of the edge-aware filter, which does not strongly contribute to the removal of invalid points. It is a known issue that Kinect V2 often produces incorrect measurements near the borders of occluded surfaces, as shown in Fig. 3.10. In this section, two types of invalid points are detected and removed from the depth map.

Points visible by the camera but falling in the shadow of an IR emitter have a low accuracy. In this context, these points are called *shadow points*. Shadow points appear in a blind area, created by the displacement between the IR emitter and the camera (Fig. 3.11), which is approximately $\Delta = 8$ cm.

The filter presented here is less concerned about depth image restoration of the regions that are not directly observed by the camera, because a 3D reconstruction system observes the same region of space from multiple viewpoints and the measured data are merged by KinFu. Instead, the main goal is the removal of invalid points, so they do not appear in the 3D reconstruction in the first place.

Shadow points appear in the regions of occlusion where a background object is observed only by the camera, but it is not illuminated by the IR emitter (yellow areas). The geometry of the sensor field is illustrated in Fig. 3.12. Let u and v be the horizontal and vertical image coordinates of the sensor, starting from the upper left corner. Let also be the intrinsic parameters of the IR camera defined as follows: $[f_u, f_v]$ the focal lengths, $[m_u, m_v]$ the principal point, $[u_{max}, v_{max}]$ the depth image size and $[\Delta, 0]$ the displacement between the IR emitter from the IR camera, which are aligned horizontally. Given a measured distance z_{uv} along the sensor axis z at image coordinates $[u, v]$, the coordinates of the measured point referred to the IR camera are

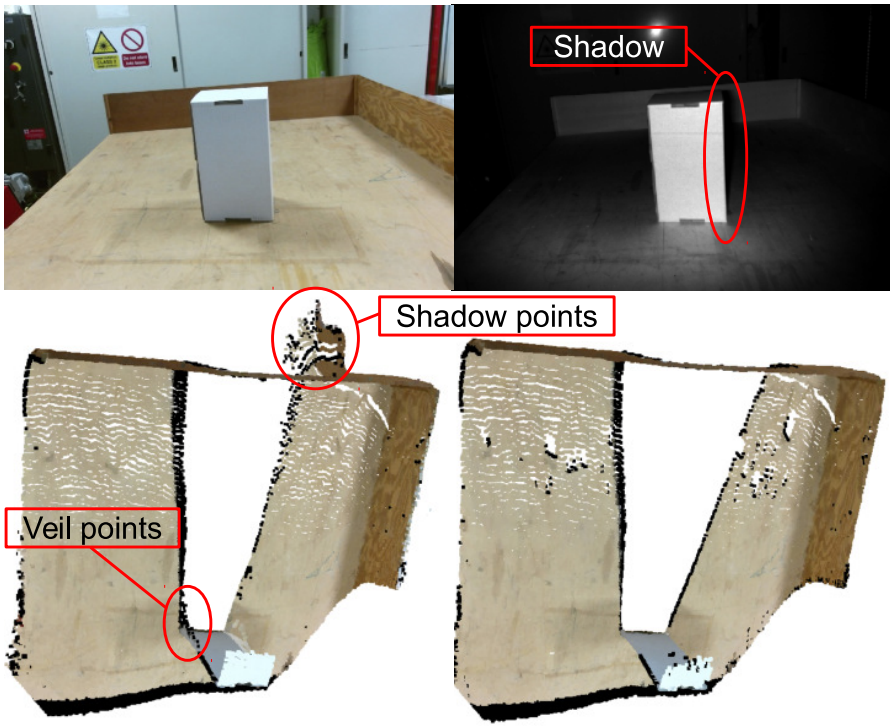


Figure 3.10: Top left: a scene as seen from the sensor. Top right: the image from the depth camera. Lower left: the point cloud acquired by the sensor, filtered by the Freenect2 driver. Lower right: the point cloud filtered by the proposed method: both shadow points and veil points are correctly removed.

Figure 3.11: The Kinect V2 sensor with IR camera, RGB camera and IR emitters.

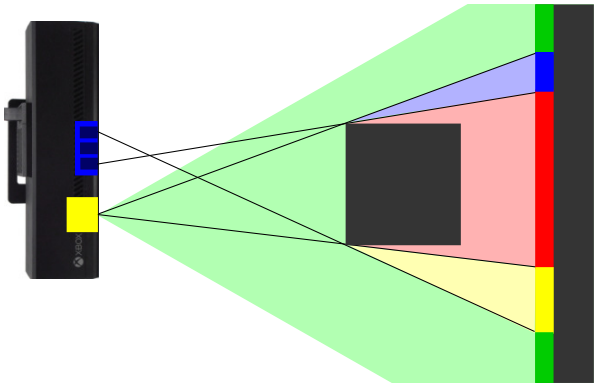


Figure 3.12: The Kinect V2 (on the left) observes a scene composed by an object (in the center) and a background plane (on the right). The object partially occludes the background plane. Three kinds of occlusions are possible: camera only (yellow), IR emitter only (blue), both (red).

given by

$$\begin{aligned} x_{uv} &= \tfrac{u-m_u}{f_u} \cdot z_{uv} \\ y_{uv} &= \tfrac{v-m_v}{f_v} \cdot z_{uv} \end{aligned} \tag{3.13}$$

while the horizontal angle, shown in Fig. 3.13, referred to the camera is

$$\alpha_{uv} = \operatorname{atan}\left(\frac{x_{uv}}{z_{uv}}\right) + \frac{\pi}{2} = \operatorname{atan}\left(\frac{u-c_u}{f_u}\right) + \frac{\pi}{2} \tag{3.14}$$

which is monotonically increasing with respect to u. However, when referred to the leftmost IR emitter, the x coordinate becomes

$$x'_{uv} = \frac{u-m_u}{f_u} \cdot z_{uv} + \Delta \tag{3.15}$$

and the horizontal angle becomes

$$\beta_{uv} = \operatorname{atan}\left(\frac{x'_{uv}}{z_{uv}}\right) + \frac{\pi}{2} \tag{3.16}$$

Unlike α_{uv}, the value of β_{uv} is not monotonically increasing with respect to u. It can be observed that an increase in u which causes a decrease in β_{uv} means that the

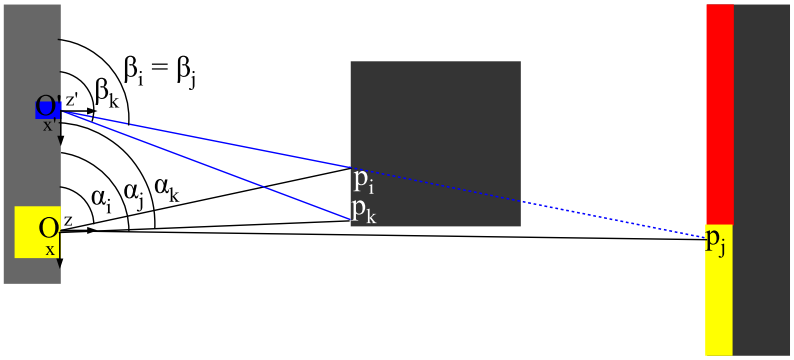


Figure 3.13: Illustration of the horizontal angles α and β of the camera and IR emitter with respect to the observed points.

depth measurement z_{uv} suddenly increased, i.e. the sensor is no longer observing an occluding object but the object behind it.

Let p_j be an observed point in the shadow of the IR emitter (yellow area in Fig. 3.13). Let also be α_j the angle from the camera origin, computed using Eq. 3.14. There exists a point p_i on the object along the illumination ray $\overline{O'p_j}$. There also exists a point p_k inside angle $\widehat{OO'p_j}$ that belongs to the object. At least $p_k \equiv p_i$ can be chosen. Since p_k belongs to $\widehat{OO'p_j}$, then $\beta_k \geq \beta_j$. Point p_k also belongs to the interior of angle $\widehat{O'Op_j}$ as the object does not intersect segment $\overline{Op_j}$. Then, $\alpha_k < \alpha_j$. Therefore, a necessary condition for a point p_j being in shadow is the existence of a point p_k that satisfies both $\beta_k \geq \beta_j$ and $\alpha_k < \alpha_j$.

Thus, the depth measurements are removed if:

$$\beta_j \leq \max_{k \,|\, \alpha_k < \alpha_j} \beta_k \tag{3.17}$$

i.e., since α is monotonic with respect to u:

$$\operatorname{atan}\left(\frac{x'_{uv}}{z_{uv}}\right) \leq \max_{k \in \{0..(u-1)\}} \operatorname{atan}\left(\frac{x'_{kv}}{z_{kv}}\right) \tag{3.18}$$

which can be efficiently computed in parallel for each $v \in \{0..(v_{max}-1)\}$ as shown in Algorithm 5.

Algorithm 5 Kinect V2 shadow points removal

Require: v: vertical coordinate
Require: z_{uv}: depth image
1: $\beta'_{max} \leftarrow -\infty$
2: **for** u **from** 0 **to** $u_{max} - 1$ **do**
3: $x' \leftarrow \frac{u-c_u}{f_u} \cdot z_{uv} + \Delta$
4: $\beta' \leftarrow \frac{x'}{z_{uv}}$
5: **if** $\beta' \leq \beta'_{max}$ **then**
6: RemovePoint(u,v)
7: **else**
8: $\beta'_{max} \leftarrow \beta'$
9: **end if**
10: **end for**

Although it is very likely that a point p_k is observed by the sensor, since the object is near the sensor and the resolution is very high, condition 3.17 is still heuristic. Indeed, in real scenarios an object may be closer to the sensor than the Kinect V2 minimum range, hence p_k may not be really observed. Moreover, only a necessary condition was demonstrated. Indeed, some valid points may be misclassified as shadow points.

In the pre-processing phase invalid points called veil points are also removed as shown in Fig. 3.10. Veil points are caused by the lidar technology, which tends to interpolate points near the object border with the background. Veil points are removed if an angle higher than $\Theta_{max} = 10^\circ$ is detected with respect to the observing ray. In particular, given a point p_i on the depth image, the point is removed if there is a point p_k in its Von Neumann (4-connected) neighborhood so that

$$\left| \frac{(p_k - p_i) \cdot p_i}{\|p_k - p_i\| \cdot \|p_i\|} \right| > \cos(\Theta_{max}) \tag{3.19}$$

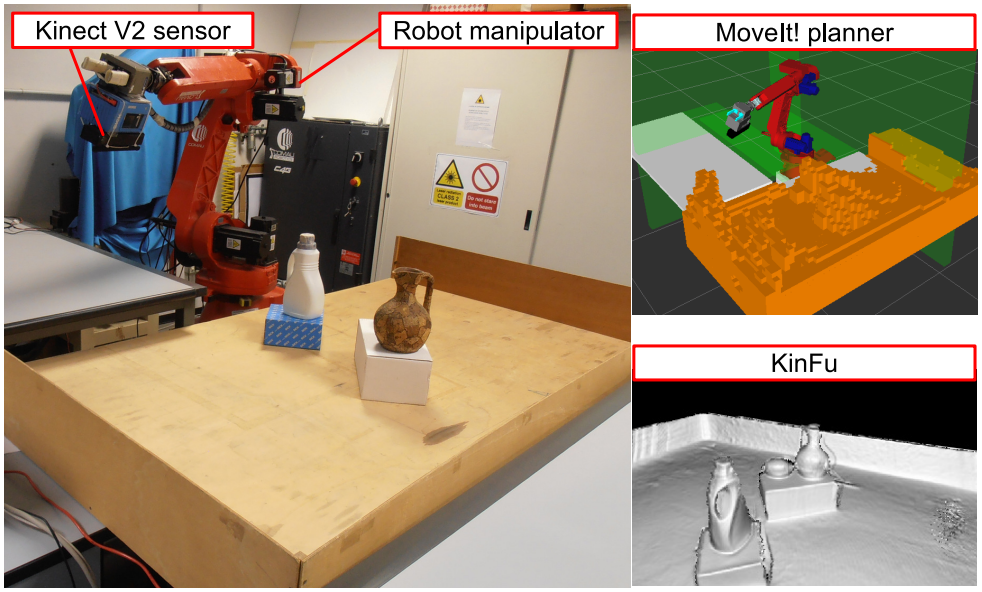


Figure 3.14: The experimental setup (left). Motion planning environment based on Moveit! (top right). Screenshot of KinFu output during the initial scan phase (bottom right).

3.2 Experimental evaluation

3.2.1 Robot setup and experimental procedure

The experimental setup (Fig. 3.14) used for the evaluation consists of a robot arm (Comau SMART SiX) with six degrees of freedom. The robot has a maximum horizontal reach of about 1.4 m. A Kinect V2 sensor is mounted on the end-effector and it has been calibrated with respect to the robot wrist. The developed software runs under the ROS framework on an Intel Core i7 4770 at 3.40 GHz, equipped with an NVidia GeForce GTX 670. Collision free robot movements are planned using the MoveIt! ROS stack.

Occupied and unknown voxels are considered as obstacles in the motion planning environment. Experiments have been performed on a workspace of size 2 m$\times$1.32 m. The volumetric representation of the environment within KinFu uses voxels of

size 5.8 mm. In the motion planning environment voxels are undersampled to 4 cm. As already shown in the previous Chapter 2, KinFu is fed with the robot forward kinematics to improve the accuracy of point cloud registration with respect to the standard sensor ego-motion tracking approach.

The experimental procedure consists of the following steps. At the beginning of each experiment the environment is completely unknown and the robot, starting from a collision-free configuration, takes a short initial scan of the scene from one side, using KinFu. Then, the system iteratively computes the NBV as described in Algorithm 4. If the motion planner finds a collision-free path the robot is moved to the NBV. Otherwise, the NBV is skipped. KinFu is turned on when the robot reaches each planned next-best view configuration, as for the KFOVP (KinFu On ViewPoint) strategy previously presented in Chapter 2, Section 2.2. The Kinect sensor is slightly tilted around the NBV by rotating the robot wrist. The volumetric representation of the environment is updated by KinFu after each observation. For the evaluation of the proposed approach for active exploration the experiments were concluded after the fifth NBV.

3.2.2 Experiments

Experiments have been performed in four different scenarios shown in Fig. 3.15. Each experiment contains multiple objects with complex geometry. In particular, in experiment 1 the environment comprises two stacks of objects, while experiment 2 has been performed in a cluttered scene with eight objects.

The performance of the proposed method was compared to the standard next-

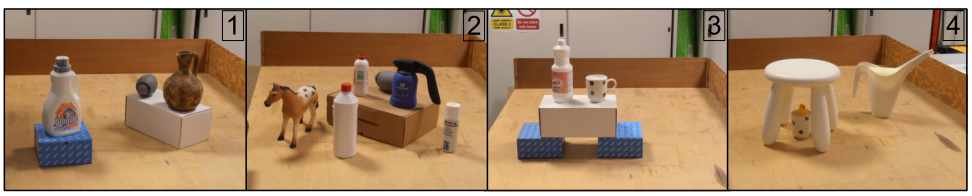


Figure 3.15: The experimental scenarios used for the evaluation.

Table 3.1: Average total time (seconds) and standard deviation over the four experiments for each phase

	Method			
Phase	Proposed		Standard	
Segm. + saliency	46.1	±2.5	-	
Views generation	2.1	±0.1	2.1	±0.1
Views evaluation	26.0	±4.1	288.5	±39.0
Subtotal	74.2	±4.8	290.6	±39.0
Planner map update	46.0	±1.6	44.6	±0.4
Motion planning	88.1	±17.9	78.3	±13.8
Robot motion	110.5	±3.9	108.5	±7.7
Total	318.7	±19.1	522.0	±42.0

best view approach where view pose is chosen at each iteration as the viewpoint that maximizes the size of the expected unknown volume of the whole environment that becomes visible. The standard approach has been developed by skipping the point cloud segmentation phase and by assigning the same saliency value to all points.

Quantitative data about the average computational time for each phase are reported in Table 3.1. The average time for point cloud segmentation and saliency computation is about 23% of the total time. A first advantage of the proposed method is that it completes the five next-best views faster than the standard approach. The average times for motion planning and robot movement are rather similar, since these are fixed costs due to the experimental setup, as well as the running time for updating the collision map of the motion planning environment (planner map update). Also, the time required for viewpoint generation is very short (2.1 seconds for five views), since the computation is performed on the GPU directly on the TSDF volume. The running time required for the computation of the NBV is reported as a subtotal. It can be noted that for the NBV computation phase the proposed attention-based approach is 3.9 times faster than the standard approach, even though the standard approach does not require point cloud segmentation and saliency evaluation.

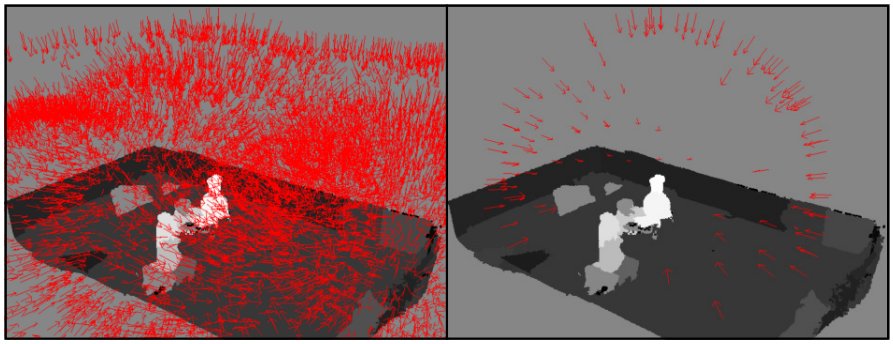


Figure 3.16: Candidate viewpoints (represented by arrows) for the proposed approach (experiment 1, third NBV). Left: candidate viewpoints of all segments. Right: candidate viewpoints for the most salient segment only.

The main difference between the two approaches in the time required to compute a NBV lies in the viewpoint evaluation phase. The standard approach evaluates all candidate viewpoints generated in the environment (on the order of thousands), most of which are located on the edges of the supporting table. On the contrary, being able to focus only on the most salient segments, the proposed method rarely evaluates more than two hundreds candidate viewpoints at each iteration. Indeed, the proposed method is strictly focused on the exploration of the salient segments, whose extension is smaller than the size of all the unknown regions of the environment.

In Fig. 3.16, an example of the generated viewpoints is shown. The total number of candidate viewpoints for all segments is 91960. Using the standard approach all viewpoints would be evaluated to find the optimal NBV. Instead, the proposed method focuses only on the most salient segment and, therefore, only 960 viewpoints are evaluated. In this case, a reachable pose for the robot was found among these viewpoints. If a reachable pose had not been found the system would have evaluated the second most salient segment, and so on. In Table 3.2 the saliency values of the point cloud segments are shown as well as the number of associated viewpoints. Had the algorithm tried other segments after the most salient one, the number of evaluated viewpoints would have increased up to 10480, which is the total number of candidate

Table 3.2: Saliency values and number of view poses for the point cloud segments in Fig. 3.16 (in descending order of saliency) up to the first segment not belonging to the objects (part of the supporting table)

Saliency	No. of poses	Description
0.589	960	Cork jug (top part)
0.565	3640	Cork jug (bottom part)
0.510	1680	Plastic jug (top part)
0.459	1560	Box under cork jug
0.424	600	Plastic jug (bottom part)
0.312	1080	Ball
0.300	400	Box under plastic jug
0.212	240	Box under plastic jug
0.177	320	Box under plastic jug
0.172	7720	Part of the table

viewpoints actually pointing towards the objects. The proposed saliency function is working properly even with some degree of over-segmentation by the LCCP algorithm. Indeed, some of the objects are segmented in multiple parts. For example, both jugs are split into two segments and one of the boxes is segmented into three parts. Nonetheless, each of those parts received a high saliency.

In Table 3.3 marks are reported that indicate whether each NBV points towards the objects or not. In the proposed approach all next-best views pointing towards the objects always occur before any other view, not focused on the objects. In the standard approach next-best views pointing towards the objects occur in an unpredictable order. Therefore, it is possible to conclude that a second and more important advantage of the proposed approach is that it allows a more rapid exploration of the objects thanks to point cloud segmentation and saliency evaluation at the segment level. This conclusion is also supported by the graphs in Fig. 3.17, which show the number of unknown residual voxels near the objects over the first five next-best views.

Images of the planned next-best views for experiment 1 are reported in Figures

Table 3.3: Marks showing NBVs pointing towards the objects (✓) or not (×), for all the experiments

		NBV				
Method	Exp.	1	2	3	4	5
Proposed	1	✓	✓	✓	✓	×
	2	✓	✓	✓	×	×
	3	✓	✓	✓	×	×
	4	✓	✓	✓	✓	✓
Standard	1	×	✓	×	✓	×
	2	✓	×	×	×	✓
	3	×	×	✓	×	✓
	4	×	×	✓	×	×

3.18 and 3.19. Images of the planned next-best views for experiment 3 are reported in Figures 3.20 and 3.21. In experiment 1 the robot focuses on the objects for the first four views. Afterwards, as there are no reachable viewing poses to observe the right side of the objects, due to kinematic constraints, the robot explores a region of space that does not contain any object. In particular, the robot observes the space on the supporting table in the front of the objects, which is incomplete due to noise. A similar behavior is evident, for the proposed approach, in experiment 3. Conversely, it can be noted that the standard approach prioritizes exploration of the unknown voxels occluded by the objects as shown, for example, in the first two views of experiment 3. In the third view of experiment 3 the standard approach takes a frontal observation of the objects, but in the fourth view the robot observes again a region of the supporting plane without any object.

In some cases at the beginning of the exploration, after one or two next-best views, the standard approach achieves a lower number of unknown residual voxels. An example can be seen in Fig. 3.17 for experiment 1, after the second NBV. This is due to the fact that in the standard approach when the robot observes the unknown voxels occluded by the objects it also partially observes the back of the objects, since

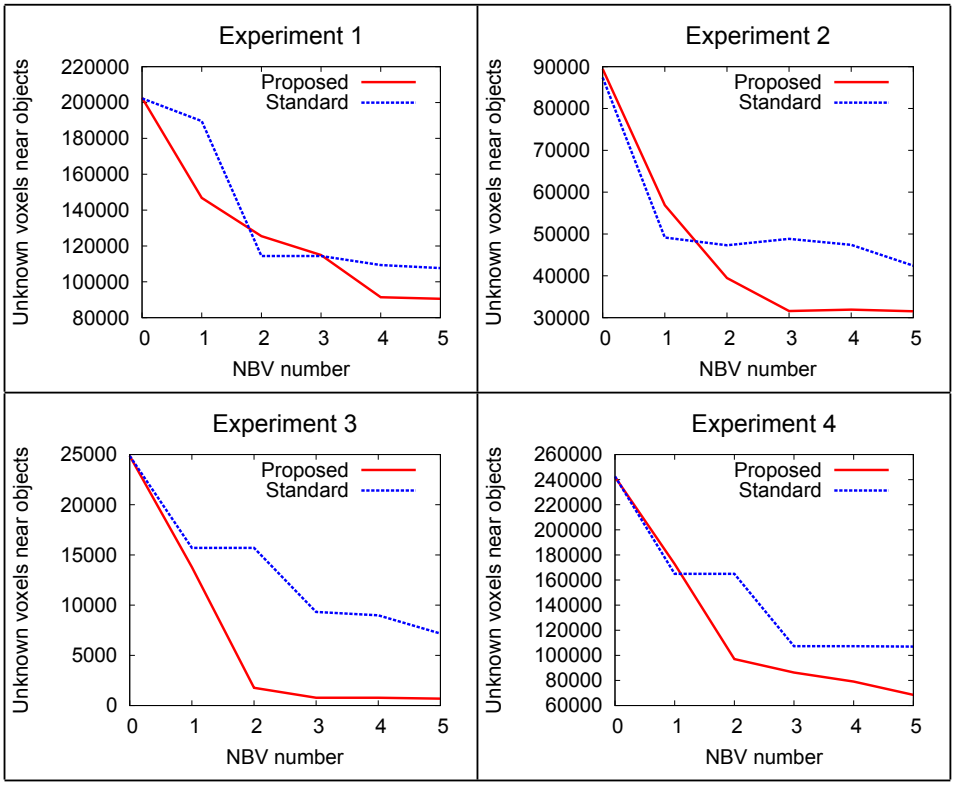


Figure 3.17: The graphs show the number of unknown voxels near the objects in the scene for the first five next-best views.

the sensor has a large field of view (70×60 degrees).

The final voxel-based reconstruction is shown in Fig. 3.22 for all experiments. The reconstruction of the objects is always more complete for the proposed method. Some unknown voxels are still present, mostly due to unreachable poses aimed at observing the back or below the objects, as stated above. Also, it can be noted that most of the irrelevant voxels around the back panel of the scene remained unknown for the proposed method, while these voxels have been observed by the standard NBV approach.

Figure 3.18: Images of experiment 1 using the proposed method (left to right). Top: saliency map of point cloud segments; middle: 3D volumetric representation; bottom: planned robot next-best views.

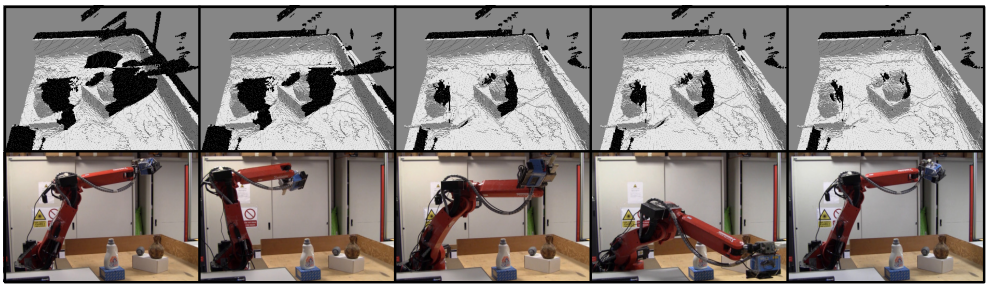


Figure 3.19: Images of experiment 1 using the standard NBV approach. Top: 3D volumetric representation; bottom: planned robot next-best views.

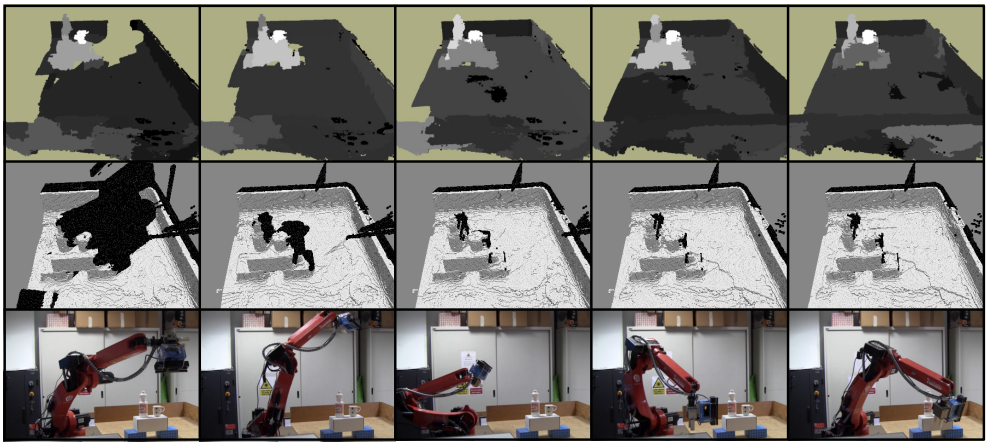


Figure 3.20: Images of experiment 3 using the proposed method (left to right). Top: saliency map of point cloud segments; middle: 3D volumetric representation; bottom: planned robot next-best views.

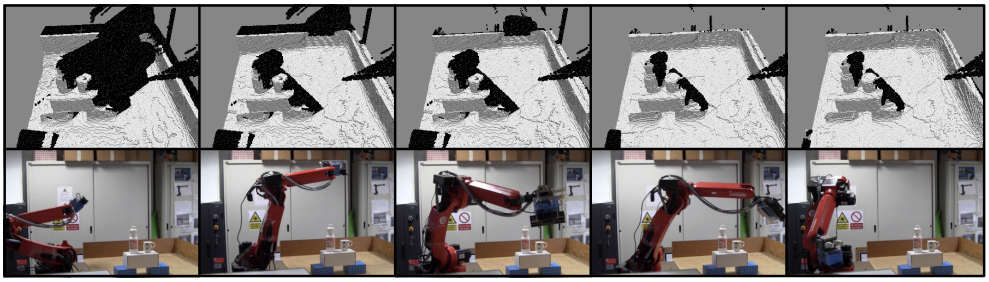


Figure 3.21: Images of experiment 3 using the standard NBV approach. Top: 3D volumetric representation; bottom: planned robot next-best views.

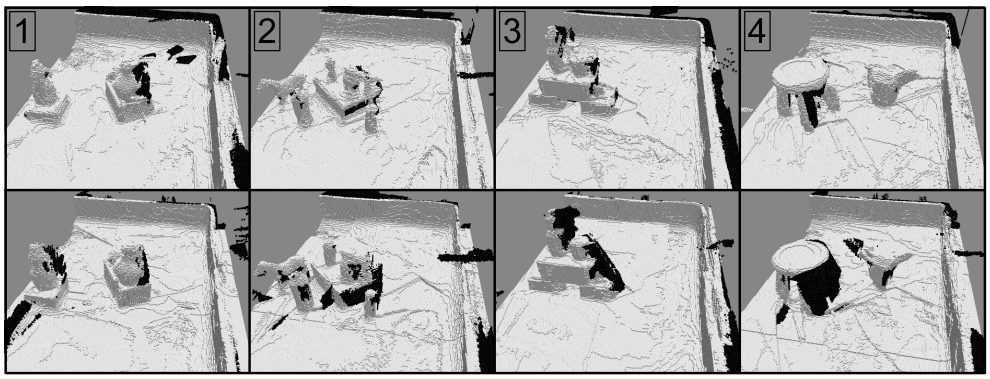


Figure 3.22: 3D volumetric representation of the environment in the four experiments after five next-best views: proposed method (top), standard approach (bottom).

3.2.3 Evaluation of depth image pre-processing

The proposed Kinect V2 depth image pre-processing filter (Section 3.1.3) has been evaluated in the scenario shown in Fig. 3.23 (top-left). The environment contains only planar surfaces to facilitate ground truth annotation. A bounding box was defined around the workspace to remove the background of the room. Thus, any point that does not belong to a plane can be considered as an outlier. Depth images were obtained by averaging 30 frames (one second) acquired by the sensor to simulate the noise-reduction effect of the KinFu algorithm. A maximum distance threshold of 3 cm was defined to consider a point as belonging to a plane.

In Fig. 3.23 it can be noted that the proposed pre-processing method successfully removes the shadow on the left of the box-shaped object. The total number of false negatives, i.e. outlier points not belonging to any plane, are reported in Table 3.4 as well as the number of measurements, i.e. the number of valid points reported by the algorithms. The proposed algorithm reports a significantly lower number of outliers compared to the standard filtering algorithms already available in the Freenect2 driver (a bilateral and an edge-aware filter). Being conservative, however, it also reports a lower number of valid measurements.

In Section 3.1.3 it was pointed out that the proposed filter for shadow points

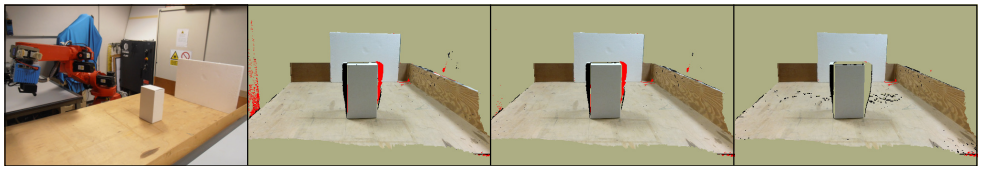


Figure 3.23: From left to right: the scenario used for testing the proposed depth image pre-processing filter, image preprocessed by the Freenect2 bilateral filter only, image preprocessed by the Freenect2 bilateral and edge-aware filters, image preprocessed by the Freenect2 bilateral filter and the proposed filter. The image is displayed in color although the algorithm operates on the depth map only. Outliers points are displayed in red.

Table 3.4: Number of measurements and false measurements produced by each algorithm.

Method	Measurements	Outliers
Bilateral	83125	1874
Bilateral+Edge-aware	81611	849
Bilateral+Proposed filter	79756	182

removal only provides a necessary condition and that false positives may still be present. Evaluation of false positives was carried out in a simulated environment shown in Fig. 3.24, which contains a ground plane, a wall, and an object (long box). The object is at a distance of 1.5 m from the wall and the Kinect V2 sensor is placed at 1 m from the object. The sensor view of the wall and ground plane is partially occluded by the object. The IR emitter and the camera were simulated as separate entities according to the Kinect V2 technical specifications. The shadow points removal filter was tested by varying the width of the object and the observation angles of the sensor around the object. Table 3.5 reports the ratio between the incorrectly removed points and all the removed points (false discovery rate). For normal-sized object ($8-16$ cm width) the false discovery rate is low. However, for thin objects (4 cm width) as the one displayed in Fig. 3.24 (left) the false discovery rate is over

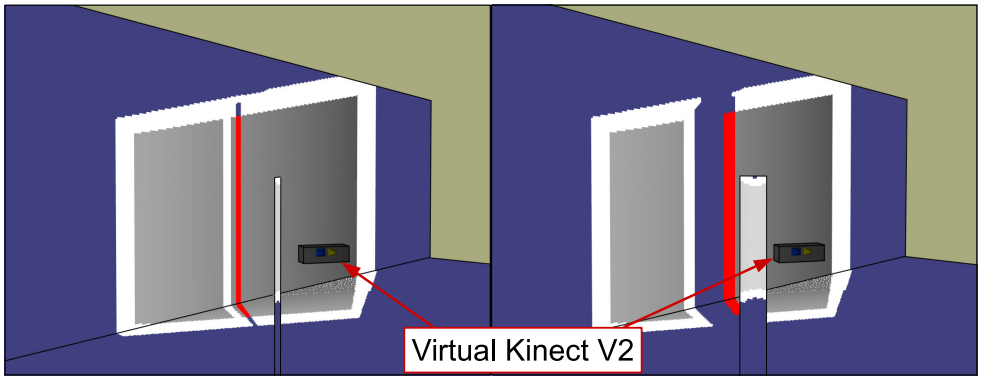


Figure 3.24: The simulated environment, with object size 4 cm (left) and 16 cm (right). White: area illuminated by the emitter only. Grey: area illuminated and properly acquired by the camera. Red: shadow visible by the camera. The vertical blue band in the right image is a region of space that is neither illuminated by the emitter nor observed by the camera.

Table 3.5: False discovery rate of shadow points

Sensor angle (deg.)		-60	-30	0	30	60
Object width (cm)	4	41%	50%	50%	47%	42%
	8	0%	1%	4%	5%	7%
	16	1%	0%	0%	14%	5%

40%. This is due to the fact that light from the emitter can pass behind a thin object and illuminate part of the background which could be correctly perceived by the real sensor, but it is actually removed by the proposed filter. It may be noted, however, that this negative result is quite rare as it happens only if a thin object is in front of a far background; moreover, in these cases only the background region is affected.

3.3 Discussion

In this chapter, an object-driven method for robot exploration based on next-best view was presented. It was demonstrated that the addition of spatial attention to an exploration system may improve its performance. As expected for an attention-based system, the advantage lies mainly in the increased focus of the resources of the system towards the desired goal.

Two resources are focused. The computation time is reduced as shown in Section 3.2.2, Table 3.3, due to the lower number of viewpoints evaluated by the next-best view algorithm. Specifically, only the viewpoints oriented towards the most salient segment are evaluated. Only if all these viewpoints are unreachable, the ones of the other segments are evaluated in decreasing order of saliency.

Moreover, in the experimental setup the the number of views is a limited resource, since the experiments have been stopped at the fifth next-best view. If the experiments had continued indefinitely, with an infinite number of views, the robot would have eventually completed the 3D reconstruction, including the objects. Thanks to attention, however, the five available views were allocated immediately to observe the objects. Therefore, the final 3D reconstruction of the objects was more complete, at the expense of general exploration, as noted in Section 3.2.2 regarding Fig. 3.22.

The last remark shows the attention heuristic tradeoff. The addition of attention to an exploration system improves its performance as long as the heuristic is compliant with the goals of the system. In the example shown in this chapter, if the robot system had aimed at a rough exploration of the whole enviroment, an opposite heuristic would have produced better results. The evaluation of alternative heuristics and their improvement represent a direction for further investigation. Moreover, the proposed heuristic depends on the LCCP segmentation algorithm, which was chosen mainly for its standardization within the Point Cloud Library. Segmentation algorithms designed specifically for saliency evaluation may be developed. Following the results about real-time segmentation by [96, 69, 97], this system may also be extended with real-time attention computation.

Chapter 4

Extension towards a surfel-based next-best view planner

The approaches presented in previous Chapters 2 and 3 operate on a volumetric 3D representation. Specifically, KinectFusion maintains a TSDF volume, which encodes in every voxel the signed distance to the nearest surface. It has been shown that the TSDF value may be easily converted into a ternary representation with empty, unknown and occupied voxels. The volumetric representation of KinectFusion is essentially implemented as a 3-dimensional array. Such implementation has large memory usage, which requires the use of a "cyclical buffer" data structure as explained in Section 2.2.2, Chapter 2, in order to download and compress data from GPU memory periodically. Moreover, GPUs are not optimized to render a volumetric representation, and the synthetic depth map is produced through ray casting.

For these reasons, in recent years scientific interest has been oriented towards surfel-based 3D representations. Surfels (Surface Elements) are disks described by a center, a normal, a radius and possibly a color. Many surfels may be assembled to describe a 3D surface. In practice, a set of surfel is represented as a point cloud, where each point has normal, color and radius attributes. The memory usage is proportional to the number of surfels, and large regions between surfaces do not require any memory. Moreover, surfel rendering is efficient, since the GPU is able to render

points and the rendering of small disks may be accelerated using shaders.

ElasticFusion [9] is a surfel-based 3D reconstruction algorithm, adopted in this chapter. It is a state-of-the-art RGB-D reconstruction algorithm, with color support, sensor egomotion tracking and loop closure. The algorithm has seen widespread use due to its open-source implementation, written in C++, GLSL (GPU shading language) and CUDA.

Surfel-based 3D representation does not allow next-best view operations to be performed. Surfels only describe a surface and there is no difference between "empty" and "unknown" space. A preliminary investigation has been performed, aiming to track the frontier between known and unknown space in surfel-based 3D reconstruction, using surfel-like elements. Such solution would facilitate integration with ElasticFusion, e.g. with the cloud deformation due to loop closure. Moreover, surfels can be projected into a simulated viewport by rendering, which may be faster on GPU than ray casting and provide a speedup for next-best view. Due to the preliminary nature of this investigation, the analysis was not included in this dissertation.

In Section 4.1, it is observed that the averaging process in ElasticFusion produces blur around color discontinuities. The blur may hamper segmentation, which is needed (for example) for the saliency method shown in Chapter 3, which is based on Supervoxel-LCCP [6]. A mode filter is proposed in order to reduce this negative effect. In order to evaluate the effect of the segmentation enhancement method, a ground truth was produced using an annotation tool, which is presented in Section 4.2.

4.1 Surfel segmentation enhancement

3D reconstruction algorithms like ElasticFusion merge multiple views from a sensor into a single environment representation. The reconstruction process usually involves a mean average. As a point in space is observed by multiple viewpoints, the attributes for that point are obtained by averaging the observations. In the final 3D model, the position, color and normal of each point are the result of this averaging process. Indeed, in presence of Gaussian noise, the averaging process is the best choice to locate the most likely attributes for that point.

However, in presence of sensor pose uncertainty, the correspondence between the observed points and the points already in the reconstruction may be uncertain as well. In regions of space close to discontinuities, the sensor may observe from either side of the discontinuity alternately. In other words, close to discontinuities the noise ceases to be Gaussian and it becomes multi-modal, each mode corresponding to a side of the discontinuity. Segmentation algorithms rely on these discontinuities in order to detect and split segments. As shown in Fig. 4.1, the blur effect may reduce the effectiveness of segmentation algorithms.

Therefore, in this section a color and position enhancement filter [14] is proposed, to be run alongside ElasticFusion [9]. The proposed filter keeps track of multiple possible color and position values for each surfel, thus modeling the multi-modal nature of the input data. At every new observation, for each surfel, only the hypothesis with most similar color to observation is updated. Only at the end of the reconstruction, the choice on one of the modes for each surfel is made through the Winner Takes All (WTA) policy. The filter thus estimates the mode of all the observed values of the surfel.

Two segmentation algorithms, a region growing (Flood Fill) algorithm and the SV-LCCP [6] algorithm, are applied to the final reconstruction, with and without the proposed enhancement filter. The final segmentation is then compared with a ground truth and evaluated in each case, using the border precision/recall and Bidirectional Consistency Error metrics. The ground truth was produced manually in environments containing small objects, using the annotation tool presented in next Section 4.2.

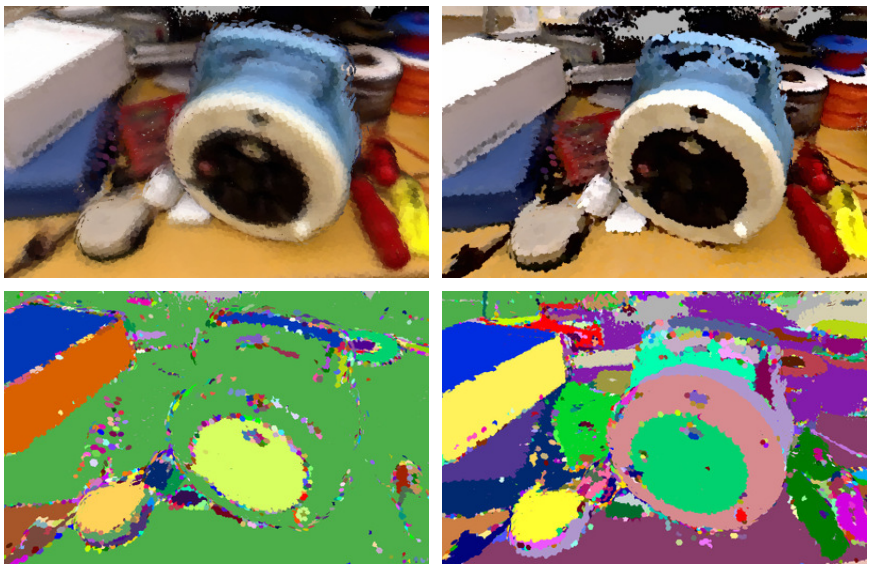


Figure 4.1: The reconstruction with mean color (top left) is more blurred than the output of the proposed filter (top right). As such, segmentation with the proposed filter (bottom right) is better than without using it (bottom left).

This section is organized into four subsections. In subsection 4.1.1, the mode filter is presented. The experimental pipeline is detailed in subsection 4.1.2. In subsection 4.1.3, the results are presented and briefly discussed in subsection 4.1.4.

4.1.1 Method

Framework overview

ElasticFusion maintains a surfel cloud. Surfels are points characterized by a 3D position $p(x,y,z)$, a normal $N(n_x,n_y,n_z)$, a radius r and a color $c(R,G,B)$. Thus, each surfel represents a small, colored disk in the 3D model. In addition, ElasticFusion keeps track of an observation counter w_s for each surfel. An ID s is associated to each surfel, which does not change during the execution of ElasticFusion.

ElasticFusion operates in two phases, which are iterated for each new frame f

acquired by the sensor. In the first phase, i.e. the tracking phase, the algorithm estimates the new pose P_f of the sensor, by comparing the new information with the existing data in the 3D model. In the second phase, i.e. the mapping phase, the new information is merged with the 3D model.

At the beginning of the mapping phase, ElasticFusion renders the 3D model onto the (estimated) current sensor viewport. The projection is required to find a correspondence between each pixel, as acquired by the RGB-D sensor, and a surfel in the surfel cloud. If some pixels of the sensor frame are not covered by any surfel, a new surfel may be created. The 3D model rendering is also used in the next tracking phase, to find the sensor movement since the last estimated pose.

Each frame F_f acquired by the sensor is composed by a color image C_f and a depth image D_f, with same resolution (u_{max}, v_{max}). These frames contain color values $c'_{f,uv}$, and depth values $d'_{f,uv}$, where (u,v) are the image coordinates. From depth values $d'_{f,uv}$ position $p'_{f,uv}$ can be computed through a standard pinhole model. ElasticFusion produces an index image S_f, composed by the surfel indices $s_{f,uv}$. This image associates each image pixel to the corresponding surfel, if any correspondence is found.

The update is essentially performed as a weighted average:

$$\begin{aligned} p_s &\leftarrow (p_s w_s + p' w_{uv}) \,/\, (w_s + w_{uv}) \\ c_s &\leftarrow (c_s w_s + c' w_{uv}) \,/\, (w_s + w_{uv}) \\ w_s &\leftarrow w_s + w_{uv} \end{aligned} \tag{4.1}$$

Weight w_{uv} of pixel (u,v) is computed as a function of the distance from the image center (u_c, v_c), as in [98], to model higher sensor noise towards the border of the image, i.e.:

$$\begin{aligned} \Delta_{uv} &= \frac{(u-u_c)^2+(v-v_c)^2}{u_{max}^2+v_{max}^2} \\ w_{uv} &= e^{-\frac{\Delta_{uv}}{\left(2\cdot\sigma_{img}^2\right)}} \end{aligned} \tag{4.2}$$

Algorithm 6 Enhancement method

Require: M_s: previous set of tuples m_i
Require: c': new color
Require: d': new depth
Require: P_f: current sensor pose
Ensure: M_s: new set of tuples

$p' \leftarrow P_f \cdot \text{BackProject}_{uv}(d')$
if $M_s = \varnothing$ **then**
  $M_s \leftarrow M_s \cup \{\langle c', w_{uv}, p' \rangle\}$
  return
end if
$n \leftarrow \text{argmin}_i \|c_i - c'\|$
$\delta_n \leftarrow \|c_n - c'\|$
if $\delta_n > K \cdot \sigma_c$ **then**
  if $\|M_s\| > (M_{\max} - 1)$ **then**
    $i_r \leftarrow \text{argmin}_i w_i$
    $M_s \leftarrow M_s - \{m_{i_r}\}$
  end if
  $M_s \leftarrow M_s \cup \{\langle c', w_{uv}, p' \rangle\}$
  return
end if
$d \leftarrow \text{Project}_{uv}\left(P_f^{-1} p_n\right)$
$\gamma \leftarrow (c_n, d)$
$d'_e \leftarrow \text{GuidedFilter}(d', \gamma)$
$p' \leftarrow P_f \cdot \text{BackProject}_{uv}(d'_e)$
$m_n \leftarrow \text{Merge}(m_n, \langle c', w_{uv}, p' \rangle)$
for each $m_i \in (M_s - \{m_n\})$ **do**
  if $\text{p}_{\mathcal{N}}(c_i, c_n, \sigma_c) \cdot w_n > \text{p}_{\mathcal{N}}(c_i, c_i, \sigma_c) \cdot w_i$ **then**
    $m_n \leftarrow \text{Merge}(m_n, m_i)$
    $M_s \leftarrow M_s - \{m_i\}$
  end if
end for

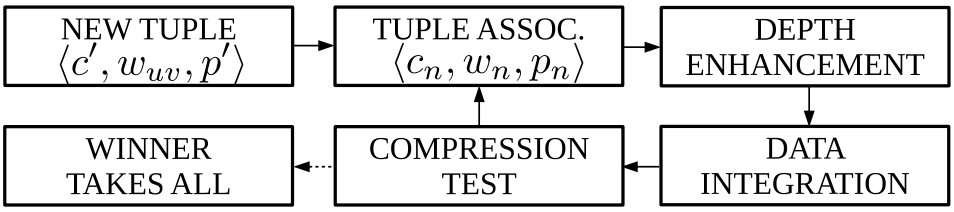


Figure 4.2: Flowchart of the method for surfel segmentation enhancement.

Enhancement Method

For each surfel with ID s, a set of possible tuples M_s is maintained. Each tuple $m_{i,s}$ in the set is defined as:

$$m_{i,s} = \langle c_{i,s}, w_{i,s}, p_{i,s} \rangle \tag{4.3}$$

where c is the color vector (r,g,b), p is the 3D position vector (x,y,z) and w is the weight. The tuple in the set represents possible colors and positions assigned to the surfel. The surfel normal and radius are not recomputed by the proposed algorithm and are computed by ElasticFusion.

During reconstruction, ElasticFusion associates to each input pixel (u,v) of frame $f(C_f, D_f)$ the corresponding surfel $s_{f,uv}$, and it also provides the current sensor pose P_f. Each input pixel provides a new possible tuple $\langle c', w_{uv}, p' \rangle$ with color, weight (as defined in Eq. 4.2) and position. Alg. 6 is executed for each pixel, in order to merge the observation into the existing set M_s. The algorithm may be decomposed into five phases, shown in Fig. 4.2 and explained next. Subscript f,uv is omitted in the following, as the algorithm is executed independently for each frame and pixel.

1) Tuple association: In this phase, the tuple in set M_s which corresponds to new tuple $\langle c', w_{uv}, p' \rangle$ is found. This tuple represents the local mode which is most likely to have generated the new tuple. If the set M_s is empty, the new tuple is added (lines 2-5 in Alg. 6). Otherwise, the tuple with the most similar color is selected at line 6. If the most similar color c_n is too different from the new color c' (lines 8-15), then the new tuple represents a different mode and should not be merged. Then, it is added to M_s with weight w_{uv} (from Eq. 4.2). To limit memory usage, if M_s exceeds a maximum

number of tuples $M_{\max}$, the tuple with lower w_i is removed (lines 9-12).

Algorithm 7 Guided filter

Require: γ: 4-channel guidance image
Require: d': input depth
Ensure: d'_e: enhanced depth
1: $\text{avg}_\gamma \leftarrow F_m(\gamma)$
2: $\text{avg}_{d'} \leftarrow F_m(d')$
3: $\text{corr}_\gamma \leftarrow F_m(\gamma \cdot \gamma^T)$
4: $\text{corr}_{\gamma d'} \leftarrow F_m(\gamma \cdot d')$
5: $\text{var}_\gamma \leftarrow \text{corr}_\gamma - \text{avg}_\gamma \cdot \text{avg}_\gamma^T$
6: $\text{cov}_{\gamma d'} \leftarrow \text{corr}_{\gamma d'} - \text{avg}_\gamma \cdot \text{avg}_{d'}$
7: $A \leftarrow \left(\text{var}_\gamma + \Sigma_E\right)^{-1} \cdot \text{cov}_{\gamma d'}$
8: $B \leftarrow \text{avg}_{d'} - A^T \cdot \text{avg}_\gamma$
9: $\text{avg}_A \leftarrow F_m(A)$
10: $\text{avg}_B \leftarrow F_m(B)$
11: $d'_e \leftarrow \text{avg}_A^T \cdot c + \text{avg}_B$

2) Depth enhancement: A new, enhanced depth value d'_e for each pixel (u,v) is estimated as in lines 16-19 in Alg. 6, Synthetic position p_{uv} is converted into sensor coordinates and projected onto the sensor frame, to obtain synthetic depth d. The values d and the color vectors c for each pixel are combined to form the guidance image $\Gamma(\gamma_{uv}) = \Gamma(c_{uv}, d_{uv})$ (line 17). A guided filter (line 18) is applied with a 4-channel guidance image. Definition and formal derivation of the guided filter may be found in [59]. The general procedure is recalled in Alg. 7. Each line in Alg. 7 is executed for each pixel (u,v). The function F_m is defined as a circular mean in the pixel neighborhood with R_m pixel radius:

$$\begin{aligned} &F_m(\Phi) = F_m(\Phi(u,v)) = \\ &\text{avg}\left(\left\{\Phi(u+\delta_u, v+\delta_v),\ \forall\ (\delta_u, \delta_v) \mid \delta_u^2 + \delta_v^2 \leq R_m^2\right\}\right) \end{aligned} \tag{4.4}$$

where Φ corresponds to various functions of (u,v) as shown in the algorithm. Parameter Σ_E (line 7, Alg. 7) is a diagonal matrix of order 4. The three top-left elements of the matrix are equal to color variance σ_c^2 and the fourth elements is depth variance σ_d^2.

3) Data integration: In the previous phases, a color and an enhanced position have been found. In this phase, they are merged with the associated tuple m_n (line 20, Alg. 6) with a weighted average:

$$c_n \leftarrow \frac{c_n \cdot w_n + c' \cdot w_{uv}}{w_n + w_{uv}} \qquad w_n \leftarrow w_n + w_{uv} \tag{4.5}$$

$$p_n \leftarrow \frac{p_n \cdot w_n + p' \cdot w_{uv}}{w_n + w_{uv}} \tag{4.6}$$

4) Compression: During data integration, a mode may have converged to a value similar to another mode. Thus, a compression test is performed, to find values in the set M which are redundant with respect to updated tuple n (lines 21-26). A color c_i may be produced by an observation of c_n affected by Gaussian noise $\mathcal{N}(c_n, \sigma_c)$, with mean c_n and standard deviation σ_c. The noise has probability density $\mathrm{p}_{\mathcal{N}}(c, c_n, \sigma_c)$. The probability density that c_i is generated by color c_n is:

$$p(c_i) = p(c_i \mid n) \cdot P(n) \tag{4.7}$$

The a priory probability of tuple n is unknown. It is estimated from the number of times that the tuple has been observed in the past:

$$P(n) = \frac{w_n}{\sum_j w_j} \tag{4.8}$$

The probability density of observing c_i given color c_n is $\mathrm{p}_{\mathcal{N}}(c_i, c_n, \sigma_c)$. Thus, the probability density that c_i is generated by c_n is:

$$p(c_i) = \mathrm{p}_{\mathcal{N}}(c_i, c_n, \sigma_c) \cdot \frac{w_i}{\sum_j w_j} \tag{4.9}$$

Colors are merged if the probability density that c_i is generated by c_n is greater than the probability density of it being generated by a noisy observation of c_i itself.

$$\mathrm{p}_{\mathcal{N}}(c_i, c_n, \sigma_c) \cdot \frac{w_n}{\sum_j w_j} > \mathrm{p}_{\mathcal{N}}(c_i, c_i, \sigma_c) \cdot \frac{w_i}{\sum_j w_j} \tag{4.10}$$

From Eq. 4.10, the condition at line 22 is obtained.

5) Winner Takes All: At the end of the reconstruction, the tuple with the highest w (i.e. the mode) for each surfel s is selected by Winner Takes All policy, i.e.:

$$i_{\max} \leftarrow \underset{i}{\operatorname{argmax}}\ w_{i,s} \quad c_s = c_{i_{\max},s} \quad p_s = p_{i_{\max},s} \tag{4.11}$$

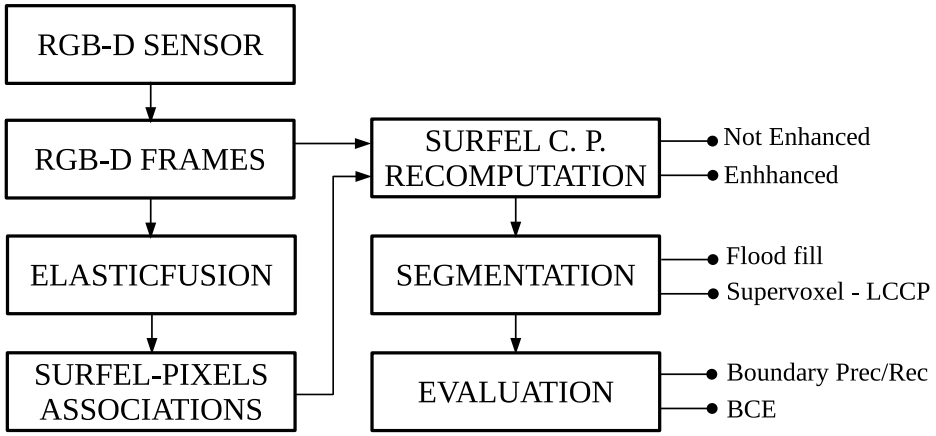


Figure 4.3: The experimental pipeline.

4.1.2 Data processing pipeline

The processing pipeline for the evaluation of the proposed method is shown in Fig. 4.3. An RGB-D frame sequence is acquired with a range sensor and it is processed by ElasticFusion [9], to generate a surfel cloud. For its internal processing, ElasticFusion needs to find a correspondence between each pixel (u,v) in each frame f and a surfel in the surfel cloud. This correspondence represents the surfel index $s(f,uv)$, required as input to the proposed mode filter.

Then, surfel colors and positions are recomputed using the mode filter ("Surfel C. P. Recomputation" in Fig. 4.3). For comparison, data are also processed by performing a simple weighted average on the colors and position as follows:

$$c_s = \frac{\sum_{f,uv} c'_{f,uv} \cdot w'_{uv}}{\sum_{f,uv} w'_{uv}} \quad (f,\ (u,v) \mid s_{f,uv} = s) \tag{4.12}$$

This average results in an output similar to ElasticFusion (as shown in equation 4.1, section 4.1.1) and it is named "Not Enhanced" in this section. By using this recomputed value instead of ElasticFusion output, it is guaranteed that the same function for w_{uv} is used for both algorithms, as defined in equation 4.2.

In order to evaluate the effects on segmentation, a segmentation algorithm is applied to the surfel reconstruction. Two algorithms have been tested, as shown in Fig. 4.3: a standard Flood Fill algorithm and the Point Cloud Library implementation of the Supervoxel-LCCP algorithm [73][6].

The Flood Fill segmentation algorithm operates on the surfel cloud neighborhood graph. Two surfels are connected if the distance between their centers is lower than the sum of their respective radii (equation 4.13). Then, a similarity graph is built by pruning the neighborhood graph. Only edges that satisfy inequality 4.14 are kept, i.e.:

$$\|p_{s_1} - p_{s_2}\| < r_{s_1} + r_{s_2} \tag{4.13}$$

$$\alpha_c \|c_{s_1} - c_{s_2}\| + \alpha_n (1 - N_{s_1} \cdot N_{s_2}) < T_{\text{ff}} \tag{4.14}$$

where c_{s_i} are the colors, with range $[0,1]$ and N are the normals of the surfels, as computed by ElasticFusion. T_{ff} is a threshold, α_c and α_n are multipliers that represent the importance of colors and normals, respectively. Connected components in the similarity graph represent the segments.

The Supervoxel-LCCP algorithm was slightly modified to take advantage of the color information. While the algorithm can exploit color when generating supervoxels, LCCP uses only position and normal information to merge supervoxels afterwards. As such, the supervoxel neighborhood graph is pruned if the color of two adjacent supervoxels is too different:

$$\|c_{s_1} - c_{s_2}\| > T_{lccp} \tag{4.15}$$

Finally, the resulting segmentation was compared to a ground truth and evaluated. A precision/recall method based on boundaries was used, as proposed by [99] and [100]. The evaluation method is defined for images, and it is extended here to handle surfel clouds. A neighborhood graph is generated, which connects each surfel with surfels closer than T_{eval}. On this graph, the set of near-boundary surfels B_N is defined as those surfels having at least one surfel with a different label from their own in their neighborhood. To find the boundary of a surfel cloud, for every surfel $b_N \in B_N$, the

nearest surfel b_i with a different label is found. The boundary B is defined as the set of these nearest surfels b_i, i.e.:

$$B = \left\{ b \;\middle|\; \exists b_N,\ b = \underset{b_i \in B_N}{\operatorname{argmin}} \|b_i - b_N\|, \text{with } l_{b_i} \neq l_{b_N} \right\} \tag{4.16}$$

where l_{b_i} and l_{b_N} are the labels of b and b_N respectively.

To compare a segmentation with boundary surfel set B and the ground truth with boundary B_{GT}, an (approximate) minimum distance assignment is performed between the elements of B and B_{GT}. Assignments with distance longer than T_{eval} are discarded. Elements without assignment count as false positives, if they belong to the segmentation being evaluated, or false negatives, if they belong to the ground truth. Successful assignments count as true positives.

Moreover, the results were also evaluated using the Bidirectional Consistency Error (BCE), as defined by D. Martin in [100]. The BCE index is an extension of Martin's own Local Consistency Error [101], sensitive to over-segmentation. Like LCE, BCE is constrained between 0 (no error) and 1 (maximum error). Each surfel can be treated as a pixel for BCE, because neighborhood information is not needed.

4.1.3 Results

The approach was tested in five scenarios, shown in Fig. 4.4. For each scenario, a manually-annotated ground truth was available. The ground truth was annotated with attention to the smallest details of the scene, which pose greater challenge to segmentation. Annotation was performed on the point cloud without enhancement. Ground truth segments which are not connected on the neighborhood graph (as defined in Eq. 4.13) were automatically split into multiple segments. Neither Flood Fill nor Supervoxel-LCCP are able to merge spatially unconnected segments.

The first, second and third scenarios were acquired with an ASUS Xtion PRO LIVE sensor. In the first and second scenarios, vignetting effect was also corrected as in [102]. The fourth and fifth scenarios were acquired using a Kinect v2 sensor.

Table 4.1 reports the number of frames for each scenario, as well as the number of ground truth points and ground truth labels. The parameters used for evaluation

are shown in Table 4.2. The only exception is scenario 3, where a σ_{img} of 0.1 was used, to compensate for higher vignetting effect.

In Fig. 4.5, 4.6, 4.7 and 4.8, close-ups of the scenarios are provided, with segmentation. The original reconstruction is shown on the left, while the enhanced version on the right. As already shown in Fig. 4.1, the enhancement reduces blur. After the enhancement, the segmentation algorithm is able to detect a higher amount of details. Moreover, a lower number of single-surfel segments are produced along image borders, since the color gradient is sharper.

In Fig. 4.9, the effect of depth enhancement is shown. The figure shows only position difference: surfel color and viewpoint are the same in both images. The thin (~ 3 cm) wooden panel has been separated from the white wall by the depth enhancement, and the black background has become visible through the gap between them. Also, some of the veil points which connect the red handle to the wall have been moved to the wall or the handle itself.

Results of the Boundary Precision/Recall evaluation for the Flood Fill algorithm and the Supervoxel-LCCP algorithm are shown in Figs. 4.10 and 4.11 respectively. The curve is drawn by varying parameters T_{ff} and T_{lccp}, for Flood Fill and LCCP respectively. It may be seen that the PR curve is higher for the proposed algorithm, denoting a better tradeoff between precision and recall. The enhancement is less evident for the Supervoxel-LCCP algorithm, as it does not depend on sharp edges during the Supervoxels generation phase.

In the case of Scenarios 4 and 5, it may also be noticed that the enhanced case has

Table 4.1: Dataset statistics

Scenario	Frames	Points	Ground truth labels
1	1737	120K	87
2	1737	114K	196
3	2209	134K	164
4	1831	287K	139
5	1698	100K	97

Figure 4.4: The five scenarios (left) and their ground truth segmentation (right).

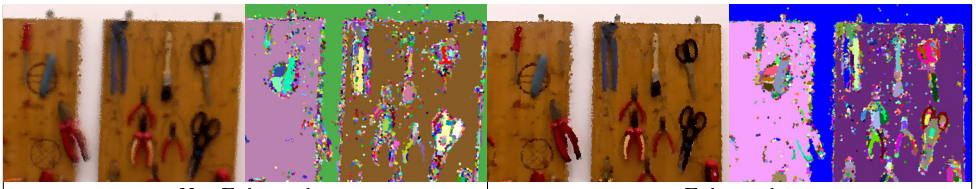


Figure 4.5: Close-up of scenario 1, reconstruction and segmentation using Flood Fill. With the enhanced method, the segmentation of the small tools on the panels is more complete.

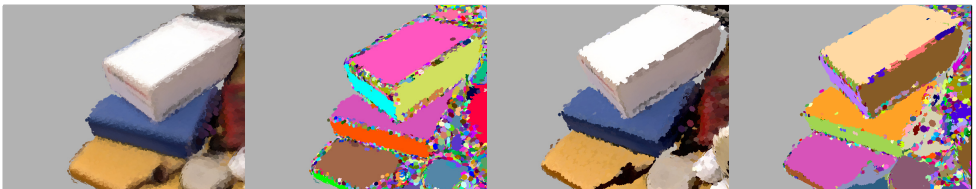


Figure 4.6: Close-up of scenario 2, reconstruction and segmentation using Flood Fill. In the enhanced case, a lower number of single-surfel segments are produced along image borders, since the color gradient is sharper.

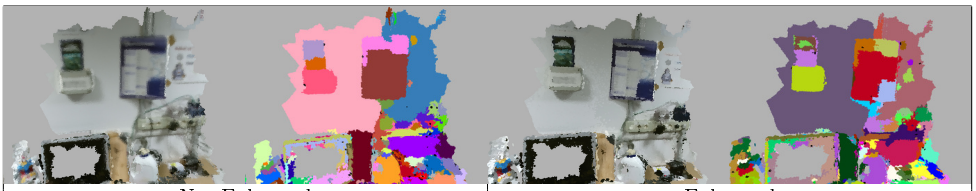


Figure 4.7: Close-up of scenario 4, reconstruction and segmentation using SV-LCCP. The difference between segmentations is less evident.

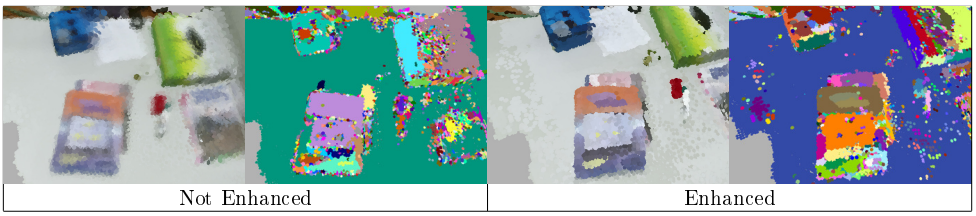


Figure 4.8: Close-up of scenario 5, reconstruction and segmentation using Flood Fill. In the enhanced case, the segmentation of the two magazines is complete, and details (e.g. the heading) have been properly detected.

Figure 4.9: Detail of scenario 1, with position not enhanced (left) and enhanced (right).

a slightly lower precision for the highest values of recall. This is due to the Kinect v2 auto-gain feature, which may cause the same point to be acquired with multiple colors in different frames. If the frequency of the colors is similar, the Winner Takes All policy may select different modes in nearby points and generates small color differences. For low values of the color thresholds T_{ff} and T_{lccp}, these differences are detected as segments. An example may be seen in Fig. 4.8. In the enhanced version, single-surfel segments appear in regions with uniform color. These segments correspond to areas on the table where color randomly switches between two similar grey tones. This observation may suggest that the mode filter is less robust to some sensor nonidealities.

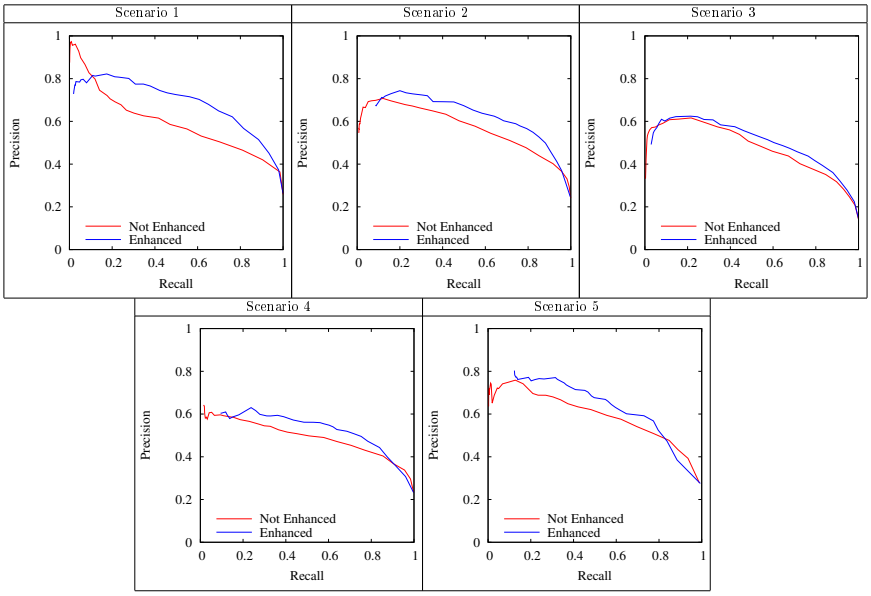


Figure 4.10: Precision/Recall curves for Flood Fill, for the five scenarios.

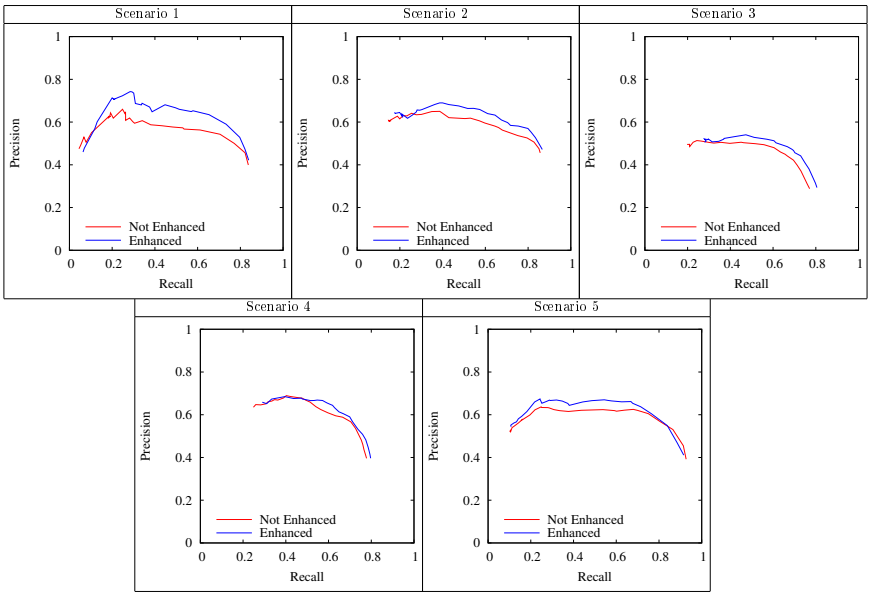


Figure 4.11: Precision/Recall curves for Supervoxel-LCCP, for the five scenarios.

Table 4.2: Fixed Parameters

Symbol	Value	Description
σ_c	0.005	Color std. dev. (in unitary RGB cube)
σ_d	5 cm	Depth std. dev.
K	5	σ_c multiplier (Alg. 6, line 8)
R_m	3 pixels	Circular mean radius (eq. 4.4)
T_{eval}	2 cm	Precision/Recall distance threshold
V_{lccp}	1 cm	Voxel size for Supervoxel-LCCP
S_{lccp}	5 cm	Supervoxel size for Supervoxel-LCCP
α_p	0.1	Supervoxel position importance
α_n	0.5	Supervoxel and eq. 4.14 normal importance
α_c	1.0	Supervoxel and eq. 4.14 color importance
σ_{img}	0.3	Radial noise model (eq. 4.2)
$M_{\max}$	10	Maximum tuples per surfel (Alg. 6, line 9)

For each value of T_{ff} and T_{lccp}, the Bidirectional Consistency Error was also computed. In Table 4.3, the minimum value (i.e. the best value) reached by BCE is reported for each algorithm, for both the not enhanced (NE) and enhanced (E) cases. The segmentation on the enhanced cloud provides a lower error by 5% for Flood Fill and 1% for LCCP.

Runtime is about 100ms per frame, with 8 threads on an Intel Core i7 4770 at 3.40 GHz. Therefore, the algorithm is not real-time for a common 30 fps RGB-D sensor. Most of the processing time is spent by the guided filter (61 ms per frame). Instead, runtime for the averaging Not Enhanced method is about 12 ms per frame.

4.1.4 Discussion

In this section, a color and position enhancement algorithm was presented for surfel-based 3D reconstruction. The algorithm is intended to enhance color and position discontinuities and facilitate segmentation. Therefore, it was evaluated by applying two segmentation algorithms, a region growing algorithm (Flood Fill) and a Supervoxel-

Table 4.3: Minimum Bidirectional Consistency Error

Scenario	Flood Fill		SV-LCCP	
	NE	E	NE	E
1	0.361	**0.259**	0.310	**0.277**
2	0.677	**0.578**	0.597	**0.569**
3	0.538	**0.486**	0.492	**0.484**
4	0.674	**0.604**	0.580	**0.565**
5	0.565	**0.518**	0.516	**0.493**

LCCP algorithm. Two metrics were used for evaluation, Boundary Precision/Recall and Bidirectional Consistency Error.

It has been shown that the segmentation produces better results after the enhancement. The result is significant especially for the Flood Fill algorithm. The Supervoxel-LCCP algorithm is more complex and is likely able to compensate the blur from ElasticFusion averaging process. In fact, the segment merging procedure is based on the average properties of surfels in the supervoxel. Such average is not affected significantly by the color at segment discontinuities.

The computation time of 100ms is not enough to make the algorithm real-time at 30fps. However, a GPU-accelerated implementation which could provide the necessary speedup is likely possible. The effect of ElasticFusion loop closure on the enhancement filter has yet to be investigated. The filter was evaluated on pre-computed poses.

The filter may be applied to real-time incremental segmentation algorithms, as they have been proposed in recent years [96]. Indeed, real-time segmentation may then be applied to the attention system described in Chapter 3.

4.2 Point cloud annotation tool

In previous Section 4.1, a segmentation enhancement algorithm was presented. To evaluate segmentation algorithms, a ground truth is needed. The ground truth is a segmentation produced and validated by a user, who guarantees that the segmentation makes sense for a human. Data annotation is notoriously a tedious and time consuming task for humans. As an example, in the previous section (Table 4.1) the ground truth consists of about one hundred label for each of the five scenarios, each of which is composed of hundreds of thousands of points. The manual assignment of each point to the right label is therefore not attainable, and a faster way has to be developed.

Annotated 3D datasets are increasingly important in research and engineering, since they are needed for algorithm evaluation. Moreover, annotated data is needed by supervised machine learning algorithms (e.g. deep learning), and can support data association and retrieval in databases.

For these reasons, an annotation tool [15], novel with respect to the state of the art, has been developed. The multi-label assisted annotation tool which has been developed has proven to be very efficient, and is a scientific contribution in itself. Indeed, while many methods have been presented for image or organized point cloud annotation, 3D unorganized data requires dealing with selection of volumes and with visualization issues.

In the proposed method, the user selects sparse “control points” on the point cloud. Multiple control points on the same object may be selected and associated to the same label. The selected control points are used as input for an automatic point cloud segmentation algorithm. The segmentation algorithm operates on the point neighborhood graph, weighted by point dissimilarity. Each non-selected point in the point cloud is assigned to the nearest control point by applying a shortest-path tree algorithm. Aiming at the annotation of surfel clouds, the tool exploits color and normal information to compute a more accurate distance function between points. However, the method may work even in absence of this information.

Fig. 4.12 shows a picture of the graphical user interface. The annotation tool was

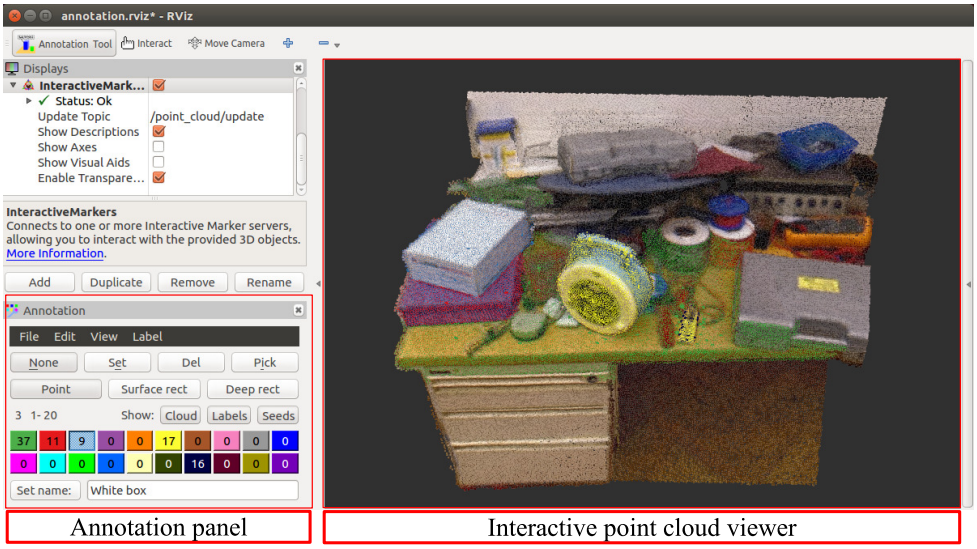


Figure 4.12: The RViz-based interface of the annotation tool.

developed on top of the RViz visualization tool [103] in ROS and has been released as open-source in [104]. The tool uses a standard mouse-based user interface, which is simple to learn and use.

A user study has also been carried out in this section, in order to verify the effectiveness and ease of use of the annotation tool. Results indicate that the proposed annotation method is simpler and faster than a standard technique, used in many 3D software tools, where all points of the objects to be segmented need to be iteratively selected by dragging 2D rectangles on the screen.

The section is subdivided into four subsection. In subsection 4.2.1 the underlying segmentation method is detailed. Optimizations are described for the common case of addition and removal of control points one at a time, in order to improve the algorithm performance. Subsection 4.2.2 describes the user interface. The results of the user evaluation are shown in Subsection 4.2.3 and discussed in subsection 4.2.4.

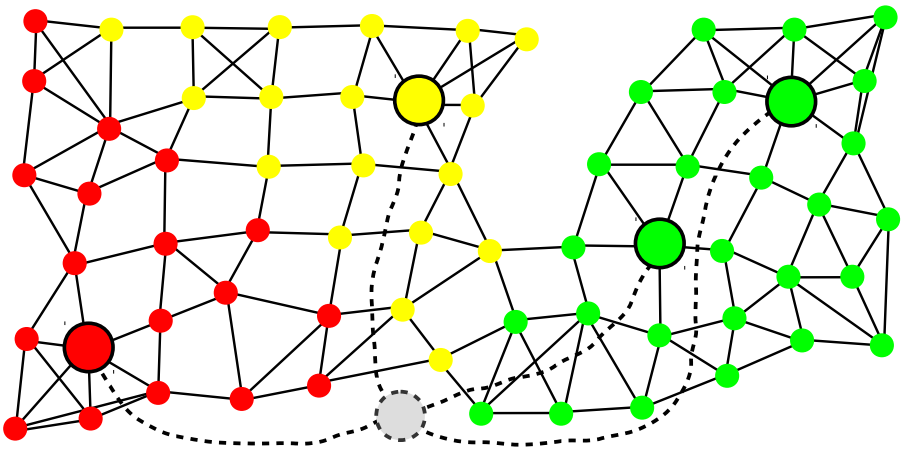


Figure 4.13: Annotation example on the neighborhood graph. The virtual point is displayed in light gray and it is connected to all the control points (dashed arcs).

4.2.1 Method

The undelying segmentation algorithm operates on the nearest neighbor graph of the points in the point cloud. Given a set of selected control points, a shortest-path tree is computed to partition the point cloud into segments. All control points are connected to a virtual node r, which acts as the root of the shortest-path tree. Each control point is the root of a subtree of the virtual node. Points in a subtree share the same label, which is the label of the root control point. Multiple control points may have the same label. Given enough control points, objects in the scene can be properly segmented.

An annotation example is shown in Fig. 4.13. The control points are displayed by larger nodes and are connected to the virtual node (displayed in light grey). Each node acquires the label (color) of the nearest control point reachable in the shortest-path tree. In the reported example two control points have the same green label, therefore, all points in both subtrees have the same label.

In detail, given a point cloud C and a set of user-selected control points $C_P \subset C$, the goal of the algorithm is to assign a label L_i to each point $i \in C$. Each point is

described by a position vector p_i, a RGB color c_i and a normal vector n_i. While position p_i is mandatory, colors and normal vectors are optional. A label L_j is assigned to each control point $j \in C_P$. Also, a subtree label l_i, which contains the index of the subtree root (i.e. the control point), is assigned to each point of the point cloud. Then, the global label L_i of a point is computed as L_{l_i}.

When the point cloud is first loaded, the undirected neighborhood graph is generated. In particular, each point i is connected with its K_{nn} nearest neighbors, which define the neighborhood set N_i. By this definition, if point i has neighbor point $j \in N_i$, it can happen that $i \notin N_j$. In these cases, graph symmetry is ensured by adding i to N_j.

For each edge, a cost is pre-computed as

$$h_{ij} = \alpha_p \|p_i - p_j\| + \alpha_n (1 - n_i \cdot n_j) + \alpha_c \|c_i - c_j\| \tag{4.17}$$

where $(\alpha_p, \alpha_n, \alpha_c)$ are constant weight parameters of position, normal and color properties. The higher the weight the more segmentation is affected by the corresponding property. For example, if a high weight is assigned to color, then segment borders will follow color discontinuities. While α_n and α_c are dimensionless, α_p should be set according to the scale of the point cloud. The edge cost is symmetric, i.e h_{ij}=h_{ji}. Moreover, unless there are duplicate points, the edge cost is always positive.

The shortest-path tree is built from the graph by using Uniform Cost Search (Dijkstra's algorithm variant). A maximum path length Ω_{max} is imposed, so that further points are left unlabeled. When a control point a is added or removed, only points j where

$$\|p_a - p_j\| < \Omega_{max}/\alpha_p \tag{4.18}$$

can change label, even assuming uniform color and normal (according to Eq. 4.17). The maximum path length is added only to guarantee locality of user's selections and to prevent label changes at too large distances. In particular, the selection of the first control point does not label the whole point cloud at once. Therefore, the parameter Ω_{max} is set to a value comparable with α_c and α_n, to prevent effects on the assisted annotation. To be useful, it should also be lower than $\alpha_p W$, where W is the largest dimension of the point cloud.

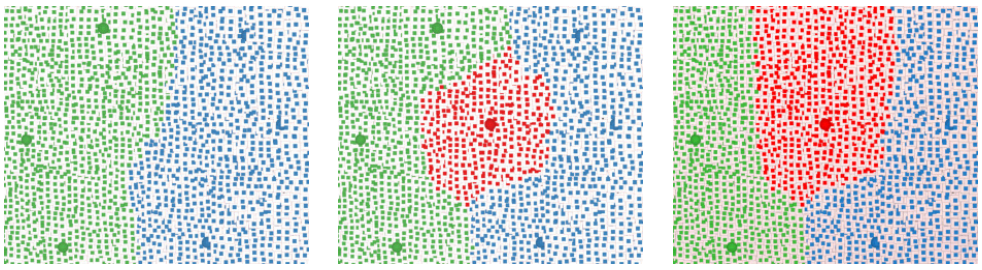


Figure 4.14: Example of point cloud labeling from control point selection and computation of the shortest-path tree. Colors correspond to global labels L_i.

Equation 4.17 depends on the point position, which originates from a sensor affected by noise. Moreover, as stated above, the size of the tree is constrained by the Ω_{max} parameter. Hence, in practice, the presence of multiple shortest paths to the same point is unlikely, i.e. the shortest-path tree is unique, and since the shortest-path tree is defined by C_P and labels by L_i, the annotation is not affected by the order in which control points are processed.

Let ω_i be the shortest-path length for each point i. When there are no control points $\omega_i{=}\infty$ and labels are null, i.e. $\forall i\ l_i{=}\emptyset$. When a control point is added or deleted, all the values ω_i and l_i are updated to reflect the new shortest-path tree configuration.

An example annotation process is shown in Fig. 4.14 for a simple 2D point cloud. Initially three green and three blue control points are selected that split the point cloud into two segments, roughly in the middle (left image). Upon addition of the red control point (center image) a new subtree is generated in the center, reducing the size of both the green and blue segments. Upon deletion of the two top-most control points (right image) the red subtree expands upward.

The procedure described in this section is performed incrementally each time a new control point is added or deleted, as detailed next.

Algorithm 8 Uniform Cost Search

Require: N_i: neighborhood set
Require: h_{ij}: edge cost
Require: S: set of starting nodes
InOut: ω_i: shortest path length for point i
InOut: l_i: subtree label for point i

1: $Q \leftarrow S$
2: **while** Q not empty **do**
3: $\quad i \leftarrow \arg\min_{j \in Q} \omega_j$
4: $\quad Q \leftarrow Q - \{i\}$
5: $\quad$ **for each** $j \in N_i$ **do**
6: $\quad\quad \omega' \leftarrow h_{ij} + \omega_i$
7: $\quad\quad$ **if** $(\omega' < \omega_j)$ **and** $(\omega' < \Omega_{max})$ **then**
8: $\quad\quad\quad l_j \leftarrow l_i$
9: $\quad\quad\quad \omega_j \leftarrow \omega'$
10: $\quad\quad\quad Q \leftarrow Q \cup \{j\}$
11: $\quad\quad$ **end if**
12: $\quad$ **end for**
13: **end while**

Control point addition

When a control point a is added, an edge e_{ar} with zero cost is created that connects the new control point to the virtual node r. This corresponds to setting the shortest-path length ω_a to zero. Since a new subtree has been created the label of the control point a is set to $l_a \leftarrow a$. Then a Uniform Cost Search is performed on the neighborhood graph (as illustrated in Alg. 8), with starting node a stored in set S. The algorithm propagates the label (line 8) and stops when the cost exceeds Ω_{max} (line 7).

The algorithm gives label a to all and only the points of the subtree rooted at control point a, as shown in the following propositions, where apex $'$ marks the state after the current iteration, P_{ij} is the path from i to j and $\left|P_{ij}\right|$ is the path length.

Proposition 1. If a point i is labeled as a by Alg. 8, then the shortest path P_{ir} from i to the root r contains a.

Proof. Alg. 8 updates a point i only if $\omega'_i < \omega_i$ (line 7). If this happens, the new path P'_{ir} includes the new edge e_{ar}. Let us assume, by contradiction, that P'_{ir} would have existed in the input tree with $|P'_{ir}| < |P_{ir}|$. Then, the input tree would not have been a shortest-path tree. □

Proposition 2. When control point a is added, if the shortest path P'_{ri} of a point i with previous label b contains a, then i is labeled as a by Alg. 8.

Proof. By hypothesis, there is a shortest path P'_{ai} with $|P'_{ai}|=|P'_{ri}|<|P_{ri}|=|P_{bi}|$. For every point $j \in P'_{ai}$, it must be $\omega'_j = \left|P'_{aj}\right| < \left|P_{bj}\right| = \omega_j$, otherwise a shorter path may be found by replacing P'_{aj} with P_{bj} in P'_{ri}. When Alg. 8 starts from control point a, there exists at least the path P'_{ai} for point i where $\omega'_j < \omega_j$ for each $j \in P'_{ai}$. Therefore, there exists at least one path, starting from a, which Alg. 8 can follow to relabel i with label a. □

Control point removal

When a control point a is removed the subtree rooted at a is visited (Algorithm 9). All labels of the points within the subtree are cleared (line 8) and their cost is set to infinite (line 9).

The neighboring points $j \in N_i$ with label $l_j \neq a$ are not visited, but they are stored in set S (line 13). These points belong to other subtrees, which may then expand in the newly unlabeled space. Then, Alg. 8 is executed using the points in S as input.

Proposition 3. Given a point i labeled as a, it is reached and cleared by Alg. 9 when control point a is removed.

Proof. All points j in the shortest path P_{ai} are labeled as a if i is labeled as a. Otherwise, if any j was labeled as b, there would exist a path P_{bi} with $|P_{bi}| < |P_{ai}|$, replacing P_{aj} with P_{bj} in P_{ai}. Thus, there exists at least path P_{ai} for Alg. 9 that clears i. □

Proposition 4. When control point a is removed, given a point i previously labeled as a, if the new shortest path P'_{ir} contains control point b, and P'_{ir} is shorter than Ω_{max},

Algorithm 9 Subtree removal

Require: N_i: neighborhood set
Require: d: index of removed control point
Require: h_{ij}: edge cost
InOut: ω_i: shortest path length for point i
InOut: l_i: subtree label for point i
Ensure: $S \subset C$: set of seeds

```
Q ← {d}
l_d ← ∅
ω_d ← ∞
while ∃i ∈ Q do
    Q ← Q − {i}
    for each j ∈ N_i do
        if l_j = l_d then
            l_j ← ∅
            ω_j ← ∞
            Q ← Q ∪ {j}
        else
            if l_j ≠ ∅ then
                S ← S ∪ {j}
            end if
        end if
    end for
end while
```

then by executing Alg. 8 i is relabeled as b.

Proof. Point i was labeled as a, so $a \in P_{ir}$ (shortest path) before removal of edge e_{ar}. Cost of $|P'_{ir}| = |P_{ib}|$, since $h_{br} = 0$. There cannot be any point $j \in P_{ib}$ for which $\omega_j > \Omega_{max}$, otherwise also $\omega_i > \Omega_{max}$ (since $h_{ij} > 0$). Moreover, there cannot be a $j \in P_{ib}$ labeled with a different label $l_j = c$. Otherwise, there would exist a sub-path P_{jc} shorter than P_{jb} and P'_{ir} would contain c with i labeled as c. Thus, any point $j \in P_{ib}$ can only have previous label $l_j \in \{a, b\}$. Then, there is some edge $e_{jk} \in P_{ib}$ which connects a point with label $l_j = b$ and a point with label $l_k = a$. By Proposition 3, the edge is reached by

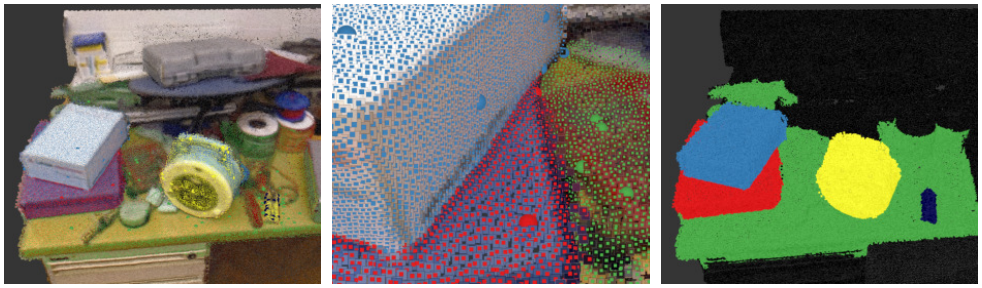


Figure 4.15: Point cloud viewer (left and middle). Labels only (right image).

Alg. 9. The condition at line 7 is false and j is added to S (line 13). Therefore, there exists at least path P_{ji} for Alg. 8 to update point i. □

4.2.2 User interface

The software tool implementing the annotation method was developed on top of the RViz visualization program of the ROS framework. The user interface, shown in Fig. 4.12, consists of two main elements: a tool panel and an interactive 3D point cloud viewer. A ROS node updates in the background the software state and manages all the events generated by the user interface. In particular, mouse events are handled by RViz Interactive Markers.

In the 3D point cloud viewer each point i is rendered as a square, with color c_i, that contains a smaller square colored as the point label (Fig. 4.15). This enables simultaneous visualization of both point colors and their labels. Control points are rendered as spheres. The user can interact with single points of the point cloud by clicking on their square or, in case of control points, on the sphere. The point cloud may be navigated using standard RViz 3D navigation.

The annotation panel (Fig. 4.17) is a RViz Panel plugin. The *menu* allows to save and load annotations. The *label selection* buttons allow selection of the current label L_i. Each button has a different color and the number on it indicates the value of the associated control point. Label colors are generated by the GlasbeyLUT [105]

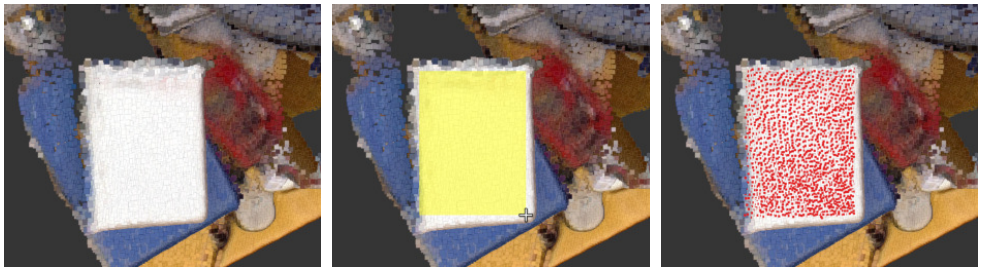


Figure 4.16: Rectangular selection: before selection (left), during selection (center), after selection (right). During selection, a yellow rectangle is displayed on the area being selected.

lookup table included in the Point Cloud Library. Subtree labels l_i are automatically generated and are not visible to the user. Only a limited number of labels can be visualized at once in the *label selection* area. The current label L_{curr} and the range of labels currently visible on the annotation panel are also displayed.

The *action selection* buttons are used to specify the action to be performed upon mouse click on a point of the point cloud: *None* for no action, *Set* to add control points, *Del* to delete control points, *Pick* to set the current label to the one of the clicked point. Thus, the current label can be specified either by pressing a label section button or by picking a previously labeled point.

The *tool selection* buttons are used to specify the selection method: *Point* for the proposed control point selection, *Surface rect* and *Deep rect* for standard rectangular selection. Rectangular selection is performed by dragging the mouse on the screen (Fig. 4.16). In particular, *Surface rect* selects only the visible points in the rectangle, while *Deep rect* selects all points whose projection lies within the rectangle. The user can perform undo/redo operations. The user can also increase and decrease the point size, which is most useful for *Surface rect* selection. If the point size is too small, gaps between points may cause the unwanted selection of background parts.

Using the *toggle visualization* buttons, point rendering mode can be changed. For example, by deselecting the *Cloud* button, each point is rendered with its square

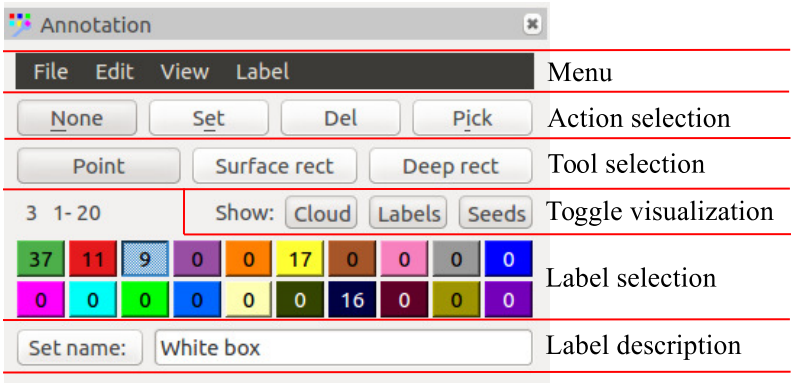


Figure 4.17: The annotation panel.

Parameter	Value	Notes
α_p	1.0	m^{-1}
α_n	0.5	
α_c	1.0	RGB cube edge is 1
Ω_{max}	1.0	
K_{nn}	10	

Table 4.4: Parameters used in the experiments.

filled by the color of its label (Fig. 4.15, right). In this mode, unlabeled points appear black. The user can use this feature to detect segmentation errors. Finally, the user can assign a descriptive text to the current label by typing in the *label description* text box.

The final segmentation can be exported in the standard PCD format used by the Point Cloud Library. An external text file is produced for the label descriptions.

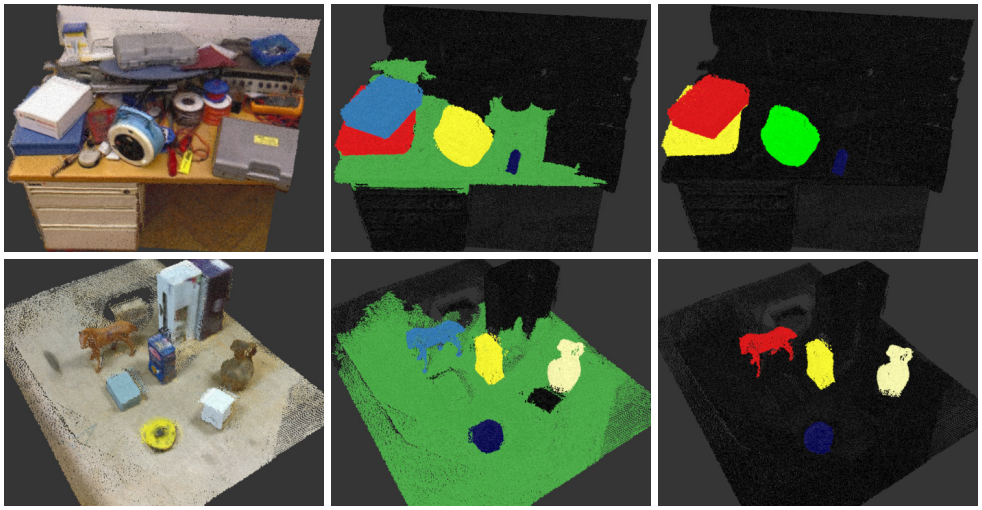


Figure 4.18: Two test scenarios (right and left column) for the user study. Point cloud (first row), ground truth annotations for Control Points selection (second row) and Rectangular Selection (third row).

4.2.3 Results

The proposed approach was evaluated in a user study. Ten persons were asked to annotate the two tabletop scenarios shown in Fig. 4.18. Each user annotated the two scenarios twice, using the proposed Control Points selection (CP) approach and using the Rectangular Selection (RS) technique, in random order. In each trial participants were instructed to annotate the same four objects.

RGB-D data were acquired by using the Asus Xtion Pro Live sensor, while in the second scenario data was acquired by a Kinect V2 sensor. The ElasticFusion 3D reconstruction algorithm [9] was used for 3D reconstruction. The full 3D reconstructions were cropped manually in a box volume, to reduce the size. The resulting point clouds consist of 150868 and 124966 points respectively. Table 4.4 reports the numerical values of the parameters defined in section 4.2.1 used in the experiments. Users were recruited among university students. They were not familiar with range sensing

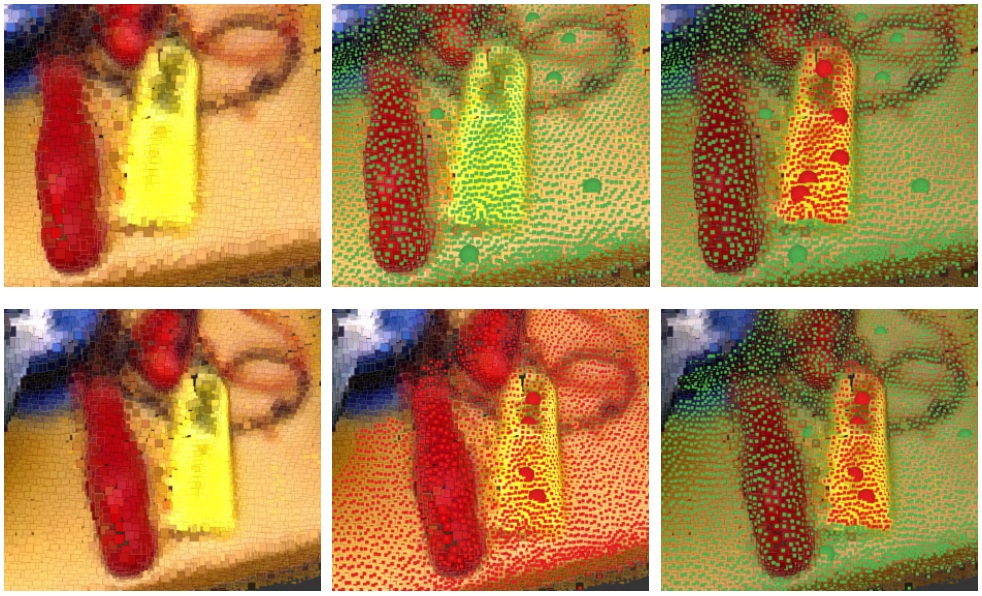


Figure 4.19: Top row: the user first places green control points on the background (middle image) and then red points on the object (right image). Bottom row: the user first places red control points (middle image) on the object and then green control points on the background (right image). The resulting annotations are equivalent.

and segmentation tasks. Most of the users had previous experience with interfaces for navigation in 3D spaces.

Each user performed an initial training session of about 30 minutes on a test point cloud to learn both annotation strategies. During training, participants were given some hints. In particular, for the RS technique users were shown how to change point size to close surface gaps. For the CP annotation approach users were instructed to select a label for the background and to alternate between placing control points on the objects and on the background, in order to refine the annotation. Users should not attempt to exploit the maximum path length (Eq. 4.18) to leave the background unlabeled. In particular, users were suggested to place at least a few control points on the background before annotating the objects themselves (Fig. 4.19). Indeed, even

Index	CP	RS	$p < 0.05$
Time (min.)	6.0 ± 2.2	9.4 ± 1.1	Yes
Undo/min.	0.72 ± 0.5	0.35 ± 0.25	Yes
Errors	1203 ± 687	3551 ± 761	Yes

Table 4.5: Average task completion time, number of Undo operations per minute, and number of annotation errors for scenario 1.

though placing control points on the object first, and then on the background, leads to the same result, it may cause confusion due to the label spreading out of the object.

For the RS technique participants were allowed to use both *Surface rect* selection as well as *Deep rect* selection. A time limit of 10 minutes was given to the users to complete each trial. Users were free to complete a trial before the time limit. Tables 4.5 and 4.6 report average results with standard deviation. Statistical significance has been assessed by standard correlated t-test and reported in the "$p < 0.05$" column. It can be noticed that participants were significantly faster when using the proposed control point annotation method. Users placed on average 26.5 ± 14.9 control points in the first scenario and 31.0 ± 11.0 in the second scenario. The difference in the number of Undo operations was also statistically significant for scenario 1. Indeed, users performed more Undo operations with the Control Point annotation technique to remove wrong control points. Instead, in the case of an incorrect rectangular selection, users often chose to refine the annotation rather than undoing the whole action.

The error score was computed by comparing the result with two ground truth segmentation produced using CS and RS, produced by myself without time limit. The two ground truth annotations differ by 860 (scenario 1) and 674 (scenario 2) point labels respectively. The error score counts the number of points having a different label with respect to the ground truth annotation, including points with null label. The background label was considered equivalent to the null label. Results indicate that the number of errors is significantly lower for the Control Point method.

Common errors with the RS method arise from stray points and missing points.

Index	CP	RS	$p < 0.05$
Time (min.)	6.8 ± 2.1	9.5 ± 1.1	Yes
Undo/min.	0.75 ± 0.76	0.53 ± 0.41	No
Errors	926 ± 346	4349 ± 1744	Yes

Table 4.6: Average task completion time, number of Undo operations per minute, and number of annotation errors for scenario 2.

Stray points (Fig. 4.20) appear in case of occluding objects whenever the *Deep rect* tool was used, or whenever the point size was not properly set to close surface gaps. Additionally, stray points may be appear when the boundaries of the objects to be segmented are wrongly estimated due to similarities with the background. In case of well separated objects, stray points may be easily deleted by changing the current viewpoint, as some users did. Instead, stray points are barely removable in case of occlusions.

Missing points may be caused by the *Surface rect* tool, which does not select points behind the visible surface. Instead, in CP mode occluded points, being connected by the neighborhood graph, are selected and included in the subtree of a control point. Fig. 4.21 illustrates this issue where (black) missing points appear inside a labeled object.

Stray points may also affect the CP method, when control points are accidentally selected through the gaps of the visible surface. In this case, however, the effect on the overall segmentation is clearly visible, and users corrected them immediately. Difficulties to find the control point have been reported. Some users also asked the feature of deleting a control point by just clicking on any point belonging to its subtree.

Users were also asked to answer a questionnaire at the end of their session to provide a qualitative evaluation of the annotation methods. Participants rated questions from 0 to 10 on a Likert scale (with 0 as the lowest and 10 as the highest) regarding learnability, ease of use, (perceived) efficiency, (perceived) effectiveness, and a global rating. Results of the questionnaire are summarized in Table 4.7. Most indices

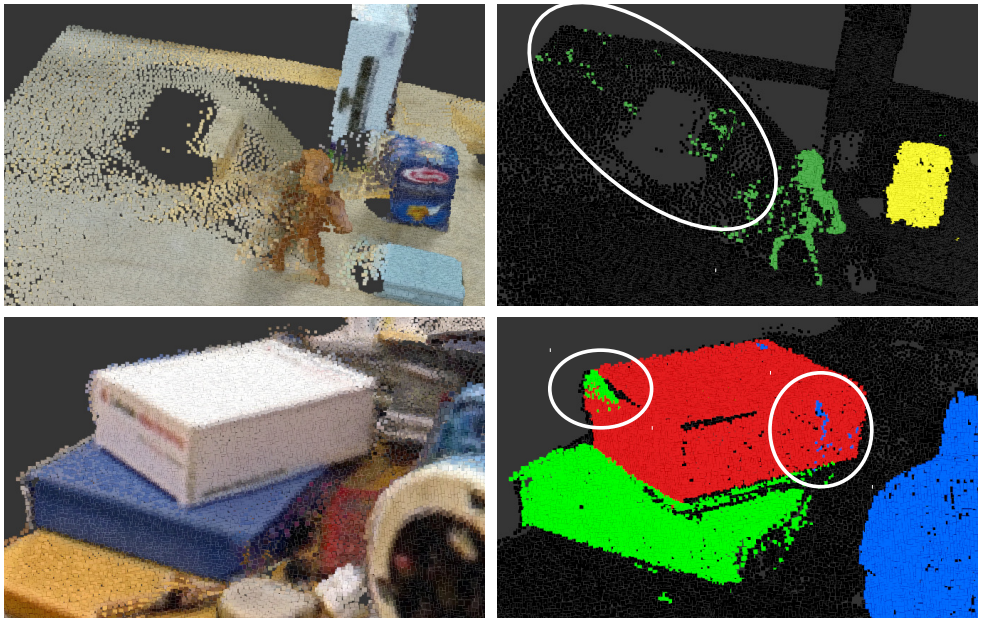


Figure 4.20: Stray points (highlighted by circles) behind the small horse object (top right image) and on the white box (bottom right image).

are statistically significant in favor of the proposed CP method. No significant difference was found in learnability, which is remarkable since Rectangular Selection is a standard selection approach for many software applications. Hence, it may be concluded that the method is intuitive and that the developed user interface is easy to use. Some participants suggested changes in the Rectangular Selection, such as the ability of using a polyline instead of a rectangular shape. Moreover, a "smarter" RS solution was requested, capable of selecting the points slightly below the surface automatically using some heuristics.

Table 4.8 shows average execution times of the underlying algorithms, for scenario 1 (150868 points). Experiments were performed on an Intel Core i7 4770 at 3.40 GHz. The neighborhood graph construction is the most expensive operation, however, it is only executed once. Memory usage is about 150 MB for the point clouds

Figure 4.21: A cork jug seen from the inside (left). RS annotation (middle) and CP annotation (right). Missing (black) points appear in the RS case.

used in the tests. Computing annotation from the control points using Uniform Cost Search takes 41 ms for 90 control points. As the number of control points increases, the delay may become noticeable by the user. Incremental addition and removal of single control points is more efficient, and is the operation that users perform most frequently.

Index	CP	RS	$p < 0.05$
Learnability	8.9 ± 0.8	8.6 ± 0.9	No
Ease of use	8.3 ± 1.1	7.1 ± 1.0	Yes
Efficiency	7.4 ± 2.1	5.8 ± 1.9	Yes
Effectiveness	8.2 ± 0.8	7.3 ± 1.3	Yes
Global eval.	8.9 ± 0.9	7.8 ± 1.1	Yes

Table 4.7: Questionnaire results, showing subjective evaluation by the users.

Operation	Time (ms)
Neighborhood graph	520
Computing annotation	41
Addition	10
Deletion	11

Table 4.8: Execution times.

4.2.4 Discussion

In this section, an annotation tool for unorganized point clouds was presented. The approach was compared to a standard rectangular selection technique by using an annotated ground truth dataset. Users provided a more positive feedback about their experience when using the developed annotation tool. Moreover, when compared with the ground truth, the resulting annotation contained less errors on average.

The proposed method may also be suitable for collaborative annotation. The use of control points may reduce accidental interference between users working on different parts of the scene. The distributed architecture of the underlying ROS framework may be exploited in order to connect the interface on multiple machines.

A limit of the proposed approach involves the parameters of the segmentation algorithm which, in the reported experiments, were fixed to constant values (Table 4.4). Indeed, active control for these parameters may be required by the user, in relation to the local characteristics and information density of the point cloud.

The current implementation requires both the point cloud and the neighborhood graph to fit in memory at once. For larger datasets, this may not be feasible. However, the updating procedure is spatially local, as defined in Subsection 4.2.1. Therefore, the algorithm could be extended in order to operate with only a part of the graph in memory at once.

Chapter 5

Conclusions

In this dissertation, novel attention-based approaches for next-best view planning have been presented, using a robot manipulator equipped with a range sensor (Kinect) in eye-in-hand configuration. The robot creates the 3D representation of a tabletop scenario using state-of-the-art reconstruction algorithms. The next-best view algorithm computes the optimal sensor placement in order to maximize information gain. Attention was used to optimize next-best view and to focus the views towards the goal of the robot.

In the first proposed approach, the goal of the robot was to update the 3D representation of the environment after a user performed object manipulation actions. Relevant manipulation actions are detected by tracking the motion of the human hand and by applying a GMM-based algorithm for saliency estimation. The approximate locations of the user actions are used to orient the sensor towards the regions where the changes are likely to have happened. It was shown that, by focusing on the salient locations, the robot is able to update the representation without re-scanning the whole environment.

Moreover, the KinFu implementation of the KinectFusion algorithm was modified to perform next-best view planning in the TSDF volume on GPU. A data structure was added to store the difference between unknown and empty space, which would have been lost during shifting operations. A high performance improvement

was demonstrated due to that modification, since next-best view could exploit the KinFu internal ray-casting algorithm.

The second approach proposed in this thesis is a non-model-based next-best view method for an exploration system. The tabletop scenario is initially unknown and the goal of the robot is to explore objects as they were discovered. A segmentation algorithm is used to split the 3D representation. A heuristic was developed to assign a high saliency value to segments which represent objects. Segments with high saliency are prioritized during exploration. Moreover, the candidate view generation for next-best view is performed using the frontiers in the KinectFusion TSDF volume.

It was shown that the robot prioritizes the completion of already discovered objects before exploring the other parts of the scene. The prioritization of some views determines a reduction of the computation time, since less views are evaluated at each next-best view iteration.

In this second application, the Kinect V2 sensor was used, in place of the Kinect V1. A novel depth enhancement filter was developed, in order to remove some artifacts produced by the sensor.

In addition to the two attention-based approaches outlined above, an initial investigation has been performed towards the implementation of next-best view in surfel-based 3D reconstruction. Surfel-based 3D reconstruction exploits the GPU rendering pipeline and promises better performance than volumetric reconstruction. However, unlike KinectFusion, surfel-based 3D representation does not track empty space.

A mode filter has been developed in order to reduce blur due to the averaging process in the surfel-based 3D reconstruction. It was demonstrated that the filter enables better segmentation to be produced from surfel clouds. To evaluate the segmentation, a user-friendly assisted annotation tool was also developed, which has been shown to outperform standard rectangle-based annotation.

Several directions for further research have been left open by this thesis. The algorithm used for next-best view finds the optimal sensor placement and disregards its trajectory to reach the desired configuration. Information acquisition during sensor movement was evaluated in Section 2.4.3, but the improvement was shown to be limited. A next-best trajectory algorithm, which finds the optimal sensor trajectory for

data acquisition, may be investigated. Moreover, real-time segmentation algorithms have been recently developed and may be integrated with the KinectFusion TSDF volume. Hence, a way to produce saliency in real-time may be investigated.

The surfel-based next-best view approach suggested in Chapter 4 will also be developed in future research.

Bibliography

[1] Simon Kriegel. *Autonomous 3D Modeling of Unknown Objects for Active Scene Exploration*. PhD dissertation, Technische Universität München, München, 2015.

[2] K. Xu, H. Huang, Y. Shi, H. Li, P. Long, J. Caichen, W. Sun, and B. Chen. Autoscanning for coupled scene reconstruction and proactive object analysis. *ACM Trans. Graph.*, 34(6):177:1–177:14, October 2015.

[3] B. Schauerte, J. Richarz, and G.A. Fink. Saliency-based identification and recognition of pointed-at objects. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 4638–4643, Oct 2010.

[4] Sang Hyoung Lee, Il Hong Suh, S. Calinon, and R. Johansson. Learning Basis Skills by Autonomous Segmentation of Humanoid Motion Trajectories. In *IEEE-RAS Intl Conference on Humanoid Robots*, pages 112–119, Nov 2012.

[5] D. Min, J. Lu, and M. N. Do. Depth video enhancement based on weighted mode filtering. *IEEE Transactions on Image Processing*, 21(3):1176–1190, March 2012.

[6] S. C. Stein, M. Schoeler, J. Papon, and F. Wörgötter. Object partitioning using local convexity. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 304–311, June 2014.

[7] A. Golovinskiy and T. Funkhouser. Min-cut based segmentation of point clouds. In *IEEE 12th International Conference on Computer Vision Workshops, ICCV Workshops*, pages 39–46, Sept 2009.

[8] R. A. Newcombe, S. Izadi, O. Hilliges, D. Molyneaux, D. Kim, A. J. Davison, P. Kohi, J. Shotton, S. Hodges, and A. Fitzgibbon. Kinectfusion: Real-time dense surface mapping and tracking. In *10th IEEE International Symposium on Mixed and Augmented Reality*, pages 127–136, Oct 2011.

[9] Thomas Whelan, Stefan Leutenegger, Renato Salas Moreno, Ben Glocker, and Andrew Davison. ElasticFusion: Dense SLAM without a pose graph. In *Proceedings of Robotics: Science and Systems*, Rome, Italy, July 2015.

[10] C. Connolly. The Determination of Next Best Views. In *IEEE Intl Conference on Robotics and Automation (ICRA)*, volume 2, pages 432–435, 1985.

[11] R. Monica, J. Aleotti, and S. Caselli. A KinFu based approach for robot spatial attention and view planning. *Robotics and Autonomous Systems*, 75, Part B:627–640, 2016.

[12] R. Monica, J. Aleotti, and S. Caselli. GMM-based detection of human hand actions for robot spatial attention. In *International Conference on Advanced Robotics (ICAR)*, July 2015.

[13] R. Monica and J. Aleotti. Contour-based next-best view planning from point cloud segmentation of unknown objects. *Autonomous Robots*, 2017, DOI:10.1007/s10514-017-9618-0.

[14] R. Monica, M. Zillich, M. Vincze, and J. Aleotti. RGB-D fusion enhancement by mode filter for surfel cloud segmentation. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2017.

[15] R. Monica, J. Aleotti, M. Zillich, and M. Vincze. Multi-label point cloud annotation by selection of sparse control points. In *International Conference on 3D Vision (3DV)*, 2017.

[16] J.E. Banta, L. R. Wong, C. Dumont, and M.A. Abidi. A Next-Best-View System for Autonomous 3-D Object Reconstruction. *IEEE Transactions on Systems, Man and Cybernetics, Part A: Systems and Humans*, 30(5):589–598, 2000.

[17] R. Pito. A Solution to the Next Best View Problem for Automated Surface Acquisition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 21(10):1016–1030, 1999.

[18] K. Klein and V. Sequeira. The view-cube: an efficient method of view planning for 3d modelling from range data. In *Proceedings Fifth IEEE Workshop on Applications of Computer Vision*, pages 186–191, 2000.

[19] P. Whaite and F.P. Ferrie. Autonomous Exploration: Driven by Uncertainty. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 19(3):193–205, 1997.

[20] K. Morooka, Hongbin Zha, and T. Hasegawa. Next Best Viewpoint (NBV) Planning for Active Object Modeling Based on a Learning-by-Showing Approach. In *Fourteenth Intl Conference on Pattern Recognition*, volume 1, pages 677–681 vol.1, 1998.

[21] Y.F. Li and Z.G. Liu. Information Entropy-Based Viewpoint Planning for 3-D Object Reconstruction. *IEEE Transactions on Robotics*, 21(3):324–337, 2005.

[22] S.Y. Chen and Y.F. Li. Vision Sensor Planning for 3-D Model Acquisition. *IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics*, 35(5):894–904, 2005.

[23] L. Torabi and K. Gupta. Integrated View and Path Planning for an Autonomous six-DOF Eye-in-hand Object Modeling System. In *IEEE/RSJ Intl Conference on Intelligent Robots and Systems (IROS)*, pages 4516–4521, 2010.

[24] S. Foix, G. Alenyà, J. Andrade-Cetto, and C. Torras. Object Modeling Using a ToF Camera under an Uncertainty Reduction Approach. In *IEEE Intl Conference on Robotics and Automation (ICRA)*, pages 1306–1312, 2010.

[25] G. Walck and M. Drouin. Automatic Observation for 3D Reconstruction of Unknown Objects Using Visual Servoing. In *IEEE/RSJ Intl Conference on Intelligent Robots and Systems (IROS)*, pages 2727–2732, 2010.

[26] M.K. Reed and P.K. Allen. Constraint-Based Sensor Planning for Scene Modeling. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22(12):1460–1467, 2000.

[27] J.I. Vasquez-Gomez, E. Lopez-Damian, and L.E. Sucar. View Planning for 3D Object Reconstruction. In *IEEE/RSJ Intl Conference on Intelligent Robots and Systems (IROS)*, pages 4015–4020, 2009.

[28] C. Potthast and G. S. Sukhatme. A probabilistic framework for next best view estimation in a cluttered environment. *Journal of Visual Communication and Image Representation*, 25(1):148–164, 2014.

[29] B. Adler, Junhao Xiao, and Jianwei Zhang. Finding next best views for autonomous UAV mapping through GPU-accelerated particle simulation. In *IEEE/RSJ Intl Conference on Intelligent Robots and Systems (IROS)*, pages 1056–1061, Nov 2013.

[30] S. Kriegel, M. Brucker, Z. C. Marton, T. Bodenmuller, and M. Suppa. Combining Object Modeling and Recognition for Active Scene Exploration. In *IEEE/RSJ Intl Conference on Intelligent Robots and Systems (IROS)*, pages 2384–2391, 2013.

[31] S. Kriegel, C. Rink, T. Bodenmuller, A. Narr, M. Suppa, and G. Hirzinger. Next-Best-Scan Planning for Autonomous 3D Modeling. In *IEEE/RSJ Intl Conference on Intelligent Robots and Systems (IROS)*, pages 2850–2856, 2012.

[32] S. Kriegel, T. Bodenmuller, M. Suppa, and G. Hirzinger. A surface-based next-best-view approach for automated 3D model completion of unknown objects. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 4869–4874, May 2011.

[33] N. Atanasov, B. Sankaran, J. Le Ny, G.J. Pappas, and K. Daniilidis. Nonmyopic view planning for active object classification and pose estimation. *IEEE Transactions on Robotics*, 30(5):1078–1090, Oct 2014.

[34] D. Stampfer, M. Lutz, and C. Schlegel. Information driven sensor placement for robust active object recognition based on multiple views. In *IEEE International Conference on Technologies for Practical Robot Applications (TePRA)*, pages 133–138, April 2012.

[35] T. Patten, M. Zillich, R. Fitch, M. Vincze, and S. Sukkarieh. Viewpoint evaluation for online 3-D active object classification. *IEEE Robotics and Automation Letters*, 1(1):73–81, Jan 2016.

[36] K. Welke, J. Issac, D. Schiebener, T. Asfour, and R. Dillmann. Autonomous Acquisition of Visual Multi-View Object Representations for Object Recognition on a Humanoid Robot. In *IEEE Intl Conference on Robotics and Automation (ICRA)*, pages 2012–2019, 2010.

[37] A. Tsuda, Y. Kakiuchi, S. Nozawa, R. Ueda, K. Okada, and M. Inaba. On-line Next Best Grasp Selection for In-Hand Object 3D Modeling with Dual-Arm Coordination. In *IEEE Intl Conference on Robotics and Automation (ICRA)*, pages 1799–1804, 2012.

[38] G. Kahn, P. Sujan, S. Patil, S. Bopardikar, J. Ryde, K. Goldberg, and P. Abbeel. Active exploration using trajectory optimization for robotic grasping in the presence of occlusions. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 4783–4790, May 2015.

[39] R. Wagner, U. Frese, and B. Bauml. Real-time dense multi-scale workspace modeling on a humanoid robot. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 5164–5171, Nov 2013.

[40] Kanzhi Wu, R. Ranasinghe, and G. Dissanayake. Active recognition and pose estimation of household objects in clutter. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 4230–4237, May 2015.

[41] H. van Hoof, O. Kroemer, and J. Peters. Probabilistic segmentation and targeted exploration of objects in cluttered environments. *IEEE Transactions on Robotics*, 30(5):1198–1209, Oct 2014.

[42] D. Beale, P. Iravani, and P. Hall. Probabilistic models for robot-based object segmentation. *Robotics and Autonomous Systems*, 59(12):1080 – 1089, 2011.

[43] L.M.G. Gonçalves, A.A.F. Oliveira, and R.A. Grupen. A framework for attention and object categorization using a stereo head robot. In *XII Brazilian Symposium on Computer Graphics and Image Processing, 1999*, pages 143–152, 1999.

[44] F. Orabona, G. Metta, and G. Sandini. Object-based visual attention: a model for a behaving robot. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 89–89, June 2005.

[45] P. Drews, P. Núñez, R. Rocha, M. Campos, and J. Dias. Novelty detection and segmentation based on gaussian mixture models: A case study in 3D robotic laser mapping. *Robotics and Autonomous Systems*, 61(12):1696–1709, 2013.

[46] P. Alimi, D. Meger, and J.J. Little. Object persistence in 3D for home robots. In *The Semantic Perception, Mapping, and Exploration (SPME) workshop*, 2012.

[47] R. Finman, T. Whelan, M. Kaess, and J.J. Leonard. Toward lifelong object segmentation from change detection in dense RGB-D maps. In *European Conference on Mobile Robots (ECMR)*, pages 178–185, Sept. 2013.

[48] E. Herbst, Xiaofeng Ren, and D. Fox. RGB-D object discovery via multi-scene analysis. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 4850–4856, Sept. 2011.

[49] E. Herbst, P. Henry, and D. Fox. Toward online 3-D object segmentation and mapping. In *IEEE Intl Conference on Robotics and Automation (ICRA)*, pages 3193–3200, 2014.

[50] Z. Yücel, A.A. Salah, C. Meriçli, T. Meriçli, R. Valenti, and T. Gevers. Joint attention by gaze interpolation and saliency. *IEEE Transactions on Cybernetics*, 43(3):829–842, 2013.

[51] S. Petsch and D. Burschka. Representation of manipulation-relevant object properties and actions for surprise-driven exploration. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 1221–1227, Sept 2011.

[52] A. Fathi and J.M. Rehg. Modeling actions through state changes. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2579–2586, June 2013.

[53] Sing Bing Kang and K. Ikeuchi. Toward Automatic Robot Instruction from Perception-Temporal Segmentation of Tasks from Human Hand Motion. *IEEE Transactions on Robotics and Automation*, 11(5):670–681, Oct 1995.

[54] M. Yeasin and S. Chaudhuri. Toward automatic robot programming: learning human skill from visual data. *IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics*, 30(1):180–185, Feb 2000.

[55] J.F.-S. Lin and D. Kulic. Online Segmentation of Human Motion for Automated Rehabilitation Exercise Analysis. *IEEE Transactions on Neural Systems and Rehabilitation Engineering*, 22(1):168–180, Jan 2014.

[56] D.R. Faria and J. Dias. 3D hand trajectory segmentation by curvatures and hand orientation for classification through a probabilistic approach. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 1284–1289, Oct 2009.

[57] D. R. Faria, R. Martins, J. Lobo, and J. Dias. Extracting data from human manipulation of objects towards improving autonomous robotic grasping. *Robotics and Autonomous Systems*, 60(3):396–410, 2012.

[58] E. E. Aksoy, A. Abramov, J. Dörr, K. Ning, B. Dellen, and F. Wörgötter. Learning the semantics of object-action relations by observation. *Int. J. Rob. Res.*, 30(10):1229–1249, September 2011.

[59] K. He, J. Sun, and X. Tang. Guided image filtering. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 35(6):1397–1409, 2013.

[60] Johannes Kopf, Michael F. Cohen, Dani Lischinski, and Matt Uyttendaele. Joint bilateral upsampling. *ACM Trans. Graph.*, 26(3), July 2007.

[61] R. Takaoka and N. Hashimoto. Depth map super-resolution for cost-effective rgb-d camera. In *International Conference on Cyberworlds (CW)*, pages 133–136, Oct 2015.

[62] Oliver Wasenmüller, Gabriele Bleser, and Didier Stricker. Combined bilateral filter for enhanced real-time upsampling of depth images. In *International Conference on Computer Vision Theory and Applications (VISIGRAPP)*, pages 5–12, 2015.

[63] T. W. Hui and K. N. Ngan. Depth enhancement using rgb-d guided filtering. In *IEEE International Conference on Image Processing (ICIP)*, pages 3832–3836, Oct 2014.

[64] J. Wasza, S. Bauer, and J. Hornegger. Real-time preprocessing for dense 3-d range imaging on the gpu: Defect interpolation, bilateral temporal averaging and guided filtering. In *IEEE International Conference on Computer Vision Workshops (ICCV Workshops)*, pages 1221–1227, 2011.

[65] J. Van de Weijer and R. Van den Boomgaard. Local mode filtering. In *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, volume 2, pages II–428–II–433 vol.2, 2001.

[66] S. Matyunin, D. Vatolin, Y. Berdnikov, and M. Smirnov. Temporal filtering for depth maps generated by kinect depth camera. In *3DTV Conference: The True*

Vision - Capture, Transmission and Display of 3D Video (3DTV-CON), pages 1–4, May 2011.

[67] C. Dorea and R. L. de Queiroz. Depth map reconstruction using color-based region merging. In *18th IEEE International Conference on Image Processing*, pages 1977–1980, Sept 2011.

[68] P. K. Rana, J. Taghia, Z. Ma, and M. Flierl. Probabilistic multiview depth image enhancement using variational inference. *IEEE Journal of Selected Topics in Signal Processing*, 9(3):435–448, April 2015.

[69] A. Uckermann, C. Eibrechter, R. Haschke, and H. Ritter. Real-time hierarchical scene segmentation and classification. In *14th IEEE-RAS International Conference on Humanoid Robots (Humanoids)*, pages 225–231, Nov 2014.

[70] K. Tateno, F. Tombari, and N. Navab. When 2.5d is not enough: Simultaneous reconstruction, segmentation and recognition on dense slam. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 2295–2302, May 2016.

[71] T. Rabbani, F. A. Van Den Heuvel, and G. Vosselman. Segmentation of point clouds using smoothness constraint. *International Archives of Photogrammetry, Remote Sensing and Spatial Information Sciences*, 36(5):248–253, 2006.

[72] Oliver Van Kaick, Noa Fish, Yanir Kleiman, Shmuel Asafi, and Daniel Cohen-OR. Shape segmentation by approximate convexity analysis. *ACM Trans. Graph.*, 34(1):4:1–4:11, December 2014.

[73] J. Papon, A. Abramov, M. Schoeler, and F. Worgotter. Voxel cloud connectivity segmentation - supervoxels for point clouds. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 2027–2034, June 2013.

[74] F. P. Tasse, J. Kosinka, and N. Dodgson. Cluster-based point set saliency. In *IEEE International Conference on Computer Vision (ICCV)*, pages 163–171, Dec 2015.

[75] D. Wolf, J. Prankl, and M. Vincze. Enhancing semantic segmentation for robotics: The power of 3-d entangled forests. *IEEE Robotics and Automation Letters*, 1(1):49–56, Jan 2016.

[76] B. C. Russell and A. Torralba. Building a database of 3D scenes from user annotations. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 2711–2718, June 2009.

[77] Bryan C. Russell, Antonio Torralba, Kevin P. Murphy, and William T. Freeman. LabelMe: a database and web-based tool for image annotation. *International Journal of Computer Vision*, 77(1):157–173, 2008.

[78] Yu-Shiang Wong, Hung-Kuo Chu, and Niloy J. Mitra. SmartAnnotator an interactive tool for annotating indoor RGBD images. *Computer Graphics Forum*, 34(2):447–457, 2015.

[79] Tianjia Shao, Weiwei Xu, Kun Zhou, Jingdong Wang, Dongping Li, and Baining Guo. An interactive approach to semantic modeling of indoor scenes with an RGBD camera. *ACM Trans. Graph.*, 31(6):136:1–136:11, November 2012.

[80] J. Xiao, A. Owens, and A. Torralba. SUN3D: A database of big spaces reconstructed using SfM and object labels. In *IEEE International Conference on Computer Vision*, pages 1625–1632, Dec 2013.

[81] Felipe Bacim, Regis Kopper, and Doug A. Bowman. Design and evaluation of 3D selection techniques based on progressive refinement. *International Journal of Human-Computer Studies*, 71(7–8):785–802, 2013.

[82] L. Yu, K. Efstathiou, P. Isenberg, and T. Isenberg. Efficient structure-aware selection techniques for 3D point cloud visualizations with 2DOF input. *IEEE Transactions on Visualization and Computer Graphics*, 18(12):2245–2254, Dec 2012.

[83] D. Coffey, N. Malbraaten, T. Le, I. Borazjani, F. Sotiropoulos, A. G. Erdman, and D. F. Keefe. Interactive Slice WIM: Navigating and interrogating volume

data sets using a multisurface, multitouch VR interface. *IEEE Transactions on Visualization and Computer Graphics*, 18(10):1614–1626, October 2012.

[84] F. Bacim, M. Nabiyouni, and D. A. Bowman. Slice-n-Swipe: A free-hand gesture user interface for 3D point cloud annotation. In *IEEE Symposium on 3D User Interfaces (3DUI)*, pages 185–186, March 2014.

[85] P. Lubos, R. Beimler, M. Lammers, and F. Steinicke. Touching the Cloud: Bimanual annotation of immersive point clouds. In *IEEE Symposium on 3D User Interfaces (3DUI)*, pages 191–192, March 2014.

[86] M. Veit and A. Capobianco. Go'Then'Tag: A 3-D point cloud annotation technique. In *IEEE Symposium on 3D User Interfaces (3DUI)*, pages 193–194, March 2014.

[87] A. Boyko and T. Funkhouser. Cheaper by the dozen: Group annotation of 3d data. In *Proceedings of the 27th Annual ACM Symposium on User Interface Software and Technology (UIST)*, pages 33–42, New York, NY, USA, 2014.

[88] A. Boyko. *Efficient Interfaces for Accurate Annotation of 3D Point Clouds*. PhD thesis, Princeton University, February 2015.

[89] K. Liu and J. Boehm. A new framework for interactive segmentation of point clouds. *ISPRS - International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, XL-5:357–362, 2014.

[90] *KinectFusion extensions to large scale environments*. `http://www.pointclouds.org/blog/srcs/fheredia/index.php`.

[91] R.B. Rusu and S. Cousins. 3D is Here: Point Cloud Library (PCL). In *IEEE Intl Conference on Robotics and Automation (ICRA)*, Shanghai, China, May 9-13 2011.

[92] D. Berenson, R. Diankov, K. Nishiwaki, S. Kagami, and J. Kuffner. Grasp planning in complex scenes. In *7th IEEE-RAS Int'l Conference on Humanoid Robots*, pages 42–48, nov. 2007.

[93] H. Roth and M. Vona. Moving Volume KinectFusion. In *Proceedings of the British Machine Vision Conference*, pages 112.1–112.11. BMVA Press, 2012.

[94] Michael Korn. *KinFu ROS wrapper*. `http://fsstud.is.uni-due.de/svn/ros/is/kinfu`.

[95] S.C. Stein, F. Worgotter, M. Schoeler, J. Papon, and T. Kulvicius. Convexity based object partitioning for robot applications. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 3213–3220, 2014.

[96] K. Tateno, F. Tombari, and N. Navab. Real-time and scalable incremental segmentation on dense SLAM. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 4465–4472, Sept 2015.

[97] A. Uckermann, R. Haschke, and H. Ritter. Real-time 3D segmentation of cluttered scenes for robot grasping. In *12th IEEE-RAS International Conference on Humanoid Robots (Humanoids)*, pages 198–203, Nov 2012.

[98] M. Keller, D. Lefloch, M. Lambers, S. Izadi, T. Weyrich, and A. Kolb. Real-time 3d reconstruction in dynamic scenes using point-based fusion. In *International Conference on 3D Vision - 3DV*, pages 1–8, June 2013.

[99] Francisco J. Estrada and Allan D. Jepson. Benchmarking image segmentation algorithms. *International Journal of Computer Vision*, 85(2), 2009.

[100] D. R. Martin, J. Malik, and D. Patterson. *An empirical approach to grouping and segmentation*. Computer Science Division, University of California, 2003.

[101] D. Martin, C. Fowlkes, D. Tal, and J. Malik. A database of human segmented natural images and its application to evaluating segmentation algorithms and measuring ecological statistics. In *Proceedings Eighth IEEE International Conference on Computer Vision ICCV*, volume 2, pages 416–423 vol.2, 2001.

[102] S. V. Alexandrov, J. Prankl, M. Zillich, and M. Vincze. Calibration and correction of vignetting effects with an application to 3d mapping. In *IEEE/RSJ In-*

ternational Conference on Intelligent Robots and Systems (IROS), pages 4217–4223, Oct 2016.

[103] Dave Hershberger, David Gossow, and Josh Faust. RViz, 3D visualization tool for ROS. URL: http://wiki.ros.org/rviz.

[104] Riccardo Monica. RViz Cloud Annotation Tool. URL: https://github.com/RMonica/rviz_cloud_annotation.

[105] Chris Glasbey, Gerie van der Heijden, Vivian F. K. Toh, and Alision Gray. Colour displays for categorical images. *Color Research & Application*, 32(4):304–309, 2007.

Acknowledgments

First of all, I wish to thank my advisor, Prof. Jacopo Aleotti, who supported my PhD and helped my research activity. Also, I thank Prof. Stefano Caselli, head of the Robotics and Intelligent Machines Laboratory, who distracted me from research to do, from time to time, something different. I thank Prof. Dario Lodi Rizzini, for his optimistic advice, to compensate for my advisor's pessimism (and mine).

Of the people in Vienna, I thank Prof. Markus Vincze, head of the Vision4Robotics laboratory at Technische Universität Wien. He allowed me to do research at his laboratory for six months, and I have learned a lot. I also thank all the PhD students and post-docs working there, for trying to befriend me. An acknowledgement goes to Sergey V. Alexandrov, for sharing his ElasticFusion dataset.

I thank my office mates, Francesco Denaro and Eleonora Iotti, who kept me company after my time in Vienna. I also wish to thank the students I helped during their bachelor thesis, because from me they just learned notions, but from them I learned about people. I expressly recall Andrea Zinelli, because he is a pain to deal with but, yes, I had fun working with him.

Special thanks go to Prof. Riccardo Roncella, of the photogrammetry laboratory, Prof. Alberto Bononi, of the Machine Learning for Pattern Recognition course, and Dr. Michael Zillich, of the Vision4Robotics laboratory. These three unrelated personalities have independently attempted to explain to me what research actually is. After the third explanation, I think I am beginning to understand.