UNIVERSITY COLLEGE LONDON

MASTER PROJECT DISSERTATION

Incremental instance-level 3D segmentation performed by fusing geometric and deep-learning cues

---

Author:
Maud BUFFIER

Supervisor:
Dr. Lourdes AGAPITO

This report is submitted as part requirement for the MSc Degree of Computer Graphic, Vision and Imaging

September 7, 2017

Declaration of Authorship

I, Maud BUFFIER, declare that this thesis titled, "Incremental instance-level 3D segmentation performed by fusing geometric and deep-learning cues" and the work presented in it are my own. I confirm that:

- This work was done wholly or mainly while in candidature for a research degree at this University.
- Where any part of this thesis has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated.
- Where I have consulted the published work of others, this is always clearly attributed.
- Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work.
- I have acknowledged all main sources of help.
- Where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself.

Signed: 

Date:

University College London

Abstract

Department of Computer Science

Computer Graphic, Vision and Imaging

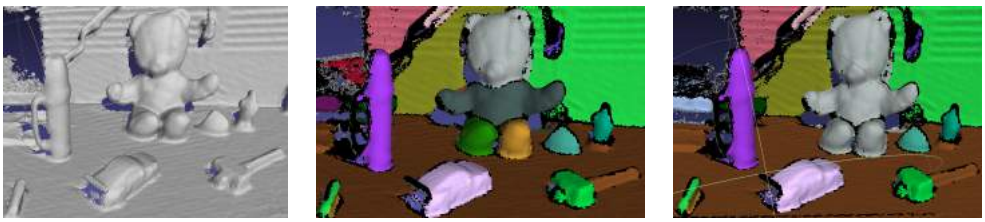
Incremental instance-level 3D segmentation performed by fusing geometric and deep-learning cues

by Maud BUFFIER

The increasing availability of hand-held RGB-D cameras has increased the interest for 3D real-time reconstruction. Accurate SLAM systems reconstructing a static or moving scene have been introduced over the past ten years and are now widely used. To go further however, and continue to improve robots intelligence, reconstruction needs to extend beyond geometry and textures. Segmentation, along with semantic information, is needed to achieve a complete scene understanding. Obtaining this segmentation at object-level is primordial for robotic purposes such as grabbing.

When reading related computer vision literature, two main working approaches can be dissociated : geometry and deep-learning. Geometric systems have been investigated for a longer time and achieve good result to detect object boundaries. However, geometry now tends to be left out for the benefit of deep-learning which computes object-aware segmentation. Whereas most works choose an approach over the other, this thesis presents a way to merge deep-learning and geometrical cues to improve the current state of the art.

This project aims at incrementally achieving an object-aware 3D segmentation by fusing geometrical and deep-learning frame-wise information.



(A) Mesh of the SLAM reconstruction

(B) 3D segmentation using geometry only

(C) 3D segmentation using geometry and deep learning

FIGURE 1: Visual evidence that fusing geometric and deep-learning cues achieved a better object-level segmentation than geometry only

Acknowledgements

I would like to deeply thank Professor Lourdes Agapito for her advices and support she gave me, and for always finding time to meet and discuss the progress of the project.

I would also like to thank Martin Ruiz for the help he provided me during this project, for his availability and for his investment to combine our works.

Finally, I wish to thank my family and friends, for their unconditional love and support

Contents

List of Figures

List of Abbreviations

RGB-D	**R**ed **G**reen **B**lue - **D**epth
GSM	**G**lobal **S**egmented **M**ap
SLAM	**S**imultaneous **L**ocalization **A**nd **M**apping
SIFT	**S**cale **I**nvariant **F**eature **T**ransform
TSDF	**T**runcated **S**igned **D**istance **F**ield
CNN	**C**onvolutional **N**eural **N**etwork
FCN	**F**ully **C**onvolutional **N**etwork
CRF	**C**onditional **R**andom **F**ield
RPN	**R**egion **P**roposal **N**etwork
ICP	**I**terative **C**losest **P**oint
PTAM	**P**arallel **T**racking **A**nd **M**apping
RANSAC	**RAN**dom **SA**mple **C**onsensus

List of Symbols

Sub-index "t" indicates at time t

$\mathcal{L}$	Global segmentation map
$\mathcal{T}_t$	Truncated signed distance field
$\mathcal{D}_t$	Depth frame
$\mathcal{V}_t$	Vertex map
$\mathcal{N}_t$	Normal map
$\mathcal{V}_t^m$	Vertex map for the model
$\mathcal{N}_t^m$	Normal map for the model
$\mathcal{L}_t^G$	Geometrical label map
l_g	Label in $\mathcal{L}_t^G$
$\mathcal{L}_t^S$	Instance level semantic segmentation label map
l_s	Label in $\mathcal{L}_t^S$
$\mathcal{L}_t$	Current label map, fusion of geometric and FCN map
l_j	Label in $\mathcal{L}_t$
$\mathcal{L}_t^m$	Label map of the model
l_i	Label in $\mathcal{L}_t^m$
$\mathcal{L}_t^p$	Propagated label map
l	Label in $\mathcal{L}_t^p$
$\Psi_t^m(l_a, l_b)$	Confidence to merge l_a and l_b in $\mathcal{L}$
$\Psi_t(v)$	Confidence in voxel v label

Chapter 1

Introduction

This project aims at incrementally building an instance-level segmented 3D map from an RGB-D sequence. In other words, at each frame, the 3D segmentation is refined by adding 2D frame-wise segmentation to output a reconstruction map where each object is assigned with a label. My approach novelty comes from the frame-wise segmentation where geometric and deep-learning segmentation are fused.
This fusion allows to achieve better result in the case of 3D segmentation, as most of thesis shows, but can also be used to improve other framework results.

1.1 Reconstruction via SLAM

The segmented 3D map, called "Global Segmentation map" (GSM), is built on the output of a SLAM system where each voxel is augmented with a label. Hence, a deep understanding of SLAM is required. This part will tackle the history of SLAM and put forward the major differences between implementations.

1.1.1 History of SLAM systems

Simultaneous Localization And Mapping (SLAM) is a chicken-or-egg problem trying in the same time to reconstruct an environment map and infer the camera position at each time. As one is supposed to be known to find the other, the SLAM problem is truly challenging.

A survey, [46], gives an overview of non-filter-based monocular Visual SLAM systems. A SLAM system can be filter-based (Kalman filter, particle filter) or non-filter-based. In the first case, information are marginalized and condensed in the last state of the filter (see figure 1.1). This state is composed of the feature points and camera positions. Moreover, each measurement accuracy is defined by a covariance representing the confidence in this measurement. Those filter-based methods become heavy when the number of feature points increases and are subject to scalability and drift issues. To cope with this problem, SLAM researchers started looking at other fields and, in particular, the bundle adjustment problem. Given a set of 3D feature

points seen from different camera positions, this technique optimizes the positions of the points and the camera. The re-projection errors are minimized in all the images using optimization technique such as Levenberg–Marquardt algorithm. However, the bundle adjustment algorithm drawback is its impossibility to run real-time.

A breakthrough idea, coming from Parallel Tracking and Mapping (PTAM [17]), separated the mapping and tracking operations. On one thread, the tracking, mandatory to know the camera position, is performed at each frame using the current points map. On an other thread, the mapping is optimized using bundle adjustment on key frames when it's computationally possible (every 10-15 frames). Separating the threads was a tremendous step forward and led to the development of non-filter-based method. In 2009, [38] proves that non-filter-based systems outperform the filter-based ones. Whereas most methods between 2003 and 2007 were filter-based, non-filter-based have become the norm (see [46]) in the following years.

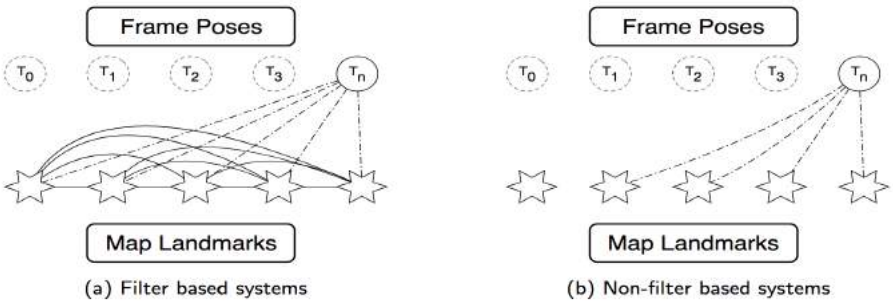
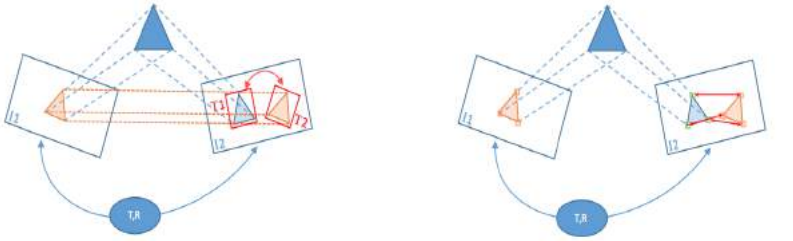


FIGURE 1.1: "Data links in filter versus non-filter based Visual SLAM systems", figure and legend from [46]

An other distinction between SLAM implementations is the choice between direct or indirect methods. In the indirect method, feature points are extracted from frames using descriptors such as SIFT (Scale-Invariant Feature Transform). Matches are found between two consecutive frames. Then, the transformation which best aligns all matching points is computed to estimate the camera position and orientation. Finally, feature point positions are optimized at each mapping thread. Direct methods use the entire frames (RGB or depth) to track the camera position. They find the camera transformation that minimizes the photometric cost between frames. The transformation is computed using the Gauss Newton Algorithm optimization to minimize step by step the photometric error. For this project, an indirect non-filter-based framework, called InfiniTam [30], has been used.



(a) direct methods using all information of the triangle to match to a query image.

(b) indirect methods use the features of the triangle to match to the features of a query image.

FIGURE 1.2: "Data types used by a Visual SLAM system", legend and figure from [46]

1.1.2 Points VS volumetric representation

Moreover, the reconstruction can be based on two representations : voxels or surfels. [16] presents surfels-based representation theory. A surfel is a disk-shaped entity with position, radius and normal information. It doesn't contain any connectivity data and describes a local planar region. On the other hand, the volumetric representation uses voxel (3D analogy of pixel) and a truncated signed distance fields (TSDF) to represent the word. The main drawback of this method is the computational and memory losses. The computational complexity lies in the constant back and forth needed between feature points, to track the camera motion, and volumetric representation, to fuse frames into the model. In contrast, surfel-based methods achieve tracking and fusion using the same surfels representation. Moreover, surfels simplify the rendering step. As point positions are directly stored in the data structure, ray-casting or a zero-crossing estimation aren't required to yield 3D point positions. And, finally, most of the common operations on point clouds such as averaging, data association or removal can be performed more easily using surfels.
However, for this project, a volumetric representation had had to be used. A framework, called InfiniTam [30], computes a dense reconstruction of a scene based on the Kinect fusion algorithm [24]. InfiniTam was used as it was the only framework compiling on my machine (mac OSX) whereas point-based frameworks investigated (such as elastic fusion [41]) did not.

1.2 Frame-wise segmentation

The output reconstruction from SLAM now needs to be augmented with labels. To do so, a 2D segmentation map is computed for each frame and the result propagated in the global segmentation map (GSM). This 2D to 3D idea was presented in [13]. However, [13] implementation only relies on geometric cues to yield the 2D segmentation. Unfortunately, this "geometry-only" implementation tends to oversegment and label complex objects in several parts. Knowing that, my implementation frame-wise segmentation is improved by adding semantic cues obtained from convolutional neural network processing.

Convolutional neural networks (CNN) have been more and more investigated to perform detection, recognition and segmentation over the past years. They have considerably improved the average results for those tasks. At the beginning, CNNs have been used to perform image classification. Afterwards, system as YOLO ([32]) has allowed to detect multiple objects with their bounding boxes in real time, solving the detection problem. To go further, articles have tried to perform classification at a pixel level, digging into pixel-wise segmentation. However, the main drawback of this method lies in its failure to detect objects at instance level. In other words, to recognize several objects of a similar class as distinct. Hence, semantic segmentation at instance level has been investigated, outputting for each object in the image, its boundaries and its class. To improve the geometric segmentation, the instance level semantic segmentation has been used.

1.3 Objectives of my project

This project main objective is improving incremental 3D segmentation state of the art by computing objects-aware 3D maps. This is done by computing object-level 2D frame-wise segmentation which are spread in the GSM. Seeing CNN fast enhancements, the first idea would be replacing 2D geometric segmentation by CNN outputs. However, CNNs systems perform well in many applications (depth estimation, semantic segmentation..) but their accuracy is variable between frames and sequences (lighting conditions, inconsistencies between training and testing data..). In the case of instance-level 2D segmentation, the main issue lies in imprecise detection of object boundaries. Knowing that, replacing completely geometric approaches by deep learning doesn't seem to be the best solution.
To tackle that, my researches were focused on finding a way to merge both 2D segmentations while keeping their respective advantages : precise boundaries using geometry and object wholeness using CNN. This fusion strategy is the key point to

achieve instance-level 3D segmentation.

The following figure highlights the fusion strategy :

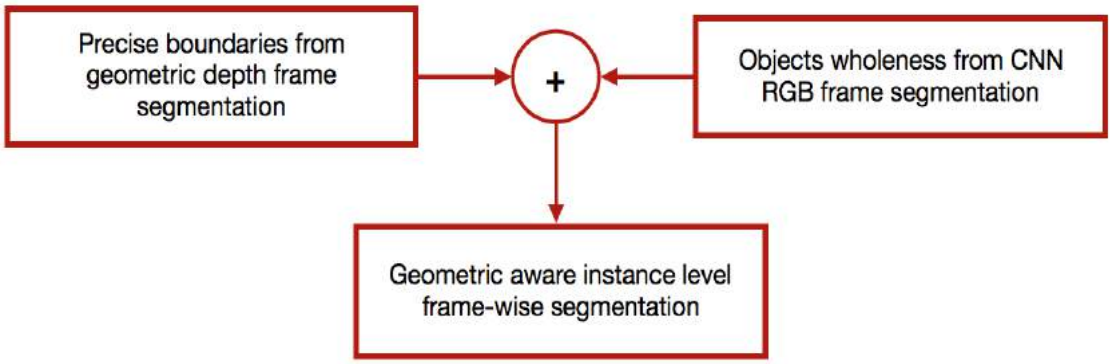


FIGURE 1.3: Objective of the fusion between semantic and geometric frame-wise segmentation

After this highlighting on the objectives, the background and related works are presented in chapter 2. Chapter 4 details the framework over-viewed in chapter 3. Chapter 5 evaluates the results on different sequences increasingly challenging, my own sequence to start and then on the ScanNet dataset [7]. Chapter 6 provides openings to possible improvements and demonstrates that this segmentation strategy can be used to improve other framework results, in particular Co-Fusion [34].

Chapter 2

Background and related works

This project aims at segmenting a 3D scene incrementally created from a RGB-D sequence. Preforming 3D segmentation yields the object occupancy in the world, crucial for moving systems. Whereas 2D segmentation has been widely investigated for decades, 3D segmentation has had to wait for depth data acquiring to become more straightforward. Hardwares like Microsoft Kinect have allowed 3D researches to flourish in the last decades by simplifying the acquisition process. Since then, many works using geometric informations have been realized. In the same time, deep-learning driven frameworks have been created to perform mostly image-based tasks.

However, those fields have been investigated separately and very few works are fusing both geometric and deep-learning cues. This project aims at presenting a solution to cope with this shortfall. First, this section will present the state of the art regarding both axes : geometry driven frameworks to perform 3D point-cloud or depth frame-wise segmentation and deep-learning driven frameworks for instance level semantic segmentation. Presenting those works actually highlights the lack of papers using both geometric and deep learning.

2.1 Geometry driven systems

2.1.1 3D point-cloud segmentation

On one hand, some works have focused on segmenting 3D point clouds obtained from a SLAM system and the Kinect Fusion algorithm. For example, [31] yields a segmentation using the assumption that 3D points belonging the same object should satisfy a smoothness constraint. To start, normals are computed at each vertex by fitting a plane to their neighbors. The neighboring vertices can be obtained either by a K-nearest neighbors algorithm or a Fixed distance neighbor algorithm. In the second case, the number of points used to find the plane equation depends on the point cloud density around the current vertex. Whereas, in the first case, this number is fixed. Then, a region growing algorithm is applied in order to segment objects.

The region growing algorithm is based on the smoothness constraint stating that the angle between normals of two points belonging to the same object should be small. Moreover, some parameters can be chosen by the user to avoid over or under segmentation. The smoothness constrain used by [31] is closely related to my framework, also partly based on normal analysis.

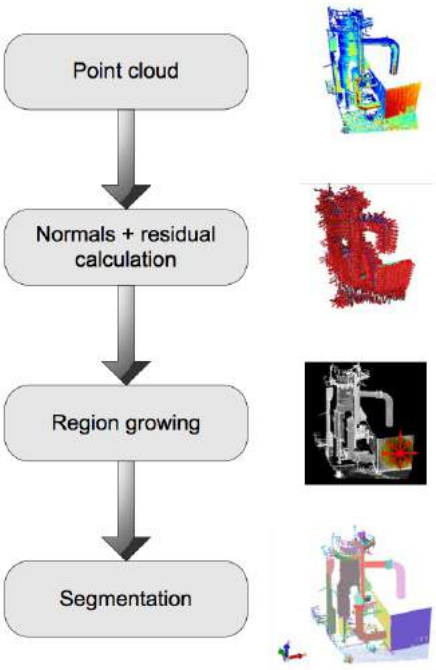


FIGURE 2.1: Flowchart of [31] segmentation algorithm

Even if my implementation also uses a smoothness constraint to segment objects, it performs on a single frame. Hence, it is computationally much faster than on the complete mesh.

[15] also tackles 3D segmentation and tries to discover objects in a point cloud. The cloud is treated as a graph to apply a graph-based segmentation approach from [10]. This step is repeated several times with different parameters in order to obtain several over-segmented results. Those results produce object candidates for which an "objectness" measurement is computed - a probability for each segment to be an object. This "objectness" is measured using shape information such as compactness, local and global convexity, symmetry, and smoothness. If a segment scores well regarding those 5 criterion, it is classified as an object (see figure 2.1).

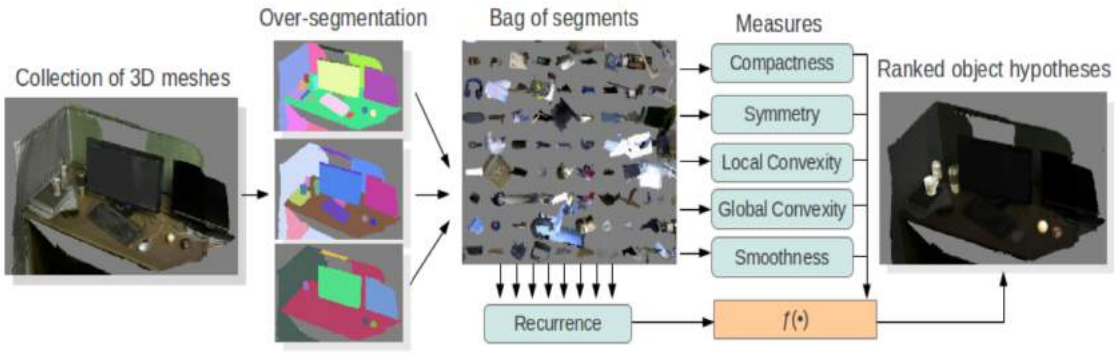


FIGURE 2.2: Every 3D input mesh is over-segmented into a large collection of segments. Each segment is ranked using an objectness measures and the final ranking is computed (image from [15])

This graph based approach is close to the one used by ScanNet, the data set evaluating my algorithm (see Chapter 6). Moreover, the pipeline is also related to mine, where a geometric over-segmentation is improved using learning cues.

[20] trains a classifier to recognize indoor environments. A "search and classified" approach is applied on the entire scene. Segmentation and classification are performed together by applying a region growing algorithm based on the output probability of the classifier. Hence, they took advantage of the 3D geometrical reconstruction and recognition to segment objects.

Several methods segmenting 3D point cloud have been seen. However, the main drawback of those methods is the computational cost which disqualifies them for real-time robotic uses.

2.1.2 Depth frame-wise segmentation

On the other hand, works enable segmenting depth 2D images in real-time. Papers, like [1] and [43], use normals computed at each vertex to infer the object edges. Edges are assumed to be found in two situations : when there is a jump in depth values or when normal directions change suddenly. Then, from this depth edge map, a high level object segmentation is performed. To obtain object segments, [1] removes the support plane using RANSAC and group together surfaces patches with high similarity. [43] uses graph-based models to group patches in order to tackle more

congested scene.

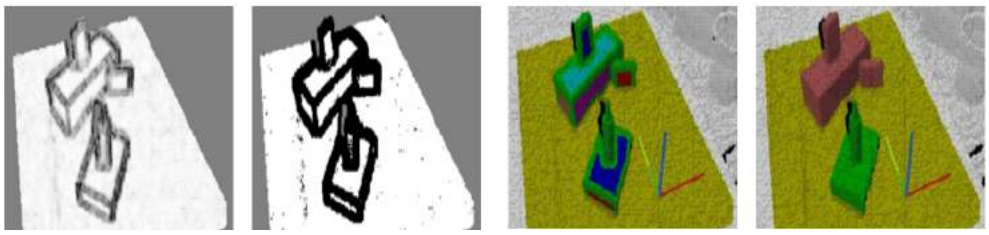


FIGURE 2.3: **Left :** Result of the surface normal detection (angular values and binarized version). **Right :** the first segmentation into surface patches and segmentation into object blobs (figure from [43])

My framework relies on a similar approach. However, [1] and [43] main drawback lies in the fact that 2 steps are needed to detect an object : normal segmentation and grouping together similar faces of the object. To cope with this issue, my implementation assumes that most objects in the world are convex, and highlights only concave regions.

[26] detects objects in the scene using an RGB-D image. A very complete pipeline is described. First, an over-segmentation of the scene with regions consistent in color is inferred. Then, regions with similarity in normals are grouped together to obtain 3D facets. 3D shapes are then computed from those 3D facets using curvature and the main assumption that 3D shapes result of a fusion of convex 3D facets. Finally, the objectness of each 3D shapes is measured using context information. A tree of support is computed for the scene, knowing that each object must obey the law of physique. At the end, 3D shapes which scored a high objectness are labeled as objects.

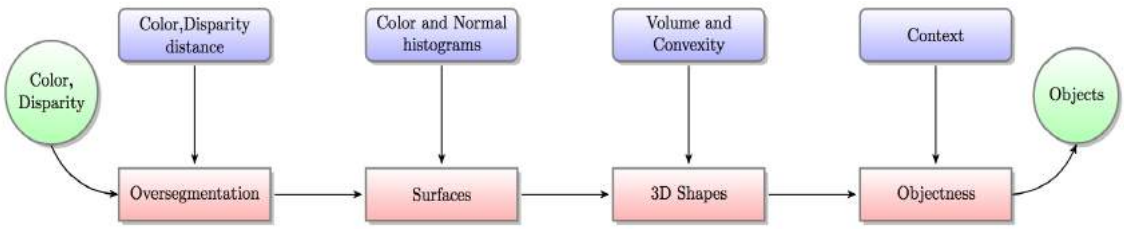


FIGURE 2.4: [26] pipeline to segment objects

This article achieves good results but is computationally heavy. It's worth noticing that the object convexity assumption is also made. However, my implementation simplifies the process using instance level semantic segmentation to unify oversegmented segments belonging to the same object.

2.1.3 From 2D to 3D segmentation

Algorithms reviewed segment point clouds but their computational burdens are too heavy for real-time applications. Moreover, this drawback grows when the point cloud size increases. On the other hand, 3D segmentation of depth map can be computed in real-time. It explains [13] main idea : incrementally building a 3D segmentation using 2D frames segmentation results. Each input image is segmented and propagated into the global segmentation map. Papers have already explored similar ideas but without using a CNN based segmentation.

The authors of [11] came out with an equivalent idea and proposed a framework to obtain an incremental segmentation based on Kintineous [45], itself built on KinectFusion. The outputs of Kintineous, cloud slices, are segmented using a graph-based method already seen ([10]). Then, slices are merged into a segmentation map. However, even if this method is a huge improvement in term of computational efficiency, at the end, the same problem remains. As [10] segmentation is needed for each output slice and because the entire slice needs to be merged into the map, this algorithm doesn't run real-time. Moreover, similarily to all 3D segmentation algorithms seen, the computational need increases with the size of the reconstruction.

[35] computes both a 3D reconstruction of the world using SLAM and a global planes segmentation map. Detected planes improve the camera pose estimation inferred from an ICP algorithm [29] in SLAM. Planar regions are segmented by processing live depth measurements and merged into the global segmentation map. This method only outputs planar surfaces and labels them with a connected components algorithm. Regions with high curvature are ignored. Hence, the approach extension is limited for objects segmentation.

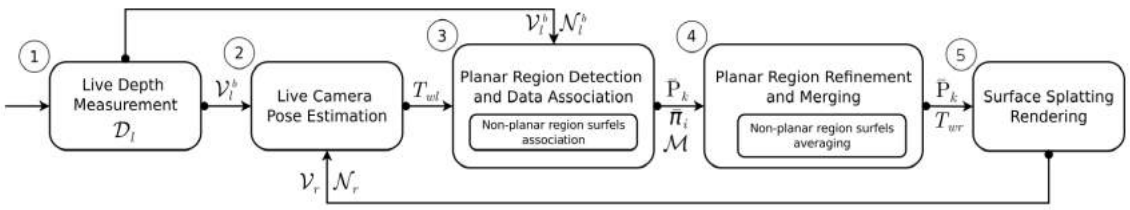


FIGURE 2.5: Pipeline of [35] to segment planar areas in real time (figure from [35])

The pipeline looks similar to mine but is not as general. It's worth noticing that this article uses a surface based representation of the world from [16], rather than a volumetric representation from the Kinect fusion algorithm.

[27] proposes a object recognition system based on monocular SLAM (Orb-SLAM). From the RGB stream, SIFT features detects 2D objects. 3D object candidates are computed taking advantage of multi-views information. However, this method outputs a sparse reconstruction of the word, hence, it's less relevant for robotic applications.

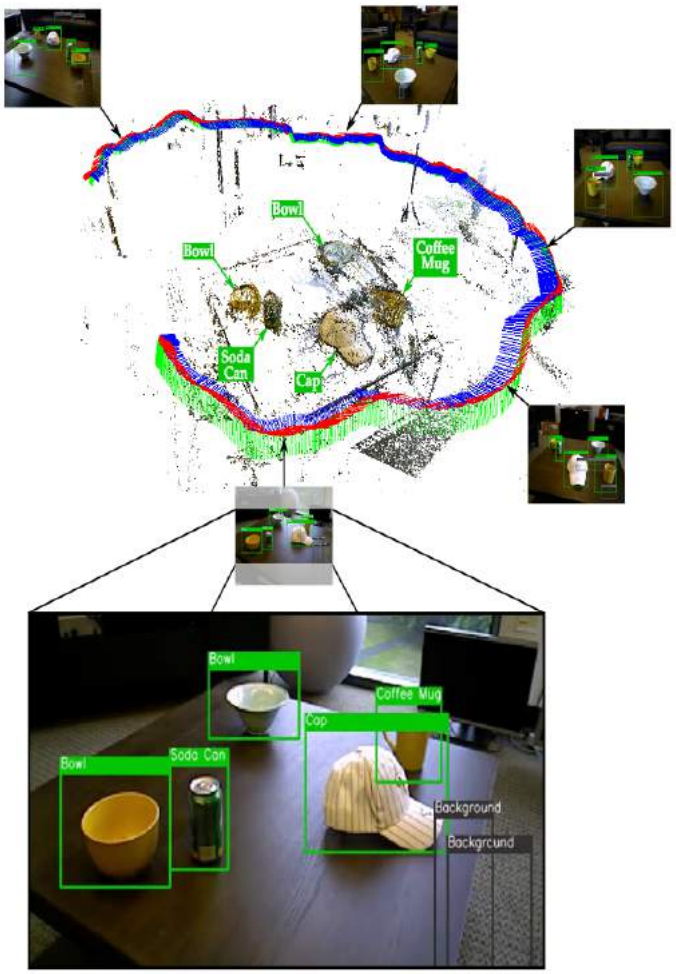


FIGURE 2.6: [27] overview to segment objects in a point cloud using frame-wise detection (figure from [27])

[14] relies on the same 2D to 3D principle. It uses a learning approach to recognize objects in a point cloud incrementally computed via SLAM. The framework combines RGB-D and voxel based information to label each 3D points using a "hierarchical sparse coding technique" to learn features. Once again, scores are computed in 2D and the result propagated in 3D.

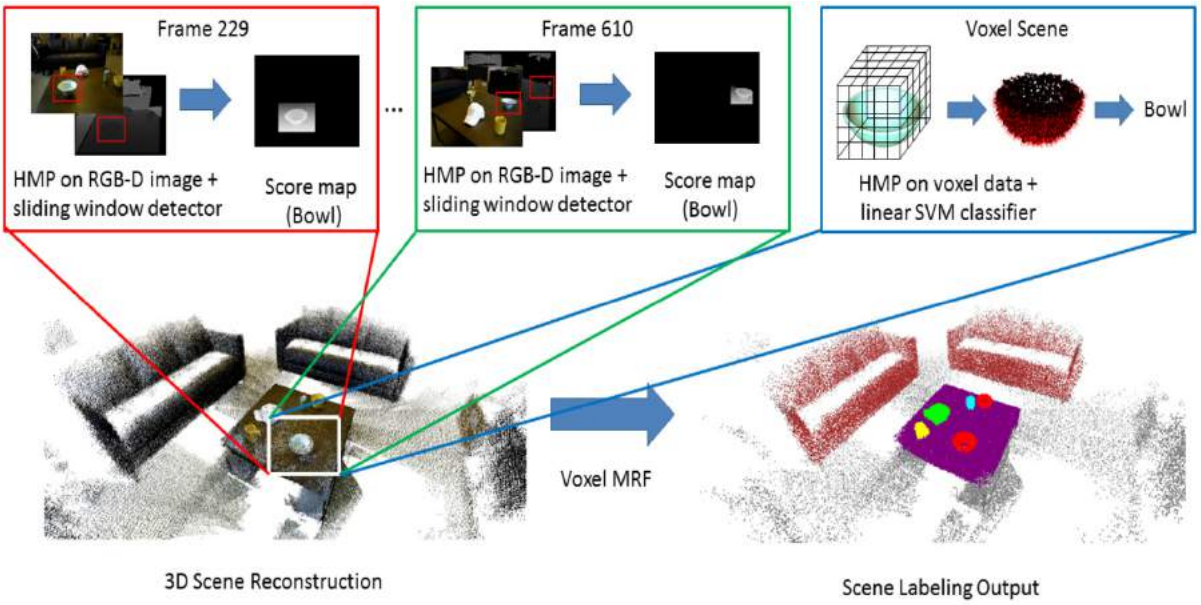


FIGURE 2.7: Overview of [14] to detect objects (figure from [14])

However, this approach is computationally way too expensive to work real-time.

It must be noted that, performing 2D recognition to segment a 3D point-cloud comes with a huge drawback : only objects recognized in 2D can be segmented. Whereas, in my implementation, recognition confirms and improves the segmentation which is provided by the frame-wise geometric segmentation.

2.2 Deep-learning driven systems

2.2.1 Instance level semantic segmentation using FCN

Fully Convolutional Networks (FCN, [36]) arrival has lead to tremendous improvements in the semantic segmentation task accuracy. A FCN differs from the CNN by its last layers. On one hand, a CNN takes an image as input and applies several fully connected layers at the end to output a vector. Hence, CNN can perform image classification but also more local tasks such as bounding box objects detection and classification. On the other hand, a Fully Convolutional Networks applies convolutional layers along with deconvolution layers at the end to output an image with the same size as the input. [36] showed that all deep architectures which have been

widely investigated (AlexNet [18], GoogLeNet [39] and the VGG net [37]) could be turned into Fully Convolutional Networks.

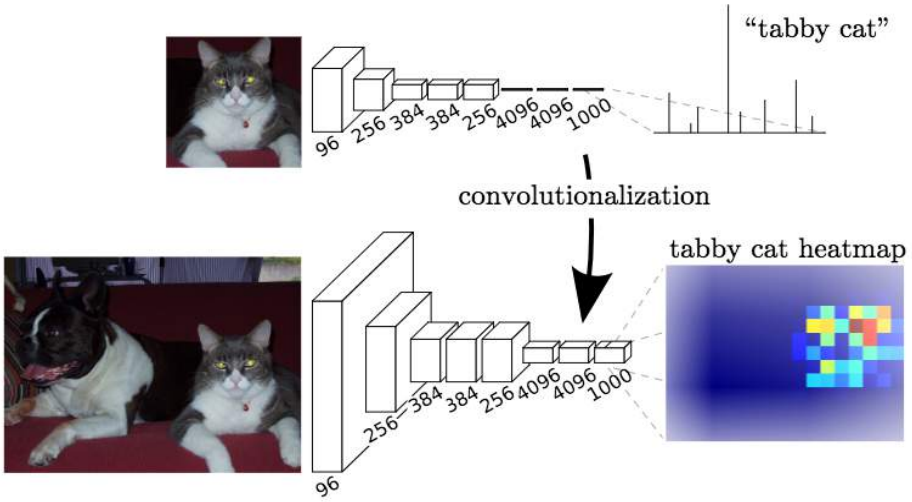


FIGURE 2.8: Transforming fully connected layers into convolution layers enables a classification net to output a heatmap (image from [36])

A FCN is used to perform dense per-pixel classification and semantic segmentation. To avoid losing information during the deconvolution process, [36] merges predictions made at different levels. Results were tested on the PASCAL VOC dataset and achieved state-of-the-art results much quicker. Several ameliorations followed, [5] is using a fully connected conditional random field (CRF) to refine the result and cope with the up-sampling losses. [8] previously computes bounding boxes to guide the segmentation process. However, the main drawback of those semantic segmentation methods are their failures to identify object instances. Indeed, those system can output all car pixels for example but are unable to distinguish one car from another. Hence, instance-aware semantic segmentation is an even more challenging problem. The COCO dataset [21] was created to encourage researches in this area and is used in papers introduced next.

The three main methods to perform instance level semantic segmentation have been investigated. The first one computes object boundaries using a first network and a second one to classify objects. For example, SharpMask ([28]) computes object proposals using a bottom-up/top-down architecture where result at each scale is refined using previous scale result. From those proposals, [47] classifies objects using a multipath architecture to cope with the scale variances in the COCO dataset. This

method will be detailed in section 4.

FIGURE 2.9: SharpMask outputs, which will be used as input in the multipath architecture (image from [28])

The second method, [9], uses a multi-task cascade network architecture. The problem is divided in 3 sub-problems : bounding boxes detection, masks estimation and classification. At each stage, the result of the previous one and the original image are considered as inputs, explaining the "cascade" term. The loss function is defined as a sum of the 3 sub-problems losses (conditional losses given the previous stages).

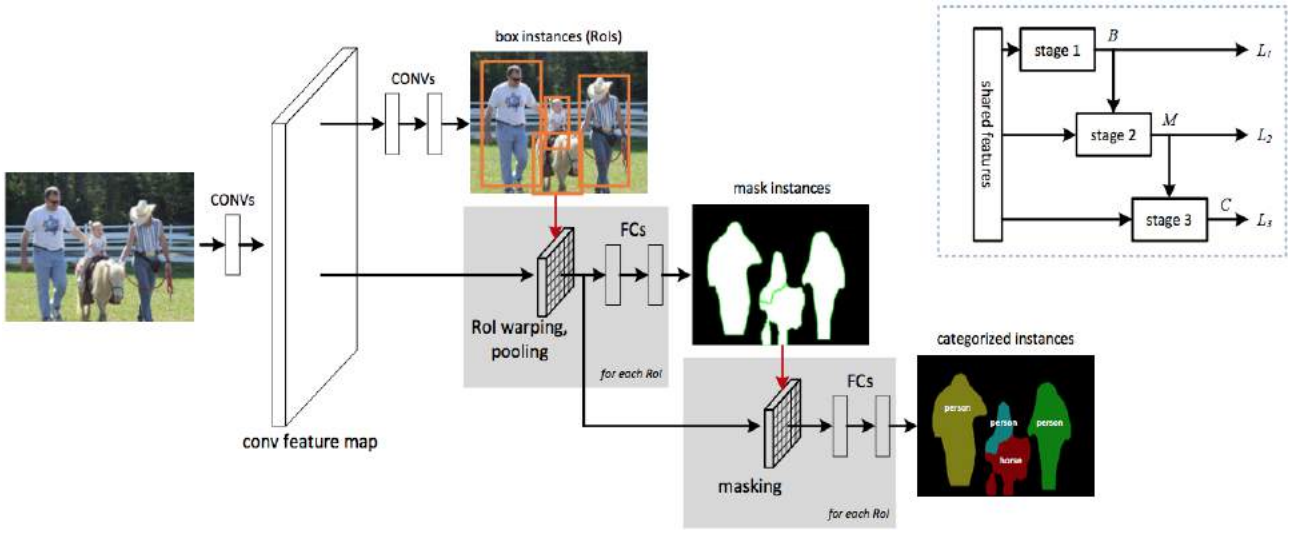


FIGURE 2.10: Multi-task Network Cascades for instance-aware semantic segmentation (image from [9])

The last method performing instance-aware semantic segmentation is called Mask R-CNN [12] and outperforms the last two by far. This method augments the Faster R-CNN [33] network which already scored well in the object detection task. Faster R-CNN implementation uses a single, unified network for object detection, whereas most methods need two : one to detect region proposals and another to classify them. In Faster R-CNN, features for both stages are shared, decreasing inference

time. Faster R-CNN output consists of objects bounding box and class. The idea behind Mask R-CNN [12] is to extend Faster R-CNN by adding a new branch computing masks in parallel. Hence, the loss function is the sum of 3 losses (class, bounding box and mask). Moreover, a new stage called RoiAlign has been added to improve the mask precision and cope with approximations during the up-sampling stage. [12] shows that considering the problems in parallel considerably improve the segmentation results.

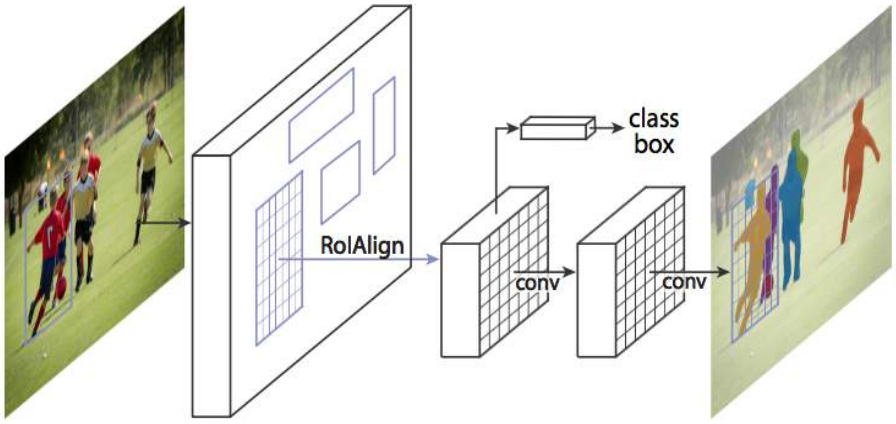


FIGURE 2.11: The Mask R-CNN framework for instance segmentation [12] (image from [12])

2.2.2 SLAM systems improved by deep-learning approach

In this section, two papers integrating CNN results into SLAM frameworks are presented. CNN-SLAM [40] combines a monocular semi-dense direct SLAM system and CNN to learn depth and semantic information. From a RGB sequence, a CNN is used at key-frames to predict semantics and dense depth maps. Those maps improve the result of a monocular SLAM system and yield a semantic label map. Moreover, it copes with one of the biggest problem of monocular SLAM systems : scale. Indeed, when reconstructing the world, a monocular SLAM system has no idea of scale. Using a CNN-learned depth allows an RGB system to reconstruct the world scale-wised.

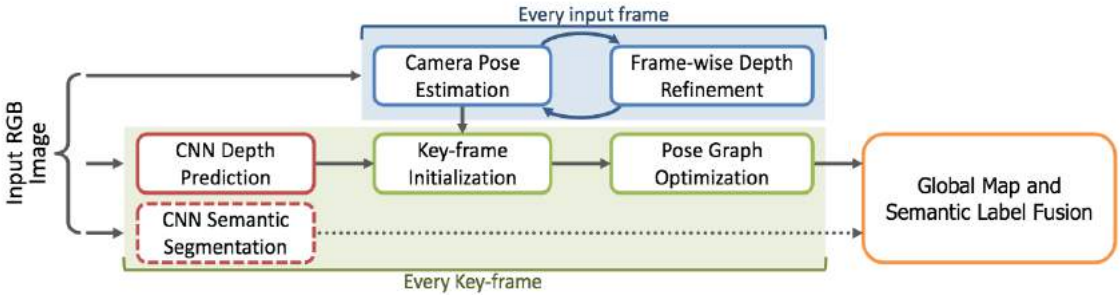


FIGURE 2.12: [40] CNN-SLAM overview (image from [40])

However, the semantic segmentation output is not used much in this framework. The article focuses more on the depth prediction part.

The SemanticFusion framework [22] is also deeply related to my approach. From a point-based SLAM system (Elastic Fusion [41]), surfels are augmented with a classes probabilities. For some key-frames, a CNN infers pixels classes used to update surfels probabilities by projection. The 3D map obtained is regularized using a Bayesian update scheme.

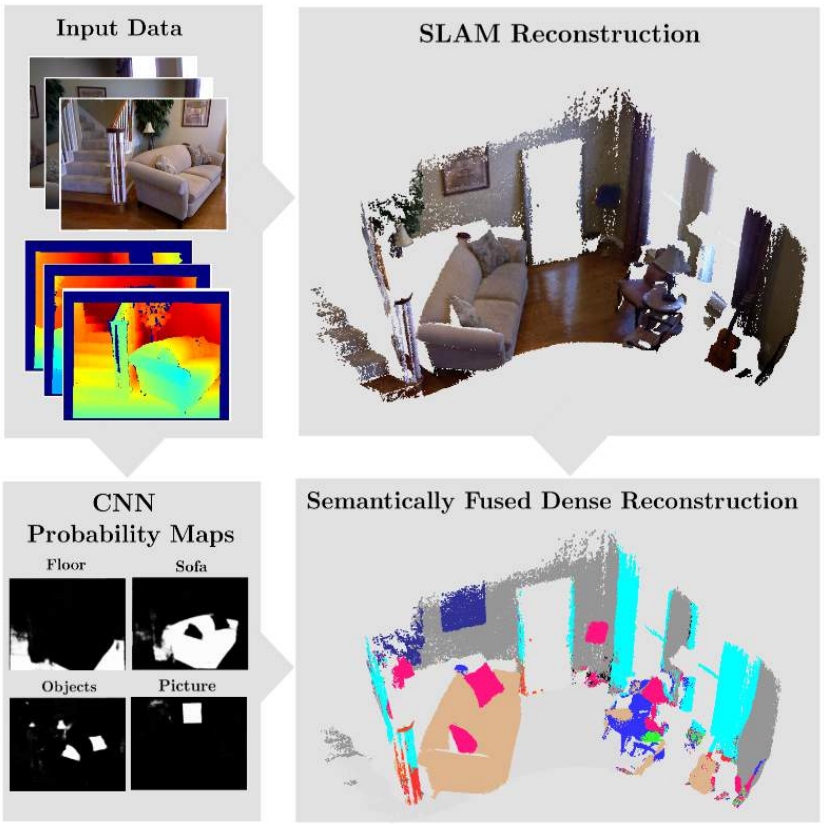


FIGURE 2.13: SemanticFusion [22] pipeline overview (image from [22])

However, as for all CNN-only based approach, the system can only detect what is recognized by the network. Whereas in my implementation, all objects are segmented and the CNN adds semantic information and refines the geometric result. Nevertheless, this article demonstrates the efficiency of the point-based method to perform 3D map refinement.

Chapter 3

System overview

This literature overview highlights the lack of papers using both geometry and deep-learning cues. The fast improvement of deep-learning approaches have pushed researches in this direction. However, geometry-based system have been investigated much longer and have achieved convincing results. Hence, this project aims at solving geometric failures using deep-learning rather than replacing one by the other. The system pipeline to achieve the 3D segmentation can be represented this way :

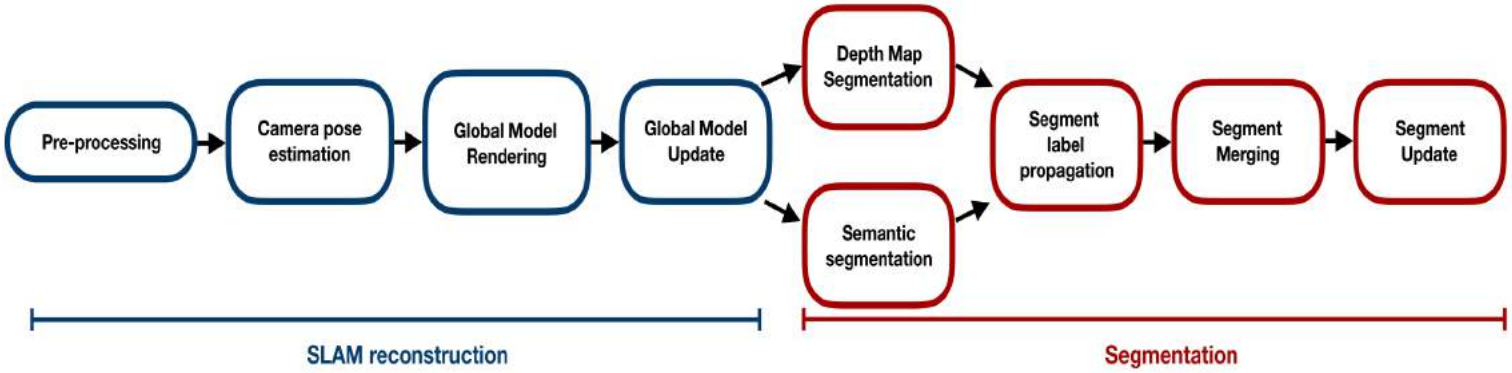


FIGURE 3.1: Pipeline of my approach performed at each new RGB-D frame

This pipeline exploits the main idea of [13] : segment 2D depth frames to propagate the result into a global 3D segmentation model. My implementation is based on the same frame-wise principle but the 2D segmentation is improved by deep-learning results to achieve an object-aware segmentation.

At each new frame, the scene reconstruction is updated using a dense SLAM framework (blue part, section 4.1). Then, geometric and semantic segmentations are performed using the current 2D RGB-D frames. Those results are then fused to be added to the 3D global segmentation map (red part, section 4.2).

Chapter 4

Incremental instance level framework

4.1 Reconstruction

Infinitam [30] defines a volumetric representation as a data structure using a truncated signed distance function (TSDF) and voxels to represent surfaces. Voxels storage has an impact on the overall performance of the system. In this section, reconstruction scene process is explained and can be visualized with this pipeline :

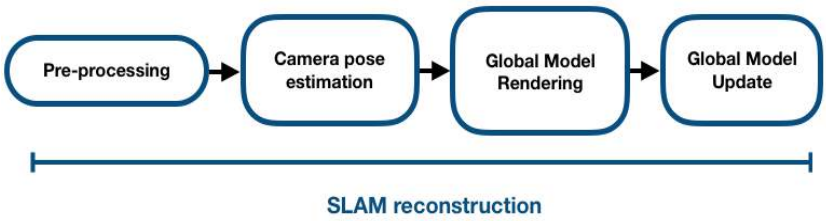


FIGURE 4.1: Pipeline for the reconstruction step

Each step is explained along with the famous Kinect fusion algorithm [24] as Infinitam is mostly built on it.

4.1.1 Calibration

InfiniTam needs intrinsics and extrinsic parameters of the device used to capture the sequence in order to run. When using on line dataset sequences, parameters are generally given directly. When capturing sequences, [4] code provides the calibration file needed by InfiniTAM. It outputs the intrinsics for RGB and depth cameras but also the rotation and translation between the two if images aren't already aligned. The scale factor between real values in meters and depth map values are also given.

To run [4], 30 captures of a chess pattern from various view points are needed. For each view point a right RGB image, and a left IR image are obtained. The infrared projector needs to be covered with glue tape to reduce noise on the IR image (it explains why the left image is darker in figure 4.2) However, to find the pattern the mean intensity is subtracted from images so darkness doesn't influence pattern detection.

Images are used to compute intrinsics for each camera along with extrinsic between the two. Patterns are detected in both images using a corner detection algorithm. Camera positions along with intrinsics can be deduced as the exact pattern size is known (each square is 38mm by 38mm). Using every pairs leads to a least square estimation problem where corners re-projection errors are minimized across all 60 frames.

Here is an illustration of computation parameters set up :

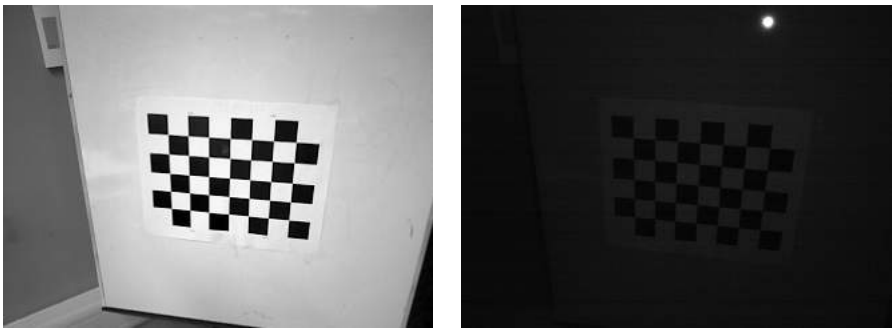


FIGURE 4.2: Right and left images from a view point to compute camera calibration parameters

4.1.2 Pre-processing

Next, the vertex and normal maps are computed from the current depth frame. First of all, a noisy vertex map $\tilde{\mathcal{V}}_t$ is obtained from depth map $\mathcal{D}_t$ using :

$$\tilde{\mathcal{V}}_t(u) = \mathbf{K}^{-1}\dot{\mathbf{u}}\mathcal{D}_t(u)$$

where $\mathbf{K}$ is the intrinsic matrix, $\dot{\mathbf{u}}$ the homogeneous coordinate of a pixel in image frame and $\mathcal{D}_t(u)$ the depth value at this position. Hence, the vertex map is computed by projecting pixel coordinates in 3D using the depth.

As the vertex map $\tilde{\mathcal{V}}_t$ is generally noisy, a bilateral filter [2] is applied. This filter performs an edge-aware smoothing using several Gaussians functions. Instead of blurring across the entire image, only edge-free regions are blurred. Open sourced code from [25] is used in my implementation (see figure 4.3).

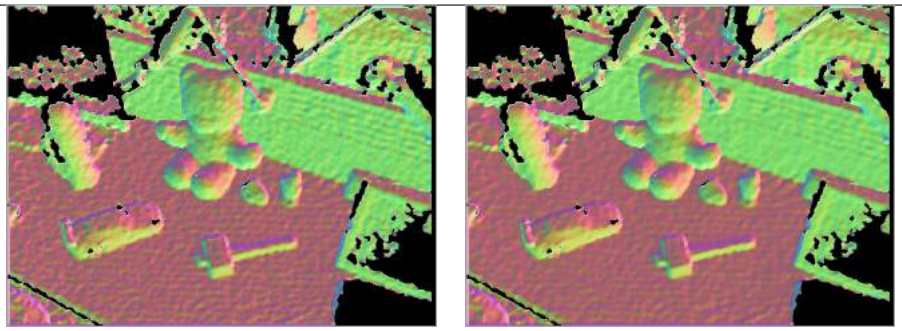


FIGURE 4.3: Result of the bilateral filter on the normal map

However, in the case of the vertex map, convex boundaries between objects are not considered as edges. The change of object isn't detected by the filter as vertices are continuous. Hence, the filter must not over-smooth to prevent those boundaries from disappearing completely. After filtering, a noise-free vertex map, $\mathcal{V}_t$ is obtained.

Next, the normal map $\mathcal{N}_t$ is derived from $\mathcal{V}_t$ using central differences. For a given pixel, two vectors are computed using neighboring 3D points corresponding to the 4-connectivity pixels in the image. Those two vectors approximate the tangent plane at this vertex and their cross-product defines the normal stored in $\mathcal{N}_t$.

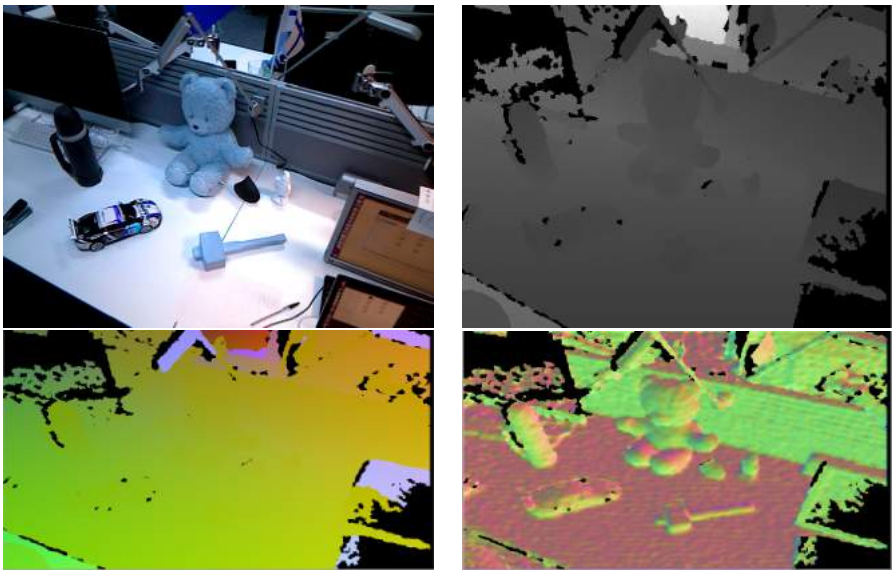


FIGURE 4.4: Outputs from the depth map processing (from left to right), **Top :** RGB and depth input frames **Bottom :** Vertex and normal map computed

The vertex is hard to interpret as it's a 3D field in 2D but the normal map gives a better visualization showing directions.

4.1.3 Camera pose estimation

To estimate the camera position, the current vertex map $\mathcal{V}_t$ is aligned with $\mathcal{V}_{t-1}^m$ obtained at the previous step. Indeed, the transformation between the two describes exactly the camera motion. This motion $\mathbf{T}_t = [\mathbf{R}_t \mathbf{t}_t] \in SE(3)$, composed of a rotation $\mathbf{R}_t \in SO(3)$ and a translation $\mathbf{t}_t \in R^3$, is derived using the ICP algorithm. This algorithm iteratively aligns two point clouds using points correspondences. Using the SVD decomposition of the point clouds covariance matrix, a rotation can be deduced. From it, the translation aligning the meshes is found.
The following figure describes the algorithm in 2D :

FIGURE 4.5: 2D ICP illustration from [23]

This alignment step is performed iteratively until the sum of square differences between the point positions reaches a threshold.

However, the ICP algorithm used in Kinect Fusion [24] isn't the original implementation described in the ICP paper[29] but a more advanced version running real-time. In the Kinect Fusion version, a small motion assumption is made : two points are matched if their projection in image frame falls on the same pixel. Hence, parallelization on GPU achieves real-time results. Moreover, Kinect Fusion ICP uses point-to-plane metric rather than point-to-point. Meaning that the error isn't computed as the distance between 2 points but, as the distance between a point and the tangent plane of its corresponding point. ICP has been shown to converge faster using this metric. Hence, this ICP implementation outputs the camera motion in real-time for further use in the SLAM framework.

4.1.4 Global Model Update

Next, the current vertex map $\mathcal{V}_t$ needs to be fused into $\mathcal{T}_{t-1}$, the previous step global model. This model is represented as a truncated signed distance field (TSDF), where each voxel is associated with a value expressing its distance to the estimated surface. If a voxel is located on the surface, its TSDF value is 0. The higher the value, the farthest from the surface the voxel is (negative inside, positive outside). This representation is called 'truncated' because only values in a fixed band are stored. Moreover, a weight is assigned to each voxel to measure its certainty, the more detected the voxel, the higher the weight.

The following figure shows a 2D grid representation of a TSDF :

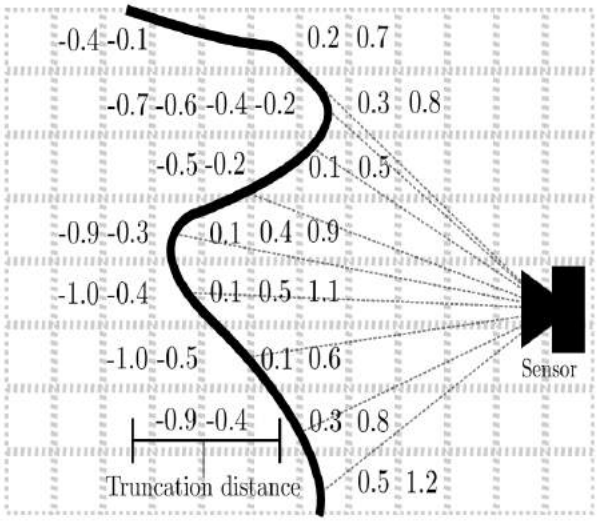


FIGURE 4.6: 2D example of a TSDF representing an implicit surface from [42]

Volumetric representation can become computationally heavy as the size of the reconstruction increases. To tackle that, InfiniTam, [30] uses a voxel block hashing function describes in [19]. This architecture alleviates the volumetric representation burden by simplifying the back and forth between voxels for fusion and 3D points for tracking.

To perform the fusion, $\mathcal{T}_{t-1}$ is projected onto the camera frame using the position computed at the previous step. After this projection, correspondences are computed between $\mathcal{T}_{t-1}(p)$ and $\mathcal{V}_t(u)$ (with p a 3D point at the voxel center). The distance between 2 corresponding points, $\overline{d}(\mathcal{T}_{t-1}(p), \mathcal{V}_t(u)$, is calculated and used to find the new TSDF value, $\mathcal{T}_t(p)$:

$$\mathcal{T}_t(p) = \frac{w_{t-1}(p)\mathcal{T}_{t-1}(p) + w_t(p)\overline{d}(\mathcal{T}_{t-1}(p), \mathcal{V}_t(u))}{w_{t-1}(p) + w_t(p)}$$

with $w_i(p)$ the confidence weight of this voxel p at time i. This weight is updated using :

$$w_t = \begin{cases} w_{t-1} + 1 \text{ if } w_{t-1} \leq w_{max} \\ w_{t-1} \text{ otherwise} \end{cases}$$

with w_{max} is the maximum confidence. This weight gives a higher importance to voxels observed several times.

4.1.5 Global Model Rendering

Finally, this step aims at outputting model vertex and a normal maps ($\mathcal{V}_t^m$ and $\mathcal{N}_t^m$) at time t. Those maps are rendered using ray casting from the camera position. Rays are traveled to find zero-crossing intersections with the TSDF - the surface (this expensive operation is avoided in a surfel-based representation as each surfel directly stores its position and normal). The resulting vertex map $\mathcal{V}_t^m$ will be used to estimate the camera position at time t+1 using ICP but also during the segments propagation step.

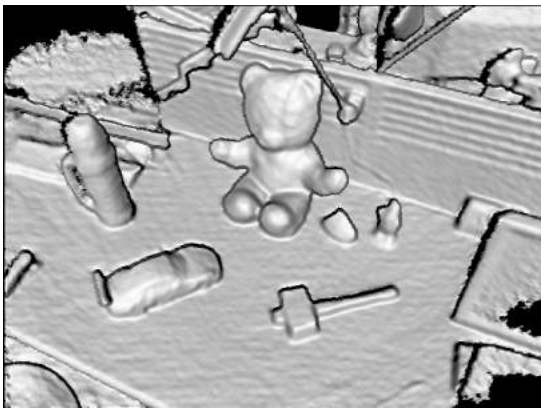


FIGURE 4.7: Output of the reconstruction stage

This section highlighted the reconstruction pipeline based on the Kinect algorithm using a volumetric representation. The following step aims at building a global segmentation map above the reconstruction.

4.2 Creation of a global segmentation map

To build the Global Segmentation Map (GSM), a frame-wise segmentation is computed by fusing geometric and semantic information. As presented in the background section, accurately segmenting a depth map can be done real-time. The instance-level semantic segmentation can also be obtained at a fast frame rate. The framework takes advantage of that and incrementally builds the GSM above the SLAM reconstruction by propagating frame-wise object labels.

The following figure illustrates the pipeline :

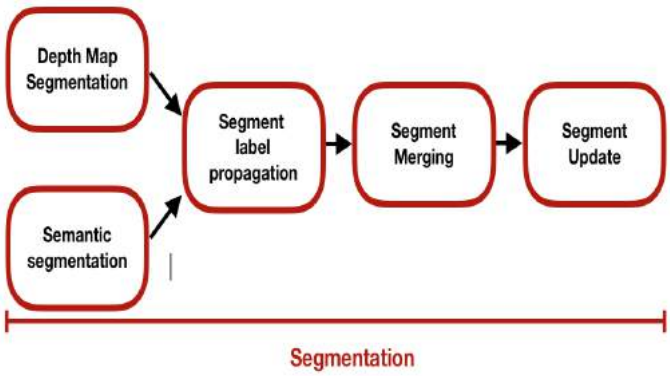


FIGURE 4.8: The segmentation pipeline

4.2.1 Frame-wise segmentation

The frame-wise segmentation results in fusion of geometric and semantic segmentation. After explaining both in details, the fusion will be presented.

4.2.1.1 Depth map segmentation

The depth map $\mathcal{D}_t$ at time t and the current GSM, $\mathcal{L}$, are assumed as inputs. The depth map segmentation relies on two "edge" analysis : a concavity-aware and a geometrical-aware segmentation. In both cases, the vertex and normal maps are needed. Hence, the 2D depth map is augmented by the depth value in order to obtain a 3D vertices field. It's done using the same formula as in the reconstruction step :

$$\tilde{\mathcal{V}}_t(u) = \mathbf{K}^{-1}\dot{\mathbf{u}}\mathcal{D}_t(u)$$

Similarly, a bilateral filter is applied to yield a noise-free vertex map. From it, a normal map $\mathcal{N}_t$ is created using the dot product method. On one hand, the concavity-aware edge map is computed using the assumption that objects in the world are convex. On the other hand, geometrical-aware edge map relies on geometrical cues by analyzing the distance between two 3D points obtained from neighboring pixel in $\mathcal{D}_t$. The final edge map is computed by union.

Concavity-aware normal edge map This part derives from an assumption made by [11] and [15] during their shape analysis stages : most objects have a convex shape. Considering that, a penalty is given at concave regions assuming they are not parts of an object. Therefore, they define an edge.
Each pixel **u** in the image is associated with its normal **n(u)**. Then, concave region are detected using neighboring pixels of u_i, numbered by i, with their associated

normals $n(u_i)$. A new operator, $\Phi_i(u)$, is defined to highlight concave direction with a low value :

$$\Phi_i(u) = \begin{cases} 1 \text{ if } (v(u_i) - v(u)).n(u) < 0 \\ n(u).n(u_i) \text{ otherwise} \end{cases}$$

with $v(u_i)$ and $v(u)$ 3D vertices in $\mathcal{V}_t$ at u and u_i. In other words, this operation can be explained as follow : if normals **n(u)** and **n(u_i)** lie on a convex region, the operator is set to its maximum value : 1. If not, it is set to the dot product between normals. The lower it is, the more concave region.
Then, a single value is computed for each vertex by applying a new operator, $\Phi(u)$ defines as follow :

$$\Phi(u) = \min_{i \in [1,8]} \Phi_i(u)$$

This way, if one of the neighboring pixel lies on a concave region, the operator takes a low value highlighting an edge. The operator is thresholded to obtain the concavity-aware binary edge map. After experimenting, the threshold was set to 0.991. The threshold produces an over-segmented result from which the algorithm can recover during next stages.

Geometrical-aware edge map The geometrical-aware edge map relies on the analyzing the distance between vertices in the scene. If 2 neighboring pixels in $\mathcal{D}_t$ give vertices far from each other in 3D, chances are high that they do not belong to the same object. In other words, they lie on an edge.
Once again, the 3D positions of the 8 neighboring pixels are analyzed. If at least one of them is far from the current 3D point, this vertex is considered as lying on a edge. It's defined using a new operator, $\Gamma(u)$:

$$\Gamma(u) = \max_{i \in [1,8]} \{|(v(u_i) - v(u)).n(u)|\}$$

However, thresholding this operator isn't trivial as the noise from the Kinect sensor is known to increase with the depth value. To address this problem, [3] defines an adaptive threshold, $\sigma_d(u)$. $\sigma_d(u)$ is a noise model taking into account the depth measurements. It integrates as variable the distance and the angle from which the point is seen to give a per-pixel threshold.
Finally the edge maps are combined by union to yield a final binary edge map. To improve the result, 2 closing operations are applied at this stage to remove black holes and make sure boundaries are closed.

Labeling the edge map A connected component algorithm is used to label the map. It extracts regions which are not separated by boundaries and gives them a similar label. Depth or breadth first search could be used. My implementation uses depth first search as it requires less memory.

Hence, it yields to this result :

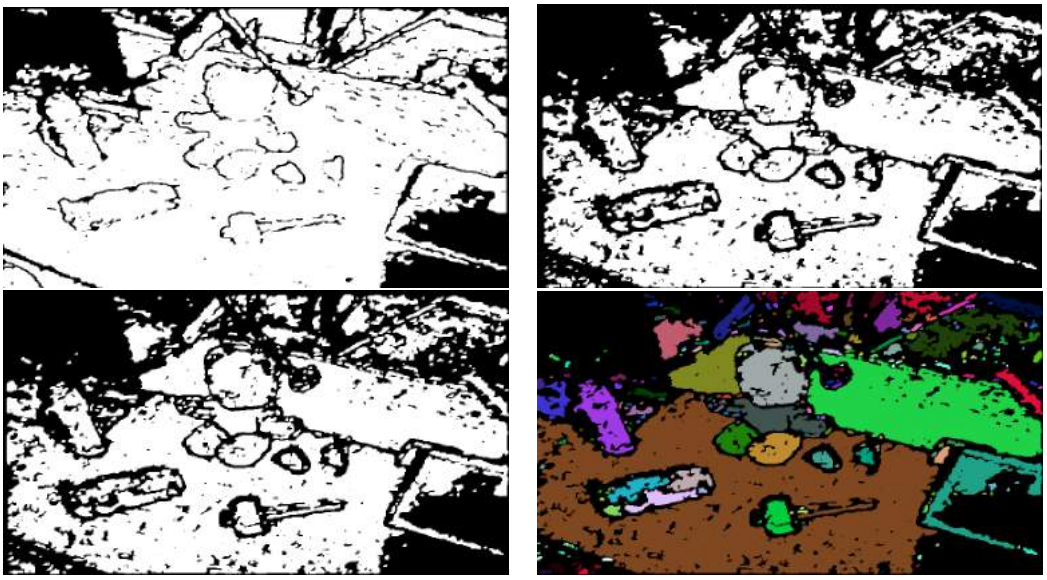


FIGURE 4.9: Result of the depth-frame geometrical segmentation, **Top** : Geometric-aware (left) and concavity-aware (right) segmentation, **Bottom** : Final segmentation (left) and geometric label map $\mathcal{L}_t^G$ (right)

4.2.1.2 Instance level semantic segmentation

My framework uses the two steps approach to perform instance level semantic segmentation. A first network, based on [28], uses bottom-up/top-down architecture to generate object masks. Next, multipath [47] network, processes those masks using a multipath network to output a object classes from the COCO dataset [21].

Sharpmask Sharpmask aims at providing a mask per object. To do so, most paper use a feedforward network or a 'skip' architectures (see figure 4.10). In the first case, upper-layers of CNN are used to output a coarse mask. In the second case, independent predictions are made per layers and averaged together. For object instance segmentation those approaches aren't well suited. Either it only outputs a coarse mask or it does not integrate enough upper and lower layer outputs. To cope with that, SharpMask [28] proposes to augment feedforward nets with a novel top-down refinement approach.
It results in a bottom-up/top-down architecture presented in figure 4.10.

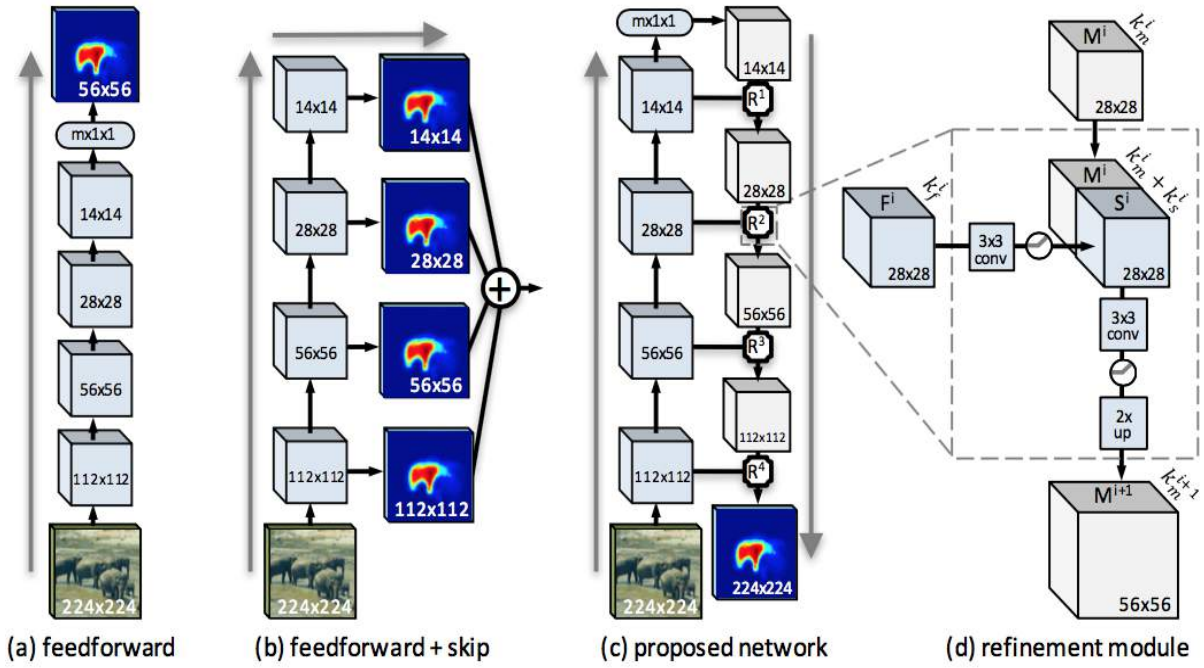


FIGURE 4.10: Bottom-up/top-down architecture from SharpMask (image from [28])

This architecture fuses lower layers, encoding spatial knowledge, with upper layers capturing object-level information. Indeed, to obtain better segmentation, both object-level and low-level pixel data are required. The refinement module mixes probabilities of the previous and current stages to keep all scale-information along the network.
Hence, Sharpmask outputs an object-level masks with the same size as the image. This segmentation is used as input in the following multipath [47] network.

Multipath Multipath [47] was inspired by the improvement in the object detection task [33] and aims at using them for recognition purposes. It drives its experiments on the COCO dataset, outlining three major changes to improve recognition process (see figure 4.11)

1. To detect objects at different scales, connections used to provide the detector with features from several network layers have been skipped
2. Add a foveal structure in the classifier to exploit context of objects at different object resolutions
3. Create a novel loss function to improve the localization

Those modifications create a flow of information along multiple paths. It can be visualized with figure 4.11.

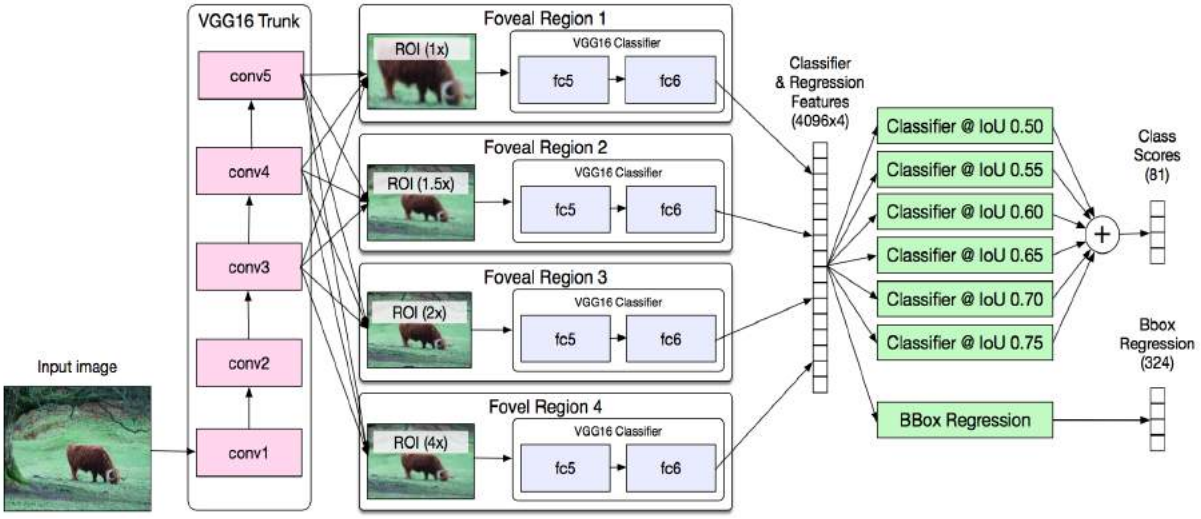


FIGURE 4.11: MultiPath architecture

Here are example for outputs classes from multipath on Sharpmask detected objects :

FIGURE 4.12: Sharpmask and Multipath network outputs

4.2.1.3 Merging geometric and semantic

Those geometry-aware and instance level segmentations are going to be merged to be propagated into the GSM. The semantic segmentation will re-unify detected objects that have been over-segmented by geometry.
From the output of multipath, a label object map, $\mathcal{L}_t^S$ is computed to be merged with $\mathcal{L}_t^G$, the labeled geometric segmentation using an operator α. If a pixel at position u is labeled l_g in $\mathcal{L}_t^G$ and l_s in $\mathcal{L}_t^S$, the score of l_s and l_g is incremented. This score is divided by the number of pixel labeled l_g to obtain a percentage of overlap for each pair such as :

$$\alpha(l_s, l_g) = \frac{\alpha(l_s, l_g)}{\#l_g}$$

For all l_g, only the l_s giving the higher score is kept.

$$\alpha_{max}(l_g) = \max_{l_s \in \mathcal{L}_t^S} \{\alpha(l_s, l_g)\}$$

All labels l_g with a percentage higher than 70% with the same l_s are fused.

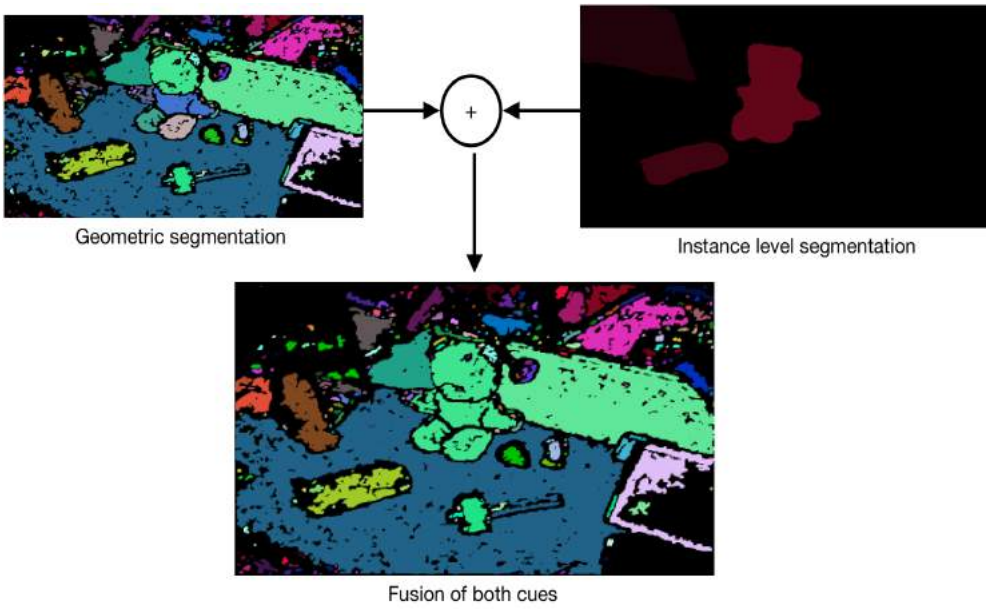


FIGURE 4.13: Label map $\mathcal{L}_t$, result of the fusion of the instance level semantic segmentation map $\mathcal{L}_t^S$ and the geometric map $\mathcal{L}_t^G$. $\mathcal{L}_t$ is an geometry-aware instance level segmentation map

The figure 4.13 shows the output of this stage and the final $\mathcal{L}_t$ which is going to be propagated in the next stage.

4.2.2 Segment label propagation

In this step, the key idea is propagating the segment of the GSM into $\mathcal{L}_t$ to be able to update the GSM in a further step. To do so, model surfaces underneath each GSM label are compared with surfaces underneath $\mathcal{L}_t$ labels. If surfaces are similar, the label value in the GSM is assigned to the segment in $\mathcal{L}_t$. This propagation simplifies the following merging step.

Here is a visual explanation :

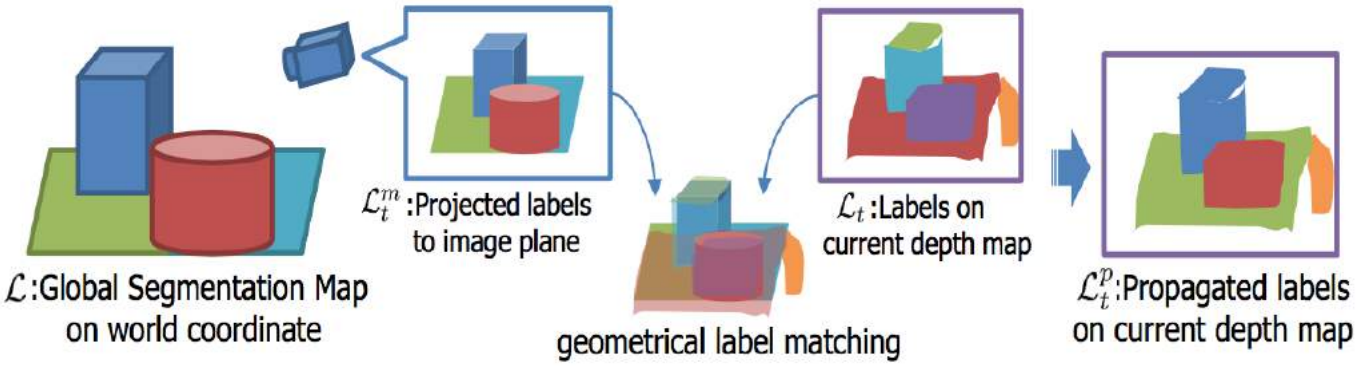


FIGURE 4.14: Toy example explaining the proposed Segment Propagation stage: first, segments from the GSM are re-projected onto the current camera plane; then, they are compared with those of the current depth map, so to identify corresponding segments between GSM and the camera plane (image and legend from [13])

4.2.2.1 Independence from the size of the model

The figure above shows that a model label map, $\mathcal{L}_t^m$, is computed from the GSM. $\mathcal{L}_t^m$ is obtained by ray-tracing using the current camera position. Next, only $\mathcal{L}_t^m$ is used to propagate the GSM labels to $\mathcal{L}_t$. Hence, no matter the model size, this segment propagation step will always keep the same complexity. This trick allows the system to continue running real-time even when the GSM grows.
Of course, the projection needed to compute $\mathcal{L}_t^m$ increases in complexity with the size of the model but it's an improvement in comparison with trying to propagate the labels from the entire GSM.

4.2.2.2 Set up for this stage

As explained before, to keep the same complexity no matters the GSM size, only 2D maps obtained by ray-tracing are used in this stage.
On one hand, the SLAM reconstruction is projected on the camera image plane to yield a vertex map, $\mathcal{V}_t^m$ and a normal map $\mathcal{N}_t^m$. The 'm' index denotes that those maps are obtained from SLAM model. On the other hand, we have the vertex and normal maps computed at the previous stage : $\mathcal{V}_t$ and $\mathcal{N}_t$.
Two label maps are used as well. The first one, $\mathcal{L}_t^m$, is obtained by ray-tracing of $\mathcal{L}$. It contains label $l_i \in \mathcal{L}_t^m$. The second one, $\mathcal{L}_t$ has been computed at the previous step and contains labels $l_j \in \mathcal{L}_t$.
The objective is to set l_j to l_i if the geometrical label matching detects similar underlying surfaces.

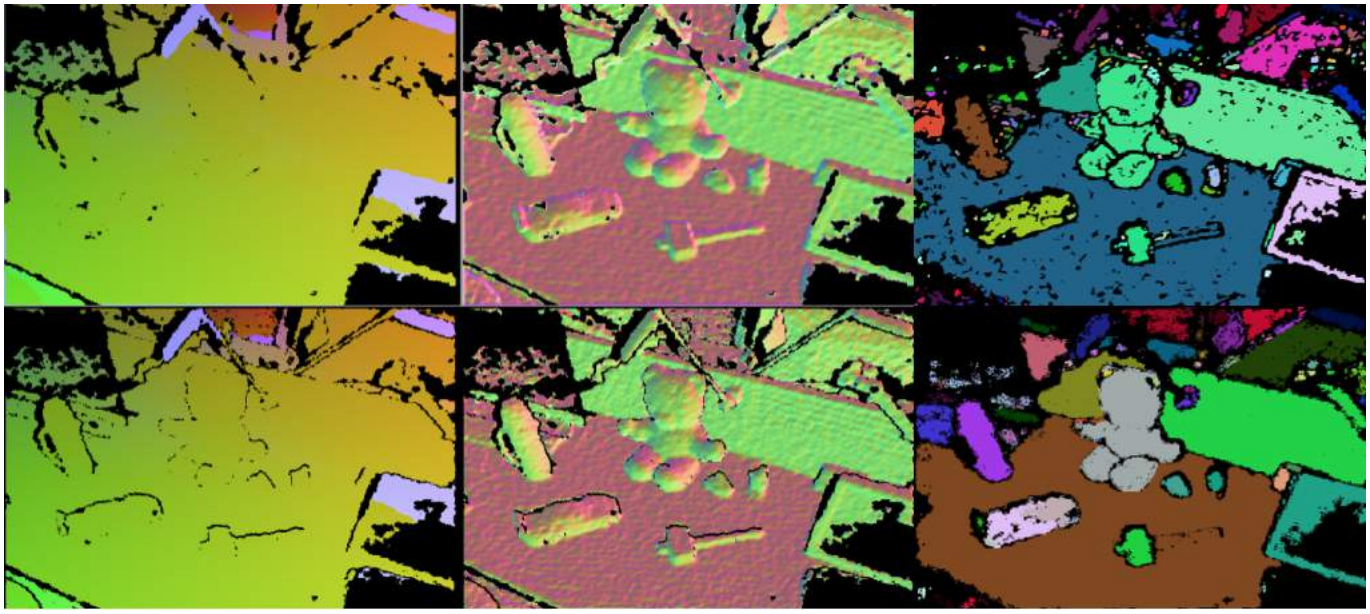


FIGURE 4.15: On top : $\mathcal{V}_t$, $\mathcal{N}_t$ and $\mathcal{L}_t$. On the bottom $\mathcal{V}_t^m$, $\mathcal{N}_t^m$ and $\mathcal{L}_t^m$

4.2.2.3 Correspondences between $\mathcal{L}_t^m$ and $\mathcal{L}_t$

To associate $l_j \in \mathcal{L}_t$ with $l_i \in \mathcal{L}_t^m$, a percentage of overlap is computed.
First, a number of corresponding points for each pair, called $\Pi(l_i, l_j)$ is calculated. To achieve that, all points in the image, $\forall u \in \mathcal{D}_t$ with their corresponding label, $l_i = \mathcal{L}_t^m(u)$ and $l_j = \mathcal{L}_t(u)$ are considered. For each pixel u in the depth map, two criterion are checked :

- The first expresses the distance between the vertices of $\mathcal{V}_t(u)$ and $\mathcal{V}_t^m(u)$ along the viewing ray. It's expressed as :

$$|(\mathcal{V}_t(u) - \mathcal{V}_t^m(u)).\frac{\mathcal{V}_t(u)}{|\mathcal{V}_t(u)|}| < \sigma_d(u)$$

  with $\sigma_d(u)$ the same adaptive threshold as in the previous step. Hence, the increasing amount of noise due to the distance to the sensor is taken into account.

- The second expresses the differences of angle between normals :

$$\mathcal{N}_t(u) \cdot \mathcal{N}_t^m(u)) > \tau_{\mathcal{N}}$$

  $\tau_{\mathcal{N}}$ is set 0.7 after experiments.

If those two conditions are satisfied, the surfaces underneath l_j and l_i associated with the pixel coordinates u are similar. Hence, $\Pi(l_i, l_j)$ is incremented.
To obtain a percentage of overlapping between l_j and l_i, $\Pi(l_i, l_j)$ is divided by the

number of points labeled by l_j in $\mathcal{L}_t$ in such as :

$$\tilde{\Pi}(l_i, l_j) = \frac{\Pi(l_i, l_j)}{\#l_j}$$

with $\#$ the cardinal operator.
From $\tilde{\Pi}$, the segment of $\mathcal{L}_t^m$ which best corresponds to $\mathcal{L}_t$ can be found :

$$\tilde{\Pi}_{max}(l_j) = \max_{l_i \in \mathcal{L}_t^m} \{\tilde{\Pi}(l_i, l_j)\}$$

The best associations between all labels $l_i \in \mathcal{L}_t^m$ and $l_j \in \mathcal{L}_t$ are now computed.

4.2.2.4 Propagation

From those associations, a propagated label map, called $\mathcal{L}_t^p$ is built. The following rules are used for each label $l \in \mathcal{L}_t^p$:

1. If $\tilde{\Pi}_{max}(l_j) > \tau_l$, the corresponding label l_i from $\mathcal{L}_t^m$ is propagated in $\mathcal{L}_t^p$ such as $l = l_i$. τ_l is set to 0.3 (meaning 30% of overlap as in the article [13]).
2. Otherwise, it means that the label is new, as no surfaces correspond. Therefore, $l = l_j$.

Furthermore, to avoid adding artifacts to the new propagate map $\mathcal{L}_t^p$, '0' label is assigned to segment in $\mathcal{L}_t$ smaller than 50 pixels.

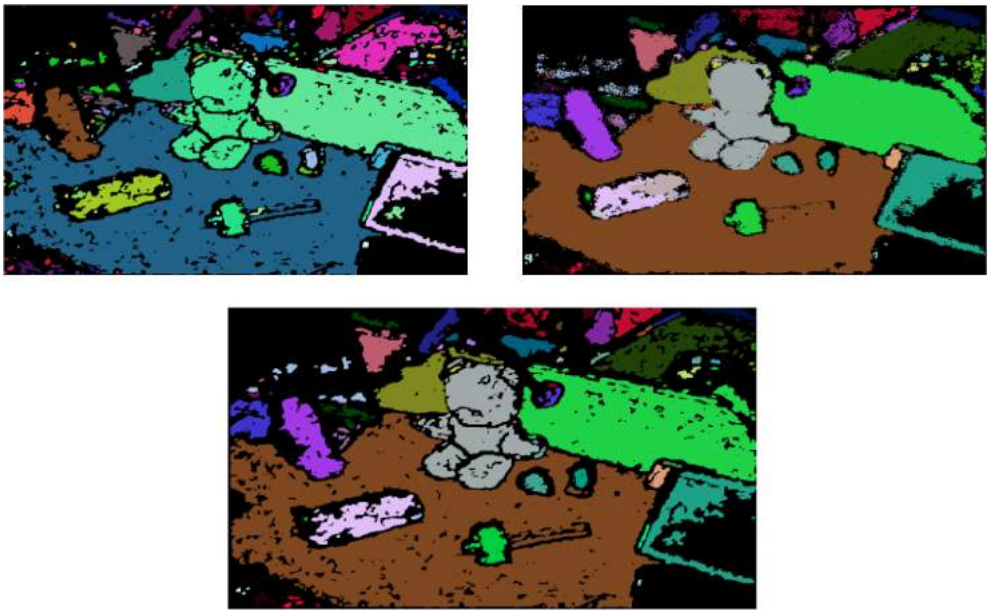


FIGURE 4.16: Illustration of the propagation process. On top left $\mathcal{L}_t$, on top right $\mathcal{L}_t^m$. On the bottom $\mathcal{L}_t^p$

In other worlds, $\mathcal{L}_t^p$ has $\mathcal{L}_t$ boundaries with $\mathcal{L}_t^m$ labels.

4.2.3 Segment merging

This part puts forward a solution to avoid over-segmented an object occluded in some frames.
Indeed, if an object is occluded by another one, it might be segmented using two different labels. However, when the camera moves, it could reveal that those two segments are actually lying on the same object and then, need to be merged.
The following figure illustrates this issue :

FIGURE 4.17: An object initially segmented into two parts due to an occluding foreground surface (left), is then merged into the same segment when a different camera view reveals the right connectivity (middle and right) (image and legend from [13])

Hence, not only $\tilde{\Pi}_{max}(l_j)$ is computed to find the best overlapping segment l_j for l_i, but all segments l_i with underneath surfaces sufficiently similar to the one below l_j are kept. In other words, segments l_i with $\tilde{\Pi}(l_j, l_i)$ higher a certain threshold are stored in memory. The threshold is set to 0.2 in my implementation. Those segments form a new set, called $\mathcal{L}_{l_j}$.
As the depth map are often noisy, merging directly segments could lead to errors in $\mathcal{L}_t^p$. Then, merging is performed when it has been seen a sufficient amount of time. It's done by computing a confidence measurement between each pair of segments l_a and l_b in $\mathcal{L}_{l_j}$. This confidence measurement, $\Psi_t^m(l_a, l_b)$ is initialized at zero and incremented each time a merging hypothesis occurs such as :

$$\Psi_t^m(l_a, l_b) = \Psi_{t-1}^m(l_a, l_b) + 1$$

Confidence of a merging observed at a previous time step but not in the current frame is decreased. Indeed, this merging could have been caused by noise in the depth frame. So, for merging not seen in this time step, $\Psi_t^m(l_a, l_b)$ is updated such as :

$$\Psi_t^m(l_a, l_b) = \max(\Psi_{t-1}^m(l_a, l_b), 0)$$

When a confidence $\Psi_t^m(l_a, l_b)$ reaches a certain threshold, set to 5, the merging is confirmed and label l_b is set to l_a in $\mathcal{L}$.
The idea of the incorporating a confidence measurement comes from [16] where the surfels in the model are added in the reconstruction only when they have been observed a sufficient amount of time.

4.2.4 Segment Update

The global segmentation map now needs to be updated with the final $\mathcal{L}_t^p$. The same procedure as above is followed and a confidence measurement is computed before adding a segment in the GSM. Indeed, updating the GSM directly from $\mathcal{L}_t^p$ would not be robust as $\mathcal{L}_t^p$ could be noisy. Hence, for each element $\mathcal{L}(p)$ in the GSM, a confidence $\Psi_t(p)$ is computed.
First of all, each element $\mathcal{L}_t^p(u)$ is associated with an voxel of $\mathcal{L}(p)$ using ray-tracing. Only confidences of voxels located in the band of the TSDF are updated. Three cases can occur :

1. $\mathcal{L}(p)$ has not been labeled yet. Hence, $\mathcal{L}(p)$ is labeled with the value of $\mathcal{L}_t^p(u)$ and the confidence $\Psi_t(p)$ is set to zero.
2. The label in $\mathcal{L}(p)$ and $\mathcal{L}_t^p(u)$ are similar. Hence, the confidence measurement is update such as :
$$\Psi_t(p) = \min(\Psi_{t-1}(p) + 1, \Psi_{max})$$
Ψ_{max} is the value where a label is thought to be sufficiently safe to be added to the model (set to 10).
3. The labels in $\mathcal{L}(p)$ and $\mathcal{L}_t^p(u)$ are not similar. Hence, the confidence measurement is decreased such as :
$$\Psi_t(p) = \max(\Psi_{t-1}(p) - 1, 0)$$

If $\Psi_t(p)$ reaches zero, it could mean either that the scene has changed due to movements or that the noise level is high. In both cases, the label of $\mathcal{L}(p)$ is set to its new value $\mathcal{L}_t^p(u)$ with a confidence of zero. If the confidence reaches Ψ_{max}, $\mathcal{L}(p)$ is updated in the model.

Chapter 5

Evaluation

To evaluate the system, two methods were used. On 2 sequences, the results were compared with the "geometry-only" implementation from [13]. It provides visual confirmations that my system computes a better object-instance segmentations than [13].
To evaluate numerically, the ScanNet [7] dataset was used as ground truth to measure the accuracy of the object segmentations using "geometry-only" or "deep-learning and geometric" frame-wise segmentation.

5.1 Teddy-bear sequence

The sequence lasts 220 frames and shows a teddy bear and a phone on a desk :

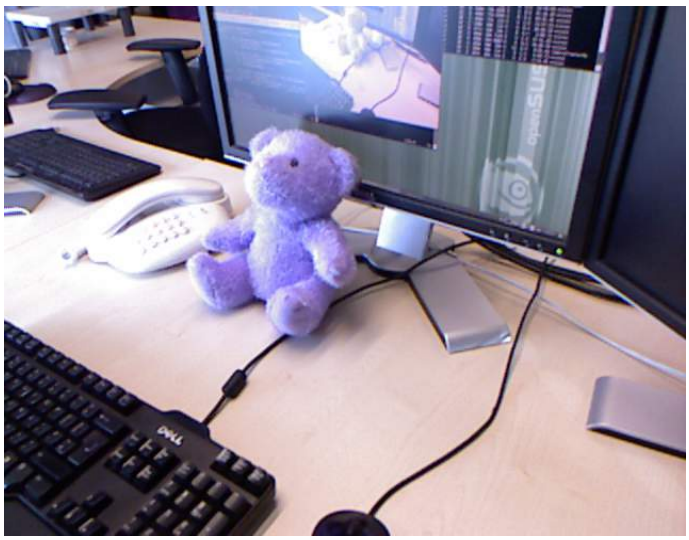
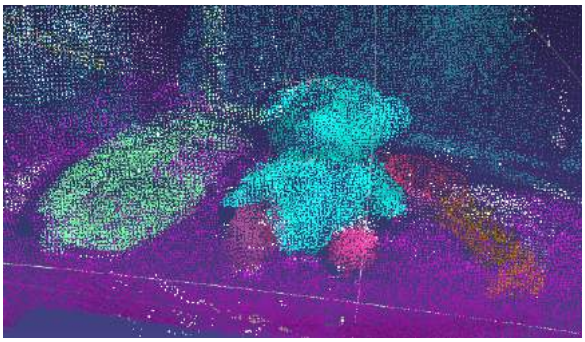


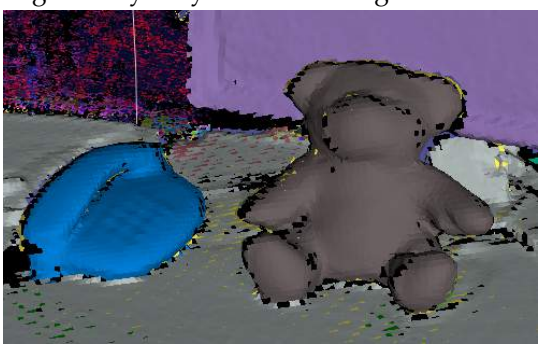
FIGURE 5.1: Frame from the teddy sequence
Frame from the teddy sequence

The mesh resulting from the [13] implementation (provided on line) is used for comparison. [13] implementation is based on a geometry only frame-wise segmentation. Hence, the improvement brought by the instance level segmentation is shown off.

[13] implementation only outputs a labeled point cloud and not a result frame by frame. Hence, a comparison of the incremental processes isn't possible.



(A) Resulting point cloud from [13], using geometry-only frame-wise segmentation



(B) Resulting mesh for my implementation, using "geometry and deep-learning" frame-wise segmentation

FIGURE 5.2: Visual comparison for the teddy bear sequence

The complete meshes can be found at [44] (teddy_bear folder). To compare visually, the teddy bear has been segmented as a single object in my output whereas it is over-segmented in the [13] result. In both cases, the phone was correctly segmented.
A video showing my implementation result frame by frame can be found in [44].

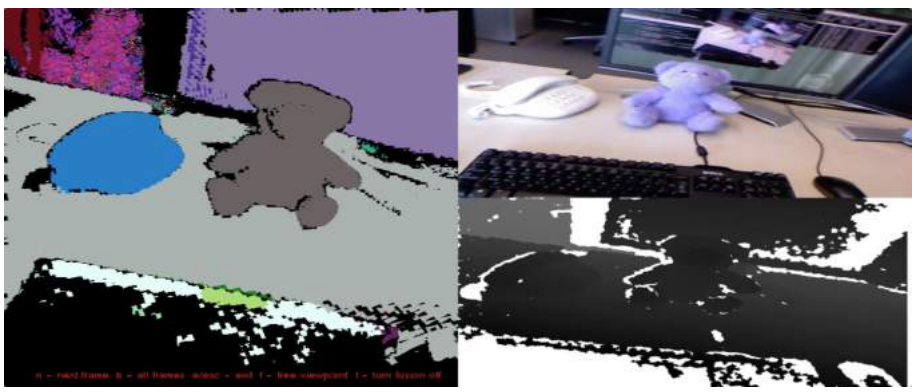
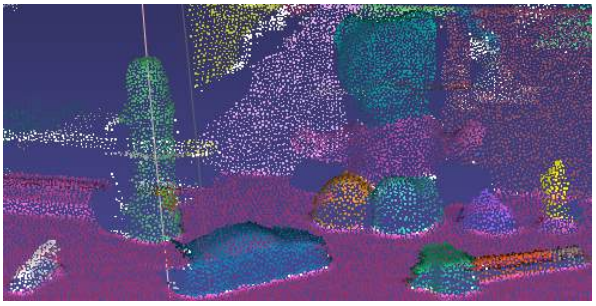


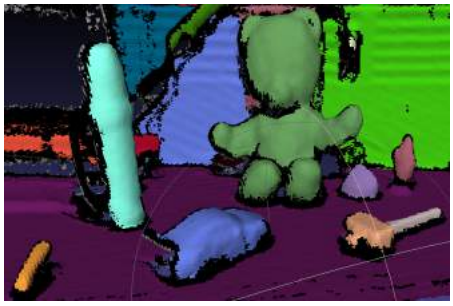
FIGURE 5.3: Frame from the video available at [44] showing the incremental reconstruction frame by frame. On the left, the GSM projected into camera frame and on the right the input (RGB and depth frame) at this time

5.2 Desk sequence

The sequence as been acquired using a ASUS RGB-D camera, calibrated with [4]. It contains 180 frames. The output meshes for both implementations are :



(A) Point cloud resulting from [13], using geometry-only frame-wise segmentation



(B) Mesh resulting from my implementation, using "geometry and deep-learning" frame-wise segmentation

FIGURE 5.4: Visual comparison for the desk sequence

As for the previous sequence, the teddy bear is segmented as a single object. (video and final meshes are available at [44], desk folder).

FIGURE 5.5: Frame from the video available at [44] showing the incremental reconstruction frame by frame. On the left, the GSM projected into camera frame and on the right the input (RGB and depth frame) at this time

Those 2 sequences highlight the improvement brought by fusing deep-learning and geometry during 2D frame-wise segmentation, in comparison to using only geometry. An 3D object-aware segmentation is now computed even in the case of complex objects. Moreover, as the teddy bear has been detected, an instance-level detection has also been performed in 3D.
To measure this enhancing quantitatively, the ScanNet dataset was used as ground truth.

5.3 ScanNet ground truth

5.3.1 ScanNet dataset

Before ScanNet, datasets available for RGB-D scene understanding were only covering few scenes with limited semantic annotations. Yet, for supervised deep-learning researches, large and labeled scenes are needed. Hence, ScanNet was created to provide data in this context. It contains 2.5M views in 1513 RGB-D scenes along with ground truth surface reconstructions, 3D camera poses, and semantic segmentations. The scenes were captured using a system computing first, a 3D reconstruction and second, a segmentation using a graph-cut algorithm. Next, semantic annotations were obtained by crowd-sourcing.
The following figure explains the pipeline :

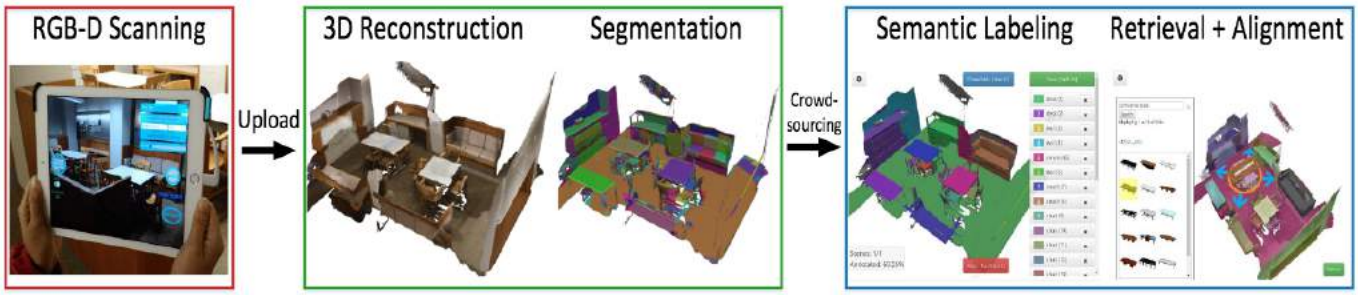


FIGURE 5.6: Overview of the RGB-D reconstruction and semantic annotation framework of ScanNet (figure and legend from [7])

Here are ScanNet scene examples :

FIGURE 5.7: Examples of ScanNet scene (figure from [7])

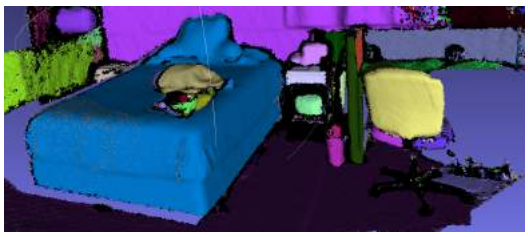
Those scenes are very challenging for my algorithm. Indeed, they are crowed and the depth range is ample. However, my results are encouraging.

5.3.2 Qualitative comparison

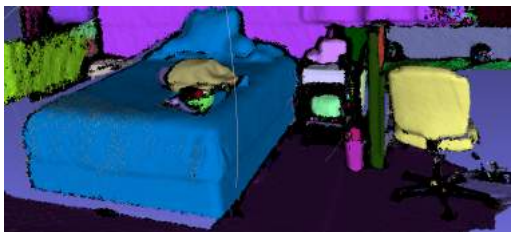
To measure the improvement brought by the cues fusion, the "geometry only" and "deep-learning and geometry" implementations are compared with the ScanNet ground-truth. For each sequence in the dataset, ground-truth 2D labeled projections on camera frame are available. Scene object segmentations from the "geometry only" and "deep-learning and geometry" results are compared to the ground-truth frame by frame. 3 sequences were exploited and complete meshes and videos are available on line at [44].
However, for memory issue, I wasn't able to use all the frames of the sequences and reconstruct the entire scenes. Hence, only regions of the rooms are reconstructed.

5.3.2.1 Bed room sequence

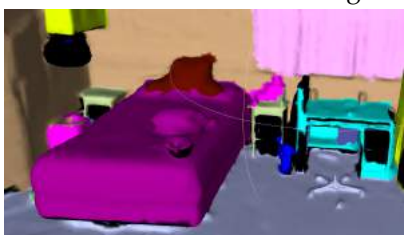
Here are some visual comparison between the "geometry only" and "deep-learning and geometry" result.



(A) Mesh resulting from the "geometry only" implementation



(B) Mesh resulting from the "deep-learning and geometry" implementation



(C) Ground truth for the sequence

FIGURE 5.8: Visual comparison for the room sequence

A default from the ScanNet dataset can be highlighted here with the missing chair in the reconstruction. However, the chair is segmented as one object in the "deep-learning and geometry" implementation.
Here is a video frame showing the difference between implementations in the incremental frame by frame reconstruction :

FIGURE 5.9: Frame of the bed room scene reconstruction video. **left :** reconstruction using "geometry only", **middle :** input of the algorithm RGB-D frame, **right :** reconstruction using "deep-learning and geometry"

To measure qualitatively the improvement of the object-aware segmentation, two objects were used : the bed and the chair.

The chair For each camera position, the projection of the 3D chair ground truth is compared with the 3D chair projection in the "geometry only" and "deep-learning and geometry".
Here is an illustration of this set up for the chair :

FIGURE 5.10: Frames of the video showing the projection of the chair reconstruction into camera frame for the sequence **left :** reconstruction using "geometry only", **middle :** ground truth from ScanNet, **right :** reconstruction using "deep-learning and geometry"

The complete video is available at [44]. Here is a graph showing the segmentation accuracy frame-by-frame. It represents a plot of the intersection divided by the union with respect to the ground truth for both implementations :

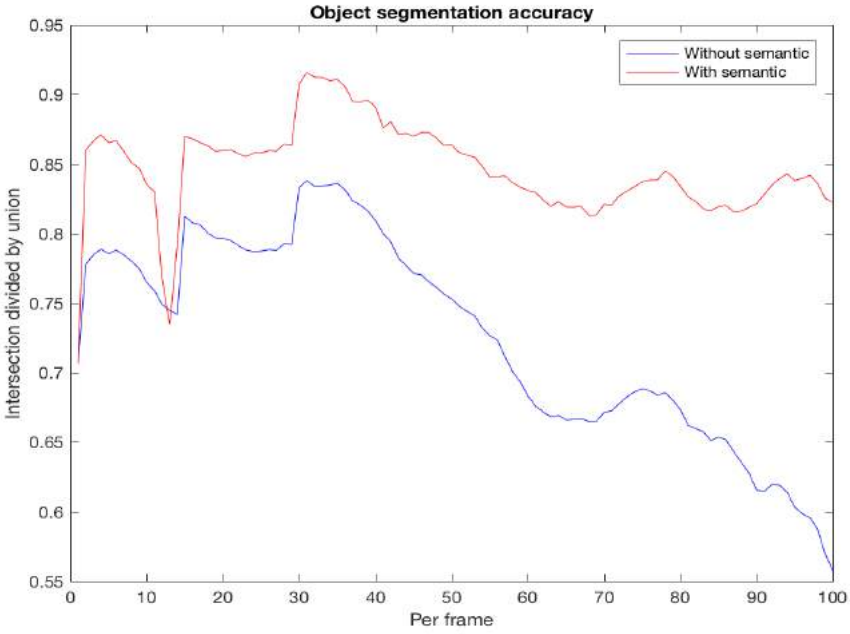


FIGURE 5.11: Segmentation accuracy of the chair for each frame of the sequence

A net improvement can be observed, 9,6% in average on the sequence.

The bed The same set up can be applied for the bed :

FIGURE 5.12: Frames of the video showing the projection of the bed reconstruction into camera frame for the sequence **left :** reconstruction using "geometry only", **middle :** ground truth from ScanNet, **right :** reconstruction using "deep-learning and geometry"

and the graph showing the segmentation accuracy (representing a plot of the intersection divided by the union with respect to the ground truth for both implementations)

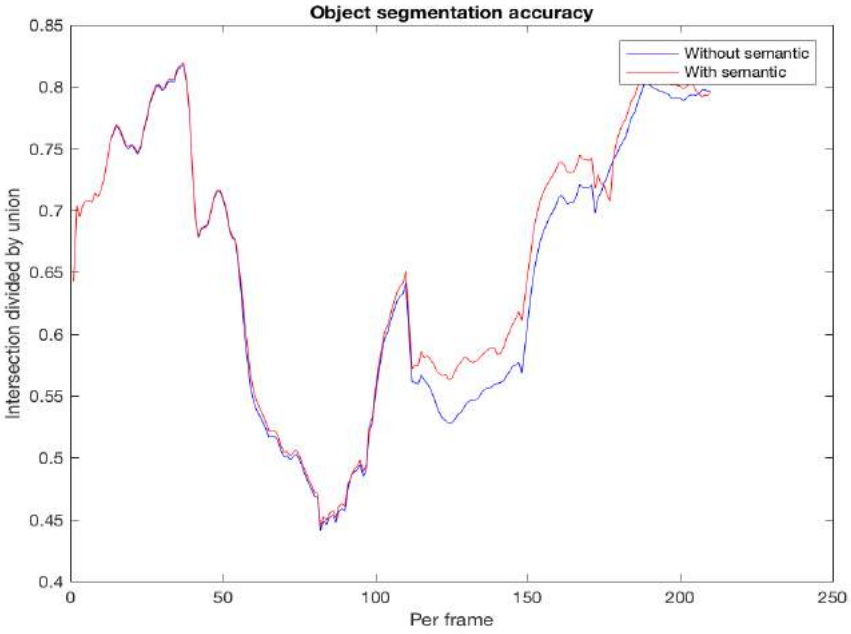
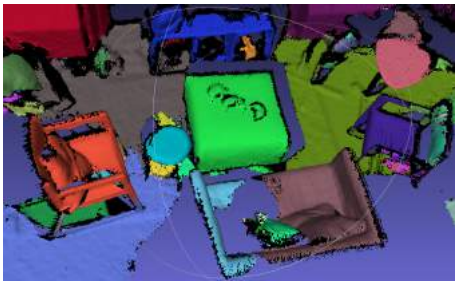


FIGURE 5.13: Segmentation accuracy of the bed for each frame of the sequence

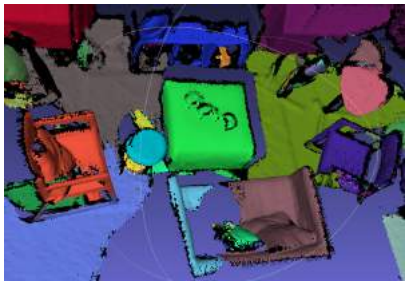
As the bed is a simple concave object, the geometry only segmentation already performed well. We can notice that the pillow is merged with the bed in both cases in my implementation. Furthermore, objects on the bed are considered as part of the bed in the ground truth but should actually be put aside. It depends if a segmentation looked for is a large overview of furnitures (like in ScanNet) or a more precise one. In the "deep-learning and geometry" case, when the bed is recognized, object on it are merged with it. Hence, the result is getting closer to the ground-truth.

5.3.2.2 Living room sequence

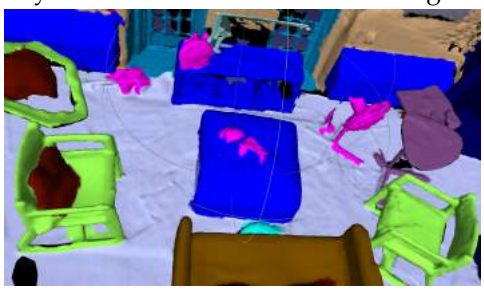
Here are meshes for visual comparison between the 2 implementations and the ground-truth.



(A) Mesh resulting from "geometry only"



(B) Mesh resulting from "deep-learning and geometry"



(C) Ground truth for the sequence

FIGURE 5.14: Visual comparison for the living room sequence

Complete meshes are available at [44]. The main difference can be spotted on the chairs. They are challenging because very thin and composed of multiple parts.

Here is the video frame highlighting the difference of reconstruction during the sequence in function of the method :

FIGURE 5.15: Frame of the living room scene reconstruction video. **left :** reconstruction "geometry only", **middle :** input of the algorithm, **right :** reconstruction "deep-learning and geometry"

The chairs For this sequence, the chairs are used for comparison.
Here is a frame of the video highlighting one chair reconstruction in both implementations.

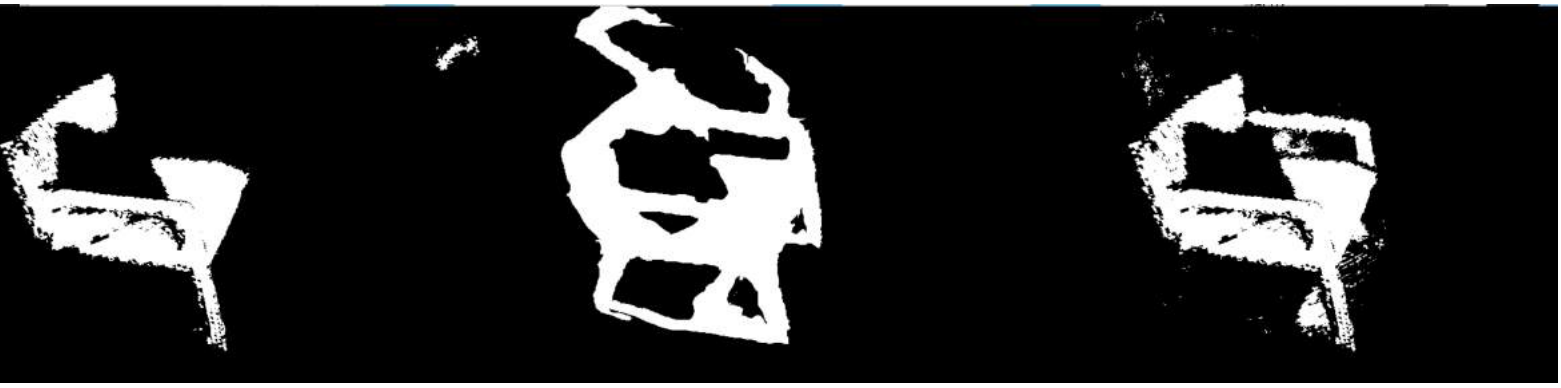


FIGURE 5.16: Frames of the video showing the projection of the chair reconstruction into camera frame for the sequence **left :** reconstruction "geometry only", **middle :** ground truth from ScanNet, **right :** reconstruction "deep-learning and geometry"

Several points can be noticed in this sequence :

- A baby seat in the back is considered to be a chair in the ground truth implementation, but isn't detected by the instance level segmentation
- There are some leaks in the "deep-learning and geometry" implementation due to imprecise boundaries of the instance level segmentation. As the chairs are very thin, geometry also struggles to obtain real boundaries.

Here is the graph of the reconstruction accuracy with respect to the ground truth :

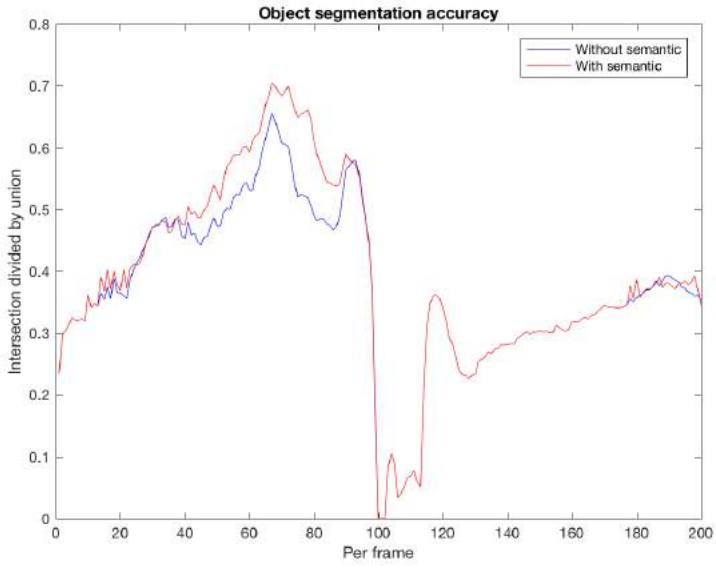
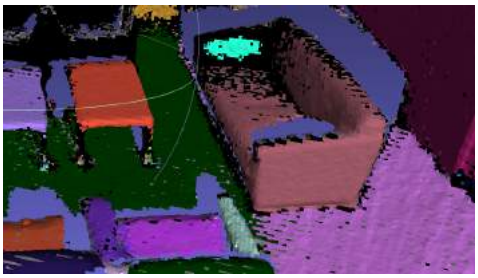


FIGURE 5.17: Segmentation accuracy for each frame of the sequence

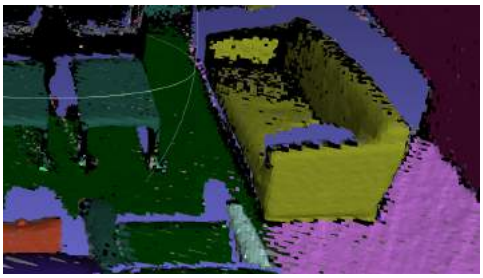
The "deep-learning and geometry" implementation outperforms the "geometry only" but less clearly than at the previous sequence.

5.3.2.3 Coach sequence

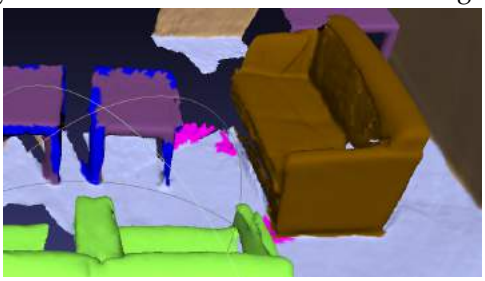
Here are the output meshes :



(A) Mesh resulting from "geometry only"



(B) Mesh resulting from "deep-learning and geometry"



(C) Ground truth for the sequence

FIGURE 5.18: Visual comparison for the coach sequence

The meshes are available at [44]. The right coach has been correctly segmented in one object. However, tables have been merged together and with the floor because they were detected as one object by the semantic segmentation. This sequence highlights one algorithm failure : when semantic segmentation unifies different objects, the algorithm is misled.

(A) RGB frame

(B) Semantic segmentation mask

FIGURE 5.19: Visual prove of algorithm failure

The left mask gathers floor, tables and coach at the back explaining why they are labeled together in the GSM.

Here is a frame from the video showing the difference in the incremental reconstruction between implementations (available at [44]) :

FIGURE 5.20: Frame of the coach reconstruction video. **left :** reconstruction "geometry only", **middle :** input of the algorithm, **right :** reconstruction "deep-learning and geometry"

The scene might look simplier but is actually more challenging. Indeed, it's mostly features-less and causes problems at InfiniTam to perform tracking. Hence, the model vertex and normal maps are very noisy and the propagation step didn't always work (especially at the end of the sequence).
However, the coach was reconstructed nicely and could be used to compare the results.

The coach Here is a frame of the video showing the coach reconstruction in both implementations with respect to the ground truth.

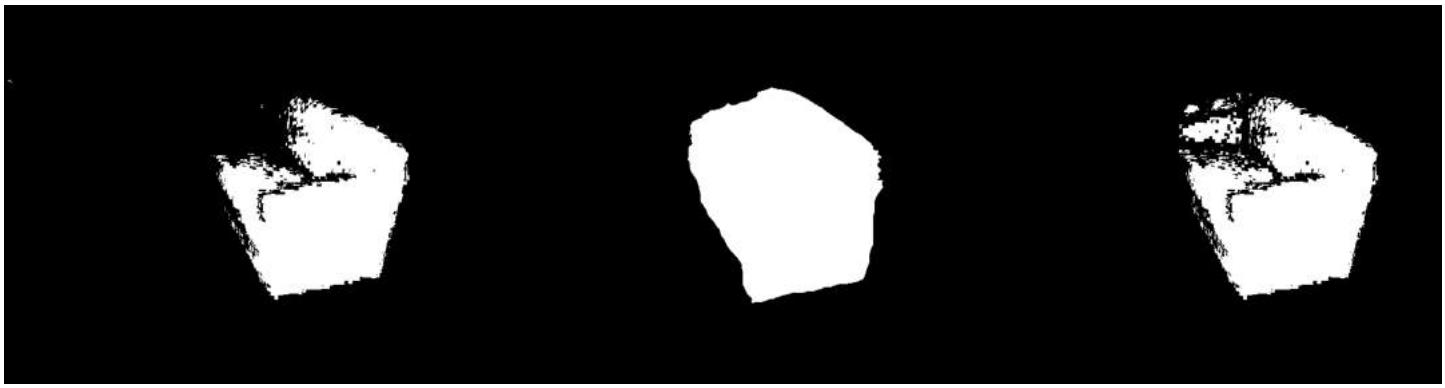


FIGURE 5.21: Frames of the video showing the projection of the coach reconstruction into camera frame for the sequence **left :** reconstruction "geometry only", **middle :** ground truth from ScanNet, **right :** reconstruction "deep-learning and geometry"

Here is the graph of the reconstruction accuracy for the coach:

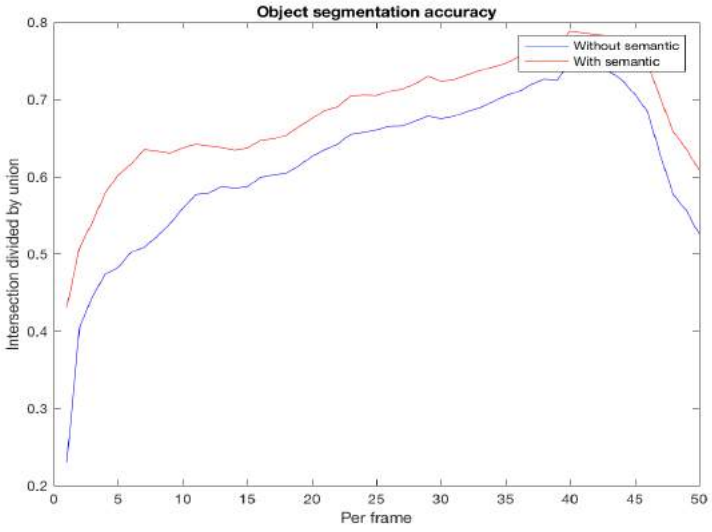


FIGURE 5.22: Segmentation accuracy for each frame of the sequence

The "deep-learning and geometry" implementation outperformed again the "geometry only" by an average of 5.8% through the sequence.

Chapter 6

Conclusion and further work

This project aims at achieving an incremental 3D segmentation by fusion of geometric and deep-learning cues. Indeed, as the background section showed, works have been mostly focused on either one or the other but no much has been done to combine both.
In my implementation, a reconstruction computed by a SLAM system is augmented by semantic labels. To add this information, a frame-by-frame analysis is performed. From the current depth frame, a over-segmented label map is computed and then, improved by instance-level semantic segmentation obtained from the RGB frame. Next, this 2D object-aware segmentation is propagated in the 3D segmentation model. During the previous chapter, the evaluation highlighted the improvement brought by the add of deep-learning results to achieve an object-aware segmentation on real-world data such as the ScanNet dataset.
To go further, some enhancements could be implemented.

6.1 Real-time implementation

My implementation is based on a frame-by-frame analysis propagated in the GSM. Hence, if this analysis is performed real-time, the system could run real-time. Even if the fusion takes too long, a system as PTAM could be implemented. The geometric segmentation would be performed at each frame and the instance-level semantic segmentation plus fusion only when possible. This way, real-time could still be achieved. However, this would required GPU implementation whereas mine is CPU-only for now.

6.2 Train a mask-RCNN network

In my implementation, to perform the instance-level semantic segmentation, Sharpmask [28] outputs a mask and the object class is found using a multi-path network from [47]. However, mask-RCNN [12] has been shown to yield better results by computing in parallel the mask, the bounding box and the object class. For my problem, more accurate the boundaries of the instance-level semantic segmentation are,

less "leaking" artifacts appear. Hence, replacing the combination of Sharpmask and multiPath by mask-RCNN should lead to better results.
Moreover, the COCO dataset [21] on which SharpMask and multiPath are trained isn't composed of room furniture pieces (except basic ones such as chair or bed). Hence, the instance-level semantic segmentation doesn't detect all objects in the scenes. Training the mask-RCNN network on the new ScanNet dataset, created specially for scene understanding, would probabily perform better and detect more objects yielding an even better segmentation.

6.3 Point-based implementation

A point-based implementation could also be an other enhancement. Indeed, my algorithm pipeline only requires a vertex and normal map from the reconstruction. Those could be obtained from a point based implementation as well. As for semanticFusion [22], the surfels would be augmented by a label. Moreover, averaging operations could be performed time to time to assure neighboring surfels to have a consistent labeling.

6.4 Moving objects

6.4.1 Co-Fusion

The other major advantage of point-based implementation is the availability to deal with moving scenes, as introduced in [16]. Indeed, in case of moving objects, removing or adding surfels is much more convenient than processing voxels. Furthermore, for real-life application, a robot needs to be able to compute the segmentation even for moving scenes. Whereas most papers consider moving regions as outliers, Co-fusion [34] uses either semantic or motion cues to infer a instance level segmentation of the scene while tracking and reconstructing 3D objects in real time. Co-fusion is a dense SLAM system based on the point based framework elasticFusion [41]. It processes following each new RGB-D frame in real-time to maintain a global reconstruction of the world with objects tracked independently (see figure 6.1).

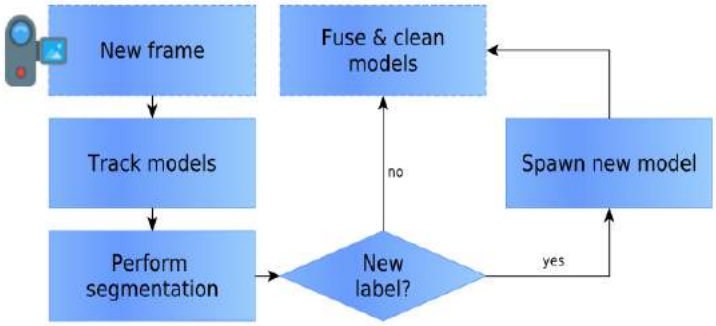


FIGURE 6.1: Overview of Co-fusion method showing the data-flow starting from a new RGBD-frame (figure and legend from [34])

Each segmented objects in the model is track independently. The current frame is segmented (using semantic or motion cues) and the result fused in the global model. Two segmenting methods were investigated :

- Motion segmentation, based on current frame pixel motions with respect to the background
- Multi-class image segmentation where an instance level segmentation is inferred from the RGB frame

The second method presents the advantage of segmenting all objects in the scene, moving or stationary. However, as boundaries of instance level segmentation aren't always accurate, it results in "leaking" in the reconstruction. Hence, the multi-class image segmentation provides the system with additional semantic information but isn't as satisfying in term of segmentation.

6.4.2 Segmentation by fusion of deep-learning and geometry

A third possibility has been investigated to segment the map : using the fusion of deep-learning and geometry presented in this thesis. This way, boundaries are constrained by geometry while semantic information are still available.

Here is a segmentation comparison :

FIGURE 6.2: Comparison between **Left :** Segmentation using deep-learning only, **Right :** Segmentation using deep-learning and geometry information

Whereas the "deep-learning only" boundaries are leaking around objects, the "deep-learning and geometry" implementation computes a sharper segmentation.
Finally, as the segmentation accuracy improved, the overall global model is enhanced as the following figure highlights :

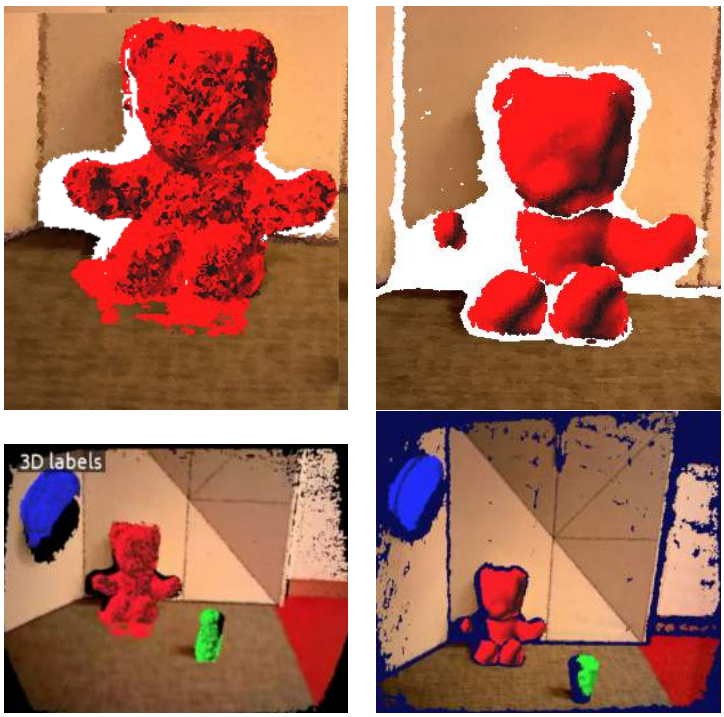


FIGURE 6.3: Comparison between **Left :** Output reconstruction with deep-learning only segmentation, **Right :** Output reconstruction with deep-learning and geometric segmentation

A comparison video can be found here [6].
Hence, the "geometry and deep-learning" segmentation strategy presented in this thesis can be applied in different cases and re-used to improve other framework results.

Bibliography

[1] A.Uckermann, R.Haschke, and H.Ritter. "Real-Time 3D Segmentation of Cluttered Scenes for Robot Grasping". In: *IEEE/RSJ International Conference on Intelligent Robots and Systems*. 2010.

[2] R. Manduchi C. Tomasi. "Bilateral Filtering for Gray and Color Images". In: *Proceedings of the 1998 IEEE International Conference on Computer Vision*. 1998.

[3] S. Izadi C. V. Nguyen and D. Lovell. "Modeling kinect sensor noise for improved 3D reconstruction and tracking," in: *Proceedings - 2nd Joint 3DIM/3DPVT Conference: 3D Imaging, Modeling, Processing, Visualization and Transmission, 3DIMPVT 2012*. 2012.

[4] "Calibration tool". In: (). URL: https://github.com/carlren/OpenNICalibTool.

[5] Liang-Chieh Chen et al. "Semantic Image Segmentation with Deep Convolutional Nets and Fully Connected CRFs". In: *CoRR* abs/1412.7062 (2014). URL: http://arxiv.org/abs/1412.7062.

[6] "Co-fusion comparison video". In: (). URL: https://drive.google.com/file/d/0B8fc1_DhoclHSlRGYzdJUXNsMjA/view?usp=sharing.

[7] Angela Dai et al. "ScanNet: Richly-annotated 3D Reconstructions of Indoor Scenes". In: *Proc. Computer Vision and Pattern Recognition (CVPR), IEEE*. 2017.

[8] Jifeng Dai, Kaiming He, and Jian Sun. "BoxSup: Exploiting Bounding Boxes to Supervise Convolutional Networks for Semantic Segmentation". In: *CoRR* abs/1503.01640 (2015). URL: http://arxiv.org/abs/1503.01640.

[9] Jifeng Dai, Kaiming He, and Jian Sun. "Instance-aware Semantic Segmentation via Multi-task Network Cascades". In: *CoRR* abs/1512.04412 (2015). URL: http://arxiv.org/abs/1512.04412.

[10] P. Felzenszwalb and D. Huttenlocher. "Efficient graph-based image segmentation". In: *International Journal of Computer Vision*. 2. 2004, 167–181.

[11] Ross Finman et al. "Efficient Incremental Map Segmentation in Dense RGB-D Maps". In: *IEEE Intl. Conf. on Robotics and Automation, ICRA, (Hong Kong)*. 2014.

[12] Kaiming He et al. "Mask R-CNN". In: *CoRR* abs/1703.06870 (2017). URL: http://arxiv.org/abs/1703.06870.

[13] Tateno K., Tombari F., and Navab N. "Real-Time and Scalable Incremental Segmentation on Dense SLAM". In: *IEEE/RSJ International Conference on Intelligent Robots and Systems*. 2015.

[14] L. Bo K. Lai and D. Fox. "Unsupervised feature learning for 3d scene labeling". In: *Int. Conf. on Robotics and Automation (ICRA)*. 2014.

[15] Andrej Karpathy, Stephen Miller, and Li Fei-Fei. "Object Discovery in 3D Scenes via Shape Analysis". In: *International Conference on Robotics and Automation (ICRA)*. 2013.

[16] M. Keller et al. "Real-Time 3D Reconstruction in Dynamic Scenes Using Point-Based Fusion". In: *International Conference on 3D Vision (3DV)*. 2013.

[17] Georg Klein and David Murray. "Parallel Tracking and Mapping for Small AR Workspaces". In: *Proc. Sixth IEEE and ACM International Symposium on Mixed and Augmented Reality (ISMAR'07)*. 2007.

[18] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. "ImageNet Classification with Deep Convolutional Neural Networks". In: *Proceedings of the 25th International Conference on Neural Information Processing Systems*. NIPS'12. Lake Tahoe, Nevada: Curran Associates Inc., 2012, pp. 1097–1105. URL: http://dl.acm.org/citation.cfm?id=2999134.2999257.

[19] Olaf Kähler et al. "Very High Frame Rate Volumetric Integration of Depth Images on Mobile Devices." In: *IEEE Trans. Vis. Comput. Graph.* 21.11 (2015), pp. 1241–1250.

[20] K. Xie L. Nan and A. Sharf. "A search-classify approach for cluttered indoor scene understanding". In: *ACM Transactions on Graphics (Proceedings of SIGGRAPH Asia 2012)*. Vol. 31. 6. 2012.

[21] Tsung-Yi Lin et al. "Microsoft COCO: Common Objects in Context". In: *CoRR* abs/1405.0312 (2014). URL: http://arxiv.org/abs/1405.0312.

[22] John McCormac et al. "SemanticFusion: Dense 3D Semantic Mapping with Convolutional Neural Networks". In: *CoRR* abs/1609.05130 (2016). URL: http://arxiv.org/abs/1609.05130.

[23] Niloy J. Mitra. "Registration slides from lectures at UCL". In:

[24] Richard A. Newcombe et al. "KinectFusion: Real-Time Dense Surface Mapping and Tracking". In: *IEEE ISMAR*. IEEE, 2011. URL: https://www.microsoft.com/en-us/research/publication/kinectfusion-real-time-dense-surface-mapping-and-tracking/.

[25] Sylvain Paris and Frédo Durand. "A Fast Approximation of the Bilateral Filter Using a Signal Processing Approach". In: vol. 81. 1. Hingham, MA, USA: Kluwer Academic Publishers, Jan. 2009, pp. 24–52. DOI: 10.1007/s11263-007-0110-8. URL: http://dx.doi.org/10.1007/s11263-007-0110-8.

[26] A. Pieropan and H. Kjellstrom. "Unsupervised object exploration using context". In: *Robot and Human Interactive Communication, 2014 RO-MAN: The 23rd IEEE International Symposium on*. 2014.

[27] S. Pillai and J. Leonard. "Monocular SLAM Supported Object Recognition". In: *Proc. of Robotics: Science and Systems (RSS)*. 2015.

[28] Pedro H. O. Pinheiro et al. "Learning to Refine Object Segments". In: *CoRR* abs/1603.08695 (2016). URL: http://arxiv.org/abs/1603.08695.

[29] Neil D. McKay P.J. Besl. "A method for registration of 3-D shapes". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence*. 1992.

[30] V. A. Prisacariu et al. "A Framework for the Volumetric Integration of Depth Images". In: *ArXiv e-prints*. 2014.

[31] T. Rabbani, F. A. van den Heuvel, and G. Vosselman. "Segmentation of point clouds using smoothness constraint". In: *International Archives of Photogrammetry, Remote Sensing and Spatial Information Sciences - Commission V Symposium 'Image Engineering and Vision Metrology*. Vol. 36. 2006, 248–253.

[32] Joseph Redmon et al. "You Only Look Once: Unified, Real-Time Object Detection". In: vol. abs/1506.02640. 2015. URL: http://arxiv.org/abs/1506.02640.

[33] Shaoqing Ren et al. "Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks". In: *CoRR* abs/1506.01497 (2015). URL: http://arxiv.org/abs/1506.01497.

[34] Martin Rünz and Lourdes Agapito. "Co-Fusion: Real-time Segmentation, Tracking and Fusion of Multiple Objects". In: *2017 IEEE International Conference on Robotics and Automation (ICRA)*. 2017, pp. 4471–4478.

[35] R. Salas-Moreno and B. Glocken. "Dense planar SLAM". In: *IEEE International Symposium on Mixed and Augmented Reality (ISMAR)*. 2014.

[36] E. Shelhamer, J. Long, and T. Darrell. "Fully Convolutional Networks for Semantic Segmentation". In: vol. 39. 4. 2017, pp. 640–651.

[37] Karen Simonyan and Andrew Zisserman. "Very Deep Convolutional Networks for Large-Scale Image Recognition". In: *CoRR* abs/1409.1556 (2014). URL: http://arxiv.org/abs/1409.1556.

[38] H Strasdat, JMM Montiel, and AJ Davison. "Visual SLAM: Why filter?" In: *Image and Vision Computing*. 2009.

[39] Christian Szegedy et al. "Going Deeper with Convolutions". In: *CoRR* abs/1409.4842 (2014). URL: http://arxiv.org/abs/1409.4842.

[40] Keisuke Tateno et al. "CNN-SLAM: Real-time dense monocular SLAM with learned depth prediction". In: *CoRR* abs/1704.03489 (2017). URL: http://arxiv.org/abs/1704.03489.

[41] Ben Glocker Andrew J. Davison Thomas Whelan Renato F. Salas-Moreno and Stefan Leutenegger. "ElasticFusion: Real-Time Dense SLAM and Light Source Estimation". In: *IJRR*. 2016.

[42] Hordur Johannsson Maurice Fallon John J. Leonard Thomas Whelan Michael Kaess and John McDonald. "Real-time large scale dense RGB-D SLAM with volumetric fusion". In: *International Journal of Robotics Research*. 2014.

[43] A. Uckermann et al. "3D scene segmentation for autonomous robot grasping," in: *IEEE/RSJ International Conference on Intelligent Robots and Systems*. 2012.

[44] "Videos and meshes for the evaluation". In: URL: https://drive.google.com/open?id=0B8fc1_DhoclHampTckNQeXpKQjA.

[45] Thomas Whelan et al. "Real-time Large-scale Dense RGB-D SLAM with Volumetric Fusion". In: *Int. J. Rob. Res.* Vol. 34. 4-5. 2015, pp. 598–626.

[46] Georges Younes, Daniel C. Asmar, and Elie A. Shammas. "A survey on non-filter-based monocular Visual SLAM systems". In: *CoRR*. Vol. abs/1607.00470. 2016.

[47] Sergey Zagoruyko et al. "A MultiPath Network for Object Detection". In: *CoRR* abs/1604.02135 (2016). URL: http://arxiv.org/abs/1604.02135.