Articulated Object Detection

Maciej Halber
MEng Computer Science
Submission Date: 26th April 2013

Supervisors
Niloy J. Mitra
Simon Julier

This report is submitted as part requirement for the MEng Degree in Computer Science at UCL. It is substantially the result of my own work except where explicitly indicated in the text. The report may be freely copied and distributed provided the source is explicitly acknowledged.
Abstract

We develop an object detection algorithm for difficult-to-detect objects with consumer-level range cameras. The increasing popularity of such devices made the process of obtaining depth data extremely easy. The availability of depth data motivates the need for algorithms that will be able to process it in meaningful ways, so that it then can be used in applications like robot navigation or object retrieval. However the quality of the point clouds obtained using these cameras often leaves much to be desired, which makes the processing stage very challenging. This project aims at developing a method that is able to process the data in a way that will conceptually alleviate the data issues. In particular we consider the problem of detecting articulated objects like laptops in the office-type, indoor environments.

This report has two main contributions. The first is a graph-based scene representation that approximates the captured environment using simple, well described geometric primitives (nodes) and describes the geometric relationships between them (edges). A series of experiments to determine set of stable features to detect these relationships has been performed. A graph created in this way serves as a global scene descriptor, and the object detection can be posed as a sub-graph matching problem. Ways of determining node-to-node correspondence between scene and object of interest graph representations were investigated. Simple method to find this node to node correspondence has been developed, exploiting the fact that most of the scenes can be described by small and simple graphs. The problem has been also posed to fit into existing, advanced framework for graph matching. The second contribution is the way of to reason about the missing data. This project focused on the laptop detection, that can suffer from dramatic data loss, due to the fact that laptop screen tend to reflect structured light pattern. The method described is able to cope with such missing data in a robust way that is also able to reason about the data loss due to other issues, i.e. partial occlusion.

The system for laptop detection in cluttered office environment has been developed as a proof of concept for the presented approach. It is argued that the graph based approach is a reliable way of performing object detection, that however suffers greatly from the poor data quality, by performing the tests using synthetically generated data as well. The performance of the detection system is presented both for synthetically generated data and the captured real-world data set. The limitations and possibilities for improvements are discussed.
Acknowledgments

First and foremost, I would like to express my sincere gratitude to my supervisors, Simon Julier and Niloy J. Mitra for providing me with an opportunity to work on this exciting project. Niloy's expertise and advice was always helpful and his remarks were extremely stimulating. Simon's enthusiasm and knowledge, and the fact that he has always been ready to talk about any issue were an invaluable help for me during this project.

I would like to thank numerous members of UCL Virtual Environments and Computer Graphics Group — especially to Fabrizio Pece for his interest in the project and always being ready to help with an advice as well as with any coding problems that I had, and to Michael Firman, for the support with stimulating discussions and pointing me to a number of interesting publications. Many thanks go also to my lecturers, Tim Weyrich and Gabriel Brostow for providing always inspiring material and often offering an insightful comments on the project. I would like to also thank my tutor, Anthony Steed for guiding me through my four years at UCL.

I would like also to thank Amnon Shashua from Hebrew University of Jerusalem for providing me with the source code from his paper "Probablisitic Graph and Hypergraph Matching".

Last, but not least, I would like to thanks to my parents for always believing in me and supporting me throughout my studies.
## Contents

1 Introduction 1
  1.1 Motivation 1
  1.2 Main Contributions 4
  1.3 Algorithm Overview 4

2 Primitive-Based Scene Reconstruction 7
  2.1 Introduction 7
  2.2 Related Work 8
  2.3 Primitive Detection 9
    2.3.1 Clipped Plane Primitive 9
    2.3.2 RANSAC-based Fitting of Clipped Plane 10
    2.3.3 Experiments 12
    2.3.4 Modified Clipped Plane Fitting 16
    2.3.5 Detection Procedure 18
  2.4 Summary 22

3 Reasoning about missing data 23
  3.1 Introduction 23
  3.2 Related Work 24
  3.3 Scene Segmentation Image Analysis 25
    3.3.1 Shadow Region Detection 26
    3.3.2 Image-based Clipped Plane Creation 27
  3.4 Summary 32

4 Graph-Based Object Detection 34
  4.1 Introduction 34
  4.2 Related Work 35
  4.3 Graph Construction 36
## CONTENTS

4.4 Object detection ........................................... 38  
  4.4.1 First Order Graph Matching .......................... 39  
  4.4.2 Probabilistic Graph Matching ......................... 42  
  4.4.3 Laptop Detection ...................................... 43  
4.5 Summary ..................................................... 43  

5 Results .......................................................... 45  
  5.1 Introduction ............................................... 45  
  5.2 Synthetic Scenes .......................................... 46  
  5.3 Real-world Scenes ........................................ 49  
    5.3.1 Good Quality Scenes ................................. 49  
    5.3.2 Entire Data Set ....................................... 50  
  5.4 Summary ..................................................... 53  

6 Future Work .................................................. 55  

Appendices ...................................................... 57  
  A Summer Internship Report ................................. 58  
  B Project Plan ................................................ 66  
  C Interim Report .............................................. 68  
  D Source Code Repository ................................. 70
1 | Introduction

1.1 Motivation

Object detection is a process of finding the instances of real world objects of all kinds in the input digital data. Being able to recognize and locate an object of interest offers a wide range of very useful applications. Face detectors [37] are widely used in digital cameras to aid photographers and even in personal computers as a security measure. Detection systems are used in cars for pedestrian detection [4] and can be employed for empty parking space detection [36]. Another very exciting applications are in the filed of robotics. Being able to recognize objects allows us to build systems for robots to perform tasks like object fetching and grasping [21, 6]. Due to ubiquity of applications object detection has been a subject of very active research for many years now.

In the past a vast majority of the work in the object detection has been done using purely two dimensional images and videos. The general focus of these algorithms was to design and extract good, descriptive local features. Such features need to be a transformation invariant since object can appear at different orientations and scales. After such features have been extracted these approaches would use various learning algorithms to learn classifiers that would be able to perform the recognitions. A lot of effort has been put into designing various features, like histograms-of-gradients [3], bag-of-words [2], SIFT [20].

These features can not only be used for the recognition, but as well to match feature points between two images robustly. Being able to find the correspondence points between two images capturing scene from different viewpoints make it possible to obtain depth data using techniques like SfM (Structure-from-Motion) [34]. SfM techniques are widely used to reconstruct the exterior environments, however are not as useful for interiors. Since interior scenes often contain a lot of texture less areas (walls) there might be not enough feature points to extract.
However with the rise of popularity of devices like Microsoft Kinect, small range cameras available at reasonable price point has led to a large amount of research in the processing of point clouds that do represent interior environments. It needs to be noted that range scanners have been available earlier, however they have been very bulky (Metis) and expensive (Mantis Vision). The Kinect and similar devices (Asus Xtion Pro etc.) has made the depth data easily obtainable, even for average user. However there is a trade off, the quality of the data offered by these devices often leaves much to be desired.

Nevertheless many techniques have been developed to perform object detection using the depth data. Some techniques aimed at using the depth data with conjunction with the RGB data, like works by Spinello and Arras [32]. Others [9], [5], [26], [18] have followed the well established direction of 2D object detection and aimed at designing robust local features to perform object detection. These techniques often do not consider any contextual information on the scene, however we believe that understanding the global structure of the scene would provide a great aid in object detection. Similar point is made in Silberman et al. work “Indoor Segmentation and Support Inference from RGBD Images” [23]. Another interesting point, that lead us to believe that we should go beyond detecting the local features and look as the scene as a whole is made in the work by Nan et al. [22]. Their argument is made that in a point clouds of interiors obtained using Kinect-like devices the amount of clutter and noise causes a lot of redundant features to be detected.

These arguments has led to the conclusion that we should consider a global view of the scene. Recently a number of papers that has focused exactly on the scene understanding has been published offering some very impressive results. Kopulla et al. [16] provides a way of segmenting the point cloud and assigning semantic labels to each of the segments, so that system is aware what kind of objects is represented by certain set of points. Works by Shao et al. [30] and Nan et al. [22] both provide a way of reconstructing the scene by also segmenting the point cloud into clusters representing different classes of objects and swapping them with models from database. In that way both papers are able to go from point clouds to actual 3D geometry and thus removing problems related to point cloud data like noise and occlusion. Still the focus of both papers is not the object recognition but rather a plausible reconstruction of the scene. No attempt is made to use the information about the scene as form of a prior for object detection. Work by Kim et al. [14] is focusing more on the object detection through developing a scene understanding. The algorithm contains
of two stages — during the first, learning phase, the models (primitive based approximations of the objects) present in the scene are learned from a set of point cloud captures. The second stage is the recognition stage that provided with a new capture is able to quickly recognize the learned models.

In our approach we follow philosophy similar to Kim et al. [14], where we propose to represent the scene from single-view capture by using high level primitives, that will describe the global relationships between objects in the scene. These relationships should be robust to local object transformations like rotation around the laptop hinge joints. Unlike Kim et al. we do not require any earlier analysis of the scene to learn the models, nor do we require multiple captures of the scene. In our approach, the scene representation will be compared with the assumed, reference representation containing the object of interest — this can be thought of a descriptor that contains a prior information on the context in which we expect to see object of interest. The decision to use just single scan is caused by the fact that we want to start analyzing objects as soon as possible, thus we do not want to force user to first acquire multiple number of scans of the scene from different views, before any object recognition can be done. The class of objects this project focuses on are laptops, which are interesting objects, that provide a set of challenges for detection(fig. 1.1). One of them is the mentioned articulation factor — the rotation of the laptop lid around the hinge joint. Second one is caused by the decision of using just a single scan — due to the reflective properties laptop screens are often not captured by the range scanners, especially when observing the laptop from side views. This can leave substantial amounts of data missing and hinder the detection rates. One could immediately argue that we could use multiple view scans and align them using algorithms like ICP [27], or use reconstruction frameworks like Kinect Fusion [24]. The problem with using such techniques is that we do not want to assume that the laptop is stationary, or that the lid has not been tilted. If any event like that would occur during continuous capture, it would result in major ghosting in the point cloud.

Figure 1.1: Laptops appearance vary greatly based on its overall orientation and the lid position.
1.2 Main Contributions

There are two main contributions of this work:

- The graph based representation of the scene — the input point cloud is represented as a set of geometrical primitives, that co-exist having various types of relationships with other primitives in the set. The primitives and the relationships are used to build graph scene descriptors.

- The missing data reasoning — due to poor quality of point clouds captured using Kinect-class cameras number of methods to reason about corrupted data has been investigated. The main focus was on the data lost due to reflective properties of surfaces, like the laptop screens.

1.3 Algorithm Overview

Figure 1.2 provides graphical presentation of the algorithm, explicitly stating what is the input and output of each stage. We argue that an indoor scenes captured using a range camera, represented as point clouds (fig. 1.2a) can be approximated by a set of simple geometrical shapes, like boxes or rectangles oriented in 3D space. Many 3D packages offer user with tools that can be used to build models of objects or environments as a set of primitives placed in the 3D space (fig. 1.2b, 1.3).

What is appealing about such form is that each of the primitives is mathematically well described, so the way primitives relate to one another is very easy to determine. However creating such approximation is a challenging task, due to rather poor quality of the point clouds obtained using consumer-level RGBD cameras — a large part of this report will focus exclusively on the ways how to perform such analysis. Once the primitives are determined (fig. 1.2c), one can easily describe such set of primitives using a graph, where each node is representing a single primitive and the connectivity is defined by the detected geometrical relationships (fig. 1.2d). A type of relationship we might be interested in detecting could be, for example, checking whether two planes are parallel or not.

Such graph descriptor can be then used to compare scenes and objects present in each of them. Given a graph \( \mathcal{R} \) with labeled nodes and a unlabeled graph \( \mathcal{G} \), we can try to establishing node-to-node correspondence between the two (fig. 1.4).
Figure 1.2: An overview of graph representation creation sequential stages of the algorithm. (a) The input is the captured point cloud (b) Point cloud is analyzed and set of primitive shapes is obtained (c) New primitives to input set after reasoning about missing data. (d) Full set of primitives is used to generate graph $G$ (e) Node-to-node correspondences between graph $G$ and reference graph $R$ established (f) Based on the correspondence a laptop is detected.

If establishing such correspondence is possible, then we can use the result as a form of object detection (fig. 1.2f). In this project we focused on the laptop detection to demonstrate the validity of our approach to perform articulated object detection. Given that laptop is represented by two primitives, one relating to laptop screen and the second to laptop keyboard, there are properties like general proximity and the common edge between the two should not change, no matter how much the screen is tilted. That means that by detecting an articulation invariant relationship we should be able to successfully do the comparison, and thus detection of laptops that have screens placed at different angles.

The choice of detecting laptops has introduced another serious challenge. Laptop screens are highly reflective surfaces, and the structured light pattern used by Kinect-like devices gets reflected from them. This causes the loss of data in
Figure 1.3: An example of simple 3D scene. Objects used to construct this scene are cuboids only, yet it is trivial for human observer to tell what kind of objects are present in such scene.

regions where the laptop screens should appear. Without the data there is no way to fit a primitive, and as a result our scene representation is incomplete. Techniques to deal with this particular problem were investigated and we will present a method to partially alleviate this issue in chapter 3.

The rest of the report is structured in a way that follows the sequential steps of our algorithm. In chapter 2 we discuss how a point cloud can be analyzed to generate a set of primitives. In chapter 3 we focus on the problem of the poor data quality, we describe how the point cloud is further investigated for the loss of data due to problematic properties of captured surfaces. In chapter we describe how can we investigate the scene representation for presence of object of interest, by performing a graph comparison with known, labeled reference graph \( R \). Two last chapters 5 and 6 present the detection results using our method, and the discussion on limitation and possible improvements in the future.
2 | Primitive-Based Scene Reconstruction

2.1 Introduction

Once we have obtained the point cloud data using RGBD camera, the primitive based reconstruction of the scene needs to be created. We wish to exploit the fact that indoor environments contain objects that can be represented as a union of very basic primitive shapes. By detection of such primitives we will obtain a very well behaved representation of the scene, that is much easier to use than the raw point cloud. In this chapter we will introduce what is the primitive type that we use and how the instances of this primitive are detected in the point cloud. We will discuss all of the issues that have been encountered and how we have resolved them.

This stage of our algorithm takes the point cloud $\mathcal{P}$ as the input. $\mathcal{P}$ is a set of unordered, unstructured points in the 3D space, obtained from real world environments using Microsoft Kinect sensor. The point cloud $\mathcal{P}$ is then analyzed for presence of the geometrical primitives using well known RANSAC algorithm. However the data acquired using Kinect is at most of mediocre quality. It suffers from obvious problems such as the sensor noise, but there are more issues that make the detection of primitives a challenging task. Among these are the non-uniform density of points per unit area, self-occlusion and missing some of the depth data due to properties of the surface that is being captured. In the following sections we will discuss all of these, and how to deal with these. The last problem is most severe, and will be discussed separately in the chapter 3. Once the primitives are detected they create a set $\mathcal{S}$ that will be the input to the next stage of the algorithm.
2.2 Related Work

Point clouds have been a subject of research for a long time now [13], however recently, with the popularization of commercial RGBD cameras, we observe even more research focusing on processing such data. Some of the work focused on extending the well known, and well performing 2D algorithms, but there is also a large number of novel methods that are designed with 3D point clouds in mind. The approach gaining more and more interest is the primitive based reconstruction of the scene. Similarly to our algorithm, these approaches aim at finding simple geometrical structures in the point clouds that will help to explain the shape or the object that given point cloud represents.

An important work in this area is the GLOBFIT method by Li et al. [19], that deals with fitting primitives on top of point clouds representing man-made objects. The authors exploit the fact that man-made objects are often just a union of a set of primitives such as planes, disks and cylinders that exist having mutual relationships, like being parallel, coaxial, coplanar etc. The authors note that due to the sensor noise, the relations might not be preserved, so they refine the initial fits by enforcing global relationships. As will be introduced in detail in chapter 4 our method also detects the relationships between the primitives, however, unlike GLOBFIT, these are not used to refine the fit quality.

Work by Shin et al. [31] provides a framework for grammar based classification, that also follows the idea of representing the objects as the set of primitive
parts. A geometric grammar is a tree that uses and/or relationships between the detected primitives as branches. For example in their method a stool can be described as the union of legs and a flat sitting surface. Then a union of a stool and a back support will provide a description of a chair. Very similar to our approach they provide system with an example grammar, then they detect primitives in the input data and aim at finding all arrangements of the primitives that match the grammar. They present very good results, yet there are some simplifications made — objects are assumed to be in the upright position to allow the parsing of the grammars. Also since the objects are represented as grammar of and/or relationships between cuboids of different sizes they rely heavily on the detection of properly shaped primitives. This is not a big issue for objects they are detecting (chairs), since legs and sitting surface are approximated by dramatically differently shaped primitives. Same cannot be said about objects one can find on an office desk.

2.3 Primitive Detection

2.3.1 Clipped Plane Primitive

As mentioned earlier, in section 1.1 our method focuses on the detection of articulated objects, specifically laptops. Laptops are essentially two planar surfaces that are joined by a hinge between the screen and the keyboard. This has led to the decision of using clipped planes as the primitives used in our method. A clipped plane \( P \) is defined by four 3D points \( \{p_1, \ldots, p_4\} \). Obviously, to define a 3D plane only three points are required, however it is a geometrical entity of infinite area. What is required however is a plane of a finite area - the four points \( \{p_1, \ldots, p_4\} \) are coplanar, and define enclosing rectangle on the 3D plane that is defined by any three of them.

Strictly speaking a clipped plane is defined by four points \( P = \{p_1, p_2, p_3, p_4\} \) such that

- All points \( p_i \) lie on the same plane with normal \( \vec{n} = \{a, b, c\} \)

\[
\forall p_i \in P \quad ax_i + by_i + cz_i + d = 0
\]
• Opposite line segments are parallel

\[ p_1p_2 \parallel p_3p_4 \cap p_1p_4 \parallel p_2p_3 \]

• Adjacent line segments are orthogonal

\[ \forall_{i,j,k \in \{1,2,3,4\}} \ p_ip_j \perp p_jp_k \]

With this formulation laptops can be easily and compactly represented as a pair of clipped planes. Such pair provide all the information needed to define the global orientation and position, as well as the laptop screen rotation with respect to the keyboard. Rest of important objects in office environments, like the desks or computer monitors are also very well described by such primitive, while all the uninteresting (for our purpose) clutter is not. Before we will explain how the whole point cloud is analyzed, we need to explain the tools used for the creation of a single clipped plane \( P \).

### 2.3.2 RANSAC-based Fitting of Clipped Plane

To fit a clipped plane \( P \) onto a set of points \( \mathcal{P} \), we need to find the plane coefficient \( P_c = \{a, b, c, d\} \) a well known RANSAC procedure is used:

The input to the RANSAC algorithm is a set of observed data values, a parameterized model which can explain or be fitted to the observations, and some confidence parameters.

RANSAC achieves its goal by iteratively selecting a random subset of the original data. These data are hypothetical inliers and this hypothesis is then tested as follows:

1. A model is fitted to the hypothetical inliers, i.e. all free parameters of the model are reconstructed from the inliers.

2. All other data are then tested against the fitted model and, if a point fits well to the estimated model, also considered as a hypothetical inlier.

3. The estimated model is reasonably good if sufficiently many points have been classified as hypothetical inliers.

4. The model is re-estimated from all hypothetical inliers, because it has only been estimated from the initial set of hypothetical inliers.
5. Finally, the model is evaluated by estimating the error of the inliers relative to the model.

This procedure is repeated a fixed number of times, each time producing either a model which is rejected because too few points are classified as inliers or a refined model together with a corresponding error measure. In the latter case, we keep the refined model if its error is lower than the last saved model.\footnote{The RANSAC procedure description from \url{http://en.wikipedia.org/wiki/RANSAC}}

![Figure 2.3: PCA provides orientation that can be used to find the minimum bounding rectangle](image)

Once the plane coefficients has been found, we need to determine the locations of the enclosing points $\{p_1, \ldots, p_2\}$. Since the RANSAC procedure provides us with the the coefficients $P_c$ of the plane the points will lie on, as well as a set of inliers $\mathcal{I}$, we need to find the orientation of bounding rectangle. Well known principal component analysis (PCA) procedure, that is defined as orthogonal linear transformation, that transform the data to a new coordinate system such that main axes are the directions of highest variance in the dataset. By running the PCA on the set of inlier $\mathcal{I}$ we obtain the main axes of local coordinate frame, in which we will find minimum area axis-aligned bounding box. The PCA procedure computes covariance matrix of $n$ data points $x_i$, defined as:

$$C = \frac{1}{n} \sum_{i=1}^{n} (x_i - \bar{x})(x_i - \bar{x})^\top$$

where $\bar{x}$ is the mean of all data points:

$$\bar{x} = \frac{1}{n} \sum_{i=1}^{n} (x_i)$$
By definition $C$ is square, so we can compute its eigendecomposition

$$C = V^{-1} \Lambda V$$

Since $C$ is also symmetric, the eigenvectors $\vec{v}_1, \vec{v}_2, \vec{v}_3$ (columns of $V$) are guaranteed to be orthonormal vectors, defining local coordinate frame that we are looking for. We will then proceed and find locations of $\{p_1, \ldots, p_2\}$ as the minimum bounding rectangle aligned with the directions given by two largest eigenvectors $\vec{v}_1, \vec{v}_2$ (fig. 2.3).

### 2.3.3 Experiments

After formulating how to fit a clipped plane primitives onto a set of points $I$ it was deemed necessary to investigate how the detection of these is influenced by properties of the point cloud data obtained using Kinect. It already has been pointed out that data acquired with Kinect is of rather poor quality. When capturing the test data, two main observation have been made:

1. The point density by unit area non-uniform. This is caused by the fact that Kinect is projecting structured light pattern onto the observed scene. This mean that for surfaces that are at steep angle with respect to the camera viewing direction, the point density on farther parts will be sparser.

2. The point cloud quality on reflective surfaces deteriorates quickly when deviating from fronto-parallel setting. The structured light pattern is not reflected back to the sensor, and no depth information is obtained.

![Figure 2.4: The quality hit with surface normal deviating from camera direction](image)

These two properties needed to be investigated to learn how they affect our primitive fitting method. We also needed to ensure the validity of the fitting method, that is to learn what are the stable features of the detected primitives.
Setup

![Figure 2.5: Setup illustration. A mounted Kinect camera was observing the surface while we have been varying the $\theta$, angle between surface normal and the camera direction.](image)

To test against the two mentioned issues three types of surfaces were investigated - the rough surface (cartoon box), the glossy computer screen and the reflective laptop screen. Each object has been captured at a varying angle with respect to the viewing direction. Figure 2.6 shows the objects and extracted scans used for testing.

Each object has been rotated around itself, with Kinect at fixed position, ant the rotation step of $\theta = 10^\circ$. The angle has been measured between the detected surface normal and the camera viewing direction. Note that the point clouds has been manually filtered so only the object of interest appears in these point clouds.

Results

The features we have investigated was the size of detected primitive compared to size of actual object, both in terms of area and side lengths, and the angle between the surface normal and viewing direction. As already seen in figure 2.4 the quality of the reflective screen has proven to be so poor, that no actual analysis was performed. Even a slight rotation ($\theta \approx 10^\circ$) causes a huge data loss, causing no clipped plane primitive to be detected.
Figure 2.6: Three types of surfaces used for experiment

Figure 3.9 presents the experiment results. First of all, these show how the number of inliers of the clipped plane primitive changes with the varying angle. As expected the rough surface capture provided better quality point cloud data and it was possible to detect primitive even when $\theta$ approached $90^\circ$. Conversely we can see that for matte monitor, already for angles higher than $50^\circ$ clipped plane primitive was no longer detected. In the next graph we can observe how the measured angle between the surface normal and the viewing direction changes. The thin red line depicts the ideal behavior, while the thick one represents our measurements. It is clear that this feature is very stable, and led us to the conclusion that, if possible, we should make the relationships between clipped planes rely on the clipped plane normal. We will come back to this fact in the chapter 4, where we will formulate what exactly are these relationships.

In the next two graphs one can see that the size of the surface, both in terms of area and side lengths. The thin lines represent the actual measured size of the object, while the thick lines show the size of the detected primitive. It can be seen that these deviate largely from the actual values. This is partially due to the sensor noise around the edges of the object. The clipped plane of the primitive will encompass all points, as seen in figure 2.8, and due this noise the side lengths and area will be consistently larger than actual values.

The second problem have been however related to way of orientation of fitted clipped plane was determined – see section 2.3.2. Principal Component Analysis
(a) The features behavior for the card box

(b) The features behavior for the matte monitor

Figure 2.7: Experiments results
Figure 2.8: The sensor noise causes the irregularities on the edges of objects represented in point clouds that was used, does not guarantee minimum bounding rectangle, so there were cases when the orientation found by it was clearly wrong (fig. 2.9).

2.3.4 Modified Clipped Plane Fitting

To alleviate this issue another method for plane orientation finding has been used. Rotating Calipers algorithm, introduced by Godfried Toussaint [35] has many different variants and the one we used guarantees the minimum bounding rectangle. The algorithm is defined as follows (from [25]):

*The input is assumed to be a convex polygon* $P = \{p_1, p_2, ..., p_n\}$ *with vertices given in clockwise order.*

1. *Compute all four extreme points for the polygon, and call them* $p_i$, $p_k$, $p_j$, $p_l$.
2. *Construct four lines of support for P through all four points. These determine two sets of "calipers".*
3. *If one (or more) lines coincide with an edge, then compute the area of the rectangle determined by the four lines, and keep as minimum. Otherwise, consider the current minimum area to be infinite.*
4. *Rotate the lines clockwise until one of them coincides with an edge of its polygon.*
5. Compute the area of the new rectangle, and compare it to the current minimum area. Update the minimum if necessary, keeping track of the rectangle determining the minimum.

6. Repeat steps 4 and 5, until the lines have been rotated an angle greater than 90 degrees.

7. Output the minimum area enclosing rectangle.

Because two pairs of "calipers" determine an enclosing rectangle, this algorithm considers all possible rectangles that could have the minimum area. Furthermore, aside from the initialization, there are only as many steps in the algorithm’s main loop as there are vertices. Thus the algorithm has linear time complexity.

RANSAC procedure provides us with the coefficients \( \mathbf{P}_c = \{a, b, c, d\} \) of a plane that the set of points \( \mathcal{I} \) approximately lie on. Thus to find the proper locations of clipped plane points \( \mathbf{P} = \{p_1, p_2, p_3, p_4\} \) we first need to derive the affine transformation \( \mathbf{A} \) that will move the the set of inliers \( \mathcal{I} \) to lie on the \( xy \) plane. The translation part \( \mathbf{T} \) is simply the vector between the origin \( \mathcal{O} \) and the centroid \( \bar{I} \) of the point set \( \mathcal{I} \). The rotation matrix \( \mathbf{R} \) is found by aligning clipped plane normal \( \mathbf{\hat{n}} = [a, b, c] \) with z axis. Then the transformation \( \mathbf{A} \) is given by

\[
\mathbf{A} = \mathbf{RT} \tag{2.1}
\]

Now we are able to run the Rotating Calipers procedure on the point set \( \mathcal{I}' = \mathbf{A}\mathcal{I} \) to find the clipped plane \( \mathbf{P}' = \{p'_1, p'_2, p'_3, p'_4\} \). Then it is straight forward to bring the \( \mathbf{P}' \) back to the 3D space by applying the inverse transformation \( \mathbf{A}^{-1} \). Thus the plane \( \mathbf{P} \) is given by

\[
\mathbf{P} = \mathbf{A}^{-1}\mathbf{P}' \tag{2.2}
\]

With Rotating Calipers algorithm, the clipped plane now snaps to the point set’s features like corners. Figure 2.10 shows the results of repeated experiments, when using Rotating Calipers. It can be seen that the measured size is closer to the actual value, and that the function is overall much better behaved — the size is more consistent with varying \( \theta \). It needs to be noted that for the rough surface the sudden decrease in area is caused by the fact that for large \( \theta \) no depth data has been stored for farther part of the surface. Due to that the found clipped plane was obviously smaller than the actual object. Notice how only length of one side, while the other is stable.
2.3.5 Detection Procedure

Having formulated how a single primitive is created it is now necessary to explain how the clipped planes are found in the input point cloud $P$. The extension is obviously straightforward, where the RANSAC procedure as defined earlier in this chapter, is called in a recursive fashion. Each time new set of inliers $I_i$ is extracted from the input point cloud, and the Rotating Calipers is used to fit a clipped plane on top of it. With that the set of clipped planes $S$ is created.

Recursive RANSAC

The RANSAC procedure is called on an input point cloud $P$. As the result we will obtain plane coefficients and the set of inliers $I_i$. The $I_i$ inliers are then removed from the original point cloud, so that $P_i = P / I_i$. Then the RANSAC is called again on $P_i$. This process is repeated until the size of the $P_i$ is sufficiently small, that is below threshold $t$, that has been set to 5% of original point cloud size. The algorithm is as follows:

1. Initialize $s = |P|
2. Run RANSAC on the input point cloud $P$ to obtain set of inliers $I$ and the model coefficients $P_c$.
3. Fit primitive $P$ onto $I$. Add $P$ to the set $S$.
4. Update $P$ by subtracting $I$ from it:
   - $P \leftarrow P \setminus I$
5. If $|P| < ts$ stop, else go to 2.
(a) The features behavior for the card box

(b) The features behavior for the matte monitor

Figure 2.10: Experiments results using Rotating Callipers
Euclidean Distance Cluster Extraction

However it needs to be noted that there is a slight issue related to the above formulation. If we were to use the $I$ directly to fit the clipped plane $P$ we would obtain bad results (fig. 2.12). This is because when using $I$ directly it contains all the points $p_i$ from the input point cloud $P$ such that:

$$d(P, p_i) < t$$  \hspace{1cm} (2.3)

Figure 2.11: Once set of inliers $I$ is obtained it contains points from various parts of the scene, "cutting" through the whole point cloud $P$

This means $I$ contains points from various regions of the point cloud $P$. This behavior is best illustrated in the figure 2.11. To alleviate this issue we need to find the cluster $C_j \in I$ that is most representative of the underlying shape. The simple heuristic used was to determine the point clusters $C_j$ within the fit plane, so that $I = \{C_1, C_2, ..., C_n\}$, and then select the cluster $C_j$ that contains the highest number of points. Then we can remove only $C_j$ from point cloud $P$. With this new formulation:

- $P \leftarrow P \cup (I \setminus C_j)$
- $I \leftarrow C_j$

This way, only the points contributing to the region of interest are used for the clipped plane fitting. The cluster detection algorithm used for this stage is the Euclidean Cluster Extraction, where a set of points $C$ is considered a single cluster if and only if

$$\forall_{p_i \in C} \hspace{0.1cm} dist(p_k, p_l) < t_c; k \neq l$$  \hspace{1cm} (2.4)
The threshold $t_c$, minimum distance between any two points within a cluster $C$, might depend on the point cloud density, as well as, the amount of cluster present in the scene. As such it has been exposed to the user as a controllable parameter. The procedure is as follows (from [28]):

1. Create a Kd-tree representation for the input point cloud dataset $\mathcal{P}$
2. Set up an empty list of clusters $C$, and a queue of the points that need to be checked $Q$
3. Then for every point $p_i \in \mathcal{P}$, perform the following steps:
   - Add $p_i$ to the current queue $Q$
   - For every point $p_i \in \mathcal{Q}$ do:
     - Search for the set $\mathcal{P}_i^k$ of point neighbours of $p_i$ in a sphere with radius $r < t_c$
     - For every neighbour $p_i^k \in \mathcal{P}_i^k$, check if the point has already been processed, and if not add it to $Q$
   - When the list of all points in $Q$ has been processed, add $Q$ to the list of clusters $C$, and reset $Q$ to an empty list
4. The algorithm terminates when all points $p_i \in \mathcal{P}$ have been processed and are now part of the list of point clusters $C$

After using the cluster extraction extension, the primitive detection results are much better, as shown in the figure 2.13.
2.4 Summary

In this chapter we have explained how to go from the unstructured point cloud $\mathcal{P}$ to the set of primitives $\mathcal{S}$. We have explained what is the primitive type that we are using, and why it has been chosen. We have investigated the issues related to the point clouds obtained using Microsoft Kinect and how these affect the detection of our primitives. However the main issue that we did not talk about is the missing data. This is a big problem for our method and it was necessary to find a way to address this serious issue. Next chapter will go into detail description of the proposed solution.
3 Reasoning about missing data

3.1 Introduction

As mentioned in the previous chapter there are cases when the Kinect is unable to obtain any depth data. This is caused by the reflective surfaces, which bounce the structured light pattern off in almost perfectly specular way, so little or no information reaches back to the sensor. This means that even slight deviation from fronto-parallel plane (i.e. when we are not looking directly at the screen) will cause dramatic data loss. As the result no clipped plane primitive will be constructed. In our setting the type of surface that has such property is the laptop screen, which is a key piece information for the detection stage — without it our method will fail to detect the object. This chapter will describe how we have decided to deal with this issue.

In general this stage of our algorithm takes the set of clipped planes $S$ and point cloud $P$ as the input. The analysis to infer the possibility of missing data is performed. If enough evidence is found we will proceed and add a new clipped plane primitives to $S$, in a location that provides the best support for it. From a probability perspective this problem can be viewed as finding what is the conditional probability $P(M|S)$, where $M$ is the event of missing some data, and $S$ is the set of detected clipped planes $P_i$ and sets of inliers $I_i$. That means that we analyze the existing info for the evidence of missing parts of point cloud.

After this analysis is performed an augmented set of clipped planes $S'$ will be ready to be used as a scene descriptor for the comparison stage. Note that it is perfectly possible that there is actually no missing data. In such case $S' = S$, and this stage is skipped.
3.2 Related Work

In general missing data and occlusion are issues that are very hard to solve — while many authors acknowledge the fact that missing data impacts their results directly \cite{14}, there is not a lot of work that provides means to deal with these problem directly. One of the works that directly tries to deal with the missing depth is by Fouhey et al.\cite{8}. In this paper authors aim at providing a system that is able to efficiently fuse the depth and RGB data and use both for object recognition. They state that depth data needs to be used with caution, since in their experiments using features coming from RGB and corrupted depth data yields worse results than when not using depth at all. The paper mentions various techniques used for dealing with corrupted data on the range image level, like recursive median filter \cite{17}, or inpainting \cite{23}. These however perform well only for relatively small regions of missing data, i.e. ones that are much smaller than the objects captured. Once large parts of the range image are missing the results of inpainting are most likely incorrect. With that in mind they propose a system that performs depth filling, but they also associate a confidence measure with each inpainted pixel. Then system is able to decide whether at give pixel only the set of 2d features should be used, or we can use both RGB and depth.
3.3 Scene Segmentation Image Analysis

Kinect is essentially a light source, so even if no data has been obtained on the object of interest, like the laptop screen, this object has "casted a shadow" on the other objects, i.e. prevented the structured light pattern to reach other surfaces.

We wish to exploit this intuition and thus a Scene Segmentation Image is used.

Scene Segmentation Image is an image created by projecting the point cloud $\mathcal{P}$ segments back onto an image plane. During the primitive detection stage, every point $p_i$ in the input point cloud $\mathcal{P}$ has been assigned either to one of the $n$ point sets $\mathcal{I}_i$, $i \in [1, n]$ or to no point set at all. The points that have not been found to belong to any planar structure can be then grouped together into a new set $\mathcal{I}_\emptyset$. The Scene Segmentation Image is initialized to be black, and then all $n + 1$ point segments $\{\mathcal{I}_\emptyset, \mathcal{I}_1, \ldots, \mathcal{I}_n\}$ are projected back onto it, each given a separate colour. With that, the following colouring scheme is used in the Scene Segmentation Image: black pixels indicate the missing data, white pixels are projected points from $\mathcal{I}_\emptyset$, other colour pixels are projected points from respective sets $\{\mathcal{I}_1, \mathcal{I}_2, \ldots, \mathcal{I}_n\}$. Results of such projection can be seen in the figure 3.2.

![Figure 3.2: Segmented point clouds and relating Scene Segmentation Images](image-url)
3.3.1 Shadow Region Detection

As stated earlier all black regions in the Scene Segmentation Image indicate the missing data. Data might be lost solely due to the sensor noise, which can be observed on segment's edges in figure 3.2. However the data loss may also be caused by the object for which we did not obtain any depth data, but it has obstructed what is behind it. For a human observer it is obvious that the black region in figure 3.2b represents the laptop screen. And in fact that black region is caused by the shadow casted by the laptop screen, as it can be seen when investigating the respective point cloud.

The advantage of using the Scene Segmentation Image is then being able to determine what the shadow regions are, as well as providing us with the approximate silhouette of the object that has casted the shadow. We now need to describe how we analyze the Scene Segmentation Image to detect the shadow regions. As stated earlier all the coloured regions, and the white regions are created by projecting the points from sets \( \{I_0, I_1, I_2, \ldots, I_n\} \) onto the image plane. This means that to detect the shadow regions we need to investigate all the black pixels in the image. These will be grouped into the set of connected components \( B = \{b_1, b_2, \ldots, b_n\} \). Now we need to perform analysis of each region in \( B \) to determine whether its occurrence has been caused by not captured object "casting a shadow", or simply due to the sensor noise.

First pass of our analysis is to simply exclude all the small regions \( b \), that is all the connected components form \( B \) that have size in terms of number of pixels below certain threshold \( t_{size} \). This reasoning follows from the fact that for our setting the missing data that we wish to account for constitutes to large chunks of the Scene Segmentation Image, as seen in figure 3.2. Once the small regions have been removed from \( B \) we will investigate the remaining regions in terms of their shape — since the clipped plane after a projective transform should appear as a quadrilateral in the image, we expect the valid shadow regions \( b \) to be approximately quadrilaterals. To explain how this check is done we first need to introduce the quadrilateral fitting procedure (see section 3.3.2). Here it is sufficient to say that we have found that if a shadow region does not appear approximately as a quadrilateral, there is often enough data for our primitive detection stage to fit all needed clipped plane properly. To see this compare the shapes of the black regions in figure 3.3a and figure 3.3b.
The regions we are detecting for the purpose of this algorithms are clipped planes. Such primitives, once projected onto the image plane will appear as quadrilaterals. The projective transform does not preserve the angles, nor the parallel lines. Therefore the properties of clipped planes defined in section 2.3.1 will not be preserved. From this also follows the fact that, as already mentioned in the previous section, we expect the valid shadow regions \( b \) to be shaped approximately like a quadrilateral. With that said our aim is then to fit a quadrilaterals \( Q_i = \{q_1, q_2, q_3, q_4\} \) into each region in \( B \). Then such quadrilateral can be raised to 3D to form a clipped plane primitive that will represent the object for which we have not captured the depth data.

First we will define the procedure that we are using to fit a quadrilateral \( Q_i \) into each \( b_i \in B \). Afterwards we will explain how we decide whether it is valid to use it for new clipped primitive creation, and how the creation process is performed for valid quadrilaterals \( Q_i \).

**Figure 3.3**: Examples of scene segmentation results
Quadrilateral fitting

A convex quadrilateral needs to be fit into each valid shadow region \( b \in \mathcal{B} \). We used a simple procedure to perform this task. For each region \( b \in \mathcal{B} \) its boundary \( B^b_i \) is extracted. \( B^b_i \) is the set of pixels that will be used to find the spatial locations of the quadrilateral corners. The algorithm to find the maximum area quadrilateral is as follows:

1. Find points \( q_1, q_2 \) in \( B^b_i \) such that euclidean distance between the two is maximum.

   \[
   \max_{\{q_1, q_2\} \in B^b_i} \text{dist}(q_1, q_2)
   \]

2. Form line \( k \) that goes through points \( q_1, q_2 \)

3. Find point \( q_3 \in B^b_i \) that maximizes signed distance to line \( k \) and a point \( q_4 \) that minimizes the signed distance to line \( k \)

   \[
   \max_{\{q_3\} \in B^b_i} \text{signed_dist}(q_3, k) \\
   \min_{\{q_4\} \in B^b_i} \text{signed_dist}(q_4, k)
   \]

4. Points \( \{q_1, q_2, q_3, q_4\} \) are the corners of maximum area quadrilateral \( Q_i \).

Such procedure is performed for each region \( b \in \mathcal{B} \), and will output a set of \( |\mathcal{B}| \) quadrilaterals \( Q_i \). The quadrilateral \( Q_i \) created in this way can be used to determine valid shadow regions (section \[3.3.1\]). We simply analyze the region within a quadrilateral \( Q_i \) and want to test if there are any coloured regions inside it (figure \[3.4\]). If yes, this means that the depth data that was already used to create some clipped plane, and thus the quadrilateral \( Q_i \) along with the shadow region \( b_i \) should be discarded. The clipped plane is already detected and no additional work is required.

With the set of valid quadrilaterals it is possible to form a sets of rays that go through the corners of each quadrilateral. The relation of these sets of rays with existing primitives \( P_i \) can be analyzed to reason about the necessity of adding new clipped plane primitives.

New Clipped Plane Creation

As mentioned in section \[1.3\] we are dealing with an indoor office environments, thus for all the subsequent reasoning we assume that the detected shadow re-
regions \( b \) have not been caused by an object that floats in the air. We assume that all the object are placed within the scene physical providing support for each other — for example desk provides support for all the objects placed on it. Thus after forming the set of quadrilaterals \( Q \), it is clear that the object of interest must be within the frustum \( F \) formed by one set of rays \( T_q = \{t_1, t_2, t_3, t_4\} \) casted through quadrilateral \( Q \), corners \( \{q_1, q_2, q_3, q_4\} \). Moreover existing, detected set of primitives \( S = \{P_1, P_2, \ldots, P_n\} \) should provide some physical support for such object. For the case of laptops, which are represented by a pair of clipped
planes having the common edge relationship, the kind of support we are looking for is the common edge relationship. From this follows that we aim to find a clipped plane $P_i$ that lies sufficiently close or partially within the mentioned frustum $F$ and would provide a support for new clipped plane $P_{new}$.

![Figure 3.6: The ray-edge distance analysis for single clipped plane](image)

The problem is then constrained to finding a pair of rays $t_n, t_m \in \mathcal{T}_q; n \neq m$, that is sufficiently close to an edge $e_s$ of currently considered clipped plane $P_i$. A schematic of such analysis can be seen in figure 3.6. This reasoning is performed for each clipped plane in $S$ and each ray set $\mathcal{T}_q$. For each clipped plane $P_i$, a pair of rays and an edge that will minimize the mutual distance is found. The error metric can be expressed as

$$
\epsilon(e_s, t_n, t_m) = \min_{p_i, p_j \in e_s} (\text{dist}(p_i, t_n), \text{dist}(p_j, t_n)) + \\
\min_{p_i, p_j \in e_s} (\text{dist}(p_i, t_m), \text{dist}(p_j, t_m))
$$

(3.1)

The distance function $\text{dist}$ used here is just a simple unsigned point to line distance. The search space for each $P_i$ is then determined by the number of ray pairs and the number of edges. This can be further reduced by noticing that not all pairs of rays are valid - rays formed by opposite corners of detected quadrilateral $Q_i$ will not form an edge in the reconstructed clipped plane. By looking at a figure 3.6 it is clear that diagonal pairs of rays $\{t_1, t_3\}$ and $\{t_2, t_4\}$ are invalid. Therefore, a $4 \times 4$ matrix that describes distance from each set of rays to each edge is sufficient. We then construct such matrix $D$ for every clipped plane(fig. 3.7).

Once the matrices $D_i$ are created for all clipped planes $P_i$, we wish to find minimum distance value among all of them. This will determine which clipped plane
Table 3.7: An edge-ray pair distance matrix $D$

<table>
<thead>
<tr>
<th></th>
<th>${k, l}$</th>
<th>${l, n}$</th>
<th>${n, m}$</th>
<th>${m, k}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>${p_1, p_2}$</td>
<td>$\epsilon_{11}$</td>
<td>$\epsilon_{12}$</td>
<td>$\epsilon_{13}$</td>
<td>$\epsilon_{14}$</td>
</tr>
<tr>
<td>${p_2, p_3}$</td>
<td>$\epsilon_{21}$</td>
<td>$\epsilon_{22}$</td>
<td>$\epsilon_{23}$</td>
<td>$\epsilon_{24}$</td>
</tr>
<tr>
<td>${p_3, p_4}$</td>
<td>$\epsilon_{31}$</td>
<td>$\epsilon_{32}$</td>
<td>$\epsilon_{33}$</td>
<td>$\epsilon_{34}$</td>
</tr>
<tr>
<td>${p_4, p_1}$</td>
<td>$\epsilon_{41}$</td>
<td>$\epsilon_{42}$</td>
<td>$\epsilon_{43}$</td>
<td>$\epsilon_{44}$</td>
</tr>
</tbody>
</table>

has been selected, and which edge will provide the mentioned support in form of \textit{common_edge} relationship. Note that if the minimum is below specified threshold $t_{\text{dist}}$ no plane will be fit. Large minimum distance is equivalent of saying that no part of detected geometry is able to provide support to a structure causing such shadow region to appear. Thus no primitive will be added to already detected geometry. Having selected the clipped plane $P_i$ and its edge $e_i$ it is now fairly straightforward to add new clipped plane. Again by exploiting the fact that we are detecting laptops we assume that the missed primitive will be the same size as the plane $P_i$. Then to create new plane $P_{\text{new}}$ we can simply copy $P_i$. Now the only thing left to find is the orientation of the new clipped plane $P_{\text{new}}$ with respect to $P_i$.

Since it also have been found which pair of rays from $T_q$ is the closest to the selected edge $e_s$, then it is clear that to properly orient the plane $P_{\text{new}}$ we need to find $\theta$ that minimizes the distance between the complement pair of rays from $T_q$ and the edge $e_o$, that is opposite to $e_s$. To use an example, looking at the figure 3.6 assume that the selected edge was $e_3$ and selected pair of rays was $\{t_1, t_2\}$, the distance we wish to minimize is between edge $e_1$ and pair of rays $\{t_3, t_4\}$. To find the $\theta$ we will use simple iterative procedure that at each will rotate $P_{\text{new}}$ by $\Delta\theta = 1^\circ$. Each iteration the distance will be calculated using equation 3.1 and if the new value is less then minimum it will be stored along with the current $\theta$. The iteration stops when $\theta = 180^\circ$. This way we obtain the $\theta \in [0, \pi]$ that minimizes the distance. An example plot of how the function behaves for different values of $\theta$ for one of the tested cases can be seen in figure 3.8.

This procedure is then repeated for each $T_q$ and new clipped planes primitives are added into the scene creating augmented set of primitives $S'$. Some of the results can be seen in the figure 3.9. These figures also show the created frustums in each of the cases.

31
3.4 Summary

In this section we have focused on a very specific problem, that has been massively hindering our technique. With the reasoning about the missing data we can reconstruct scene fully, even if the Kinect failed to capture some of the data due to the reflective surface properties. Note however that the ongoing development of the range camera technology might solve this issue on the hardware level. In case of such event our method will benefit directly, and there will be no more need for analysis described in this chapter. That is why some of the techniques used were not very involved in any way, since we do expect the quality of data obtained by range cameras to improve in the future.

With the augmented set of clipped plane primitives $S'$ we are ready to create a graph representation of the scene and perform the graph comparison that will lead to the object detection, as described in the next chapter.
Figure 3.9: Some results, new clipped planes added to the existing scene representation
4 Graph-Based Object Detection

4.1 Introduction

In the previous chapters we have introduced a way of creating primitive based representation of a point cloud $\mathcal{P}$ using a set $\mathcal{S}'$ of clipped plane primitives. We will now describe a way of representing the set of clipped planes $\mathcal{S}' = \{P_1, \ldots, P_n\}$ as a graph $\mathcal{G} = \{\mathcal{V}, \mathcal{E}\}$. Each of the $P_i$ will be represented as a node $n_i \in \mathcal{V}$, while the connectivity will be defined by investigating primitive-to-primitive relationships. Section 2.3.3 introduced the notion of the clipped plane primitive features. These features have been used to determine the set of relationships that can be detected, and thus types of the edges in $\mathcal{G}$. Such graph representation is then used to perform object detection by comparison with the reference graph $\mathcal{R}$.

Graph $\mathcal{R}$ is a graph with labeled nodes, that represents the object that we wish to detect in the scene context, in our case the laptop standing on a desk. By determining node-to-node correspondence between $\mathcal{R}$ and $\mathcal{G}$ we hope to properly label the nodes in $\mathcal{G}$ that relate to the object of interest. As a result we are able to provide meaning to the primitives in $\mathcal{S}$ and detect the object of interest.

To sum up this algorithm stage takes the set of primitives $\mathcal{S}'$ and the reference graph $\mathcal{R}$ as an input and uses it to generate graph representation $\mathcal{G}$. $\mathcal{G}$ is then compared with $\mathcal{R}$ to create node-to-node correspondence, that will determine the meaning of primitives in $\mathcal{S}'$. We will discuss the different ways this comparison can be performed and in the next chapter we shall assess how the different techniques influence the detection rate.
4.2 Related work

There is a tremendous amount of papers considering problem of object detection, here we will bring up just a few of them. Lai and Fox [18] introduced the Instance Distance Learning - the distance function is learned per object instance, where the weighted average distance from novel view \( x \) to a set of known views of the same object instance \( Y \) is used. This is strictly machine based approach, offering very good recognition rates, however it only considers object instances on a turntable not in a whole cluttered scenes. Works by Redondo-Cabrera et al. [26], Fehr et al. [5], Frome et al. [9] are all examples of similar approach, where the aim is to design a good local feature descriptor and then learn a classifier to recognize the objects. Redondo-Cabrera et al. extends the SURF descriptor[11] to 3D and by quantizing the local descriptors generates Bag-Of-Words representation. Such representation is then used to learn the classifier using Support Vector Machines. Fehr et al. introduces extension of the Shape Context to 3D and also modification of it, called Harmonic Shape context, that is obtained by applying spherical harmonic transformation to 3D Shape Context bins. Fehr et al. also introduces the new kind of local feature descriptor, the covariance descriptor, which are covariance matrices of features computed at each point. These features capture simple geometrical relations between neighbours, like the angle between point normals, euclidean distance from the projection onto tangent plane, the euclidean distance in 3D etc.

All these techniques however are used either on clean, uncluttered scenes, or objects sitting on turntables. Interesting work that aims at detecting objects in cluttered scenes is by Hintertoisser et al. [12]. The paper describes a method for detecting objects using multi-modal approach that fuses the RGB and depth data. Once multi-modal (based on colour gradients and estimated surface normals) template of an object of interest is generated it can be robustly used for detection of said object. Their method offers extremely good detection rates, and has been extended in [11] to generate templates from 3D CAD models. A template match in their setting not only provides the result on the presence of object in the scene, but if an object is detected a 3D pose is automatically obtained, similarly to our technique.

Steder et al. [33] work also used template-based approach. A point cloud representation of an object that is to be recognized is acquired and a spin images are generated out of for matching against the range image of the scene. This method also offers very good recognition rates, however it requires user to first obtain a
Another interesting method is described in a work by Grabner et al. [10]. In Grabner's work a triangulation of input point cloud data is created and an sitting actor is ghosted throughout the scene to determine "good places" to sit. At each point a score is stored, which in return creates a map of sittable objects, that not surprisingly relates to chairs and sofas in the scene. His approach is much different from ours, still it shows that the degree of freedom provided by additional channels enables incorporating detection techniques that strongly deviate from the classical 2D approaches to recognition problem.

The graph matching to perform shape detection can be seen in Schnabel et al.[29]. The scene is also approximated by the set of primitives and the relationships between neighbouring primitives are investigated to crate a topology graph. This graph is to identify the redundant shapes in the point cloud by finding similar subgraphs in the topology graph. We will see that this is very similar to our approach to object detection where we wish to determine if the captured scene graph representation contains a subgraph that is similar to reference graph.

A graph representation of the scene has been also used in Fisher et al. [7]. This work uses 3D CAD models, and aims at generating and using the graph representation for database model search and for generating similar looking variations of the scene automatically. Interestingly they mention a list of spatial relationships that are used by humans for reasoning about the scene. In our case the set of relationships between primitive shapes has been chosen based on the clipped plane features analysis. However if our framework would be extended a set of the inter-primitive relationships mentioned in Fisher et al. paper should be used.

### 4.3 Graph Construction

As mentioned in the introduction to create graph $G$ first a node $n_i$ is created for each primitive in $S = P_1, ... P_n$. The connectivity is then obtained by detection of simple geometrical relationships between each pair of clipped planes. From our experiments on features of clipped plane primitive we came to realization that the set of relationships that we use should mainly depend on the clipped plane normal $\hat{n}_{P_i}$. We also have mentioned that to represent the object of interest,
the laptop, a common edge relationship needed to be used. With that the set of relationships types that we detect between each pair of nodes $n_i, n_j$ is:

- **Orthogonal** – given a pair of clipped planes $P_1, P_2$, if $\hat{n}_{P_1} \cdot \hat{n}_{P_2} = 0$, then the $P_1$ and $P_2$ are said to be orthogonal to each other, i.e. have orthogonal relationship with one another.

- **Parallel** – given a pair of clipped planes $P_1, P_2$, if $\hat{n}_{P_1} \cdot \hat{n}_{P_2} = 1$, then the $P_1$ and the $P_2$ are said to be parallel to each other, i.e. have parallel relationship with one another.

- **Coplanar** – given a pair of clipped planes $P_1, P_2$, given that they are parallel, define vector $\vec{m} = \vec{k} \times \vec{l}$, where vectors are defined by points of the clipped planes $P_1 = \{p_{11}, p_{12}, p_{13}, p_{14}\}$, $P_2 = \{p_{21}, p_{22}, p_{23}, p_{24}\}$. Now let $\vec{k} = p_{11} - p_{12}$ and $\vec{l} = p_{11} - p_{21}$. Then, by construction if $P_1$ and $P_2$ are coplanar, both $\vec{m} \cdot \hat{n}_{P_1}$ and $\vec{m} \cdot \hat{n}_{P_1}$ should be equal to 1.

- **Common edge** – given a pair of clipped planes $P_1, P_2$, if one of the edges of $P_1$ is collinear with one of the edges of $P_2$, and they are of equal lengths. For any pair of $P_1$ and $P_2$ if these conditions hold they are said to share the common edge.

After performing the detection of the relationships between all nodes $n_i, n_j$, the graph $G$ is generated. Figure [4.1] shows the set of primitives and the relating graph $G$ that we use to represent the scene.

![Figure 4.1: The graph representation of the scene. Edges types are colour coded](image)

It needs to be noted that above set of relationships allows for multiple edges of different types between a pair of nodes - for example nodes $n_i$ and $n_j$, $i \neq j$ can have orthogonal and common edge relationship at the same time. Most algorithms to perform graph comparison only allow an unique edge between each two nodes. With that we create a formulation that would allow only an unique
edge between any given \( n_i, n_j \), but one that would still allow for multiple relationship types. We have decided that each edge \( e = \{n_i, n_j\} \) will be represented as vector of length \( k \), where \( k \) is the number of defined relationships that can exist between any two primitives (in our case \( k = 4 \)). If multiple relationships are detected between \( n_i \) and \( n_j \), the representative vector needs to reflect that. The elements in the representative vector are boolean values that flag occurrence of given relationship between \( n_i \) and \( n_j \). For consistency each position in the vector needs to represent specific type of relationship. It is then assumed that:

1. Position 0 relates to parallel type
2. Position 1 relates to orthogonal type
3. Position 2 relates to common edge type
4. Position 3 relates to coplanar type

Figure 4.2 shows an example of node with edges represented as vectors.

\[
[0, 1, 0, 0]
\]

\[
[1, 0, 0, 0]
\]

**Figure 4.2:** A node \( n \) with two outgoing edges. The type of the edges are parallel (lower one) and orthogonal (upper one)

### 4.4 Object detection

The reference graph \( \mathcal{R} \) is the graph representation of the scene containing object of interest, that has been created either from a good quality point cloud, or synthetically. Each node in \( \mathcal{R} \) is labeled with a name of the object it represents. Figure 4.3 presents the \( \mathcal{R} \) that we have used for laptop detection in this project. Note that with more primitive types being detected, and more relationships defined, it should be perfectly possible to extend this method to various types of
objects, so that $\mathcal{R}$ can represent more complex scenes.

![Reference graph $\mathcal{R}$ with labeled nodes](image)

**Figure 4.3:** Reference graph $\mathcal{R}$ with labeled nodes

We have developed a simple graph matching procedure based on local node-to-node comparisons. With the way we pose the matching problem it is perfectly possible to use any other method for the graph comparison. To demonstrate that, we show the use of Probabilistic Graph Matching method [38] by Zass and Shashua. Both methods aim at providing a soft matching matrix $S$ between the two graphs, so even after a binary matching is obtained, an information of other matching possibilities is retained. First we shall introduce simple, first order comparison. It exploits the fact that in the presented formulation graphs are fairly small, and such straightforward formulation have been found sufficient. Afterwards we present a short description of other, more advanced method by Zass and Shashua [38].

### 4.4.1 First Order Graph Matching

Given a reference graph $\mathcal{R} = \{V_\mathcal{R}, E_\mathcal{R}\}$ and input graph $\mathcal{G} = \{V_\mathcal{G}, E_\mathcal{G}\}$ a first order comparison method will compare each node in $V_\mathcal{R}$ to each node in $V_\mathcal{G}$. The output of this stage is a right stochastic matrix $S$ that describes the probability of node-to-node matches. Size of $S$ is then $|\mathcal{R}| \times |\mathcal{G}| - \text{th}$ row will represent $n^{\text{th}}$ node in $\mathcal{R}$, and $m^{\text{th}}$ column will represent $m^{\text{th}}$ node in $\mathcal{G}$.

To compare two nodes first a node $n \in V_\mathcal{R}$ is considered. The representative
vectors of all outgoing edges will be concatenated into one vector $\vec{v}_n$. The information on the ordering of the edge types that has been concatenated into $\vec{v}_n$ needs to be stored. With that for each node $m$ in $G$ a vector of zeros $\vec{v}_m$ of the same length as $\vec{v}_n$ is created. If $m$ have any edges of the same type that $n$ posses, their representative will be used to fill $\vec{v}_m$, using the stored, type based ordering. If $n$ have an edge of mixed type, we follow the rule of selecting the edge from $m$ that will minimize the Hamming distance between the two. Ideally that would be an edge of the same mixed type, but if such are unavailable we will select any edge of $m$ is just partially of the same type. For example consider $n$ and $m$ as in figure 4.4.

![Figure 4.4: The node $n$ and $m$ that are being compared](image)

In such case $\vec{v}_n = [1, 0, 0, 0, 0, 1, 1, 0]$. This informs how $\vec{v}_m$ should be constructed. Notice that there is no outgoing edges of types parallel in $m$, so first 4 entries of $\vec{v}_m$ will stay all zeros. $m$ does have outgoing edge of type orthogonal, that can be compared with $n$’s mixed type edge - thus the for remaining 4 elements $\vec{e}_2$ are used. With that $\vec{v}_m = [0, 0, 0, 0, 0, 1, 0, 0]$. After such procedure comparison of node $n$ and $m$ is a straightforward process, where the norm of the representative vectors difference can be used as similarity measure:

$$d_{nm} = ||\vec{v}_n - \vec{v}_m||$$

(4.1)

With $d_{nm}$ calculated for every $n \in V_R$ and every $m \in V_G$, matrix $S'$ can be created, with entry $S'_{nm}$ given as

$$S'_{nm} = \exp(-d_{nm}/\sigma)$$

$\sigma$ is a parameter that controls how much we penalize the edge difference. More on the results of using various $\sigma$ values can be found in the chapter 5. Matrix $S'$ contains strictly positive entries, that are $\leq 1$. However to create a valid soft matching matrix $S$, a normalization of $S'$ needs to be performed. Using the well-known Sinkhorn algorithm [15] we can obtain a right stochastic matrix — a matrix with strictly nonnegative, real-valued entries, with rows summing up to
1. Sinkhorn’s theorem states that for any square matrix $M$ with strictly positive entries there exist diagonal matrices $D$ and $E$ such that

$$A = DME$$

is doubly stochastic. We violate the assumption on the square matrix, and thus our matrix $P$ after finding $D$ and $E$ is only right stochastic.

**Sinkhorn algorithm**

A formulation of the Sinkhorn algorithm can be find in technical report by Philip Knight [15]:

Sinkhorn algorithm is perhaps the simplest method for finding a doubly stochastic scaling of a nonnegative matrix $A$. It does this by generating a sequence of matrices whose rows and columns are normalised alternately. The algorithm can be thought of in terms of matrices

$$A_0 = A, A_1, A_2, \ldots$$

whose limit is the doubly stochastic matrix we are after, or in terms of pairs of diagonal matrices

$$(D_0, E_0), (D_1, E_1), (D_2, E_2), \ldots$$

whose limit gives the desired scaling of $A$. (...) To describe the algorithm more formally, we introduce the operator $D : \mathbb{R}^n \to \mathbb{R}^{n \times n}$, where $D(x) = \text{diag}(x)$. Starting with $D_0 = E_0 = I$, we let

$$r_k = D_{k-1}AE_{k-1}$$

where $e$ is a vector of ones, and $D_k = D(r_k)^{-1}$. Now let

$$c_k^T = e^T D_k AE_{k-1}$$

and $E_k = D(c_k)^{-1}$

The algorithm has been found to stabilize quickly in the right stochastic form for us, so as a rule of thumb the $|V_R| + 1$ iterations are used each time it is performed.

After creating the soft matching matrix $S$ we might be interested in the hard matching matrix $H$. $H$ is essentially a permutation matrix, that describes which node in $R$ corresponds to which node in $G$. To construct such matrix we assume
that we want to find $|V_R|$ matches, i.e. we wish to find a best possible match for each of the nodes in $R$. To do so simple iterative procedure is used. We start by creating matrix $S'$ that is exact copy of $S$, and a matrix $H$ with all entries initialized to zeros. At each iteration a maximum entry $\{r, c\}$ of $S'$ is found. Then $H(r, c) := 1$ and all entries in row $r$ and column $c$ in $S'$ are assigned to be $-\infty$. The iteration terminates when $|V_R|$ entries of $H$ are equal to one.

This simple formulation has been found to perform surprisingly good, which is a result of having limited number of primitives and edge types. We understand that from more complex scenes a more involved method might be necessary. The next section will provide description on how our framework can be posed to serve as an input to general graph matching algorithm.

### 4.4.2 Probabilistic Graph Matching

Detailed explanation of the algorithm can be found in Zass and Shashua paper [38], this section will only provide a general overview of their method. The algorithm described in "Probabilistic Graph and Hypergraph Matching" provides a way to obtain the probabilistic (soft) node-to-node matching given a function that is able to compare (hyper)edges of two graphs. Main contribution of the paper is an elegant algebraic connection between a matrix $S$, that describes edge-to-edge similarity, to matrix $X$, that is a desired soft node-to-node matching matrix. The connection described is

$$S = \otimes_d X$$

Which holds true under the assumption that the vertex-to-vertex matches are conditionally independent ($d$ is the hypergraph dimensionality):

$$S_{e,e'} = \Pr(m(e) = e'|G,G') = \prod_{i=1}^d \Pr(m(v_i) = v'_i|G,G')$$

$$= \prod_{i=1}^d X_{v_i,v'_i}$$

Then the method of retrieving $X$ is presented, the solution is provided in the context of relative entropy error measure - matrix $X^*$ that minimizes the relative entropy $D_{KL}(S||\otimes_d X)$ will be obtained. Thus to use this algorithm a function to describe edge-to-edge difference is necessary. As mentioned in previous section
two such functions are naturally available for edges in provided formulation (equation 4.1). Thus no further modification of original method is needed.

In the case of this method the hard matching matrix $H$ is also obtained using the Sinkhorn algorithm.

### 4.4.3 Laptop Detection

Once a hard matching matrix $H$ has been obtained, it provides a semantic labeling for the nodes in graph $G$ — if a node $n \in G$ is matched with a node $m \in \mathcal{R}$, then $n$ is given the label of $m$. We then perform a simple check to determine whether nodes $n_a, n_b \in G$, that are labeled as laptop keyboard and laptop screen, have a *common edge* relationship. If yes then we mark $n_a$ and $n_b$ as the detected laptop. If no, then either the matching from $H$ was false, or the scene does not contain a laptop.

![Successfully detected laptop is highlighted in colour](image1)

![No laptop present in the scene](image2)

### 4.5 Summary

This chapter concludes the description of our algorithm. We have shown how to generate graph-based scene descriptors that represent the scene approximated by the set of primitives $S'$. Once such graph $G$ is created we argue that object detection can be performed by obtaining node-to-node correspondence with known reference graph $\mathcal{R}$. We have introduced two methods that can be used to
establish such correspondence, the first order comparison and the probabilistic graph matching method by Zass and Shashua.

In the next chapter we will discuss how using different methods affects the detection rates of our method, we will provide example results and point out cases that are problematic for our method.
5 | Results

5.1 Introduction

In this chapter we will talk about the performance of our algorithm on both synthetic and real-world scenes. The convention used in this chapter is as follows — if in the specific example laptop has been successfully detected we will re-colour the clipped planes, so that subset representing the object is clearly highlighted in red. If however the laptop has not been detected the original colouring from detection stage is preserved, so it easier to see what have caused the detection failure.

To reiterate on the intuition used for the object detection — given a reference graph $R$ with labeled nodes, and the input graph $G$, that is the representation of the captured scene, we want to establish node-to-node correspondence between the two. Once such correspondence is obtained we are able to reason what are the objects represented by nodes of $G$. In following sections we will show the confusion matrices — it is necessary to stress that these present success in classifying all objects from the reference graph $R$, however detection of a laptop is deemed to be successful as mentioned in section 4.4.3 when the nodes labeled as laptop screen and laptop keyboard have a common edge relationship. This means that even if nodes representing desk and wall have been mismatched, the laptop could still be detected correctly. For confusion matrices we use following convention for each cell — the brighter the cell is, the more cases were found, where two respective object categories have been matched. The confusion matrices are provided for two values of $\sigma$, the variable used by the edge-distance function 4.1. The additional class, Other, seen in confusion matrices, describes all the cases where one of the nodes in reference graph $R$ has been matched with a node not representing any of the key objects in $R$.

We will first present the results on a synthetic data set and then move to more
challenging cases of the real-world data set. This is done to demonstrate that our method performs well provided a good quality of the point clouds and that the main challenge for us is not to establish the node-to-node correspondence, but to be able to robustly detect primitive in the scene. All the results will be presented using both methods described in chapter 4. We will discuss the differences between the results obtained by both methods and what is causing these differences to occur. All the data set cases contain the scenes with laptop in them, the results present percentage of cases when the laptop has been successfully detected.

5.2 Synthetic Scenes

First we wish to assess the performance of our algorithm on the perfect quality data, not suffering from the problems mentioned and tackled in chapter 3. The synthetic scenes have been created using Blender 3D software and a virtual scanner have been created to obtain the series of point-clouds representing the scene (fig. 5.1).

Tests performed (fig. 5.2, 5.3) on this data set returned perfect results, detecting the laptop from all generated views. It needs to be noted that in all the views tested the laptop was entirely visible. The aim of this experiment was not to test against issues like self-occlusion, but rather to show that once we are able to detect all the primitives, the subsequent graph comparison stage performs very well.

For the case of the synthetic data set, the confusion matrices show that for both methods the laptop primitives — laptop screen and laptop keyboard — were detected properly in all cases. There are however cases where the desk and wall have been mismatched. Figure 5.4 showcases a situation in which such mismatching may arise. Two nodes may have exactly the same relations with the rest of the scene causing such ambiguities to arise. As we will discuss in chapter 6 such problems should be address in the future work, by creating larger families of possible relationships. Even with such clear limitations, due to the fact that our graphs are relatively simple, this problem have not hindered our detection rates (table 5.1), however in the future a larger family of relationships should be used.

The difference between First Order Comparison and the Probabilistic Graph Matching is caused by the fact that the latter is looking at the graph in global
fashion, considering many matching options. Without strongly penalizing even small differences between the edge types the results provided by this method are more ambiguous. The First Order Comparison on the other hand is rather crude technique, that just tests all the local difference between the two nodes.
Figure 5.3: Confusion Matrices using Probabilistic Graph Matching. Varying $\sigma$ value visibly improves matching results

<table>
<thead>
<tr>
<th></th>
<th>$\sigma = 0.5$</th>
<th>$\sigma = 0.01$</th>
</tr>
</thead>
<tbody>
<tr>
<td>First Order Comparison</td>
<td>100%</td>
<td>100%</td>
</tr>
<tr>
<td>Probabilistic Graph Matching</td>
<td>80%</td>
<td>100%</td>
</tr>
</tbody>
</table>

Table 5.1: Detection results on the synthetic data set

Figure 5.4: On the left you can see reference graph $R$ and on the right the graph $G$. Notice that in the case like that for $G$ the node 0 have exactly same connectivity like the node 4. In such situation there is clear ambiguity that causes nodes to be mismatched

However for such small graphs like ours (average number of detected primitives is 7) this local approach is more certain, since the number of possible matches
is small. It however needs to be noted, that once we would have more complex
scenes and allowed for more primitives to be detected, the global methods like
Probabilistic Graph Matching would most likely perform much better than the
local formulation.

5.3 Real-world Scenes

The results in the previous section have shown that for a good quality point
cloud data our method performs very well. To further convey this point in this
section we will first present the results on a subset of our data set where laptops
were clearly visible, and the quality of the point clouds allowed for detection of
all representative primitives without the need of using techniques from chapter
3. After that we shall present results on the whole data set and will identify the
classes of cases that cause our method to fail.

5.3.1 Good Quality Scenes

![Figure 5.5: Laptops viewed from the front result in good quality point cloud data. Note that the colour information has not been used by our framework](image)

The subset of cases where the laptop is clearly visible is the set of captures where
we have been observing the laptop from the front (fig. 5.5). In such settings
the point cloud quality obtained by Kinect is very good and thus all needed
primitives are detected.

As expected, the confusion matrices (fig. 5.6, 5.7) show similar classification re-
sults as in the case of the synthetic data set. The detection rates also provide very
similar results (see table 5.2). This shows that the main issue for our method is the initial stage of detecting primitives. As we will see when testing the method on the entire data set, the cases where our method fails are mostly caused by the corruption of the point cloud data, and not by the improper matching during the graph comparison stage.

5.3.2 Entire Data Set

Here we will present the results obtained by our algorithm when using the entire data-set we have captured. This data set contains of three different scenes, where the laptop have been captured from various camera angles, various laptop configurations, as well as varying amount of scene clutter. The figure 5.8 shows some examples of used scenes.
\[
\sigma = 0.5 \quad \sigma = 0.01
\]

<table>
<thead>
<tr>
<th></th>
<th>( \sigma = 0.5 )</th>
<th>( \sigma = 0.01 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>First Order Comparison</td>
<td>100%</td>
<td>100%</td>
</tr>
<tr>
<td>Probabilistic Graph Matching</td>
<td>81.8%</td>
<td>100%</td>
</tr>
</tbody>
</table>

Table 5.2: Detection results on the good quality subset data set

The results (figures 5.9, 5.10 and table 5.3) immediately show a significant performance hit in terms of detection. The previous stages has led us to believe that the graph matching stage is performing well, and any fail cases are caused by the fact that there have been not enough data to reconstruct the laptop. In general we have identified six classes based on the captured point cloud quality (fig. 5.12). These classes varies from having nearly perfect quality, where
we are able to see the entire laptop, with hardly no data lost (fig. 5.12a), to cases where the data is corrupted either to the level where we still are able to detect the primitives using RANSAC (fig. 5.12b), or where we need to employ our missing data reasoning procedure (fig. 5.12c). The other type of cases are where the point cloud quality prevents the proper primitive detection. We may have the data corrupted to the point where our missing data reasoning fails (fig. 5.12d) or where the data quality is plausible, yet there are other factors like self-occlusion (fig. 5.12e) that prevent the primitives from being fit properly. We can also suffer from the situation where the clutter impacts the detection stage, and causes a bad primitive-based representation to be created (fig. 5.12d).

We have also gathered a set of captures that contain very high amounts of clutter, to investigate behavior for such extreme cases. We have found that for such, our reconstruction stage fails completely, and we are unable to create any meaningful representation of the scene (fig. 5.11). These captures have not been used in the previously presented tests, as for these the detection rate would be near zero. However such configurations like that are rather rare to see in actual indoor situations. We however still have to acknowledge that our method does not deal with heavy clutter well.
5.4 Summary

The tests have shown the classes of the scene configurations that have caused our method to perform badly. In the next chapter we will provide discussion on the ways that in our opinion should be plausible solutions to the problems identified and shown in the figure. The general conclusion however is that our method is conceptually a good approach to do object recognition, however more thought and care needs to be put into the scene representation and reconstruction to make our method more robust to self occlusion and very heavy clutter.
(a) Perfect case, good point cloud quality, we are able to detect all the necessary primitives

(b) Point cloud quality is worse, but there is still enough data to detect the primitives

(c) A lost of data lost, but our missing data reasoning is able to recover the missing primitives

(d) The point cloud is too severely corrupted to detect the primitives

(e) A substantial self occlusion prevents the primitives from being detected

(f) The scene clutter causes a bad primitive fit

Figure 5.12: Classes of scenes from our dataset showcasing cases where the features of the scene cause our method to succeed or fail.
6 | Future Work

Previous chapter has shown that the presented method performs well, and can be considered a valid way to perform object detection. As the range scanner technology matures, the performance of our method is sure to increase, as shown by the results performed on the synthetic and good quality data sets. However, the method presented in this report does suffer from some clear limitations. Even with a perfect quality depth data the problems of self-occlusion and heavy occlusion still exist. Presented method does not perform well under the heavy clutter, since the primitive detection stage is not able to recover all the needed data, nor it is able to detect a laptop when it is observed from behind.

In the future the main focus should be designing a better primitive detection procedure, that will be able to detect primitives more robustly, and in heavy clutter. RANSAC procedure can be biased by the scene clutter, often creating a non-optimal fits. Some over segmentation techniques, such as the ones used by \[22\] or \[30\] could be used to improve the primitive fitting accuracy, especially under the heavy clutter.

An obvious extension would be making use of the RGB data. Since our algorithm only uses the depth data, it is possible to setup a "trick" object that will have the same representation in the depth data as laptops — for example two books placed in the right way could result in very similar depth information. Clearly, introducing the RGB information could be then used to verify, if the detected laptop is really a laptop, or just a "trick" case. RGB information could also be used to drive the detection in the cases when we need to deal with lost depth data, just as in \[?, FouheyCHS12\] Such extensions have been considered as a part of this project, however in the interest of time the needed to be abandoned.

We have also found that using just four relationships to describe how two primitives correspond is sometimes not enough to represent the objects uniquely. In
heavy clutter the detected planes can cause a re-occurring pattern of exactly same relationships, that will cause ambiguities during comparison stage. Such cases were causing some graph mismatching, as discussed in section 5.2. To alleviate this problem, a larger family of relationships should be designed. A simple example of a new type of relationship would be to have a relationship that would determine whether two clipped plane primitives are parallel and close to each other - this would only capture the relation between the desk and the laptop keyboard, leaving out floor and desk (even though the two are parallel).

Despite all the mentioned limitations, even a relatively simple formulation as the one presented in this report is offering good detection rates. This proves that the graph based recognition method holds promise, and in the future additional work to overcome the limitations of the current framework will be needed to improve the method and possibly extend it to perform recognition of objects of other types.
Appendices
This is the report submitted as a fulfillment of my summer 2012 internship, that included a preliminary work that has been continued in this project.
Project Report -
Laptop recognition using point cloud data
Maciej Halber

Abstract
This report outlines the work carried out during a student internship during summer 2012. The project goal was to explore robust ways of object detection and recognition techniques using 3d point cloud data, collected using range cameras. Processing point cloud data can be challenging, due to occlusion issues and poor capture quality. In particular we considered the problem of detecting laptops in office-type environments. We developed graph-based recognition technique which decomposes the scene into set of clipped planes and relationships between them. We describe the algorithm and present test results. We summarize with conclusions and possible future work.

1 Introduction

The graph-based recognition technique that has been investigated during the course of this project is designed to be able to detect objects by looking at simple geometrical relations between the objects in simplified scene. The point cloud input is abstracted as a set of clipped planes and the relations between these planes are detected. However the main problems this method is suffering from are the occlusion-related problems and overall poor quality of the captured cloud (Fig.1). The revised problem was then to be able to, having two different captures, reliably match objects present in the simplified versions of the two. With such matches we then will be able to hypothesize about presence of specific objects in the scene, even if they were not captured by the range camera, or partially occluded by different objects.

2 Algorithm

The algorithm consist of four main stages: The point cloud data segmentation, inter-plane relation detection, graph creation followed by the graph comparison (Fig.2). The scene augmentation and object recognition stage are yet to be investigated (see section 4).

![Figure 1: The aim of our method is to identify if a laptop is present in the scene. However, the appearance of the captured point cloud data may vary considerably. In this example, the return of points from the laptop screen varies significantly with the angle of the screen with respect to the scanner. The only way a match can be successfully made is if the system can deduce that there is a screen which was not detected in the scan.](image)
2.1 Scene segmentation

The first stage of the algorithm takes the point cloud data as an input and generates the set of clipped planes, that represent main planar areas in the captured data set. The raw data from range camera is often downsampled for performance reasons, the filter used to decrease the number of points is pcl::VoxelGrid(pcl::VoxelGrid docs).

To extract the set of clipped planes, the following recursive method is used:

1. Perform RANSAC with user specified threshold $rt$ on initial point cloud $C$ to determine get the largest subset $SC$ of $C$ with points that lie on the same plane.

2. Find largest cluster $L$ of points, where cluster is set of points $\in SC$ in which the distance between two neighboring points is not more than user-specified threshold $ct$.

3. Apply RANSAC again on $L$ with $rt/2$ to refine results. This prevents algorithm from finding points lying on the same plane, but belonging to different objects, i.e. preventing method from finding unfavorable planes that cross through multiple objects.

4. Extract the cluster $P$ from $C$. Extraction is performed by taking set of points that are within the distance $rt/2$ from plane with coefficients returned by step 3. The largest cluster in this set will become $P$. The $C$ is now split into $P$ and $R = C - P$.

5. Clipped plane $CP$ is placed around $P$. $CP$ orientation is determined by vectors outputted by PCA performed on $P$.

The plane fitting routine is performed on the cloud data until $R$ passed to recursive call of the function is less than 20% of original point cloud size. Also if the number of points in $P$ is less that 3% of original point cloud size $CP$ is disregarded. Example of scene segmentation is shown in (Fig.3).

2.2 Relation Detection

Given the set of clipped planes finding the relations between them is straightforward. Algorithm is finding following types of relations:

- orthogonal
- parallel
- coplanar
- overlapping
- common edge

First three relations are self-explanatory. Overlapping relation is more specific case of coplanar relation, where two clipped planes

![Flowchart of consecutive algorithm stages](image)

Figure 2: Flowchart of consecutive algorithm stages

geometrical relations between these clipped planes are then found and used to construct a graph $G$, where each clipped plane is a node, and the geometric relationship between two clipped planes is represented by an edge of different types. After the graph creation stage, the algorithm is able to compare this newly generated graph $G$ against some reference graph $R$ to create a minimal common subgraph $S$. $S$ then can be used to reason whether we can hypothesize if laptop is present in the scene or not. Reference graph $R$ is simply a graph generated from good quality capture of certain scene.
need to be coplanar and overlapping, for this relation to be detected. The common edge is detected if two clipped planes $P_1 \land P_2$ have a common edge of similar length.

This relation set represents set of very simple geometrical relations (orthogonal, parallel, coplanar) and a some relations that are representative for our case of laptop recognition (common edge, overlapping). First three enable us to determine relative orientation of each clipped plane and thus differentiate between clipped planes found in the scene. The common edge relation is a very specific case that should be present for all laptops (edge between the lid and laptop keyboard). It enables to differentiate between the clipped plane that represent desk and the one that represent laptop keyboard, and serves as the hook for scene augmentation - if the common edge relation is absent in the graph we might have missed the laptop lid for occlusion reasons. The overlapping relation is intended to take the plane segmentation into account. In various scenes the desk plane might have been segmented due to occlusion (Fig. 4). Overlapping help us to reason whether these two clipped planes represent same object or not. In the current version of algorithm if two clipped planes share the overlapping relation, then the are assumed to represent same object. This part of our method requires further investigation (see section 4).

**Figure 3:** Scene with clipped planes found and placed on top of point cloud data. Each node in the graph represents different clipped plane. The colour scheme has been introduced to easily distinguish between clipped planes.

The output of this stage is the graph $\mathcal{G} = (\mathbf{V}_G, \mathbf{E}_G)$, where $\mathbf{V}_G$ is the set nodes, where each node relates to some clipped plane in the scene and $\mathbf{E}_G$ is set of edges, where edge $e_{G_i}$ is a relation of type $i$ detected between two clipped planes.

**Figure 4:** The desk plane segmented into two parts - red and yellow rectangles

### 2.3 Graph comparison

Due to a partial occlusion or poor overall capture quality, some object in the scene might not have good representation in the input point clouds, as seen in the (Fig. 1). Then the clipped plane that would represent this object would not be found. In return our method would clearly be unable to do any recognition. We then require means to account for such possibility. We would like to be able to compare any graph $\mathcal{G}$ generated from corrupted or noisy scene captures to some reference graph $\mathcal{R}$, that has been generated from complete capture containing all important objects (in our case by important objects we mean laptops on desks).

We propose a method of node-wise graph comparison, that enables us to create a set of corresponding nodes and then insert hypothetical nodes that will represent the clipped planes that were missing due to poor capture quality. Given $\mathcal{R}$ and $\mathcal{G}$, each node $n \in \mathbf{V}_R$ is assigned a score $s$ that describes its likelihood to correspond to a node $m \in \mathbf{V}_G$. $N \times M$ matrix is created to hold the scores, where $N$ is the number of nodes in $\mathbf{V}_R$ and $M$ is number of nodes in $\mathbf{V}_G$. Set of corresponding nodes is a set of triples...
\((n, m, s)\), where \(n \in V_R\), \(m \in V_G\) and \(s\) is the score that describe the likelihood that \(n\) and \(m\) correspond to each other.

The presence of such triple means that \(n\) and \(m\) might represent the same kind of object in both respective scenes. For example let node 0 be a clipped plane that represents desk object in the scene described by graph \(R\), let 1 be a plane that represents desk object in the scene described by graph \(G\) and let \(s\) be sufficiently high for \((n, m)\) to be selected match. Then the graph comparison stage will return a set of triples \(S\) that contains triple \((n, m, s)\).

The comparison routine has four stages: node grouping, initial score gather, hypothetical node detection and final score gather. Each stage is described in detail below:

1. **Node grouping** - this stage generates preliminary groups of nodes in \(R\) and \(G\) that are being compared. These node groups, \(NG_R\) and \(NG_G\), contain nodes that are more likely to represent to the same objects in their respective scenes, than the nodes outside these groups, i.e. the score \(s\) in triple \((n, m, s)\), where \(n \in NG_R\) and \(m \in NG_G\), will be boosted. \(NG_R\) and \(NG_G\) are created using some external knowledge about the scene, before looking at the graphs themselves. This project has been focusing on the laptop object recognition, so node grouping stage is looking for such group of nodes \(NG\), that \(\forall n \in NG\) each node \(n\) relates to a clipped planes that either might represent a desk or a laptop keyboard.

2. **Initial score gather** - the initial score gathering will create the \(N \times M\) score matrix and will fill it according to simple rules:

\[
\forall n \in NG_R, m \in NG_G \quad S_{n,m} \leftarrow S_{n,m} + 2;
\]

\[
\forall n \in (V_R - NG_R), m \in (V_G - NG_G) ((\exists n \in e_{Ri} \land \exists m \in e_G) \Rightarrow S_{n,m} \leftarrow S_{n,m} + 1)
\]

Where \(S_{n,m}\) is a score stored in the cell \((n, m)\), \(e_{Li}\) is and edge in some graph \(L\) of type \(i\). The possible types are orthogonal, parallel, coplanar, common edge and overlapping as described in section 2.2. \(\exists n \in e_{Ri}\) means that there exist node \(n\) that is a one of the two nodes that create edge \(e\) of type \(i\) in graph \(R\).

3. **Hypothetical node insertion** - during this stage the presence of hypothetical planes is determined. When looking at nodes \(\in NG\) there is number of cases to consider (we assume that \(|NG_R| \neq 0\):

- \(|NG_G| = 0\) - the first stage failed to create a group of nodes that could be matched to nodes in \(NG_R\). In our case this means that node grouping stage has failed to find any nodes that could represent the laptop keyboard. Then we try to insert the hypothetical node \(h\) that would represent keyboard. For such node to be inserted it has to hold these condition

\[
\exists n_a \in V_G \Rightarrow (h, n_a) = e_{orthogonal} \\
\land \exists n_b \in V_G \Rightarrow (h, n_b) = e_{common, edge} \\
\land \exists n_c \in V_G \Rightarrow (h, n_c) = e_{parallel} \\
\land ||P_{n_a}, P_n|| < threshold \\
\land n_a \neq n_b \neq n_c
\]

,where \(P_n\) is a clipped plane associated with node \(n\).

- \(|NG_G| \neq 0\) - node grouping has been able to find some nodes that could be matched to nodes in \(NG_R\). In our case this means that nodes representing keyboard and desk planes has been found. We then assume that the smaller of the two, node \(n_{Gk}\) represents the laptop keyboard. We then compare the outgoing edges of \(n_{Gk}\) to the outgoing edges of \(n_{Rk}\). If there is some edge of type \(i\) that \(n_{Rk}\)
have while nGk does not, a hypothetical node h' is added to G, s.t. (h', nGk) = eGi

4. Final score gather - node-to-node comparison using graphs augmented with hypothetical nodes (if added during step 3) form is performed during this stage. The score matrix is enlarged, if needed, for the scores for hypothetical nodes, and the initial scores are incremented according to following rule:

\[
\forall n \in V_R, m \in V_G ((\exists n \in e_{Ri} \land \exists m \in e_{Gi}) \\
\Rightarrow S_{n,m} \leftarrow S_{n,m} + 1)
\]

After this stage we are presented with a N × M matrix with scores for each pair of nodes. Now in order to extract the set S of corresponding nodes we perform greedy per row extraction of all non-zero scores, and creating triples of corresponding nodes along with their scores, then we sort them in ascending order. The highest score pairs are extracted and the rest of the S is checked, i.e. after extracting a triple (i, j, s), where s = S_{i,j}, i is a node \in V_R, j is a node \in V_G. If there exists another triple with node j, but its score is s' < s then this triple is removed from S.

However, greedy extraction may not return most optimal match between two graphs, and contain various ambiguities (the triples (n, m, s) and (n, k, s), where m \neq k).

We introduced a second stage comparison that uses the up-vector to resolve these. The up-vector has been determined by taking the normal of such plane, that relate to the largest clipped plane. Then the normal of such plane becomes our up-vector.

Let’s assume clipped plane R_1 has been chosen from scene described by graph \mathcal{R} and plane G_0 from scene described by graph \mathcal{G}. This enables us to resolve two types of ambiguities (let D be a clipped plane represented by node d):

1. Plane orientation - using the information given by the up-vector we can compare the geometrical relationship of the clipped planes represented by the nodes in ambiguous triples, with D. The reasoning here is that a pair of nodes (n, m) cannot be a match, if the clipped planes they represent, N and M, have different geometrical relation with D. As an example, provided we have ambiguous triples (r_4, g_2, 3) and (r_4, g_3, 3), we can now check the orientation of relating clipped planes - R_4 to R_1 - assume that R_4 is orthogonal to R_1. Then we can check the geometrical relation between G_2 and G_0 and between G_3 and G_0. Again, assume that G_2 is parallel to G_0, while G_3 is orthogonal to G_0. Clearly, in such case the triple (r_4, g_3, 3) is the correct match, thus (r_4, g_2, 3) have to be discarded.

2. Plane location - with up-vector we can also perform z-sorting and remove ambiguities by looking at the clipped plane location in relation to D. Clearly a pair (n, m) can be a match iff both N and M are on the same side of D.

After this stage we are able to create a matching between \mathcal{R} and augmented graph \mathcal{G}'. (Fig. 5) shows the scene and associated graph \mathcal{G} before and after the comparison.

3 Results

During the course of the project the total of 106 captures of laptops has been made to test the algorithm. In the beginning, the capturing of different scenes has been performed loosely in order to obtain the data needed to test the implementation of early stages of the method. At the later stage of the project the capture of desk scene has been performed in a more controlled manner. The scene has been captured from different angles (front, 45° to left, 45° to right, camera placed lower with the look-at vector parallel to floor), with laptop in six different orientations(0°, 30°, 60°, 90°, 120°, 180° - where 0° is laptop facing camera directly), and finally with more and more clutter introduced into the scene. The scene with four
classes of different levels of clutter has been captured (fig. 6):

- no clutter (upper left corner) - simple scene with no objects besides the laptop present. The planes detection have no troubles with this kind of simple scene, the only problem is the lack of points captured on laptop lid, when capturing the scene when laptop is not facing the camera directly. This kind of problem can be seen in the lower row of figure 1.

- low clutter (upper right corner) - only few simple object, as well as the standalone monitor has been introduced to the scene. Our method is still able to find the clipped planes very effectively in this case. It of course also suffers from the problem described above, but no other problems with finding planes has been encountered.

- medium clutter (lower left corner) - more random objects has been introduced to the scene. Since some objects was directly adjacent to the laptop, the $ct$ value(See section 2) needed to be lowered to find proper clipped planes planes, that were not encompassing the points belonging to other objects than laptop lid.

- high clutter (lower right corner) - in this setting the amount of objects on the desk starts to occlude the desk. Our method is still able to fit the clipped plane, but it is only formed around the front part of the desk. Due to severe occlusion, in cases where laptop was rotated is unable to detect the laptop

The testing has shown that the scene segmentation stage is performing very well, even with heavily down-sampled scenes (down from ca. 0.25 million points to ca. 10000 points) it is still able to pick up all important planes present in the scene. However, the main problem encountered was the principal component analysis not returning the minimal bounding rectangle around the detected planar cluster of points. With un-

Figure 5: Top row is a reference graph $\mathcal{R}$, the middle row is $\mathcal{G}$ prior to the comparison, bottom one is $\mathcal{G}'$ - it is $\mathcal{G}$ outputted by the comparison stage. Note the laptop lid plane that has been fit in the scene. Matching nodes in both graphs have been drawn with the same colours to make the assessment of the results easy.
favorably oriented clipped plane, the comparison stage may fail to find the common edge relation (fig. 7). This, in turn, causes the algorithm to fail in finding proper matching - not properly aligned rectangle is ignored and our method introduces hypothetical node that represents properly aligned laptop lid. Currently there is no functionality to merge such hypothetical node with the one that is present in the scene, or to change the orientation of not properly aligned plane.

Our method is able to introduce the hypothetical node in 100% of cases, however the way the hypothetical clipped plane is inserted back in the scene is very naive, and currently works only in a few cases. The proper fitting of this hypothetical geometry is needed to carry out the tests whether the hypothetical node should stay in the graph, or if it was wrong to introduce it. Moreover the current implementation of inserting new planes into the scene may place the new node in the wrong location or with wrong orientation, which may cause the second stage of comparison (see section 2.3) to disregard proper node matches.

4 Future Work

Current version of the algorithm needs to be considered as a work-in-progress. The main shortcomings of current method is very naive hypothetical plane insertion that prevents it to work in a general case, and not ideal orientation of the clipped planes. Furthermore, the later stages of the algorithm that would perform the test whether the hypothetical node has been placed properly are not developed yet.

The work will be continued as 4th year Individual Project (M091), the algorithm require additional work to make the scene augmentation stage return proper results in general case. With the reconstructed scene we will be able to start working on actual object recognition stage and hopefully on expanding our method beyond recognition of single class of objects. Other things that need further investigation are the relation importance grading, introduction of additional relationship types and re-assessment of the ones currently used. The testing has shown that our method is very prone to any capture-quality issues, so additional effort will need to be put into solving any occlusion-related problems.
B | Project Plan
Final year project plan

Articulated object detection

Author:  
Maciej Halber

Supervisors:  
Dr. Simon Julier  
Dr. Niloy J. Mitra

1 Aim

To develop graph based object-detection algorithm capable of robust laptop recognition in indoor, office environments from RGBD images captured using Microsoft Kinect like device, that incorporates prior knowledge regarding these types of scenes. Project will build upon the work done during summer internship, and will investigate the use of RGB data in addition to the Depth data as well as use of probabilistic methods.

2 Objectives

1. Review current state-of-the-art techniques used for object recognition and scene segmentation using RGBD data.
2. Review the code state from as left after summer internship and make necessary extension for it to enable program to work with RGB point clouds.
3. Obtain a test set of captures that shall be used for algorithm verification.
4. Extension of current framework to be probability based.
5. Develop an appropriate machine vision components that shall make an effective use of RGB information

3 Deliverables

- A literature review that summarizes current state-of-the-art techniques in object recognition (as part of main report).
- The data set of RGBD point clouds.
- The algorithm evaluation strategy and results.
- The developed algorithm description and functional C++ implementation.
- The summer internship report
Interim Report
1 Progress to date

1. Literature review - papers regarding the advantages of combining the RGB and depth information, the approaches to object recognition in point cloud data and the approaches to graph matching problem.

2. Re-implementation of the system created during internship. The main reasons for this decision were:
   - problematic part of the old implementation was VTK-based data visualization part.
   - the need of using RGB point clouds, instead of depth-only.
   - decoupling of the capture and point cloud analysis functionality.
   - improvement of the source code quality.

3. Implementation of capture program that enables user to capture RGBD point clouds along with related 2D picture. User is able to capture the image and the point cloud simultaneously, filter the point cloud based on the depth data and save image and point cloud in .png and .pcd formats respectively.

4. Investigation of the relation between the actual angle $\theta_{\text{true}}$ (angle between the surface normal and the camera direction), and the features of the detected clipped plane $P$. The captures have been taken in a controlled environment, increasing the angle $\theta_{\text{true}}$ by 10 degrees at a time. The captures involved 3 types of planar surfaces - a card box side, a matte screen and a glossy screen.

The features detected $\vec{f}$ included number of points in $P$, the detected angle $\theta_{\text{det}}$, the detected plane area $A_p$ and the detected width and height $P_w, P_h$. It has been found that the angular information $\theta_{\text{det}}$ is quite robust and we are able to recover the angle of the surface, even if other detected features are not representative of the surface that has been captured.
The source-code for this project is available both on the CD included with this report, as well as from the online SVN repository

svn: //amy.cs.ucl.ac.uk/student_projects/2012/object_recognition/LaptopRecognition

Provided code depends on PCL and OpenCV libraries for the point cloud and image processing purposes. The 3D capabilities are implemented in OpenGL 4.0, and use GLM library to handle the graphics related mathematical operations in programmable pipeline. For the GUI it uses Qt framework.

The project has been compiled and tested on Windows 7, and thus also to run it on Windows GLEW library is needed to create OpenGL context.

The provided CD also contains a working subset of captured dataset, to test the method firsthand.


