Master Degree in Computer Engineering
Software Engineering Track

Politecnico di Torino

KTH
Royal Institute of Technology

Extracting contact surfaces from point-cloud data for autonomous placing of rigid objects

Supervisors

Prof. Alessandro Rizzo

Joshua A. Haustein, Ph. D.

Candidate

Carolina Bianchi

AY 2019–2020

Abstract

Nowadays, thousands of human workers engage daily in non-creative and physically demanding tasks such as *order-picking-and-placing*. This task consists of collecting different goods from warehouse shelves and placing them in a container to fulfill an order. The robotics research community has put much effort into investigating the automation of the picking problem for robotic manipulators. The placing problem, however, has received comparably less attention.

A robot tasked with placing a grasped object has to choose a suitable pose and motion to release the item inside a container, that may be partially filled with other goods. The aim of this thesis is to develop and evaluate a system that automates the placing of objects in a container, whose content is perceived through an RGB-D camera. To accomplish this goal, we develop a perception module that estimates the volume of the objects inside the container and extracts horizontal surfaces as potential supporting regions for the object from RGB-D data. We integrate this module with a state-of-the-art placement planner to compute placement poses for the grasped object, that are stable and reachable by the robot among the perceived obstacles.

We evaluate the system by manually reproducing the computed placements in different test scenarios. Our experiments confirm that with the developed pipeline it is possible to automatically compute feasible and stable placements poses for different objects in containers filled with various objects, perceived through an RGB-D camera.

Contents

Chapter 1

Introduction

One of the main changes that our economy, industry, and society is undergoing in the present years is the so-called fourth industrial revolution, or Industry 4.0. Although it is too soon to provide a formal definition of this process, it is clear that Information and Communication Technologies are the main drivers of this change [1]. A milestone in this process is the automation of warehouses, which constitutes a step towards the full implementation of smart factories. Smart factories leverage the latest innovations in the fields of communication, data science, and robotics to automatize and optimize processes that have historically been carried out by humans.

A central task in this context is *order-picking-and-placing*, which is the operation of collecting different goods from warehouse shelves and placing them in containers to fulfill an order [2]. This non-creative and physically demanding task is hard to automate, due to the inherent difficulties of developing a technology that is able to tackle multiple object shapes, potentially in collaboration with humans in a dynamic and unstructured environment.

E-commerce corporations like Amazon have a strong interest in warehouse automation and thus launched initiatives like the Amazon Picking Challenge [3] to encourage the development of new technologies to automate the pick-and-place process. This includes solving the problem of identifying a single item in an unstructured collection of similar items on warehouse shelves or on pallets, planning

and executing a collision-free motion to approach and grasp the object stably, and selecting a stable placement pose and a feasible motion to realize the placement.

The object identification and localization problem has received much attention by the computer vision research community [4]-[5]. Similarly, the robotics research community has extensively investigated the grasp and motion planning task, for which the developed methods range from analytical approaches to data-driven strategies [6]-[7] and, more recently, vision-based solutions [8]. In this work, we focus on the third problem, placing, which has received comparably less attention.

Placing consists of deciding where and how to place a grasped object. Here, a robot is presented with several challenges:

1. If the robot has no prior knowledge about the environment, it has to interpret noisy sensor data and reconstruct a model of its environment.

2. It has to find a suitable pose in the environment that affords a stable placement for the object. To this aim, it has to reason about the item and the physical characteristics of the environment.

3. Not all the possible placements are equally desirable. The robot has to rank the available placements according to a quality metric that encodes an objective that should be maximized, e.g., semantic preference, clearance from obstacles, or space optimization.

4. It has to compute a collision-free motion to reach the placement.

5. Finally, the robot needs to execute the planned motion and release, push, throw, or nudge the object to its target location.

In this thesis, we will focus on the perception and planning challenges (1-4) of this placement problem. The control of the execution (challenge 5) is not explicitly addressed in this thesis.

1.1 Problem Statement

We address the placement problem for scenarios as shown in Figure 1.1: a robotic manipulator is tasked to place a grasped rigid object o in a partially-filled roller container. To describe the placement problem, we adopt the notation of Haustein, Hang, Stork, *et al.* in [9].

Placement search space: Initially, the robot is at a pre-place configuration $q_o \in C$ that offers it a top-view over a target volume $V_t \subset \mathbf{R}^3$ and it has to place the object in this volume. The robot can sense a volume $V_s \subset V_t$ and it has to find a placement pose $\hat{x}_o$ for the object. The search space of the object placement position is thus limited to V_s. The robotic manipulator that is employed in this thesis can re-orient the object only around the world z-axis and therefore the orientation search space is $SO(2)$. The total search space for object placements is thus $V_s \times SO(2)$.

Motion reachability constraint: We focus on prehensile manipulation: the robot will hold the object, move it to the final pose and release it. Thus, the placement pose $\hat{x}_o$ has to be reachable by the robotic manipulator. Given the robot's configuration space C, there must be a continuous path $\tau(q_o, q_1)$ that connects the robot initial configuration $q_0 \in C$ to a final configuration $q_1 \in C$ that results in the object being at $\hat{x}_o$. Let the predicate $r(x_o)$ express this constraint: if a pose x_o is reachable, then $r(x_o)$ evaluates to 1, else to 0.

Collision-free constraint: During the execution of the motion, the object and the robot must not collide with the environment. If this condition is satisfied, we say the no-collision constraint $c(\tau)$ evaluates to 1, otherwise to 0. To evaluate this constraint, the robot has to interpret the sensor data to understand which part of V_s is occupied by rigid objects. For example, in the scenario in Figure 1.1 it has to infer the volume V_{obs} occupied by the cans, the box and the bag in the roller container, and by the roller container itself.

Stability constraint: Furthermore, the manipulator has to choose a target pose that affords a stable placement for the grasped item. For example, it needs to know that a placement across objects at very different heights (the two stacks of cans) is not stable. This additional constraint $s(x_o)$ evaluates to 1 if the placement is stable,

to 0 if not.

Placement objective: Lastly, the robot needs to choose among multiple placement possibilities. For this, we define with $\xi(x_o)$ a scalar metric that expresses a placement preference score that has to be maximized.

(a) A partially filled roller container with common goods: cans, a bag of pet food, and a box.

(b) A new object (water bottles package) is placed automatically in the roller container.

Figure 1.1: The robot needs to find a suitable placement for a package of water bottles in a semi-filled roller container. In the picture on the right, the resulting placement is realized by the robot.

With these definitions, the placement problem can then be formulated as a constrained-optimization problem:

$$\begin{aligned} \max_{x_o \in V_s \times SO(2), \tau(C_0, x_o)} \quad & \xi(x_o) \\ \text{subject to:} \quad & r(x_o) = 1 \\ & c(\tau) = 1 \\ & s(x_o) = 1 \end{aligned} \tag{1.1}$$

Most of the positions in the sensed volume V_s are not valid for a placement: they will be in mid-air, violating the stability constraint, or inside objects volume, violating the non-collision constraint. The goal of this thesis is therefore to develop a perception module that restricts the position search space V_s to positions that are more likely to satisfy the problem's hard constraints. This perception module is then integrated with a state-of-the-art placement planning algorithm [9] to solve the above optimization problem.

1.1.1 Scope and Limitations

In our target scenario, we operate a robotic manipulator, of which the full geometric and kinematic model is known. The manipulator can perceive the environment through an RGB-D camera, and the environment is rigid and static over the time frame needed to carry out the placement task. Initially, the robot is at a pre-place configuration, from which it has a view over the placement volume, and the robot is grasping the target object, which is rigid and has a known geometry. To evaluate the computed placements, we manually reproduce them in the real world scenario.

1.2 Outline

The remainder of this thesis is organized as follows. Chapter 2 presents related work relevant to challenges 1-3. The motion planning (challenge 4) is solved by the placement module in [9]. In Chapter 3, we detail the employed RGB-D sensor and the format of the data it produces and we explain the algorithms that are used to implement the perception module and its integration with the planning algorithm in [9]. In Chapter 4, we illustrate the results of the experiments conducted to assess

the system performance, and in Chapter 5 we analyze the results and the system limitations. Lastly, in Chapter 6 we summarize the work done in this thesis, and we suggest possible future developments.

Chapter 2

Related Work

In this chapter, the scientific literature relevant to the challenges 1-3 introduced in Chapter 1 is presented. Section 2.1 overviews the methods that are commonly employed for obstacle estimation in applications that require motion planning in unknown scenes. Section 2.2 illustrates the strategies that state-of-the-art placement algorithms employ to ensure a stable placement. Section 2.3 reviews the methods that have been developed to extract primitive shapes from point clouds. Section 2.4 lists some of the metrics that can be employed to evaluate the quality of a placement. Finally Section 2.5 illustrates the placement sampling strategies encountered in literature.

2.1 Obstacle Volume Estimation

Many robotics applications require having a tractable model of the environment. Mobile robots need it to plan their navigation dynamically without crashing into obstacles and similarly do robotic manipulators. In our application, we need a map to plan the motion of the robot so that it does not collide with the objects in the scene.

When the surroundings are not previously known, the robot has to perceive the environment through sensors. Commonly employed range-measuring sensors serving this purpose are: tilting laser range finders (LIDAR), ultrasonic distance sensors,

Time of Flight cameras, or RGB-D sensors.

Measurements taken by such sensors are affected by noise, therefore the map building procedure has to deal with uncertainty. Common ways to do this are to either employ a probabilistic framework or to apply obstacle inflation, i.e., to estimate the obstacle volume and add some padding to ensure extra safety.

The main trade-off when it comes to mapping is between memory and time efficiency and precision. In some cases, mapping can be reduced to a 2D problem, for example, if the robot motion is constrained to be on a planar surface or if it is in a structured environment.

Commonly employed data-structures in this context are occupancy grids [10]. Each cell in the grid holds the probability of being occupied, which is initialized to be 0.5 and can range from 0 to 1. When the sensor perceives an obstacle, the occupancy probability of the cells on the ray that connects it to the obstacle decreases, and the occupancy probability of the cell containing the obstacle increases. This simple model has been extended to elevation maps [11], which store in each cell the height of the perceived obstacle, or multi-level surface maps, which offer a multi-surface view of the environment [12].

The natural extension of two dimensional occupancy grids to the 3D case, is the use of occupancy voxel grids. This approach, however, scales poorly as it is very memory-demanding. To mitigate this issue, Hornung, Wurm, Bennewitz, *et al.* [13] proposed an octree-based data structure, that enables temporal measurement integration, handles unseen space, and is time and memory efficient. The data structure holds the map representation in an octree (see Figure 2.1), where each node in the tree represents a cube (voxel) in the space and can have zero or eight children. The node contains a value that expresses its voxel occupancy probability; children nodes are created only when necessary. The occupancy probabilities are updated similarly to the two dimensional case.

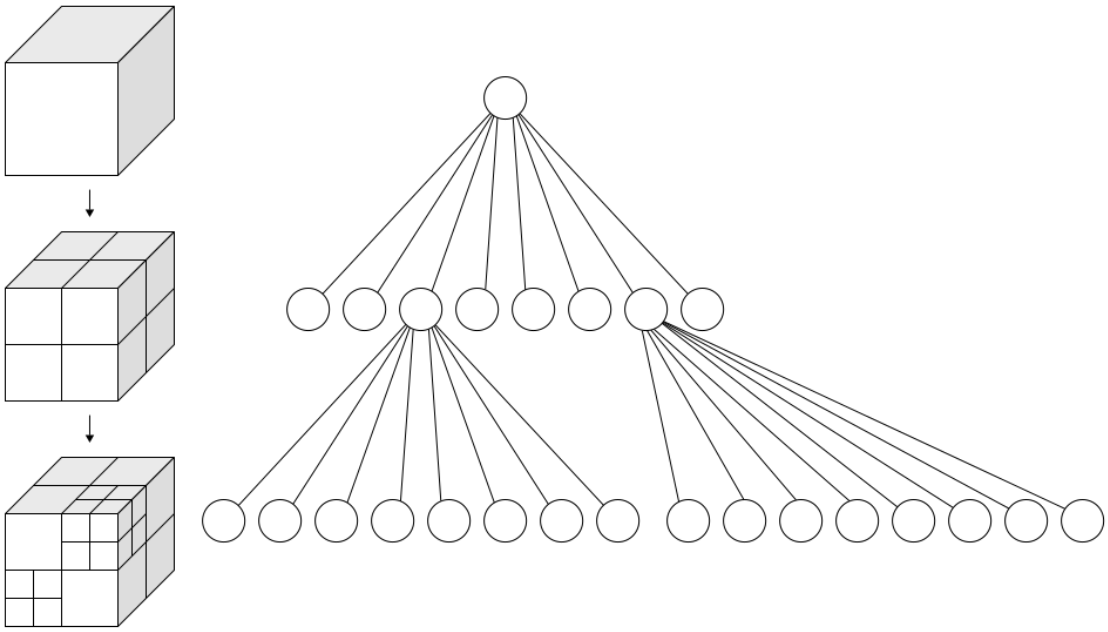


Figure 2.1: Octree data structure. Each node in the structure represents a three dimensional cube in the space. The root of the tree and each intermediate node in the tree have exactly eight children. Image from [14].

In this thesis, we will take advantage of the prior knowledge that we have about the placement scenario. The robot has a top-view on the target placement volume, therefore we use a reconstruction of the scene similar to the elevation map approach. The obstacle resolution depends on the local resolution of the point cloud. The developed method is illustrated in Section 3.2.

2.2 Placement Stability

In order to solve the placement problem, it is necessary to reason about the stability of a placement pose. A rigid body rests in equilibrium if the forces that act on it sum to zero and if they do not induce any momentum. The forces that we consider in this context are gravitation, support reaction forces and friction. Gravity acts from the object center of mass towards the ground. Reaction forces act at the points of contact between the object and the support to impede the motion of the object towards the support. Friction is exerted between the object and the support contact

surfaces in the direction that hinders a sliding motion of the object on the surface.

Baumgartl, Werner, Kaminsky, *et al.* in [15] developed a placement planner that bases its stability analysis by explicitly reasoning about friction. When computing a placement, the authors identify the set of contact points between the object and the supporting surface. They project the object center of mass along the gravity direction and verify that the projection lies within the contact points' convex hull, which ensures rotational stability. Furthermore, they assume a minimum friction coefficient μ, which defines the maximum allowed slope of the contact surface that guarantees a static equilibrium.

If it is not possible to make any assumption about the object or environment materials, the most general approach remains to consider only planar surfaces as candidate placement regions to ensure force equilibrium. In this case, the gravitational force acts perpendicularly to the placement surface that, given its rigid nature, will counteract with an equal and opposite normal force, regardless of the friction coefficient, thus ensuring force equilibrium.

Following this method, Harada, Tsuji, Nagata, *et al.* in [16] identify candidate placement areas by deriving polygon models of the object and the environment from point cloud data, and considering only horizontal surfaces. The authors cluster the object and the environment in approximately planar surface patches and base their stability evaluation on a convexity test between the contact surface of the object and the placement region. Haustein, Hang, Stork, *et al.* in [9], similarly, do not assume any friction coefficient and restrict placements to horizontal surfaces. An analogous method was employed by Schuster, Okerman, Nguyen, *et al.* in [17], who focus their research on the perception of flat surfaces on cluttered tabletops that can afford placements.

Jiang, Lim, Zheng, *et al.* in [18] develop a strategy to compute more complex placements such as hanging and caging. To this aim, they manually design several features to characterize a placement, which aim to capture the placement stability and preferred orientation. They then produce a training set of placement combinations by employing a rigid-body simulator, and train a Support Vector Machine to rank candidate placements automatically.

In this thesis, we build on the work of Haustein, Hang, Stork, *et al.* in [9] to compute stable and reachable placements among clutter, see Section 2.5. Therefore, also in this work we constrain placements to the horizontal case and thus develop a method to detect planar surfaces from noisy point clouds. As a drawback, complex placements (such as hanging or caging) will not be considered.

2.3 Plane Detection in Point Clouds

Range measuring sensors produce point clouds, i.e. sets of points in space. Several strategies can be found in the scientific literature to identify primitive shapes in a point cloud, and we focus on the ones which aim at isolating flat surfaces. Given a point cloud $P = \{p_i\}, p_i \in \mathbb{R}^3$, the goal is to find the regions $R_j \subset P$ that were generated by a planar surface in the scene.

We can divide the most popular state-of-the-art point cloud segmentation strategies into two classes: attribute-based methods and model-fitting methods. In the following, we give an overview of each of these strategies.

2.3.1 Attribute-based Methods

Region growing procedures segment point clouds in the spatial domain, a typical region-growing algorithm is summarized in Algorithm 1.

Key concepts in this approach are:

- **Neighborhood** N_q of a point p_q: a collection of points in P that satisfy some proximity condition to p_q (Algorithm 1 line 2).
- **Local features**: the set of geometric attributes that are computed to characterize each point p_i in the cloud.
- **Seed point** p_s: reference point from which the region is currently growing (Algorithm 1 lines 1 and 4). An alternative is using voxels, rather than single points, as working units, thus preserving only one measurement or the average measurement in each voxel [19], [20], [21].

Algorithm 1: Region growing algorithm

Data: $P = \{p_{1,\dots,N}\}$ point cloud
Result: $R_{1,..,n}$ set of horizontal regions
initialize $R = \emptyset$;
1 select an initial seed point p_s;
$i = 1$;
$R_i = \{p_s\}$;
$P_r = P \setminus p_s$ set of points that do not belong to any region;
while $P_r \neq \emptyset$ **do**
2 $N_s =$ `neighborhood` (P_r, p_s) ;
for $p_n \in N_s$ **do**
3 **if** p_n and p_s belong to the same region **then**
$R_i \leftarrow R_i \cup \{p_n\}$;
$P_r \leftarrow P_r \setminus \{p_n\}$;
end
end
if R_i can still grow **then**
pick p_s in R_i;
else
grow a new region: $i \leftarrow i + 1$;
4 choose $p_s \in P_r$;
$R_i \leftarrow \{p_s\}$;
end
end
Procedure `neighborhood`(P_r, p_s)
N_s is the set of points in P_r that are close to p_s;
return N_s;

- **Region growing criterion**: a condition that the point p_i has to satisfy in order to be joined to the region R_j, taking into consideration the source point p_s (Algorithm 1 line 3). This condition is evaluated by considering the two points' local features.

The choices regarding the definition of the local neighborhood of a point, its local features, the seed point and the region growing condition differentiate each algorithm and its performance. In the following, we present some of the strategies encountered in literature.

Neighborhood definition

Defining a suitable **neighborhood** is one of the most challenging aspects of the algorithm, as it influences the computed local features. Some options are:

- Define a fixed-size k neighborhood of p_q: N_q^k contains the closest k points to p_q. Closeness is evaluated in the sense of Euclidean distance [22].
- Define a fixed-radius ρ sphere around p_q: N_q^ρ is the set of points within this sphere. Fan, Wang, Geng, *et al.* in [23] propose a method to automatically determine ρ based on global consideration of the sparseness of the point cloud.
- Dynamically define the geometry of the neighborhood based on local considerations such as the currently estimated slope [24].
- Divide the point cloud into voxels and define each voxel as a neighborhood [20].
- TOF cameras and RGB-D cameras produce a point cloud organized in a matrix shape, like a standard picture. In this case it is then possible to use a fixed pixel-sized square neighborhood [19] to avoid the complex neighbors search or alternatively to adapt the size of the pixel-neighborhood to the local depth of each point [25].

Local Features

The **local features**, that are computed to characterize each point, include:

- The local surface normal n_q of point p_q.

  It can be computed by the Principal Component Analysis (PCA) as proposed in [26] and done in [22], [20], [27] and [21]. It corresponds to finding the normal of the least-squares-fit plane π_q of p_q's neighborhood N_q .

  Let C_q be the covariance matrix $C_q = \frac{1}{k}\sum_{i=1}^{K}(p_i - \bar{p})^T(p_i - \bar{p})$ of neighborhood N_q of the queried point. $\bar{p}$ is the average of p_i. Let λ_0, λ_1, λ_2 be the eigenvalues of C_q, which satisfy $0 \leq \lambda_0 \leq \lambda_1 \leq \lambda_2$, and let ν_i be the eigenvector that corresponds to λ_i for $i = 0, 1, 2$. Then:

$$n_q = \begin{Bmatrix} \nu_0 & \text{if } \nu_0 \cdot (\nu_p - p_q) > 0, \\ -\nu_0 & \text{if } \nu_0 \cdot (\nu_p - p_q) < 0 \end{Bmatrix} \tag{2.1}$$

  ν_p is the sensor pose when the point cloud was acquired.

  This method fails in case of surface discontinuities and noise. An alternative proposed by Fan, Wang, Geng, *et al.* is to use a constrained nonlinear least-squares algorithm that weights each point contribution proportionally to its *fit* to the plane π_q [23].

  Holzer, Rusu, Dixon, *et al.* focus on picture-shaped point clouds (TOF and RGB-D data). The authors adopt a square neighborhood definition $N_{p_{m,n}} = \begin{pmatrix} p_{m-r,n-r} & \cdots & p_{m-r,n+r} \\ \cdots & p_{m,n} & \cdots \\ p_{m+r,n-r} & \cdots & p_{m+r,n+r} \end{pmatrix}$. The authors adjust the size of the neighborhood r to the local point depth, based on the observation that farther measurements tend to be affected by heavier noise. Moreover, they implement an edge detection mechanism to avoid averaging depth data in the presence of depth discontinuities.

  A further time-improvement by Holz, Holzer, Rusu, *et al.* in [19] is based on the use of *integral images*, which allows averaging measurements over any rectangular-shaped pixel neighborhood in constant time. An integral image I of a source image $S = S_{i,j}$ of dimension $h \times w$ is defined in Equation 2.2:

$$I_{i,j} = \sum_{l=0}^{l=i}\sum_{m=0}^{m=j} S_{l,m} \text{ for } i = 1,..,\ h\ j = 1,..,\ w \tag{2.2}$$

Each element of the integral image can be computed iteratively as follows in Equation 2.3 and shown in Figure 2.2:

$$I_{i,j} = S_{i,j} + I_{i-1,j} + I_{i,j-1} - I_{i-1,j-1} \tag{2.3}$$

thus needing a single pass over the image for the full integral image retrieval.

	1	2	3	4	5
1	3	3	2	3	1
2	2	2	3	1	1
3	3	2	2	4	1
4	1	3	4	4	4
5	1	2	3	2	1

(a) The source image S.

	1	2	3	4	5
1	3	6	8	11	12
2	5	10	15	19	21
3	8	15	22	30	33
4	9	19	30	42	49
5	10	22	36	50	58

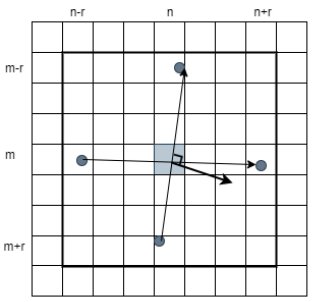
(b) The integral image I.

Figure 2.2: Toy example of an integral image. On the left we have the original image S, on the right its integral image I. As an example of Equation 2.3 $I_{4,4} = S_{4,4} + I_{3,4} + I_{4,3} - I_{3,3} = 4 + 30 + 30 - 22 = 42$.

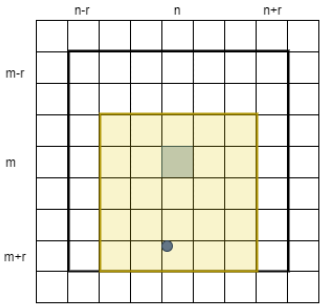
By considering the matrices containing the x, y and z coordinates of the points in the organized cloud as source images S^x, S^y, S^z, then it is possible to compute the sum of measurements (hence the average) over any square neighborhood of size r in the picture with only four memory accesses:

$$\sum_{i=m-r}^{m+r} \sum_{j=n-r}^{n+r} S_{i,j} = I_{m+r,n+r} - I_{m-r,n+r} - I_{m+r,n-r} + I_{m-r,n-r}. \tag{2.4}$$

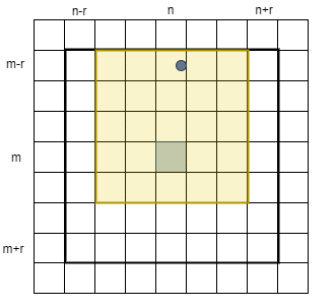
Given these premises, once the neighborhood size r in pixels of $p \in P$ at pixel position m, n is computed, the normal vector is retrieved as the cross product between the vectors connecting pixels $p_{m-r,n}$ and $p_{m+r,n}$ and $p_{m,n-r}$ and $p_{m,n+r}$, where the coordinate of each of the four points is smoothed over a $r-1$ side square within the neighborhood of $p_{m,n}$ as illustrated in Figure 2.3.



(a) Cross product of the vectors connecting the points at the horizontal and vertical boundaries of the neighborhood (thick line).



(b) In yellow the measurement smoothing region for the point at the bottom limit of the neighborhood.



(c) In yellow the measurement smoothing region for the point at the upper limit of the neighborhood.

Figure 2.3: Normal computation over an organized point cloud with local smoothing. r is the size of the neighborhood (in pixels).

- The local curvature, as introduced by Pauly, Gross, and Kobbelt in [28] and used by Wang, Zou, Shen, *et al.* in [22], is computed as:

$$\sigma_q = \frac{\lambda_0}{\lambda_0 + \lambda_1 + \lambda_2} \tag{2.5}$$

  λ_i are the eigenvalues of the neighborhood covariance matrix with the same order considered in the computation of the surface normal. Another method to assess the local curvature is to compute the distance of the point to the fitting plane [29], or by summing the squared residuals of the points in the neighborhood of p_q w.r.t. the computed fitting plane π_q [20], [27].

Seed Point Definition

The methods to choose a new **seed point**, include:

- Randomly pick a point in the cloud.
- Choose the point with the smallest curvature, when explicitly searching for planar clusters [22].

- Identify points which present high-density peaks [30] and are sufficiently distant one from the other, following the assumption that cluster centers are surrounded by neighbors with lower local density.

Strategies for the choice of subsequent seed points include:

- Keep the same seed point until it is impossible to aggregate more points to the current cluster.
- Choosing a new seed point in the region p_j if the angle between its local normal vector and the region normal vector does not exceed a threshold [22].

Region Growing Criterion

A similarity condition is needed to decide whether a point in the neighborhood of p_s belongs to the same region (Algorithm 1 line 2). Standard strategies are:

- Defining a maximum angular threshold θ_{max} between the normal of the seed point and the new point [20], [22], [23], [27], [31]. Given the seed point p_s and its normal ν_s, the neighboring point p_i is added to the region if its normal ν_i is such that:

$$cos^{-1}\left(\frac{\mid \nu_s \cdot \nu_i \mid}{\|\nu_s\|\|\nu_i\|}\right) < \theta_{max} \tag{2.6}$$

  θ_{max} is a parameter that limits the allowed local roughness, as it is evaluated between two close points.

- Using a threshold on the sum of residuals, that enforces smooth areas to be broken at edges [27].
- If the focus is on detecting horizontal clusters, testing the difference of the z coordinate of the seed point p_s and the neighboring point p_i [21].

Attribute-based methods are a robust approach to point cloud segmentation [32]. However, the definition of neighborhood in an unevenly sampled space can be difficult. Moreover the computation time can be unfeasible in case of multidimensional attribute spaces and highly dimensional input point clouds.

2.3.2 Model-based Methods

When models of the different parts composing the scene are known, it is possible to employ model-fitting methods to detect those shapes. The two standard ways to implement model-fitting are the Hough Transform (HT) and RANdom SAmpling Consensus (RANSAC) methods.

Hough Transform

Hough Transform was first introduced by Hough in [33] to detect patterns in binary images, and it was later successfully applied in 2D greyscale images to recognize lines and circles. This method is useful to fit parametric shapes to sampled data, and its application was extended to 3-D data as well.
To work with the Hough Transform for shape detection on the point cloud $P = \{p_i\} \in \mathbb{R}^3$, one has to define:

- A parametric shape $\Gamma \subset \mathbb{R}^3$ defined over a set of parameters $\alpha \in \mathbb{R}^N$.
- A relation $\Lambda : \mathbb{R}^3 \to 2^{\mathbb{R}^N}$ between each point and the parameters in $\mathbb{R}^N$ that describes the family of shapes to which the point could belong.
- A suitable discretization of the parameter space $\mathbb{R}^N$ and an N-dimensional data structure (*accumulator*) with a one-to-one correspondence to each cell of the discretized parameter space.

The most straightforward implementation of the Hough Transform for primitive shape detection follows Algorithm 2.

Algorithm 2: Hough transform algorithm

Data: $P = \{p_{1,\dots,N}\}$ point cloud; $\Gamma : \mathbb{R}^N \to \mathbb{R}^3$, $\Lambda : R^3 \to 2^{\mathbb{R}^N}$;
zero each entry of the accumulator;
for $p_i \in P$ **do**
    find the parametric shapes to which the point could belong $\Lambda(p_i)$;
    increment the corresponding cells in the accumulator;
end
look for peaks in the accumulator and extract the corresponding shapes;

Some improvements of the algorithm are reviewed in [34]:

- Select *active peaks* at regular intervals instead of sweeping and updating the whole parameter space.
- Instead of working point-by-point, use randomly drawn minimal set of points which identifies one single corresponding shape (three points for a plane), only update that cell in the accumulator.
- Create a one-to-one correspondence between the point and the shape (a plane), by locally estimating the normal [35], eventually, smooth the vote with a Gaussian distribution in parameter space [36].

RANSAC

RANSAC strategies stem from the algorithm introduced by Fischler and Bolles in [37]. It is considered the state-of-the-art method for model fitting. Some key concept in this technique are:

- The *consensus set* of $\Gamma(\alpha)$ in P is the set of points $p_i \in P$ within a certain threshold from a shape $\Gamma(\alpha)$.
- The size M of the **minimal set** of points in P which uniquely identify one shape.
- The inverse function $\Gamma^{-1} : \mathbb{R}^{3M} \rightarrow \mathbb{R}^N$ that allows retrieving the single shape identified by a minimal set: $\Gamma^{-1}(p_1, ..., p_M) = (\alpha_1, ..., \alpha_N)$.

A typical RANSAC algorithm is presented in Algorithm 3.

Several improvements have been studied about each step of the algorithm. Schnabel, Wahl, and Klein in [38] use this method to efficiently extract primitive shapes such as planes, torus, cylinders, cones, and spheres from point clouds. The authors implement a sampling criterion which follows the assumption that the a-priori probability that two points belong to the same shape is higher for closer points. Torr and Zisserman in [39] formulate the problem in a probabilistic framework, thus basing the shape re-fit (Algorithm 3 Line 1) by finding the maximum-likelihood shape that fits the points.

Algorithm 3: RANSAC algorithm

Data: $P = \{p_{1,...,N}\}$ point cloud; Γ: $\mathbb{R}^N \rightarrow \mathbb{R}^3$, Γ^{-1}: $\mathbb{R}^{3M} \rightarrow \mathbb{R}^N$
$nc = 0$;
$i = 0$;
while $nc <$ cons_min and $i <$ max_i **do**
  Sample a subset $S \in P : |S| = M$;
  Find the parametric shape that fits the points $\alpha = \Gamma^{-1}(S)$;
  Find $S^* = p_1, ..., p_k \in P$ within a tolerance from $\Gamma(\alpha)$;
  $nc = \mid S^* \mid = k$;
  $i \leftarrow i + 1$;
end
if $i <$ max_i **then**
  refit the whole consensus set S^* minimizing some cost function;
else
  failure
end

Overall, model-based methods are fast and robust to outliers [32] and in particular RANSAC is considered the state-of-the-art primitive shape extraction method.

Given the works illustrated in this section, we decide to adopt a region-growing algorithm for horizontal surfaces detection presented by Dong, Gao, Zhang, *et al.* in [21]. The reason behind this choice is twofold: the strategy presented by the authors specifically aims at finding *horizontal* surfaces in contrast to the other reviewed methods. Moreover, the algorithm is simple and its results are comparable with the state-of-the-art RANSAC algorithm, while the runtime is lower [21]. We adapt the algorithm to the case of $organized$ point clouds like the ones generated by RGB-D or TOF cameras, by integrating the contributions of Holz, Holzer, Rusu, *et al.* in [19] and Holzer, Rusu, Dixon, *et al.* in [25] about fast neighbors retrieval and normal vectors estimation in organized point clouds.

2.4 Placement Quality

If the robot manages to find multiple solutions to the placement problem, it has to be able to rank them and choose the most suitable one to solve the problem at hand.

One way to rank different placement poses is to take into account their semantic preference: not all the object's orientation feel equally desirable to humans, and neither do all locations. Baumgartl, Werner, Kaminsky, *et al.* in [15] define an object preferred orientation by aligning the object's volume first dominant principal axis to the placement surface normal. This encourages vertical placements over horizontal ones. A complementary work on this topic is presented in [40], where the authors train a Support Vector Machine on hand-made geometric features to predict the upright orientation of objects. A similar method is implemented in [18]. The authors predict the semantically preferred placement areas and orientations for objects. For example, they learn that a plate is preferably placed on a table or in a dish-rack and that in the first case it should be placed horizontally, and in the second case it should be caged by the racks.

Other placement objectives are to maximize the clearance from obstacles to ensure a safe placement, or to minimize it to implement a packing procedure [9].

In this work, we choose to minimize the object placement height, following the empirical observation that to fill up roller containers it is beneficial to start placing objects at the lowest layer available, hence:

$$\xi(x_o) = -x_{o,z} \tag{2.7}$$

2.5 Placement Search Space Exploration

We operate in the presence of several obstacles, hence it is important to efficiently explore the search space, as most of the poses will not satisfy the problem constraints. Most of the works presented in Section 2.2 do not explicitly mention the strategy adopted to sample candidate placement poses. Baumgartl, Werner, Kaminsky, *et al.* in [15] guide their search starting from an arbitrary point in the space, and they progressively explore the nearby positions and orientations; they stop the search as soon as a valid pose is found. Haustein, Hang, Stork, *et al.* in [9] perform an efficient search biased towards regions which are expected to be more favorable to satisfy the problem hard constraints, while maximizing a user-provided objective function $\xi(x_o)$. This algorithm aims to find quickly a feasible solution and to progressively optimize it in an anytime fashion. We adopt this second strategy as it is tailored for placements in cluttered scenes and explores the solution space while exploiting the local information progressively collected during the search.

To explore the solution space, the authors employ a tree structure to hold the hierarchical subdivision of the position search space (in our case R_H) and the orientation search space (in our case $SO(2)$). At the first level of the tree, we find the clusters that were produced by the horizontal region detection algorithm, and each of them is associated with the complete orientation solution space of $[-\pi, \pi)$. At each deeper layer in the tree, the region is recursively split in four regions along its principal axes and the orientation space is divided in k equal-width segments. The splitting process is continued until a user-defined resolution either in space or orientation is reached. Each node in the tree, therefore, restricts the search to a small area in the space and a subset of orientations.

The authors use this tree-structure for a Monte Carlo tree search to incorporate feedback to bias the sampling process towards more promising regions, while keeping the search explorative of untested spaces. Every time a point is drawn from a region, a heuristic that takes into account the hard constraints and the user-provided objective is computed. The reward is propagated back to the parents of the region (see Figure 2.4), thus indirectly informing the sampling process about the expected results in the nearby space as well. Each time a kinematically reachable collision-free and stable placement pose that improves the objective $\xi(x_o)$ is found, the motion planning algorithm is invoked to verify its path-reachability and compute the motion $\tau(C_0, x_o)$.

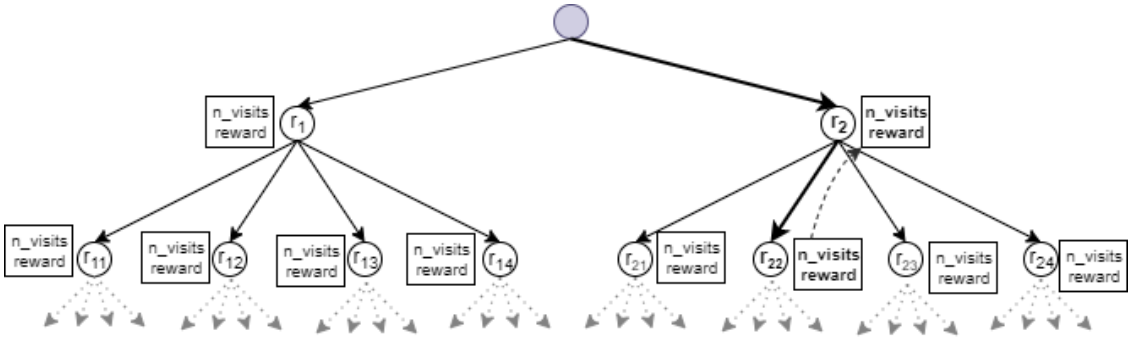


Figure 2.4: Hierarchical representation of the search space. Each node in this tree represents a region (or a sub-region). When a region is sampled, the number of visits increases and the corresponding reward is computed according to an heuristic function which evaluates the hard and soft constraints of the problem.

The observation that motivates the algorithm in [9] is that there exists a spatial correlation between the poses that satisfy the hard constraints: for example, if a pose produces a collision with the nearby environment, close poses will likely do the same and vice-versa. Similarly, the adopted sampling strategy is helpful in the case $\xi(x_o)$ has a limited local growth within contiguous regions in the solution space, which is certainly true for the z coordinate of points belonging to the same horizontal region.

By integrating the perception of the obstacle volume and using the extracted planar surfaces from the point cloud as candidate placement regions with this placement module, we are able to compute placement poses in new scenes for known objects.

Chapter 3

Method

Figure 3.1 illustrates the overall system that was developed in this thesis. This chapter covers the method that was developed to implement the perception module and the integration with the planning algorithm in [9]. Throughout this chapter we use the notation introduced in Section 1.1. Section 3.1 briefly describes the sensor that produces the input data, its format and how the data is pre-processed. Section 3.2 presents the strategy adopted to estimate the volume occupied by the obstacles in the sensed space. Section 3.3 illustrates the method used to extract horizontal flat regions from a point cloud, which restricts the search space for the placement position. Section 3.4 details how the work in [9] was integrated with the perception module.

3.1 Input Data

The robot perceives the environment through an RGB-D camera. In the following subsections we briefly describe the working principle of the employed sensor.

3.1.1 The Sensor

The robot is provided with an Intel® RealSense™ D435 depth camera with the technical specifications in Table 3.1 [41].

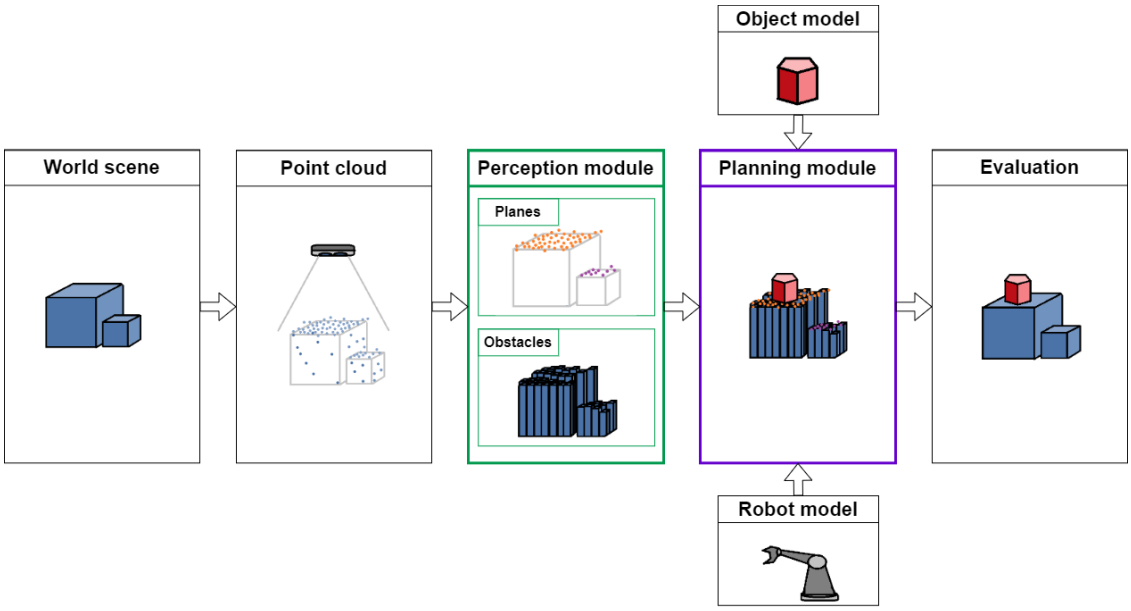


Figure 3.1: The overall system that was developed in this thesis is summarized in this figure. The robot can perceive the environment through an RGB-D sensor, that produces a point cloud. The point cloud is the input of the perception module, that produces an estimation of the obstacle volume in the environment and extracts horizontal planes that restrict the position search space. The obstacles and the horizontal regions are then fed to the planning module, together with the robot and object model. The planning module samples a feasible placement for the object in the environment, and computes a motion for the robot. Lastly the placement is evaluated by manually reproducing the computed placement in the real scenario.

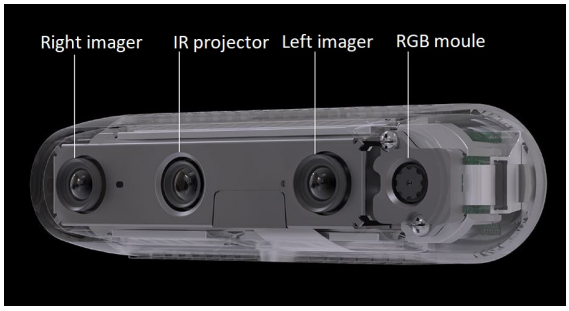


Figure 3.2: Intel® RealSense™ D435 depth camera. The picture shows the Infrared stereo module (right and left imager), the Infrared projector and the RGB Module. Picture taken from [41]

.

Working Principle

The sensor has two infrared (IR) cameras, an RGB sensor, and an IR pattern projector (see Figure 3.2). It is a stereo camera, which means that depth information estimation

Features	Environment: indoor/outdoor Image Sensor Technology: Global Shutter, 3 μmxμm pixel size Maximum range: approx 10 meters.
Depth	Depth technology: Active IR Stereo Depth field of view (FOV) : $87° \pm 3° \times 58° \pm 1° \times 95° \pm 3°$ Maximum range: approx 10 meters. Depth output resolution and frame rate: Up to 1280 $\times$ 720 active stereo depth resolution. Up to 90 fps.
RGB	RGB Sensor Resolution and Frame Rate: 1920×1080 RGB frame rate: 30 fps RGB Sensor FOV (H x V x D): $69.4° \times 42.5° \times 77°(\pm 3°)$
Major components	Camera Module: Intel RealSense Module D430 + RGB Camera Vision Processor Board: Intel RealSense Vision Processor D4
Physical	Form Factor: Camera Peripheral Connector: USB-C* 3.1 Gen 1* Length $\times$ Depth $\times$ Height: 90 mm $\times$ 25 mm $\times$ 25 mm One 1/4-20 UNC thread mounting point. Two M3 thread mounting points.

Table 3.1: Technical specification of Intel® RealSense™ D435 depth camera.

is based on the comparison of the output produced by two identical (IR) sensors, displaced along one of their axis with a known offset (left and right imagers in Figure 3.2), as to mimic human vision.

After having acquired two IR images at the same time, the procedure to infer the per-pixel depth is the following:

1. Solve the *correspondence* problem: find a per-pixel correspondence between the two images (i.e., match pixels which were originated by the same world location). For each corresponding pair of pixels, its *disparity* is computed, i.e., the shift that separates the two pixels in the two images along the cameras displacement axis (see Figure 3.3 and 3.4). The objects which are closer to the camera will experience a larger displacement with respect to the objects that are far away in the background.

2. Triangulate each pixel to find its location in the camera frame, leveraging the knowledge of the spatial offset between the two cameras.

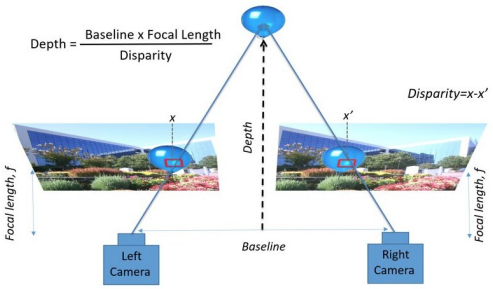
The result of this operation is an IR image with associated a per-pixel depth information, hence their location in the camera reference systems. The image is enriched with the color data retrieved by the RGB camera.

(a) A landscape as seen from the left camera of a stereo system.

(b) A landscape as seen from the right camera of a stereo system.

Figure 3.3: A landscape as seen from two cameras of a stereo system. The orange (red) square highlights a point in the world and its different locations in the two camera's frames. The orange (red) line connecting the two squares represents the *disparity* between the two pixels. Images from [42].

(a) An example of a disparity map.



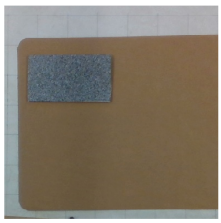
(b) Retrieving depth from disparity maps.

Figure 3.4: The image on the left contains a toy example of a disparity map. Pixels belonging to closer objects have higher disparity values. Images from [42].

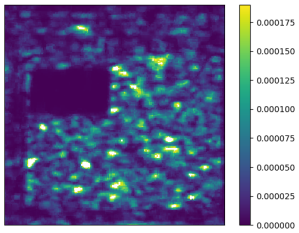
Limitations

The key operation that enables the stereo camera to retrieve the depth of each pixel is finding correspondences between the two pictures, which is easier if the scene in

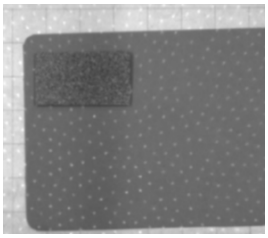
the field of view as a highly varied texture. Therefore the camera is more appropriate for applications in external environments, whereas it may be challenging to retrieve reliable data when dealing with homogeneously textured patches (such as walls or floors). An example of this effect can be observed in Figure 3.5.



(a) A flat homogenously colored surface, with a random noise patch in its top-left corner.



(b) Heatmap of the variance of the estimated per-pixel depth over ten frames.



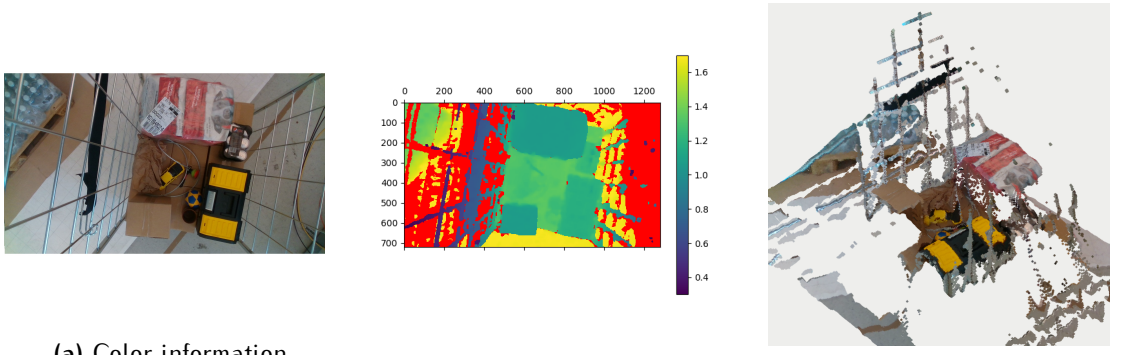
(c) Infrared pattern projected on the scene.

Figure 3.5: The depth estimation is more consistent when the surface has a varied texture.

To overcome this limitation, the camera is provided with an IR projector, which projects a semi-random light pattern on the scene consisting of five thousand points [43] (see Figure 3.5c). This pattern enriches the scene with more recognizable key-points that help finding correspondences in low-textured scenes. If the scene material is poorly reflective such as transparent plastic or dark surfaces, however, the projector will bring no advantage.

3.1.2 Data Format

The data produced by this sensor is a collection of points in camera frame organized in a two dimensional matrix $P = \begin{pmatrix} p_{1,1} & \cdots & p_{1,w} \\ \cdots & p_{m,n} & \cdots \\ p_{h,1} & \cdots & p_{h,w} \end{pmatrix}$, hence called *organized* point clouds. h is the height of the cloud (1280 in our sensor) and w is its width (720). Each point $p_{i,j}$ carries its location in camera frame $(x, y, z) \in \mathbb{R}^3$ together with the color $(r, g, b) \in [0.0,1.0]^3$ information provided by the RGB camera. This data can be interpreted as a general picture (3.6a), a depth map (3.6b) or a point cloud in space (3.6c).



(a) Color information.

(b) Depth information $[m]$. Red pixels represent missing data.

(c) Color and depth information combined in a point-cloud.

Figure 3.6: Three ways to interpret the output of the RGB-D camera.

3.1.3 Preprocessing

Stereo cameras like the one that we employ in this project are heavily affected by local noise, and we need a way to mitigate the derived uncertainty.

If the sensor and the scene are steady, each pixel measures the same point in the space in subsequent frames. Hence, it is possible to average several sensor measurements over time in a per-pixel basis. To do this, we apply the temporal filter provided by the Realsense Library [44] and implemented in an exponential moving average (Algorithm 4). This strategy allows us to mitigate the per-pixel noise, as well as to recover partially missing frames.

Algorithm 4: Exponential temporal filtering

Data: $D^{t_1} = \{d_{i,j}^{t_1}\}$ depth matrix acquired at t_1;
$\hat{D}^{t_0} = \{\hat{d}_{i,j}^{t_0}\}$ filtered depth data at the previous time step t_0;
Result: $\hat{D}^{t_1}$ filtered depth data at t_1
for $d_{i,j}^{t_1} \in D^{t_1}$ **do**
  if $\mid d_{i,j}^{t_1} - \hat{d}_{i,j}^{t_0} \mid < \delta_{max}$ **then**
    $\hat{d}_{i,j}^{t_1} = \alpha d_{i,j}^{t_1} + (1-\alpha)\hat{d}_{i,j}^{t_0}$
  else
    $\hat{d}_{i,j}^{t_1} = d_{i,j}^{t_1}$
  end
end

To make the computation feasible in short times sequentially on a CPU, we will use only a downsampled version of the point-cloud (originally made of $1280 * 720 = 921600$ points) both in the obstacle volume estimation operation (Section 3.2) and in the horizontal region extraction (Section 3.3). To reduce the size of the point cloud we employ a *decimation* step: given a resolution d, only the points at a row/column which is multiple of d are kept in the cloud (Equation 3.1):

$$P = \{p_{i,j} \mid i = k * d, j = n * d, k = 0, 1, ..., \frac{h}{d}, n = 0, 1, ..., \frac{w}{d}\} \tag{3.1}$$

Obviously decimating the point cloud worsens the spatial resolution of the available data. This effect is linearly dependent on the depth of the points in the camera frame (Figure 3.7). To choose a suitable decimation factor, we consider the fact that our maximum working depth is of 1.5 m. Therefore we decide to use a decimation factor of 4, which means having a spatial resolution at 1.5 m distance of 0.0066 m, while bringing down the number of points to 57600.

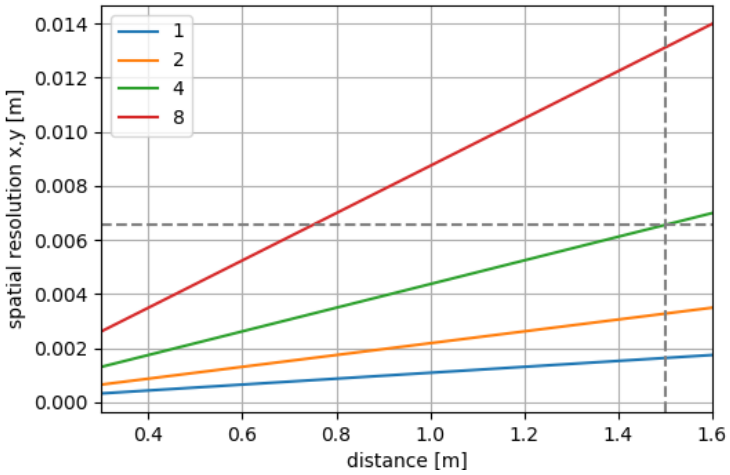


Figure 3.7: Resolution along the camera x and y-axis for different decimation factors at different depths. The spatial resolution is computed according to the equations in [45].

Using these strategies we produce a reduced version of the original point-cloud which is robust to local noise. Neither filtering nor decimating merges any measurements spatially, thus preserving edges.

3.2 Obstacle Volume Estimation

Obstacle volume estimation is needed to assess the non-collision constraint of the motion of the robot $c(\tau)$. As described in Chapter 1, initially the robot is grasping an object and it is at a pre-place configuration that allows it to have a top view over the placement volume (Figure 3.8). The robot end-effector grasping the object is at the same height of the camera.

Figure 3.8: Point cloud acquired from the RGB-D camera showing the view of the robot over the placement volume.

We restrict the motion of the robot to be within the camera field of view (FOV), therefore all the obstacles that have to be take into account are visible. The placement volume is at the center of the FOV, such that we can approximate occlusion to be vertical.

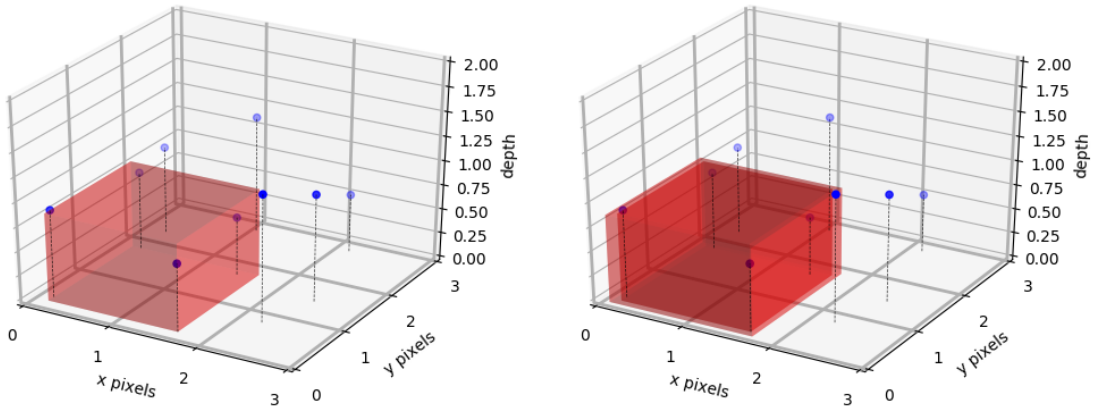
To place the object, the robot will realize a vertical descending motion that brings the object to the target placement pose. In addition, the mounting point of the camera limits the view to top surfaces in the placement volume. Therefore, we can consider the space that lies underneath the perceived surfaces and above the floor as occupied or unknown. In this way, we allow placements only on the top surfaces of the visible objects, and we prevent the robot to move underneath visible surfaces.

To do this, we employ a simple obstacle representation similar to a heightmap. Namely, we use a composition of axis-aligned cuboids to describe the occupied

space, which allows fast collision checking. The method to retrieve this volumetric description is detailed in Algorithm 5 and Figure 3.9.

Algorithm 5: Estimate obstacle volume from a point cloud.

Data: $P = \{p_i\}$, k inflation factor
Result: $V_{obs} \subset V_s$ volume occupied by obstacles
initialize $V_{obs} = \emptyset$;
for $i = 1, ..., h-1$ **do**
  for $j = 1, ..., w-1$ **do**
    p_r, p_d, p_{dr} = `neighbors` $(p_{i,j})$;
    find $V_{obs,p_{i,j}}$, the axis-aligned rectangular cuboid that fits the four points;
    inflate its base by k;
    $V_{obs} \leftarrow V_{obs} \cup V_{obs,p_{i,j}}$;
  end
end
Procedure `neighbors`$(p_{i,j})$
  $p_r = P_{i,j+1}$;
  $p_d = P_{i+1,j}$;
  $p_{dr} = P_{i+1,j+1}$;
  return $[p_r, p_d, p_{dr}]$;



(a) Rectangular cuboid fit to the first four pixels of the image.

(b) Original and inflated obstacle. The base is inflated by a 1.10 coefficient.

Figure 3.9: Obstacle volume approximation. Each pixel contains one measurement (a point in the space). Pixels are processed four-by-four; a rectangular axis-aligned cuboid is fit to each group of pixels. The obstacle base is then inflated by a coefficient (right image), which in the example is 10%. This operation ensures further safety during motion planning.

3.3 Horizontal Regions Extraction

To ensure placement stability, we follow the strategies presented in Section 2.2: the planning algorithm in [9] ensures that the center of mass of the target item falls into the candidate placement face, and we restrict placements to flat surfaces.

We need to detect horizontal segments in the input point cloud, to restrict the placement position search space. We choose the world frame orthonormal reference system $\{\hat{x}, \hat{y}, \hat{z}\}$ in which gravitation acts in the negative direction of the z-axis $\vec{F}_g = -|F_g|\hat{z}$. A plane in this reference system is horizontal if its normal vector is parallel to $\hat{z}$. Given $P = \{p_i\}$, the output of the sensor brought to the world frame, we want to cluster together those points which belong to the same horizontal region and therefore will share a similar z coordinate. We want to filter out the points that do not belong to horizontal segments.

To this aim, we keep the overall structure of the algorithm in [21] and summarized in Algorithm 6 for horizontal clusters extraction. We change the pre-processing method (Section 3.1.3), post processing strategies (Section 3.3.2) and the normal vectors estimation method (Section 3.3.1).

3.3.1 Normal Vectors Estimation

The two features used in Algorithm 6 are the point z coordinate (line 4) and its local normal vector (lines 1-2). Dong, Gao, Zhang, *et al.* estimate this vector by employing a fixed-size neighborhood using standard PCA as illustrated in Section 2.3.1.

Instead, we adopt the method presented by Holzer, Rusu, Dixon, *et al.* in [25], who implement a depth-adaptive normal estimation mechanism precisely for *organized* point clouds. Table 3.2 summarizes the time-improvement experienced when adopting the decimation and integral-images-based normal computation method, with respect to the voxel filter and PCA based on KD-tree strategy adopted in [21].

We adopt the square-neighborhood definition used in [25] to speed up the neighbor search needed in line 2 of Algorithm 6. This concept is implemented in the Point Cloud Library [46] by Radu B. Rusu et al. with the name of OrganizedNeighbor.

Algorithm 6: HoPE: Horizontal Plane Extractor for Cluttered 3D Scenes [21]

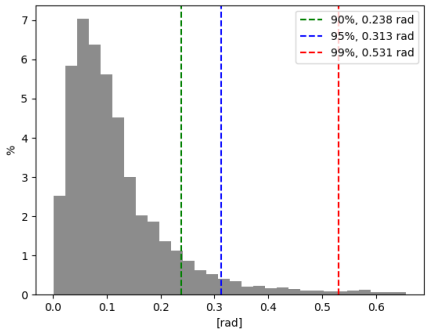
Data: $P = \{p_i\}$, $N = \{N_{p_i}\}$ neighbors of each point in P
Result: $R_H = \{R_j\}$ horizontal segments: $R_j = \{p_k\}$ set of points belonging to the same horizontal surface
initialize $R_H = \emptyset, j = 0$;
`preprocess(`P`);`
compute point-wise normals $\{n_i\}$;
keep points with almost vertical normals: $P = \{p_i\} \mid cos^{-1}(\frac{\langle n_i, \hat{z} \rangle}{\|n_i\|\|\hat{z}\|}) < \theta_{max}$;
select a seed point p_s;
$P \leftarrow P \setminus p_s$;
$R_j = \{p_s\}$;
while $P \neq \emptyset$ **do**
  for $p_n \in N_{p_s}$ *neighbor of* p_s **do**
    if $|p_{n,z} - p_{s,z}| < \delta z_{max}$ **then**
      $R_j \leftarrow R_j \cup \{p_n\}$;
      $P \leftarrow P \setminus p_n$;
      p_n can be chosen as new seed to grow region j;
    end
  end
  if There is a point in R_j that has not been used as seed yet **then**
    choose p_s from R_j, keep growing this region;
  else
    $R_H \leftarrow \{R_H \cup R_j\}$;
    Choose p_s from P;
    $j = j + 1$;
    Start growing a new region $R_j = \{p_s\}$
  end
end
`postprocess(`R_H`);`
Procedure `preprocess(`P`)`
  remove points out of the camera working depth ranges ;
  remove outliers ;
  down-sample P using a voxelized filter;
  return P;
Procedure `postprocess(`R_H`)`
  detect false positives by analyzing each region's points normal distribution;
  return R_H;

Method	Time/point(avg) $[\mu s]$	Std dev $[\mu s]$
Voxel filter + KD-tree	3.23	0.135
Decimation + Integral image	0.156	0.0298

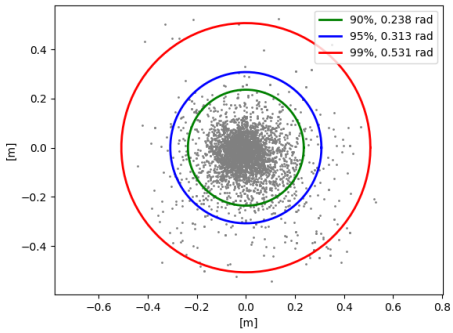
Table 3.2: Per-point processing time to downsample and estimate the normals of an organized point cloud initially made of 921600 points. We compare our method with the one in [21]. Dong, Gao, Zhang, *et al.* in [21] use a voxel-approach to reduce the point cloud dimensionality (Section 2.3) and a standard KD-tree and PCA strategy for normal vectors estimation (Section 2.3). The results are taken by using a voxelized filter with $1.0cm$ resolution (output 14615 points), and a the decimation method with decimation factor 8 (output 14400 points). The experiment were run on an Intel Core i7-8550U CPU, 1.80 GHz of standard frequency and up to 4.00 GHz of turbo frequency. The second method gives a $\sim$ 20x time improvement.

This definition of neighborhood takes down the temporal complexity of neighbors extraction from $O(log(N))$ in a K-D tree data structure [47] to constant time.

Lastly, to choose the maximum value for the tilting angle θ_{max} between $\hat{z}$ and n_i (Algorithm 6 line 2), we examine the distribution of those angles at a planar surface (Figure 3.10) and pick the one corresponding to its 99th percentile, namely 0.531 rad $(30.42°)$.



(a) Distribution of the angle formed by the local normal and the z axis on a flat surface. The vertical lines cut the distribution at the 90th, 95th and 99th percentile of the distribution.



(b) Distribution of the x and y components of the normal vectors on a flat surface. If the surface is planar with white noise, the distribution should be gaussian, as the one in the picture. The colored circles represent the 90th, 95th and 99th percentile of the distribution.

Figure 3.10: Distribution of the tilting angle of the estimated normals at a planar surface.

3.3.2 Postprocessing

Once the candidate planar clusters $R_H = \{R_i\}$ have been extracted, we conduct a post-processing procedure to exclude false positives. An example of a false positive, can be a slightly tilted planar surfaces: the per-point normal vectors are almost vertical, but the cluster is not overall horizontal. Similarly, the upper face of a sphere presents almost vertical normal vectors.

To effectively detect sloped planar clusters from our final result, we compute the least-squares plane fit π_q of each cluster and enforce a threshold on the allowed overall slope, which in this case can be much finer than θ_{max}. Namely, we filter out those point clusters which overall normal is tilted of more than 0.08 rad $(5.0°)$ from the reference vertical vector $\hat{z}$ (Figure 3.11).

(a) Pitch: 0 rad $(0\ °)$ **(b)** Pitch: -0.05 rad $(-3.0\ °)$ **(c)** Pitch: -0.07 rad $(-4.0\ °)$

Figure 3.11: To test the post processing analysis, we progressively tilt the scene by adding some pitch to the camera transform. We highlight in green the points that belong to planar clusters. When the tilting reaches about 4 degrees, almost no point in the original planar surface is detected as planar.

To tackle the second case of false positives, we test the cluster curvature [28]: given the cluster R_i, the covariance matrix of the cluster is computed as in Equation 3.2.

$$C_i = \frac{1}{K}\sum_{j=1}^{K}(p_i - \hat{p})^T(p_i - \hat{p}) \tag{3.2}$$

being $\hat{p}$ the centroid of the points set. We let $\lambda_0, \lambda_1, \lambda_2$ be the eigenvalues of C_i, such that $0 \leq \lambda_0 \leq \lambda_1 \leq \lambda_2$. If the cluster is planar, we expect it to have a

principal component decomposition such that the two principal components would explain most of the cluster variance, therefore $\lambda_1 >> \lambda_0$ and $\lambda_2 >> \lambda_0$. Given this principle, the curvature is computed as in Equation 3.3.

$$\sigma = \frac{\lambda_0}{\lambda_0 + \lambda_1 + \lambda_2} \tag{3.3}$$

We put an upper bound on the curvature of 0.02, i.e. only 2.0% of the cluster variability can be explained by the third principal component (the normal) of the cluster.

3.4 Integration with the Placement Planner

The set of horizontal regions R_H, restricts the search space for the candidate placement positions from V_s to R_H. Next, we integrate the perceived regions with the planner in [9] and reviewed in Section 2.5.

We provide a way to hierarchically split the placement regions, as required to build the Monte Carlo Search Tree. To do this, we use the Principal Component Analysis to find the placement region principal axis and its centroid. We assign each point to a different sub-region, according to their position w.r.t. the center and the two principal axis of the region (Figure 3.12).

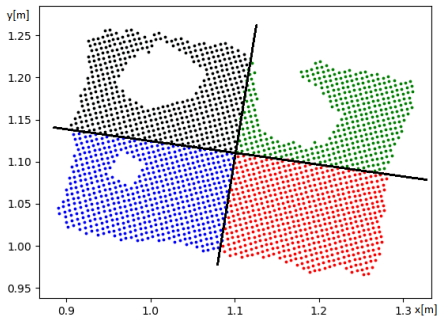


Figure 3.12: Creation of children regions from an initial set of points. The points in the plot are the projection of the points in the space into the xy plane. Each color represents the assignment of those points to a specific subregion. To subdivide the regions produced by the segmentation algorithm hierarchically, we use the Principal Component Analysis to find the central point of the area and the two horizontal principal axes. Then, we consider a local reference system centered in the mean position and with the axes found by PCA: each point is assigned to one of the four child regions based on the sign of the local coordinates in this new reference system.

The pose that is sampled in the solution space refers to a reference system that is centered on one of the points in the base surface of the target object's convex hull (Figure 3.13). To assess the support constraint, we verify that each of the other vertices in the placement face is in contact with some surface in the scene. To do this, we perform a radius search for each of the vertices of the object on the original input point cloud. If all of them have a minimum amount of neighbors within a radius r, we consider the placement to be stable.

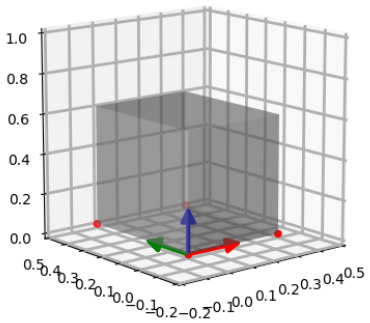


Figure 3.13: Cubic object that has to be placed in the scene. The red points are checked to assess the stability constraint. The arrows symbolize the object's local reference system.

The integration of the perception and placement planning module, allows the computation of a placement pose $\hat{x_o}$ and a corresponding motion τ to place a known objects in a scene perceived through an RGB-D sensor. The pose satisfies the no-collision and stability constraint. Moreover, it maximizes the objective function $\xi(x_o)$, i.e. is the lowest feasible placement.

Chapter 4

Results

In the following, we illustrate the result of the plane extraction process and placement computations in some placement scenarios of increasing difficulty. To evaluate the computed placements, we manually reproduce them in the real scenario.

4.1 Placing on a Single Surface

To assess the performance of the pipeline, we first test if it is possible to compute simple placements like placing small boxes on a single object. Each supporting object stands on a layer of boxes and in this phase we force the plane extraction module to ignore the boxes.

4.1.1 Toolbox

As first supporting object, we choose a rigid toolbox. The toolbox has an upper rigid surface of 15.0 cm $\times 40.0$ cm and is slightly sloped on the side. We compute a placement for a small carton box with a bottom face of 8.0 cm $\times 19.0$ cm. Table 4.1 shows the process of plane extraction. Table 4.2 shows the result of the obstacle volume estimation applied to the content of the roller container (in red); the volume of the roller container (in grey) is known. Table 4.3 shows multiple planned placement poses and the reproduction of one of those in the real setup.

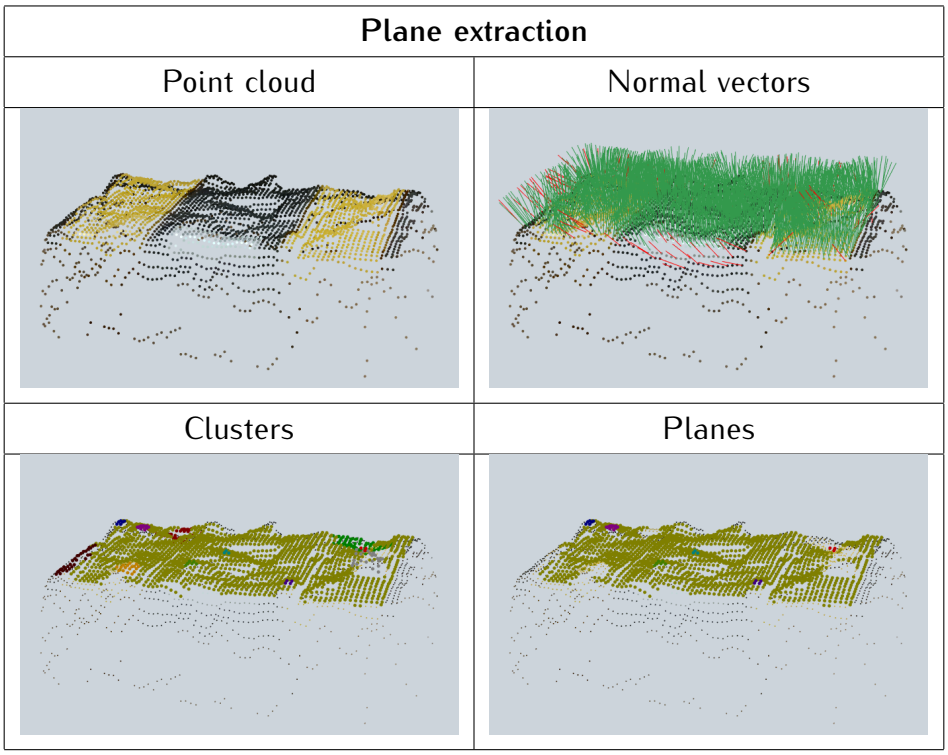


Table 4.1: Plane extraction process applied to a toolbox. In the upper-left corner the original point cloud. In the upper-right corner the estimated normal vectors: the red ones are filtered out according to line 2 of Algorithm 6. In the bottom-left corner the remaining points are divided in clusters, each color corresponds to a new cluster. The bottom-right corner shows the cluster that pass the post-processing test.

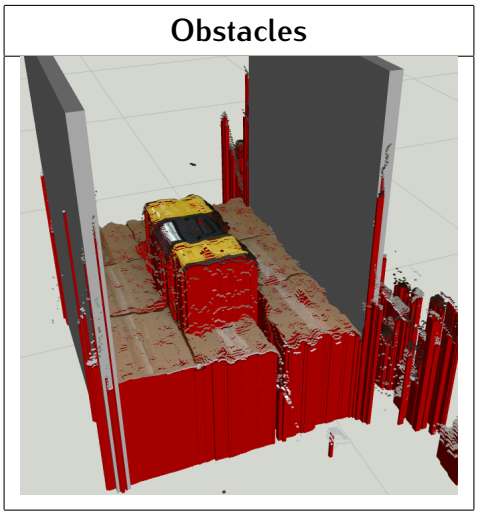


Table 4.2: Obstacle estimation (in red). In grey the obstacle volume of the roller container, its pose and volume is known.

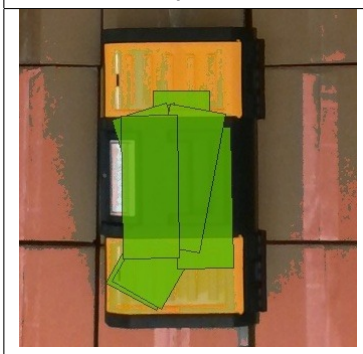
Planning		
Planned placements	Example	Real placement
		

Table 4.3: Computed placement for a small box. The leftmost picture shows the footprint of the object in different computed placement poses. In the middle one of the computed placements, on the right the reproduction of the placement.

4.1.2 Cans

As a second experiment, we use as support a package of 24 cans. Cans have circular planar surfaces of 2.5 cm radius and are wrapped in a plastic film that is overall planar. The size of the top layer of the can box is circa 38.0 cm $\times 29.0$ cm. We plan a placement for a box with a base of 26.0 cm $\times 20.5$ cm. Table 4.5 shows the result of the obstacle volume estimation. Table 4.6 shows multiple planned placement poses and the reproduction of one of those in the real setup.

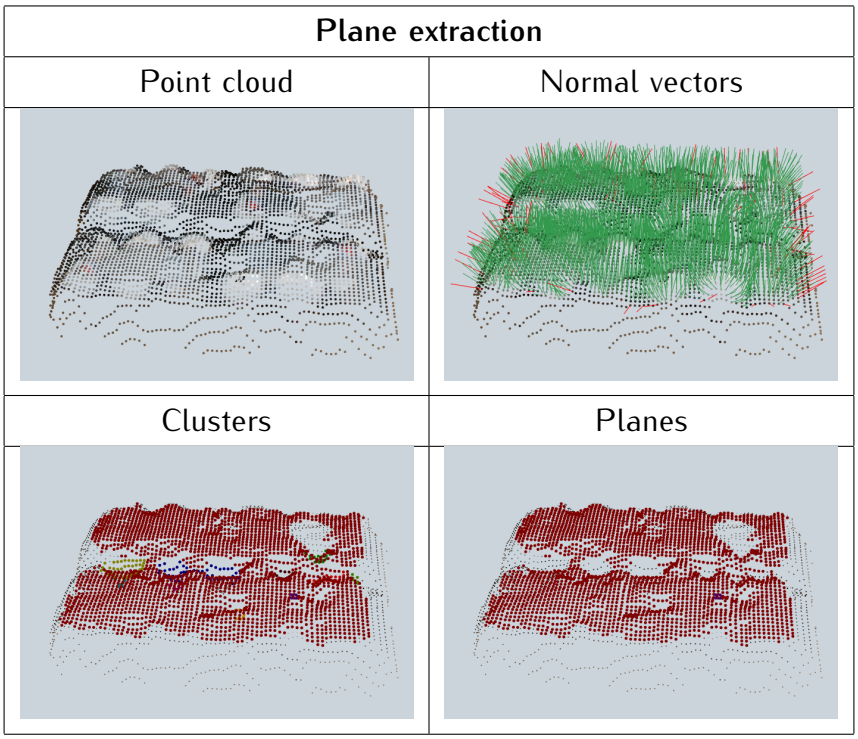


Table 4.4: Plane extraction process applied to a box of cans. In the upper-left corner the original point cloud. In the upper-right corner the estimated normal vectors: the red ones are filtered out according to line 2 of Algorithm 6. In the bottom-left corner the remaining points are divided in clusters, each color corresponds to a new cluster. The bottom-right corner shows the cluster that pass the post-processing test: curved clusters between the packages are discarded.

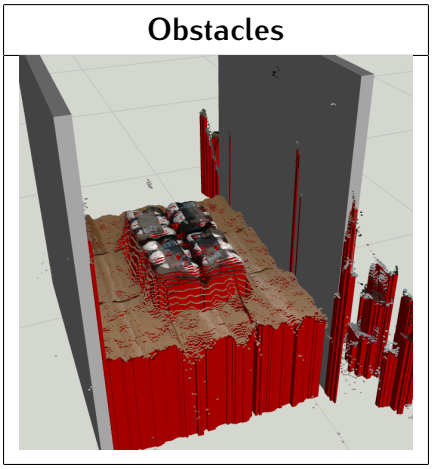


Table 4.5: Obstacle estimation (in red). In grey the obstacle volume of the roller container, its pose and volume is known.

Planning		
Planned placements	Example	Real placement
		

Table 4.6: Computed placement for a box with a base of 26.0 cm ×20.5 cm. The leftmost picture shows the footprint of the object in different computed placement poses. In the middle one of the computed placements, on the right the reproduction of the placement.

4.1.3 Bag of Pet Food

The next supporting surface that we use is a pet food bag. The bag is not rigid, therefore it has several local bumps and is overall curved. The size of the bag can be approximated to be 42.0 cm ×26.0 cm. The sides of the bag are slightly curved, therefore it is not possible to find placements that require having all the supports on its most external sides. For this reason, it is not possible to find a placement of the 26.0 cm ×20.5 cm side box. We compute placement poses for a box of size 26.0 cm ×16.0 cm. Table 4.7 shows the result of plane extraction: the local bumps constitute small clusters at different heights. Table 4.8 shows the results of obstacle volume estimation applied to the content of the roller container. Table 4.7 illustrates multiple solutions found by the planning algorithm and the realization of one of them.

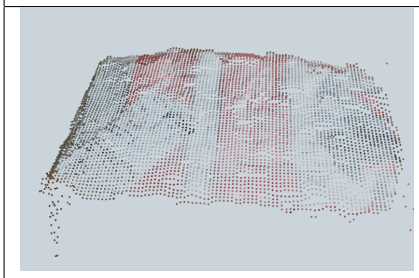
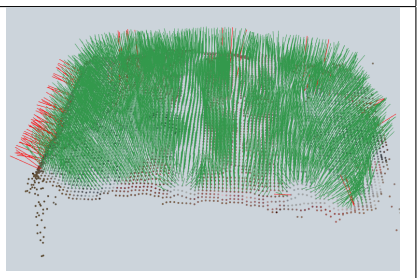
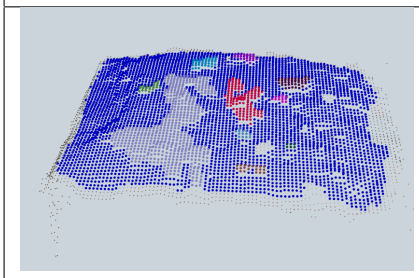
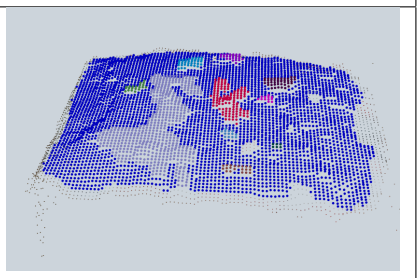
Plane extraction	
Point cloud	Normal vectors
	
Clusters	Planes
	

Table 4.7: Plane extraction process applied to a the bag of pet-food. In the upper-left corner the original point cloud. In the upper-right corner the estimated normal vectors: the red ones are filtered out according to line 2 of Algorithm 6. In the bottom-left corner the remaining points are divided in clusters, each color corresponds to a new cluster. The bottom-right corner shows the cluster that pass the post-processing test.

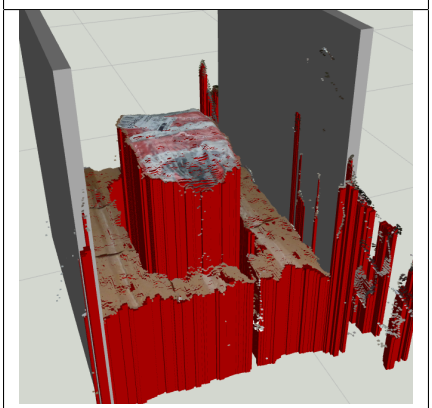
Obstacles


Table 4.8: Obstacle estimation (in red). In grey the obstacle volume of the roller container, its pose and volume is known.

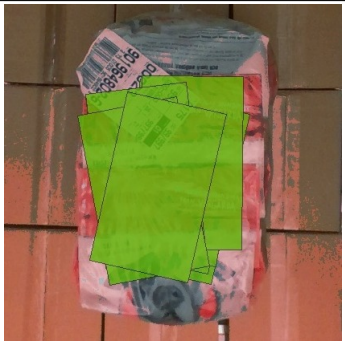
Planning		
Planned placements	Example	Real placement
		

Table 4.9: Planned and real placement. The object appears to be bigger in the real placement scenario, due to perspective distortions.

4.1.4 Water Bottles

Lastly we use as supporting objects four packs of water bottles, the size of one package is approximately 15.0 cm $\times 21.0$ cm. Water bottles are wrapped in a plastic film which is in fact overall almost planar. However, the plastic film is transparent and as a result it is hardly perceived by the sensor. Thus, only a small section of the top surface is detected as planar (Table 4.10). Nonetheless, it is possible to compute multiple placements for a box of size 26.0 cm $\times 20.5$ cm (Table 4.12).

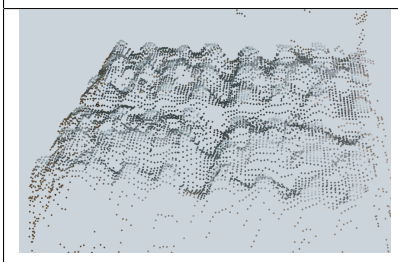
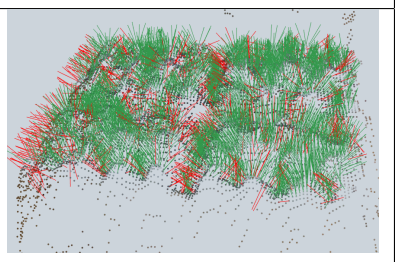
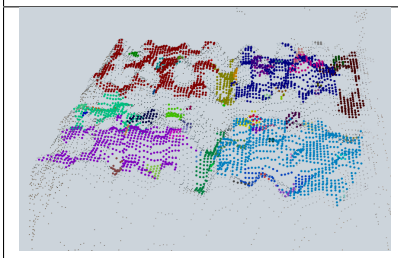
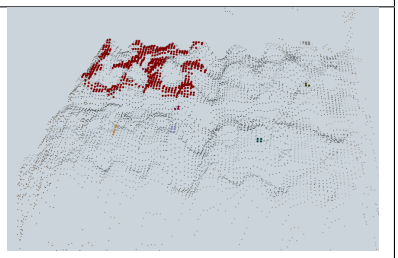
Plane extraction	
Point cloud	Normal vectors
	
Clusters	Planes
	

Table 4.10: Plane extraction process applied to a water bottle package. In the upper-left corner the original point cloud. In the upper-right corner the estimated normal vectors: the red ones are filtered out according to line 2 of Algorithm 6. In the bottom-left corner the remaining points are divided in clusters, each color corresponds to a new cluster. The bottom-right corner shows the cluster that pass the post-processing test. Almost all the clusters found by the algorithm do not pass the post-processing step, due to the high z-variation.

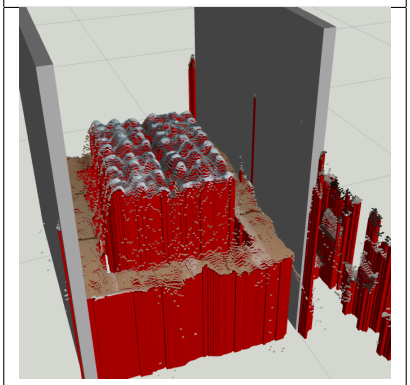
Obstacles


Table 4.11: Obstacle estimation (in red). In grey the obstacle volume of the roller container, its pose and volume is known.

Planning		
Planned placements	Example	Real placement
		

Table 4.12: Placement for a box. On the left several proposed placement poses, in the center and right picture the realization of one of them. The box in the real scenario appears to be bigger than the Planned one, due to perspective distortion.

4.2 Placing in Semi-filled Roller Containers

Next, we test the system on more complex scenes, namely roller-containers filled with several objects. We use three different target objects: the package of water bottles (base 18.0 cm $\times 24.0$ cm), a box (base 26.0 cm $\times 20.5$ cm), and a can package (base circa 40.0 cm $\times 27.0$ cm). The objects are approximated with their bounding boxes.

4.2.1 Roller Container filled with Boxes

As first scenario, we choose a roller container filled with boxes. This first scenario is easy, as the top surface of the boxes is planar. Table 4.13 illustrates the process of the plane extraction: the surfaces of the boxes are recognized as planar. The upper layer is perceived as two different clusters due to a perceived local discontinuity. Table 4.14 illustrate the result of obstacle volume estimation. Tables 4.15-4.17 show some planned placements for the water bottles, a box and the cans package.

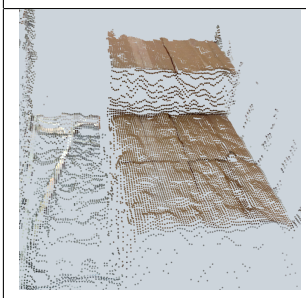
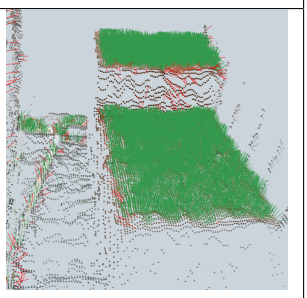
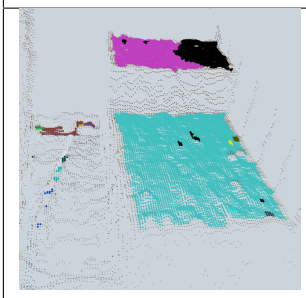
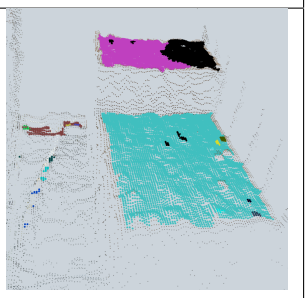
Plane extraction	
Point cloud	Normal vectors
	
Clusters	Planes
	

Table 4.13: Plane extraction process applied to a roller container filled with boxes. In the upper-left corner the original point cloud. In the upper-right corner the estimated normal vectors: the red ones are filtered out according to line 2 of Algorithm 6. In the bottom-left corner the remaining points are divided in clusters, each color corresponds to a new cluster. The bottom-right corner shows the cluster that pass the post-processing test.

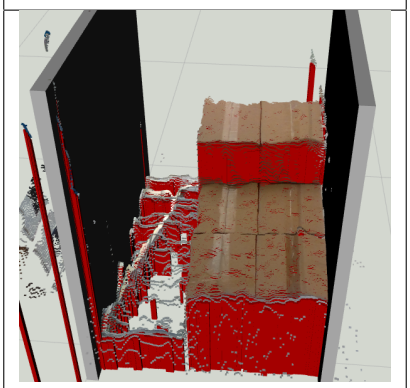
Obstacles


Table 4.14: Obstacle estimation (in red). In grey the obstacle volume of the roller container, its pose and volume is known.

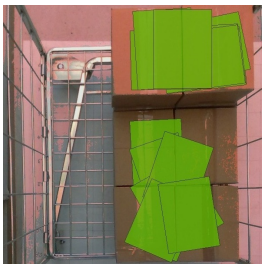
Planning - water bottles		
Planned placements	Example	Real placement
		

Table 4.15: Placement for a water bottle package. Placement for a water bottle package. On the left, the footprint of the bounding box of the package is plotted on the scene in several planned poses. The right image is the manual reproduction of the placement in the central image.

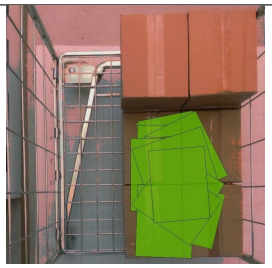
Planning - box		
Planned placements	Example	Real placement
		

Table 4.16: Computed placement for a box (26.0 cm $\times 20.5$). On the left, the footprint of the bounding box of the package is plotted on the scene in several planned poses. The right image is the manual reproduction of the placement in the central image.

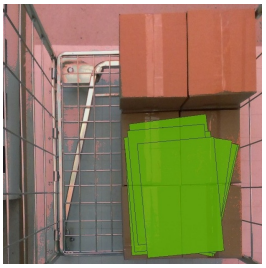
Planning - cans		
Planned placements	Example	Real placement
		

Table 4.17: Computed placement for a package of cans. On the left, the footprint of the bounding box of the package is plotted on the scene in several planned poses. The right image is the manual reproduction of the placement in the central image.

4.2.2 Semi-filled Roller Container

In the next scenario, we fill up the roller container with a combination of cans, boxes and pet food. We compute a placement for the water bottles (Table 4.20) and the box (Table 4.21). No placement is found for the cans.

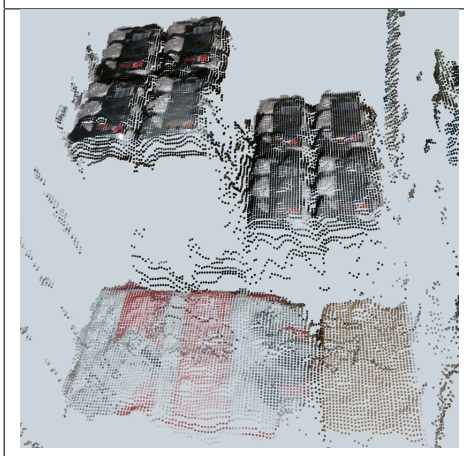
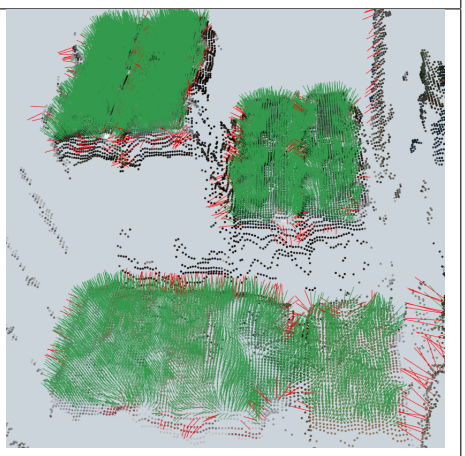
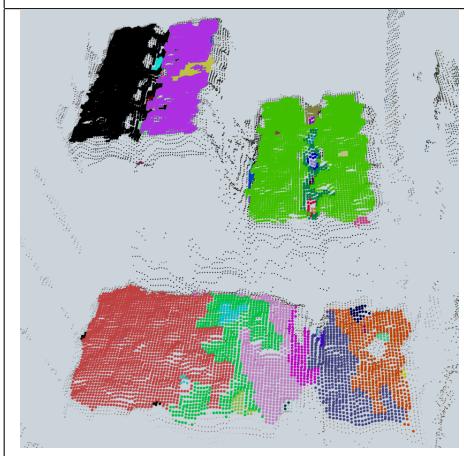
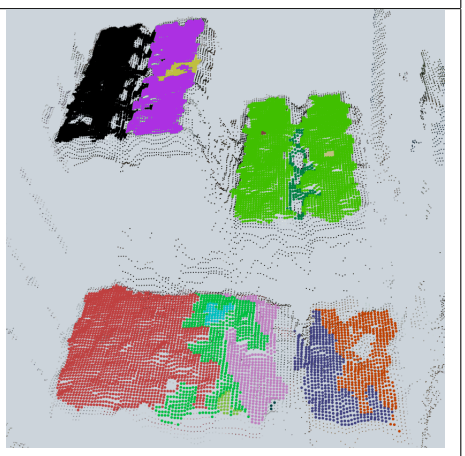
Plane extraction	
Point cloud	Normal vectors
	
Clusters	Planes
	

Table 4.18: Plane extraction process applied to a roller container filled with different objects: two stacks of beverage cans, a bag of pet food and a carton box. In the upper-left corner the original point cloud. In the upper-right corner the estimated normal vectors: the red ones are filtered out according to line 2 of Algorithm 6. In the bottom-left corner the remaining points are divided in clusters, each color corresponds to a new cluster. The bottom-right corner shows the cluster that pass the post-processing test. The sloped cluster at the rightmost edge of the pet-food bag is discarded, as well as the clusters that separate the two columns of cans in the lower layer.

Obstacles


Table 4.19: Obstacle estimation (in red). In grey the obstacle volume of the roller container, its pose and volume is known.

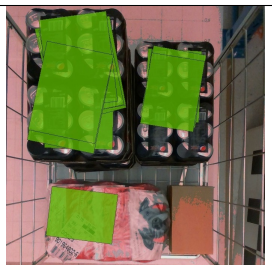
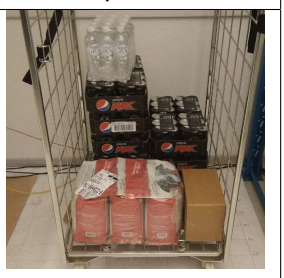
Planning - water bottles		
Planned placements	Example	Real placement
		

Table 4.20: Placement for a water bottle package. On the left, the footprint of the bounding box of the package is plotted on the scene in several planned poses. The right image is the manual reproduction of the placement in the central image.

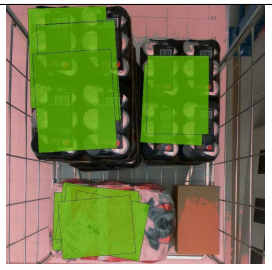
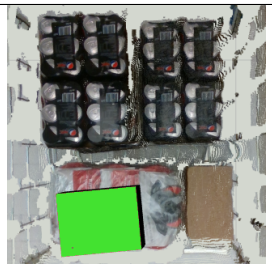
Planning - box		
Planned placements	Example	Real placement
		

Table 4.21: Placement for a box. Placement for a box. On the left, the footprint of the bounding box of the package is plotted on the scene in several planned poses. The right image is the manual reproduction of the placement in the central image.

4.2.3 Clutter

Finally, we create a highly cluttered scene with several objects (Table 4.22-4.23) and we show that it is still possible to find a placement for the box (Table 4.24). The perception module is able to find planar clusters in this highly cluttered an doisy environment: namely it detects the top surface of the bag of pet food, of the toolbox, of the can package and of the box in the bottom left corner.

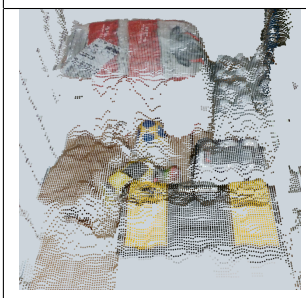
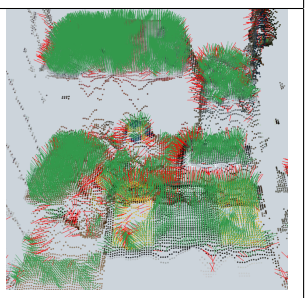
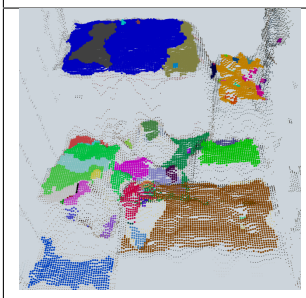
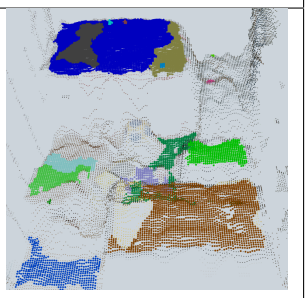
Plane extraction	
Point cloud	Normal vectors
	
Clusters	Planes
	

Table 4.22: Plane extraction process applied to a roller container filled with different objects. In the upper-left corner the original point cloud. In the upper-right corner the estimated normal vectors: the red ones are filtered out according to line 2 of Algorithm 6. In the bottom-left corner the remaining points are divided in clusters, each color corresponds to a new cluster. The bottom-right corner shows the cluster that pass the post-processing test.

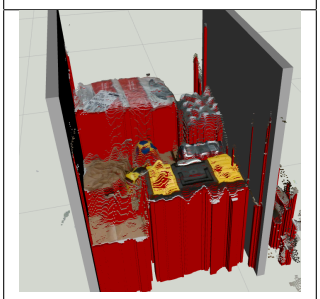
Obstacles


Table 4.23: Obstacle estimation (in red). In grey the obstacle volume of the roller container, its pose and volume is known.

Planning - box		
Planned placements	Example	Real placement
		

Table 4.24: Placement for a box. On the left several proposed placement poses, in the center and right picture the realization of one of them.

Chapter 5

Discussion

In this section, we discuss the results of the experiments in Chapter 5 and highlight the main findings and limitations.

5.1 Plane Extraction

The experiments in Section 4 show that the plane extraction algorithm recognizes planar clusters and discards surfaces that are not planar for the different objects that we have employed. A meaningful case is the last example in Table 4.22. The scene is highly cluttered and the data is particularly noisy, nonetheless the plane extraction algorithm succeeds in finding planar surfaces on the bag of pet food, the toolbox, the cans and the boxes. Moreover it does discard the surfaces belonging to the ball and the toy car in the roller-container.

In order to be successful, the plane extraction algorithm has to be able to tackle the source of erroneous depth estimation presented in Section 3.1. In table 4.1 and 4.13 it is possible to observe the local error in depth estimate at homogeneously colored surfaces. From these examples it is clear that noise is not spatially independent, and a wrong estimate at pixel i, j affects the estimates at the nearby pixels as well. Therefore planar surfaces present several local bumps. In both cases in Tables 4.1 and 4.13 the extraction algorithm does recognize the surfaces as planar, and local bumps are integrated in bigger overall-planar clusters. Two different clusters

are created on the higher layer of boxes in Table 4.13. This is due to the fact that none of the pixels on the border between the two clusters are perceived as being at similar heights, hence there is no evidence that the two clusters are connected according to proximity condition in line 4 of Algorithm 6.

A challenging case is the case of soft surfaces like the bag of pet-food in Table 4.7. This surface is not rigid, hence it has small local deformations and it is slightly curved on the sides. The algorithm is able to assign most points in the surface to the same planar cluster. Few small clusters that do not satisfy the proximity condition (line 4 of Algorithm 6) are isolated from the main cluster.

Another very challenging scenario is the case of water bottles. The depth estimate in case of translucent materials like the plastic film that wraps the water bottles (Table 4.10) is affected by the light conditions. In most of the cases the sensor perceives correctly the center of the bottles cups. The data in the space between the cups measures the depth of the bottles, rather than the plastic film. In the example in Table 4.10, the package in the upper-right corner of the image is affected less by light reflection, and the algorithm manages to find the planar cluster that connects the bottles cups.

Overall the results demonstrate that the plane extraction algorithm is able to tackle small local deformations, both in the case of sensor noise, or partially planar surfaces as in the case of non-rigid bags. Some sensor limitations such as consistent noise or significant deformations (as the rightmost edge of the package of cans in Table 4.4) cannot be mitigated in absence of prior knowledge about the objects in the scene.

5.2 Integration with the Placement Planner

The integration with the placement planner shows promising results. The computed placements do satisfy the reachability, no-collision and support constraints of the problem. No faulty placement were ever computed, and the simple obstacle estimation strategy used seems to be sufficient to guarantee feasible placements.

It is always possible to compute placements where enough planar space is available on the scene. Moreover, it is possible to compute placements even in the case that very small planar clusters are found, as the in the experiment with bottles (Table 4.10).

Chapter 6

Conclusion

In this thesis, we addressed the problem of automating the placement of known objects in an unknown and unstructured environment such as partially-filled containers. To do this, we used an RGB-D sensor to perceive the working scene, and we extracted planar surfaces as candidate placement regions. Then, we made use of the algorithm developed in [9] to find a suitable placement pose.

Under the hypothesis of dealing with rigid objects and a static environment, our experiments show that given enough space, the overall system succeeds to find stable placements for simple objects like boxes. Overall, this work constitutes a step towards the automation of placement tasks.

6.1 Possible extensions

To develop this work we relied on some simplification. For example, we assume that the pose and volume of the target container is known. In some placement scenarios, this is not the case, and a container-localization step must be included.

Moreover, we assume that a model of the object that has to be placed is available, or we approximate box-shaped objects with their bounding boxes when needed. If a model of the object is not available, an extension could be to estimate the object shape from perceived data. https://doi.org/10.1007/s11042-018-6912-6

Additionally, we consider a region of interest at the center of the camera field of

view, which allow us to make use of a simple obstacles volume estimation method. If a wider area has to be considered, a more sophisticated volume estimation method has to be applied, possibly integrated over time [13].

A further improvement could be to design a placement objective function that aims at producing placements that are favorable to the goal of filling up the roller container tightly. Alternatively, such a mapping could be learnt empirically, possibly through synthetic data, similarly to what was done in [48] or [49], where the authors use physics simulators to learn robust grasps from RGB-D images.

Bibliography

[1] Y. Lu, «Industry 4.0: A survey on technologies, applications and open research issues», *Journal of Industrial Information Integration*, vol. 6, pp. 1–10, Jun. 2017, ISSN: 2452414X. DOI: 10.1016/j.jii.2017.04.005. [Online]. Available: https://linkinghub.elsevier.com/retrieve/pii/S2452414X17300043 (visited on 01/24/2020).

[2] T. Ketelaars and E. van de Plassche, «An Industrial Solution to Automated Item Picking», in *Automation in Warehouse Development*, R. Hamberg and J. Verriet, Eds., London: Springer London, 2012, pp. 105–115, ISBN: 978-0-85729-968-0. DOI: 10.1007/978-0-85729-968-0_8. [Online]. Available: https://doi.org/10.1007/978-0-85729-968-0_8 (visited on 09/06/2019).

[3] *Robot Challenges - ICRA 2015*, http://icra2015.org/conference/robot-challenges. (visited on 01/25/2020).

[4] H.-Y. Kuo, H.-R. Su, S.-H. Lai, and C.-C. Wu, «3D object detection and pose estimation from depth image for robotic bin picking», in *2014 IEEE International Conference on Automation Science and Engineering (CASE)*, Taipei: IEEE, Aug. 2014, pp. 1264–1269, ISBN: 978-1-4799-5283-0 978-1-4799-5282-3. DOI: 10.1109/CoASE.2014.6899489. [Online]. Available: http://ieeexplore.ieee.org/document/6899489/ (visited on 03/23/2020).

[5] R. D. Singh, A. Mittal, and R. K. Bhatia, «3D convolutional neural network for object recognition: A review», *Multimedia Tools and Applications*, vol. 78, no. 12, pp. 15 951–15 995, Jun. 2019, ISSN: 1380-7501, 1573-7721. DOI: 10.

1007/s11042-018-6912-6. [Online]. Available: http://link.springer.com/10.1007/s11042-018-6912-6 (visited on 09/19/2019).

[6] A. Sahbani, S. El-Khoury, and P. Bidaud, «An overview of 3D object grasp synthesis algorithms», *Robotics and Autonomous Systems*, Autonomous Grasping, vol. 60, no. 3, pp. 326–336, Mar. 1, 2012, ISSN: 0921-8890. DOI: 10.1016/j.robot.2011.07.016. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0921889011001485 (visited on 02/27/2020).

[7] J. Bohg, A. Morales, T. Asfour, and D. Kragic, «Data-Driven Grasp Synthesis - A Survey», *IEEE Transactions on Robotics*, vol. 30, no. 2, pp. 289–309, Apr. 2014, ISSN: 1552-3098, 1941-0468. DOI: 10.1109/TRO.2013.2289018. arXiv: 1309.2660. [Online]. Available: http://arxiv.org/abs/1309.2660 (visited on 02/27/2020).

[8] G. Du, K. Wang, and S. Lian, «Vision-Based Robotic Grasping from Object Localization, Pose Estimation, Grasp Detection to Motion Planning: A Review», *arXiv:1905.06658*, May 16, 2019. arXiv: 1905.06658 [cs]. [Online]. Available: http://arxiv.org/abs/1905.06658 (visited on 02/27/2020).

[9] J. A. Haustein, K. Hang, J. Stork, and D. Kragic, «Object Placement Planning and optimization for Robot Manipulators», *arXiv:1907.02555*, pp. 7417–7424, 2020. DOI: 10.1109/iros40897.2019.8967732. arXiv: 1907.02555.

[10] S. Thrun, «Probabilistic robotics», *Communications of the ACM*, vol. 45, no. 3, pp. 221–242, Mar. 1, 2002, ISSN: 00010782. DOI: 10.1145/504729.504754. [Online]. Available: http://portal.acm.org/citation.cfm?doid=504729.504754 (visited on 03/23/2020).

[11] R. Hadsell, J. A. Bagnell, D. F. Huber, and M. Hebert, «Accurate rough terrain estimation with space-carving kernels.», in *Robotics: Science and Systems*, vol. 2009, 2009.

[12] R. Triebel, P. Pfaff, and W. Burgard, «Multi-Level Surface Maps for Outdoor Terrain Mapping and Loop Closing», in *2006 IEEE/RSJ International Conference on Intelligent Robots and Systems*, Oct. 2006, pp. 2276–2282. DOI: 10.1109/IROS.2006.282632.

[13] A. Hornung, K. M. Wurm, M. Bennewitz, C. Stachniss, and W. Burgard, «OctoMap: An efficient probabilistic 3D mapping framework based on octrees», *Autonomous Robots*, vol. 34, no. 3, pp. 189–206, Apr. 1, 2013, ISSN: 1573-7527. DOI: 10.1007/s10514-012-9321-0. [Online]. Available: https://doi.org/10.1007/s10514-012-9321-0 (visited on 12/02/2019).

[14] Wikipedia, *Octree*, https://en.wikipedia.org/wiki/Octree#/media/File:Octree2.svg. (visited on 01/25/2020).

[15] J. Baumgartl, T. Werner, P. Kaminsky, and D. Henrich, «A fast, GPU-based geometrical placement planner for unknown sensor-modelled objects and placement areas», in *2014 IEEE International Conference on Robotics and Automation (ICRA)*, Hong Kong, China: IEEE, May 2014, pp. 1552–1559, ISBN: 978-1-4799-3685-4. DOI: 10.1109/ICRA.2014.6907058. [Online]. Available: http://ieeexplore.ieee.org/document/6907058/ (visited on 09/19/2019).

[16] K. Harada, T. Tsuji, K. Nagata, N. Yamanobe, and H. Onda, «Validating an Object Placement Planner for Robotic Pick-and-Place Tasks», *Robotics and Autonomous Systems*, vol. 62, no. 10, pp. 1463–1477, Oct. 1, 2014, ISSN: 0921-8890. DOI: 10.1016/j.robot.2014.05.014. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0921889014001092 (visited on 08/13/2019).

[17] M. J. Schuster, J. Okerman, H. Nguyen, J. M. Rehg, and C. C. Kemp, «Perceiving Clutter and Surfaces for Object Placement in Indoor Environments», in *2010 10th IEEE-RAS International Conference on Humanoid Robots*, Dec. 2010, pp. 152–159. DOI: 10.1109/ICHR.2010.5686328.

[18] Y. Jiang, M. Lim, C. Zheng, and A. Saxena, «Learning to Place New Objects in a Scene», *arXiv:1202.1694*, Feb. 8, 2012. arXiv: 1202.1694 [cs]. [Online]. Available: http://arxiv.org/abs/1202.1694 (visited on 08/17/2019).

[19] D. Holz, S. Holzer, R. B. Rusu, and S. Behnke, «Real-Time Plane Segmentation Using RGB-D Cameras», in *RoboCup 2011: Robot Soccer World Cup XV*, T. Röfer, N. M. Mayer, J. Savage, and U. Saranlı, Eds., ser. Lecture Notes in

Computer Science, Springer Berlin Heidelberg, 2012, pp. 306–317, ISBN: 978-3-642-32060-6.

[20] A.-V. Vo, L. Truong-Hong, D. F. Laefer, and M. Bertolotto, «Octree-Based Region Growing for Point Cloud Segmentation», *ISPRS Journal of Photogrammetry and Remote Sensing*, vol. 104, pp. 88–100, Jun. 1, 2015, ISSN: 0924-2716. DOI: 10.1016/j.isprsjprs.2015.01.011. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0924271615000283 (visited on 08/17/2019).

[21] Z. Dong, Y. Gao, J. Zhang, Y. Yan, X. Wang, and F. Chen, «HoPE: Horizontal Plane Extractor for Cluttered 3D Scenes», *Sensors (Switzerland)*, vol. 18, no. 10, 2018. DOI: 10.3390/s18103214.

[22] X. Wang, L. Zou, X. Shen, Y. Ren, and Y. Qin, «A region-growing approach for automatic outcrop fracture extraction from a three-dimensional point cloud», *Computers & Geosciences*, vol. 99, pp. 100–106, Feb. 2017, ISSN: 00983004. DOI: 10.1016/j.cageo.2016.11.002. [Online]. Available: https://linkinghub.elsevier.com/retrieve/pii/S0098300416306471 (visited on 09/13/2019).

[23] Y. Fan, M. Wang, N. Geng, D. He, J. Chang, and J. J. Zhang, «A self-adaptive segmentation method for a point cloud», *The Visual Computer*, vol. 34, no. 5, pp. 659–673, May 2018, ISSN: 0178-2789, 1432-2315. DOI: 10.1007/s00371-017-1405-6. [Online]. Available: http://link.springer.com/10.1007/s00371-017-1405-6 (visited on 09/13/2019).

[24] S. Filin and N. Pfeifer, «Segmentation of airborne laser scanning data using a slope adaptive neighborhood», *ISPRS Journal of Photogrammetry and Remote Sensing*, vol. 60, no. 2, pp. 71–80, Apr. 2006, ISSN: 09242716. DOI: 10.1016/j.isprsjprs.2005.10.005. [Online]. Available: https://linkinghub.elsevier.com/retrieve/pii/S0924271605000638 (visited on 09/16/2019).

[25] S. Holzer, R. B. Rusu, M. Dixon, S. Gedikli, and N. Navab, «Adaptive neighborhood selection for real-time surface normal estimation from organized point cloud data using integral images», in *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, Vilamoura-Algarve, Portugal: IEEE, Oct.

2012, pp. 2684–2689, ISBN: 978-1-4673-1736-8 978-1-4673-1737-5 978-1-4673-1735-1. DOI: 10.1109/IROS.2012.6385999. [Online]. Available: http://ieeexplore.ieee.org/document/6385999/ (visited on 01/29/2020).

[26] J. Berkmann and T. Caelli, «Computation of surface geometry and segmentation using covariance techniques», *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 16, no. 11, pp. 1114–1116, Nov./1994, ISSN: 01628828. DOI: 10.1109/34.334391. [Online]. Available: http://ieeexplore.ieee.org/document/334391/ (visited on 09/13/2019).

[27] T. Rabbani, F. Van Den Heuvel, and G. Vosselman, «Segmentation of Point Clouds Using Smoothness Constraint», *International Archives of Photogrammetry, Remote Sensing and Spatial Information Sciences*, vol. 36, no. 5, pp. 248–253, 2006.

[28] M. Pauly, M. Gross, and L. Kobbelt, «Efficient simplification of point-sampled surfaces», in *IEEE Visualization, 2002. VIS 2002.*, Boston, MA, USA: IEEE, 2002, pp. 163–170, ISBN: 978-0-7803-7498-0. DOI: 10.1109/VISUAL.2002.1183771. [Online]. Available: http://ieeexplore.ieee.org/document/1183771/ (visited on 09/13/2019).

[29] J. M. Biosca and J. L. Lerma, «Unsupervised robust planar segmentation of terrestrial laser scanner point clouds based on fuzzy clustering methods», *ISPRS Journal of Photogrammetry and Remote Sensing*, vol. 63, no. 1, pp. 84–98, Jan. 2008, ISSN: 09242716. DOI: 10.1016/j.isprsjprs.2007.07.010. [Online]. Available: https://linkinghub.elsevier.com/retrieve/pii/S0924271607000809 (visited on 09/17/2019).

[30] A. Rodriguez and A. Laio, «Clustering by fast search and find of density peaks», *Science*, vol. 344, no. 6191, pp. 1492–1496, Jun. 27, 2014, ISSN: 0036-8075, 1095-9203. DOI: 10.1126/science.1242072. pmid: 24970081. [Online]. Available: http://science.sciencemag.org/content/344/6191/1492 (visited on 09/15/2019).

[31] Y. Ge, H. Tang, D. Xia, L. Wang, B. Zhao, J. W. Teaway, H. Chen, and T. Zhou, «Automated measurements of discontinuity geometric properties from a

3D-point cloud based on a modified region growing algorithm», *Engineering Geology*, vol. 242, pp. 44–54, Aug. 2018, ISSN: 00137952. DOI: 10.1016/j.enggeo.2018.05.007. [Online]. Available: https://linkinghub.elsevier.com/retrieve/pii/S0013795217317477 (visited on 09/13/2019).

[32] A. Nguyen and B. Le, «3D Point Cloud Segmentation: A Survey», in *2013 6th IEEE Conference on Robotics, Automation and Mechatronics (RAM)*, Nov. 2013, pp. 225–230. DOI: 10.1109/RAM.2013.6758588.

[33] P. V. C. Hough, «Method and Means for Recognizing Complex Patterns», U.S. Patent 3069654A, Dec. 18, 1962. [Online]. Available: https://patents.google.com/patent/US3069654A/en (visited on 09/17/2019).

[34] D. Borrmann, J. Elseberg, K. Lingemann, and A. Nüchter, «The 3D Hough Transform for plane detection in point clouds: A review and a new accumulator design», *3D Research*, vol. 2, no. 2, p. 3, Jun. 2011, ISSN: 2092-6731. DOI: 10.1007/3DRes.02(2011)3. [Online]. Available: http://link.springer.com/10.1007/3DRes.02(2011)3 (visited on 09/16/2019).

[35] R. Hulik, M. Spanel, P. Smrz, and Z. Materna, «Continuous Plane Detection in Point-Cloud Data Based on 3D Hough Transform», *Journal of Visual Communication and Image Representation*, Visual Understanding and Applications with RGB-D Cameras, vol. 25, no. 1, pp. 86–97, Jan. 1, 2014, ISSN: 1047-3203. DOI: 10.1016/j.jvcir.2013.04.001. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S104732031300062X (visited on 09/03/2019).

[36] X. Leng, J. Xiao, and Y. Wang, «A Multi-Scale Plane-Detection Method Based on the Hough Transform and Region Growing», *Photogrammetric Record*, vol. 31, no. 154, pp. 166–192, 2016. DOI: 10.1111/phor.12145.

[37] M. A. Fischler and R. C. Bolles, «Random sample consensus: A paradigm for model fitting with applications to image analysis and automated cartography», *Communications of the ACM*, vol. 24, no. 6, pp. 381–395, Jun. 1, 1981, ISSN: 00010782. DOI: 10.1145/358669.358692. [Online]. Available: http:

//portal.acm.org/citation.cfm?doid=358669.358692 (visited on 09/17/2019).

[38] R. Schnabel, R. Wahl, and R. Klein, «Efficient RANSAC for Point-Cloud Shape Detection», *Comput. Graph. Forum*, vol. 26, pp. 214–226, 2007. DOI: 10.1111/j.1467-8659.2007.01016.x.

[39] P. Torr and A. Zisserman, «MLESAC: A New Robust Estimator with Application to Estimating Image Geometry», *Computer Vision and Image Understanding*, vol. 78, no. 1, pp. 138–156, Apr. 2000, ISSN: 10773142. DOI: 10.1006/cviu.1999.0832. [Online]. Available: https://linkinghub.elsevier.com/retrieve/pii/S1077314299908329 (visited on 09/18/2019).

[40] H. Fu, D. Cohen-or, G. Dror, and A. Sheffer, «Upright Orientation of Man-Made Objects», *ACM Trans. Graphics*, pp. 1–7, 2008.

[41] *Depth Camera D435*, https://www.intelrealsense.com/depth-camera-d435/. (visited on 01/27/2020).

[42] *Stereo depth cameras for mobile phones*, https://dev.intelrealsense.com/docs/stereo-depth-cameras-for-phones. (visited on 02/24/2020).

[43] *Projectors for D400 Series Depth Cameras*, https://dev.intelrealsense.com/docs/projectors. (visited on 01/28/2020).

[44] I. RealSense™, *Intel® RealSense™ SDK 2.0*, https://github.com/IntelRealSense/librealsense. (visited on 02/24/2020).

[45] *Camera Calibration and 3D Reconstruction — OpenCV 2.4.13.7 Documentation*, https://docs.opencv.org/2.4/modules/calib3d/doc/camera_calibration_and_3d_reconstruction.html. (visited on 02/24/2020).

[46] *Point Cloud Library (PCL): PCL API Documentation*, http://docs.pointclouds.org/1.8.1/index.html. (visited on 09/10/2019).

[47] J. L. Bentley, «Multidimensional binary search trees used for associative searching», *Communications of the ACM*, vol. 18, no. 9, pp. 509–517, Sep. 1, 1975, ISSN: 00010782. DOI: 10.1145/361002.361007. [Online]. Available: http:

//portal.acm.org/citation.cfm?doid=361002.361007 (visited on 01/29/2020).

[48] E. Johns, S. Leutenegger, and A. J. Davison, «Deep Learning a Grasp Function for Grasping under Gripper Pose Uncertainty», in *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Oct. 2016, pp. 4461–4468. DOI: 10.1109/IROS.2016.7759657.

[49] J. Mahler, J. Liang, S. Niyaz, M. Laskey, R. Doan, X. Liu, J. A. Ojea, and K. Goldberg, «Dex-Net 2.0: Deep Learning to Plan Robust Grasps with Synthetic Point Clouds and Analytic Grasp Metrics», Mar. 27, 2017. arXiv: 1703.09312 [cs]. [Online]. Available: http://arxiv.org/abs/1703.09312 (visited on 09/24/2019).