Alma Mater Studiorum – Università di Bologna

DOTTORATO DI RICERCA IN

Ingegneria elettronica, telecomunicazioni e tecnologie dell'informazione

Ciclo 29

Settore Concorsuale: Area 09 - Ingegneria industriale e dell'informazione > 09/G - Ingegneria dei sistemi e bioingegneria > 09/G1 Automatica

Settore Scientifico Disciplinare: Area 09 - Ingegneria industriale e dell'informazione > ING-INF/04 Automatica

Scene Mapping and Understanding by Robotic Vision

Candidato: Daniele De Gregorio

Coordinatore Dottorato

Vanelli Coralli, Alessandro

Supervisore

Di Stefano, Luigi

Co-Supervisore

Melchiorri, Claudio

Esame finale anno 2018

I would like to dedicate this thesis to my Father who is so far away from me, to my Mother who is so close to me, to the family in which i was born and finally to Giulia the family i choose.

Table of contents

Introduction

Although life has existed for several billion years, animals advanced enough to make good use of vision have only been around for little more than half a billion years (Land and Nilsson, 2012). Similar fate has concerned – somehow proportionally – *Computer Vision* and *Robotics*. The first mechanical *Automaton* concept was found in a Chinese text written in the 3rd century BC (Needham, 1974), while *Computer Vision* began in the late 1960*s*. Therefore, visual perception applied to machines (i.e. the *Machine Vision*) is a young and exciting alliance.

When robots came in, however, the new field of *Robotic Vision* was born, and these two terms begun to be erroneously interchanged. In short, we can say that Machine Vision is an engineering domain, which concern the industrial use of Vision for tasks like quality control, automatic inspection, counting systems, rejection machines or robot guidance. The Robotic Vision, instead, is a research field that tries to incorporate robotics aspects in computer vision algorithms. Visual Servoing (Chaumette and Hutchinson, 2007), for example, is one of the problems that cannot be solved by computer vision only.

Accordingly, a large part of this work deals with boosting popular Computer Vision techniques by exploiting robotics: e.g. the use of *kinematics* to localize a vision sensor, mounted as the robot end-effector, in order to realize a kind of *moving eye*. Following the terminology used by Muis and Ohnishi (2005), this work concerns the study of the *eye-on-hand* configuration, where the robot behaves as hand and the vision sensor as eye. The *eye-on-hand* configuration largely solves the *Camera Localization* problem that is one of the largest headache in applications like *Augmented Reality* or *Environment Mapping*, because through *kinematics* we can compute the exact 6 DoF pose of every artifact rigidly attached to a robotic agent. At the same time, however, this kind of configuration allows not only to know the location of the *eye* but also to explore the environment in a controlled manner: e.g. so as to move the sensor if something is occluding the visual field.

The remainder of this thesis is dedicated to the counterparty, i.e. the use of computer vision to solve real robotic problems like grasping objects or navigate avoiding obstacles. In particular, Part I concerns the *Mapping*, namely the ability of a robotic agent to build

a virtual representation (the model) of the surrounding environment. Chapter 1 presents a brief survey about data structures most widely used in robotics to create a *map* of the environment, thereby providing the necessary background to understand the rest of the first part. Chapter 2 describes an heuristic approach through which is possible to grasp unknown object by reasoning over a map of the workspace only, without the need for an object detection stage. Chapter 3 introduces *SkiMap*, a completely new sparse data structure, designed for multi-core CPUs, used both for robotic mapping and as a general purpose 3D spatial index, which is faster, for certain tasks, compared to similar solutions. Chapter 4 exploits the aforesaid *SkiMap* data structure to prove how it is possible to pursue Object Detection and Pose Estimation by reasoning over a sparse 3D map of features in real-time on a CPU.

Part II deals with the *Machine Teaching* field, described by Simard et al. (2017), by introducing a new method to train an artificial intelligence by deploying Robotic Vision[1].

Notations

In this work we will intensively use the notation ${}^{A}\mathbf{T}_{B} \in \mathbb{R}^{4\times4}$ to define a 3D reference frame (briefly RF) B expressed in the base A, or the equivalent roto-translation operator. For this reason, for example, the symbol ${}^{0}\mathbf{T}_{camera}$ will mean the RF of the camera in the *zero* reference frame. Same consideration will be applied to other operators like rotations ${}^{A}\mathbf{R}_{B}$, or geometric primitives like vectors ${}^{A}\mathbf{p}_{B}$. This notation allows the correct composition of spatial transformations by matching previous right-subscript with the next left-subscripts, for example:

$$ {}^{A}\mathbf{T}_{D} = {}^{A}\mathbf{T}_{B} \cdot {}^{B}\mathbf{T}_{C} \cdot {}^{C}\mathbf{T}_{D} \tag{1} $$

[1] Filing a patent application concerning this technique is currently under consideration by the University Of Bologna.

Part I

Mapping with a Robot

Chapter 1

Mapping: state of the art

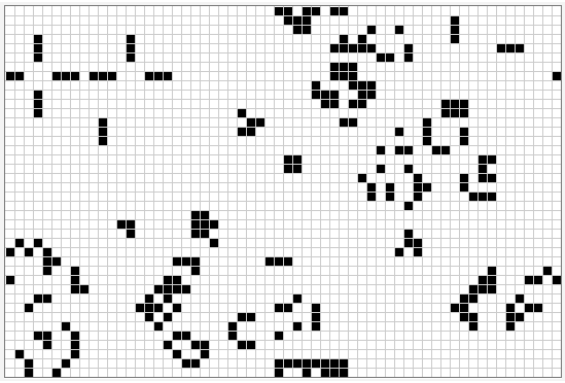
1.1 Mapping and Robotics

Most robotic applications require a sufficiently rich perception of the environment in order to carry out high-level tasks *e.g.* manipulation with an industrial robot or navigation with a mobile one or, why not, both together. This kind of perception is realized through suitable sensors and algorithms. As for the former, laser rangefinders have traditionally been employed to capture a planar view (2D) of the surroundings, while visual sensors, and in particular RGB-D [1] cameras, are becoming more and more widespread on account of their potential to model the environment in 3D. In the space of algorithms, instead, we can explore several solutions grouping them by their level of representation required by the specific robotic task.

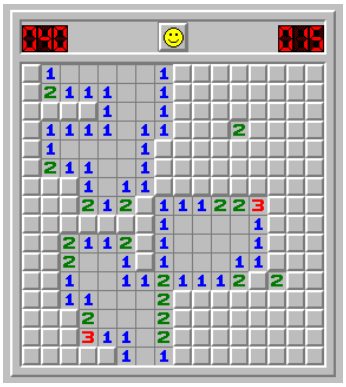
For what concern mobile robots, as described in Thrun et al. (2005), the classical mapping approach for the navigation task is the 2D occupancy grid. Accordingly, sensors measurements (coming, typically, from a planar laser scanner) are fused into a 2D Grid wherein each *Tile* (i.e. a square chunk of the space) contains an occupancy probability which can be interpreted as the likelihood that the tile belongs to an obstacle. Many robot navigation systems, often referred to as *Grid-Based SLAM* (Simultaneous Localization And Mapping), rely on this 2D occupancy grid as in the popular work of Grisetti et al. (2007) that can be considered as a baseline for robot navigation, provided that many commercial solutions employ them. Yet, planar sensing and related 2D mapping may not be reliable enough due to defective reconstruction of the environment, e.g. when dealing with a MAV (micro aerial vehicle) for indoor navigation, or, more generally with any Mobile Robot that cannot be modeled as a bi-dimensional agent[2]. A conceptually straightforward approach to pursue 3D mapping

[1] https://en.wikipedia.org/wiki/Kinect , https://en.wikipedia.org/wiki/PrimeSense
[2] http://www.willowgarage.com/pages/pr2/overview

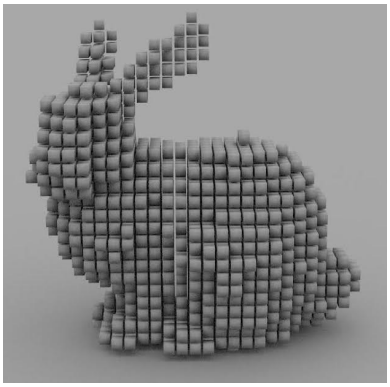
when deploying sensors, e,g. visual sensors, capable of delivering 3D measurements would consist in extending the occupancy map to a 3D Volume by cutting the space into *Voxels* (i.e. small cubes), like in the early work of Roth-Tabak and Jain (1989), each of which storing the probability for an obstacle to be located therein. More generally the information stored in a *Tile* or in a voxel is application-dependent, for example in the mapping framework of Hornung et al. (2013) a *Voxel* can contain information like $\{Occupied, Free, Unexplored\}$; in the *6-DoF SLAM* system of Endres et al. (2014), instead, each voxel contains also Color information.



(a) 2D Map from Game Of Life



(b) 2D Map from Minesweeper



(c) 3D Map of the Standford Bunny

(d) 3D Map from Minecraft

Table 1.1 (a) A simple 2D Map from the Game Of Life of Conway (1970): here each *Tile* stores only boolean information $\{1 = occupied; 0 = free\}$; (b) A more complex 2D Map from the windows game *Minesweeper*: hear each *Tile* contains at least 11 values $\{0 = void; [1,8] = adiacency; 9 = bomb; 10 = unexplored; 11 = flag\}$; (c) A simple 3D Volume representing the Standford Bunny: also here the volume stores only boolean information $\{1 = occupied; 0 = free\}$; (d) A complex 3D Volume from the popular game Minecraft: each voxel here can map up to 300 different values (terrain, water, wood etc.).

Table 1.1 depicts popular 2D and 3D mapping strategies – not directly linked with robotic but – very useful to understand the trade-off between the richness of

representation and the memory footprint: for example in Table 1.1a and Table 1.1c, where the stored information is boolean, just 1 *bit* is necessary for each *Tile* or *Voxel*; in richer maps like Table 1.1d, where we need to store 300 distinct values approximately, we need at least $\lceil log_2(300) \rceil = 9$ *bits*. It is rather intuitive that the more complex is the representation of the environment, the larger is the memory footprint of the model, but it is important to say that also choosing the correct data structure, on the basis of which our 2D/3D model is built, may affect – dramatically – the performances of our representation. For this reason, in the reminder of this chapter we will examine the state of the art of both *Data Structure* (Section 1.2) and *Environment Representation* (Section 1.3) in the robotic field.

For what concern the *Data Structure*, it is important to say that several authors investigated the use of non-cubic subdivision of the space, like Ryde and Brünig (2009) and Fridovich-Keil et al. (2017); but this topic is not directly treated in this work.

1.2 Data Structures

As mentioned earlier, the richness of our environment representation affects heavily the memory footprint of our model but the data structure which contains them is equally undermining the efficiency. For this reason in the following sections we will examine the most important data structures deployed as fundamental bricks for several mapping solutions in the robotic vision field, More precisely, in Section 1.2.1 we will introduce the simplest one that represents the foundations in all our considerations; in Section 1.2.2 we will describe the *Octree*, one of the most used sparse data structure not only in the robotic field but in the computer graphics field in general; in Section 1.2.3 we will present the *Voxel Hashing* technique, that is in a better position in terms of memory efficiency and in Section 1.2.4 we will introduce briefly our novel data structure referred as *SkiMap* (De Gregorio and Di Stefano, 2017) which will be explained in detail in Chapter 3.

1.2.1 Dense 3D Volume

The model of the environment through a Dense 3D Volume is represented by a 3D volumetric grid, superimposed over the entire space, with cubic voxels as base unit. As depicted in Figure 1.1 the 3D space is discretized in small polyhedric units with side r. This kind of 3D Volume cannot be not infinite so we need to define a maximum extension for each dimension of that grid: for the sake of simplicity we can use a cubic volume with side of $s = s_s = s_y = s_z$. So it is simple to compute how many voxels belong to our environment model:

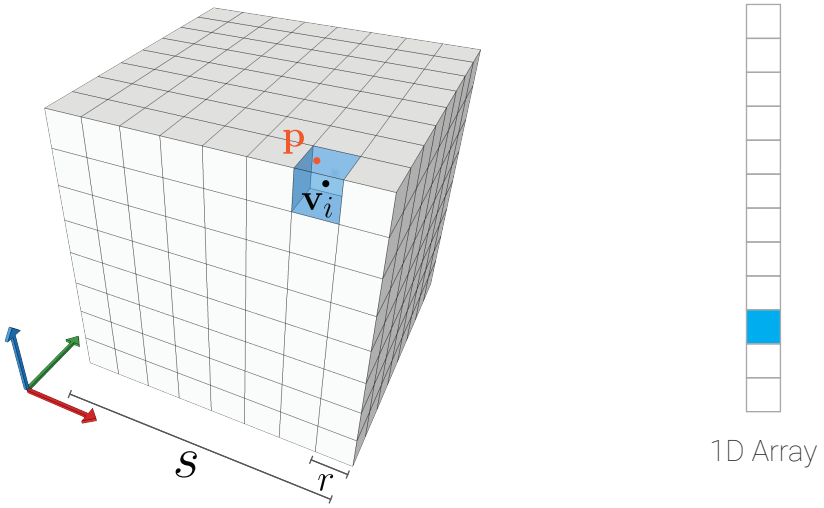


Fig. 1.1 A dense 3D Grid with side s and resolution of r: the entire volume is populated of voxels regardless of whether or not they map an occupied portion of the space (the blue voxel). The corresponding data structure usually is a simple 1D *Array*. The size of the array is $\frac{s^3}{r^3}$.

$$|\mathbb{V}| = \frac{s_x s_y s_z}{r^3} = \frac{s^3}{r^3} \tag{1.1}$$

where $\mathbb{V} = \{\mathbf{v}_i = \begin{bmatrix} ar\mathbf{u}_x & br\mathbf{u}_y & cr\mathbf{u}_z \end{bmatrix} \mid a,b,c \in [0,...,i_{\max}]\}$, with $i_{\max}$ the max allowed index, in each dimension, which complies, for example for the x axis, with the condition: $i_{\max} r\mathbf{u}_x < s_x < (i_{\max}+1)r\mathbf{u}_x$. So the coordinates of each voxel $\mathbf{v}_i$ are integer multiples of r given $\begin{bmatrix} \mathbf{u}_x & \mathbf{u}_y & \mathbf{u}_z \end{bmatrix}$ a basis of $\mathbb{R}^3$. Thereby setting out r and $\begin{bmatrix} \mathbf{u}_x & \mathbf{u}_y & \mathbf{u}_z \end{bmatrix}$ then the vector $\begin{bmatrix} a & b & c \end{bmatrix}$ indexes each voxel in the 3D lattice described by $\mathbb{V}$.

$$\mathbf{v}_i = r\begin{bmatrix} a & b & c \end{bmatrix}\begin{bmatrix} \mathbf{u}_x \\ \mathbf{u}_y \\ \mathbf{u}_z \end{bmatrix} = \mathbf{v}_{a,b,c} \tag{1.2}$$

realizing that just one linear index i is not enough to correctly identify a 3D voxel. More precisely there is a simple conversion between a linear index to a 3D index: $i = a + b(\frac{s}{r}) + c(\frac{s}{r})^2$; this conversion is useful whenever we decide to store our 3D Voxel Grid in a linear data structure like a common 1D *Array* (and *vice versa*).

Now we have a model able to transform – or, better, discretize – each random point $\mathbf{p} = \begin{bmatrix} p_x & p_y & p_z \end{bmatrix} \in \mathbb{R}^3$ with a voxel $\mathbf{v}_{a,b,c}$ with the equation:

$$\rho(\mathbf{p}) = \begin{bmatrix} a & b & c \end{bmatrix} = \lceil \frac{\begin{bmatrix} p_x & p_y & p_z \end{bmatrix}}{r} \rceil \tag{1.3}$$

discovering that $\rho : \mathbb{R} \mapsto \mathbb{Z}^+$ is actually a quantization of the space. In simple terms each point $\mathbf{p}$ will be mapped into the nearest voxel's center $\mathbf{v}_i$. In Figure 1.2, a continuous set of points belonging to the surface of the Standford Bunny 3D model are discretized over two 3D lattices with two different resolution r_1, r_2, showing how the discretization level affects the original model. An important consideration is required here: 3D features smaller that the resolution will disappear during the mapping procedure, as the bunny's eye; so when we design our mapping solution we need to be sure that our discretization process does not remove information of the environment necessary for the addressed high level task.

Thus, the Dense 3D Volume is a model with a very high memory footprint but also with a very rich geometrical content and the lowest time complexity in accessing voxels ($\mathcal{O}(1)$). Each voxel knows its neighborhood: *e.g.* the voxel $V_{a+1,b,c}$ is near the voxel $V_{a,b,c}$. On the other hand, however, the space complexity (Arora and Barak, 2009) of the data structure is $\mathcal{O}(n = \frac{s^3}{r^3})$ so it does not take into account occupied or void space but grows with the cube of the side of the mapped volume. To understand exactly what the memory footprint is let us consider an application similar to Table 1.1d, where we need to store in each voxel an integer (4 bytes) representing the voxel semantic type (*e.g.* terrain, road, wall etc.) and imagine to map an environment with a side of $s = 100m$ (100 meters) with a resolution of $r = 0.05m$ (5 centimeters) the memory usage will be $\mathbf{S} = 4\frac{100^3}{0.05^3} = 32GB$ (GigaBytes); this is surely not a solution for large-scale environment mapping but, thanks to the dense structure, is quite compliant with GPGPU architectures (Newcombe et al. (2011) used it for the very popular *KinectFusion* SLAM system).

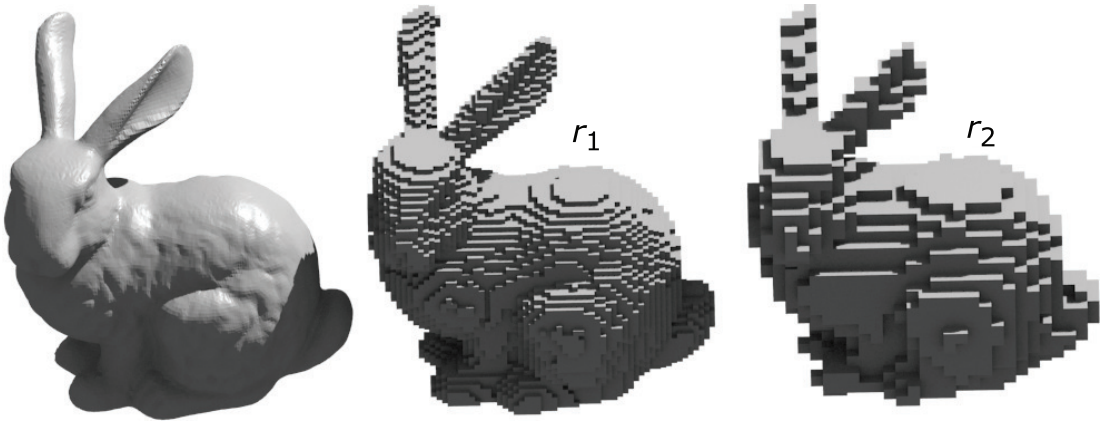


Fig. 1.2 The Standford Bunny in its original version and with the discretization over a 3D Voxel Grid with two different resolutions $r_1 < r_2$.

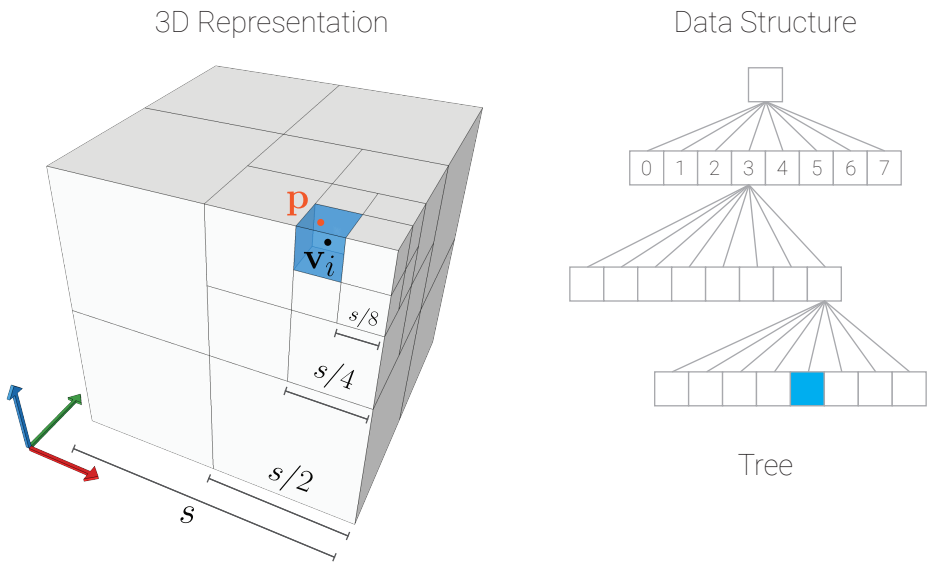


Fig. 1.3 An Octree graphical representation with its equivalent data structure. This model can be represented as a Tree with precisely 8 children for each middle node. Only filled leafs (blue cube) stores useful informations, everything else can be represented as simple pointer.

1.2.2 Octree

An octree, in contrast with the 3D Dense Volume, is a sparse data structure used for spatial subdivision of the 3D space. With reference to Figure 1.3, we can see the 3D representation of an octree with its equivalent data structure that is a simple 8-*ary* tree (a tree where each node has exactly 8 children). The Octree can be seen as a recursive partition of the space: the whole map is a big cube with side s subdivided in 8 smaller cubes (*octants*) with side $s/2$ and so on. More generally the size of each level $s_i = \frac{s_{i-1}}{2}$. With reference to Wilhelms and Van Gelder (1992) we can see this kind of data structure as a *Branch-on-Need* tree: we can effectively subdivide a region of the space only if that region contains information about environment. This technique allows us to save a lot of memory (space complexity) compared to a dense data structure at the expense – obviously – of the time complexity. In fact the simple addressing system seen in Section 1.2.1 can not be applied here because the *accessing* procedure of a tree is more complex: if we are searching for the position of a generic point $\mathbf{p}$ then at each level, starting from the root node (the biggest cube), we need to compute which of the eight children contains the query point. Addressing each children with a simple index $c_l \in \{0,\dots,7\}$ (*i.e.* the children at level l) we can build the entire path to a generic voxel $\mathbf{v}_i$ by an iterative search: $\mathbf{v}_i = \mathbf{v}_{c_0,c_1,\dots,c_d}$ where d is the max depth chosen for the octree model. In this case, the time complexity, for a random search performed through the tree, is *logarimic*: $\mathcal{O}(log(n)) = \mathcal{O}(d)$ and it's proportional – only – to the depth

of the representation. Although the computation of the exact space complexity of an octree depends on the sparsity of the environment, we can use a visual metaphor to understand the memory savings compared with the dense approach: looking at both Figure 1.1 and Figure 1.3 and considering that a white cube occupies the same ammount of memory despite its size, it is clear that a dense structure contains much more blocks than an octree. In order to quantify these savings, we can rely on the experiments carried-out by Hornung et al. (2013) which use the Octree as base data structure for their mapping framework *Octomap*: mapping the Freiburg Campus[3] with a volume of $s_x = 292m \times s_y = 167m \times s_z = 28m$ with a resolution $r = 0.1m$ the occupied memory is about $1.3Gb$ compared to the $5Gb$ of a dense grid (computed by Equation 1.1).

A distinctive feature of an Octree is the *multi-resolution* query: if we want to visit the whole tree structure up to the max depth d_{max} we will obtain the maximum richness of representation, but at the same time we want to stop our visit to the depth d_{max-1} we will obtain a less refined representation – saving time – of our map; to understand this we can look at Figure 1.2 where the rightmost bunny is the result of a visit down to a depth below the max depth that is represented instead by the middle bunny.

In conclusion with the Octree model we have reduced the memory footprint of our environment representation to the detriment of time efficiency and richness of geometrical information content (two adjacent voxels may have two different *octants* as parent resulting in an indirect relationship).

[3] http://ais.informatik.uni-freiburg.de/projects/datasets/fr360/

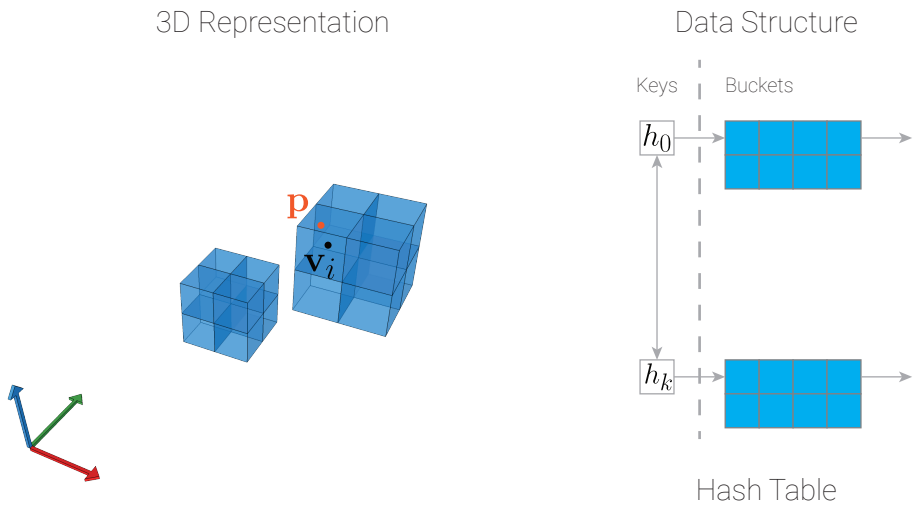


Fig. 1.4 A VoxelHashing 3D representation is completely void aside the occupied portion of the space (compare it with Figure 1.1 and Figure 1.3). The related data structure is an Hash Table where each entry is a Voxel Block (in this case $2 \times 2 \times 2$, but could also be just a single Voxel) list; the lateral list data structure (the right arrows pointing nothing) is mandatory to tackle the problem of the collisions in the *hashing* procedure (Bellare and Rogaway, 1993).

1.2.3 Voxelhashing

The basic idea of the *Voxel Hashing* approach is to avoid both the dense structure (Section 1.2.1) and the hierarchical structure (Section 1.2.2) reducing in this way both space and time complexity. Thus, as depicted in Figure 1.4, there is no superstructure around our target voxels and this result is achieved thanks to the *Spatial Hashing* technique well described by Teschner et al. (2003) or García et al. (2011) in their works. First of all we need to discretize the space like in Section 1.2.1, so for a random point $\mathbf{p}$ we can use Equation 1.3, chosing a target resolution r, to obtain a set of integer indices. For the sake of simplification, we can define this set of indices as $\begin{bmatrix} x & y & z \end{bmatrix}$ as if they were integer 3D coordinates. The spatial hashing idea is to design a function $H(x,y,z) = h \in \{0,...,n\}, n \in \mathbb{Z}^+$ in order to map our 3D integer coordinates to a linear index of an Hash Table, where our voxel data is stored. Obviously the function $H : \mathbb{Z}^3 \mapsto [0,n)$ (with $n \in Z^+$ is the size of the hash table) is a one-way function, that is, a function which cannot be inverted for the simple reason that the cardinality of the image of H is kept smaller than $\left|\mathbb{Z}^3\right|$. An example *hash* function could be:

$$H(x,y,z) = (xp_1 \oplus yp_2 \oplus zp_3) \; mod \; n \tag{1.4}$$

where p_1, p_2, p_3 are large prime numbers and $\oplus$ is the *bit-wise xor* operator (Teschner et al., 2003). Hence if we want to store information about a point $\mathbf{p}$ we first compute the related hash table index $h = H(\mathbf{p})$ then we retrieve the corresponding table value where are stored our custom data (*e.g.* Occupancy, Color etc.). This is a sort of packing procedure able to compress a 3D space into a linear space, but with several drawbacks. First of all the function $H(\cdot)$ – that is not perfect – could result in a collision ($H(\mathbf{p}_1) = H(\mathbf{p}_2)$ with $\mathbf{p}_1 \neq \mathbf{p}_2$) increasing in this way the time complexity to retrieve our correct data because we need to store in the same entry of the hashtable more than one voxel and, in addition, is not enough to save only environmental information in a voxel but also geometrical information (*i.e.* its implicit 3D coordinate) is needed, in contrast with a Dense Volume or an Octree where the accessing procedure of a voxel implies directly the knowledge of its 3D position in the space (the Equation 1.3 is – lossy – invertible). Another drawback is the loss of the geometrical information content in the hash table: two voxel of the same entry in the hash table could be very distant in the 3D space because the function $H(\cdot)$ is a sort of *Random Oracle* (Bellare and Rogaway, 1993) that remove any spatial relation intrinsically linked with voxel integer coordinates.

To summarise, the *VoxelHashing* technique is the best data structure with regard to the memory footprint for very large environment mapping and for well-designed *hash* function is also very good in the *access* operation because, on the assumption that collisions are very infrequent, the cost is $\mathcal{O}(1)$, i.e. the same as a Dense Grid. The tradeoff here is the lack of possibility to perform a spatial query over the data structure: for example if we want to know which voxel are in the neighborhood of a target query point the only chance is to perform a *Brute Force* – costly – search over the hashtable. In the intereseting work of Nießner et al. (2013), where the richness of reconstruction is preferable to the geometrical structure of the environment, the Voxel Hashing technique was well exploited for the first time for a very comprehensive reconstruction of several large scale scenes in real-time with a GPU, where space (complexity) is more important than time.

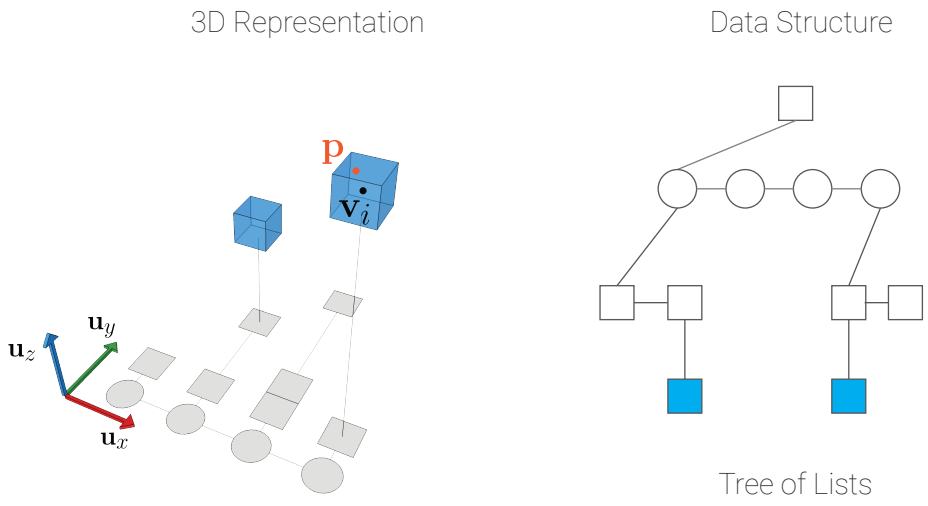


Fig. 1.5 A SkiMap 3D graphic representation. Compare it with Figure 1.1, Figure 1.3 and Figure 1.4. The related data structure is a *tree* with max depth 3 and where real voxels are represented by the leaves. Inner nodes of the structure represent the projection of the 3D information onto the $x-y$ plane.

1.2.4 Skimap

We proposed a completely new sparse data structure (De Gregorio and Di Stefano, 2017), dubbed *SkiMap*, similar in performance to the *Octree* model but with several key aspects that make it better in several robotic-oriented tasks. The overall *SkiMap* method will be described in detail in Chapter 3, instead here we will focus on its data structure so as to compare it with the other solutions described in this chapter.

Figure 1.5 represents the hierarchical 3D and 2D superstructure of *SkiMap* hidden behind voxel data. The complete structure is a *Tree of List* where each list is – conceptually – parallel to one of the vectors $\begin{bmatrix}\mathbf{u}_x & \mathbf{u}_y & \mathbf{u}_z\end{bmatrix}$ forming a basis of $\mathbb{R}^3$. To be more precise, there is just one list along $\mathbf{u}_x$, than for each of its nodes we have many lists along $\mathbf{u}_y$ and iteratively the same procedure for the nodes along $\mathbf{u}_z$. Also with this approach, the first step to store information about a random point $\mathbf{p}$ is the discretization of the 3D space by Equation 1.3, obtaining the 3D integer indices $\begin{bmatrix}x & y & z\end{bmatrix}$. Then we need to perform a search over the tree of lists that can be splitted in three sub-searches $\sigma_x, \sigma_y, \sigma_z$, each of which occurring along one dimension, *i.e.* σ_k denotes the search for $\mathbf{p}$ along the k axis. So, the overal search is the composition in reverse order $\sigma(\rho(\mathbf{p})) = \sigma_z(\sigma_y(\sigma_x(\rho(\mathbf{p}))))$: in other words we first search the x node corresponding to the projection of $\mathbf{p}$ over the axis $\mathbf{u}_x$, the y node along the projection over $\mathbf{u}_y$ and the same with $z, \mathbf{u}_z$. The novel idea, in this kind of approach, is to employ an efficient data structure to represent the

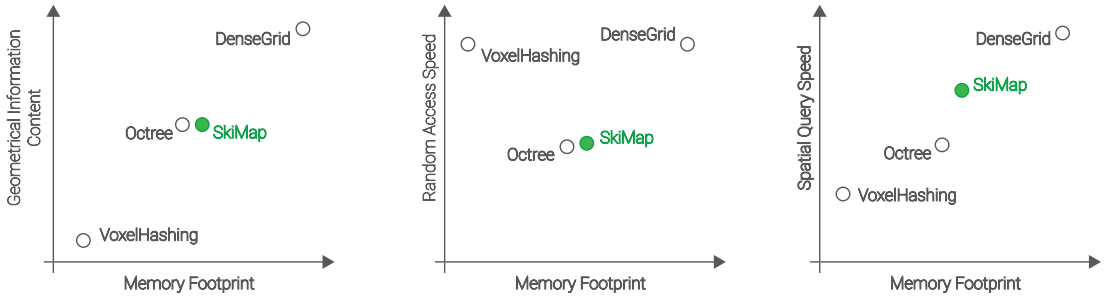


Fig. 1.6 These plots represent a qualitative comparison between *SkiMap* and other algorithms, mainly against *Octree* that is the target competitor. The comparison is just qualitative and is coming from experiments of the Chapter 3 to give an idea of benefits of *SkiMap* especially dealing with Spatial Queries.

lists (or better the *Ordered linked lists*) in order to minimize the *access* time. Had we used a simple list, we would have needed a linear time to find an element, running into the worst performances $\mathcal{O}(n)$ compared to $\mathcal{O}(log(n))$ of the *Octree* or to $\mathcal{O}(1)$ for both *Dense* and *VoxelHashing*. It was for this reason that, in *SkiMap*, we employed the *SkipList*, introduced by Pugh (1990), that is a valid alternative to *Balanced Trees*. This data structure ensures – probabilistically – a logarithmic time $\mathcal{O}(log(n))$ for basic operations like *insert*,*modify* and *delete*. In detail the correct computational complexity is $\mathcal{O}(3log(n))$ because in each search we involve at least 3 lists (one for each dimension) so *SkiMap* is theoretically slower than an *Octree* but thanks to the high parallelizable architecture it turns out faster in practice. This high parallelization is very useful also to perform very fast spatial queries, a key feature of this mapping algorithm. In Chapter 3 we will explain in detail how *SkiMap* is faster and why, in addition to other unique features that make it a valid solution for environment mapping in *Mobile Robotics*.

A very general graphical classification of the abovementioned mapping structures, compared with *SkiMap*, is depicted in Figure 1.6.

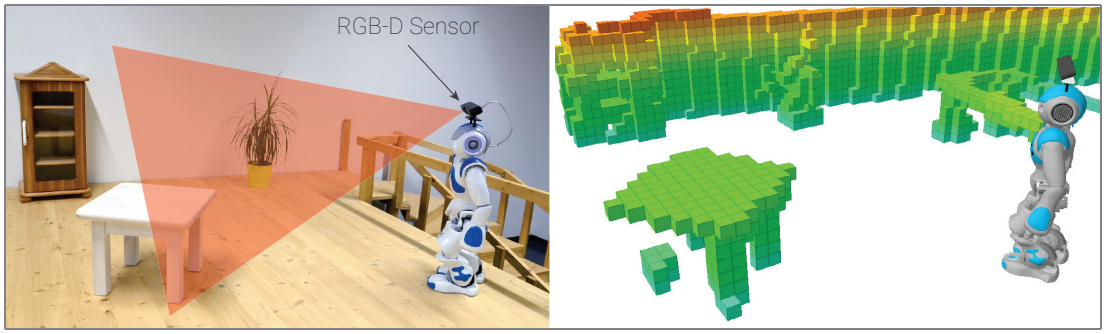


Fig. 1.7 A live snapshot from the system described by Maier et al. (2012). The right image depicts a graphical representation of a voxel grid mapping the occupancy of the environment represented in the left picture. Each voxel will be displayed only if its occupancy information q is over a given threshold q_{th}, and its color maps its height w.r.t. the ground.

1.3 Environment Representation

As mentioned at the beginning of this chapter, other than the hidden data structure of a map representation, the information content itself affects the richness of the model. In this section we will examine some of the most used representation in the field of Robotic Vision. The representation in this case is just the information content stored in each voxel, be it one of a Dense Grid (Figure 1.1) or of any other kind of data structure, and could be anything useful to represent the mapped environment. As seen in Table 1.1 just a numerical information could be sufficient in several applications, in particular in the robotic field we will see how different environment representation lead to different detail level based on the specific task for the robot.

1.3.1 Occupancy Grid

One of the simplest approach is to store in the voxels a sort of *occupancy* information, considering that the map will be continuously updated with measurements coming from a generic 3D sensor. We can define, in general terms, a sensor measurement (*sensor* = *camera* for simplicity), at time t, like a set of 3D points ${}^{cam}\mathbf{P}_t = \{p_0, ..., p_n\}$ and a set of corresponding rays ${}^{cam}\mathbf{U}_t = \{r_0, ..., r_n\}$ with $p_i, r_i \in \mathbb{R}^3$. Defining the sensor pose, at time t, as ${}^{map}\mathbf{T}_{cam_t} = ({}^{map}\mathbf{R}_{cam_t}, {}^{map}\mathbf{t}_{cam_t})$ (*i.e.* the pose of the camera cam_t in the *map* reference frame) we can easily compute the point-to-ray conversion as:

$$ {}^{map}\mathbf{p}_i = {}^{map}\mathbf{t}_{cam_t} + {}^{map}\mathbf{R}_{cam_t} \cdot {}^{cam}\mathbf{r}_t = {}^{map}\mathbf{t}_{cam_t} + {}^{map}\mathbf{r}_t \tag{1.5} $$

$$ {}^{cam}\mathbf{p}_i = \begin{bmatrix} 0 & 0 & 0 \end{bmatrix} + \mathbf{I} \cdot {}^{cam}\mathbf{r}_t = {}^{cam}\mathbf{r}_t \tag{1.6} $$

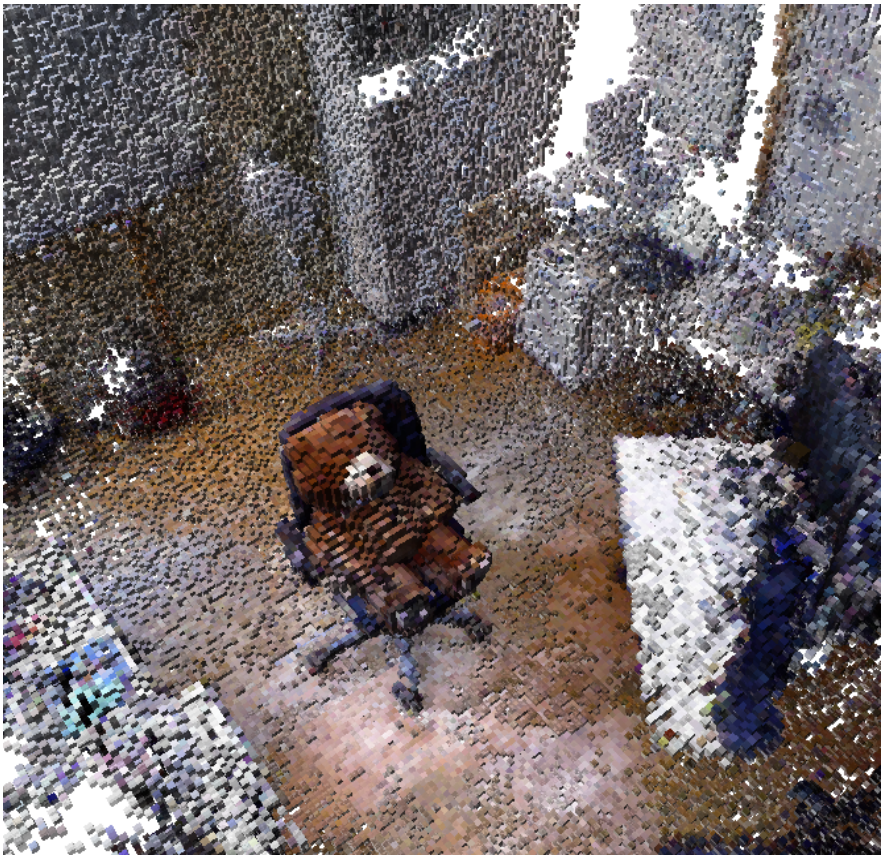


Fig. 1.8 This is a graphical representation of a 3D grid, of a teddy bear on a swivel chair, where each voxels tries to map the real color of the corresponding portion of the environment.

So a generic ray r_i is the vector connecting the corresponding point p_i and the sensor center, which overlap each other when represented in the sensor reference frame. For the simplest occupancy grid scheme we can use just a binary information $\{0,1\}$ where 0 means free and 1 means occupied. We need, then, to retrieve the voxel indices $\mathbf{v}_i = \rho({}^{map}\mathbf{p}_i)$ corresponding to a measured point ${}^{map}\mathbf{p}_i$, where $\rho(\cdot)$ is the a generic voxelization function (Equation 1.3), and update its content setting $V(\mathbf{v}_i) = 1$, where $V(\mathbf{v}_i)$ represents the voxel content corresponding to indices $\mathbf{v}_i$.

The above mentioned binary-occupancy mapping scheme is quite unhelpful in a real robotic scenario, because in the real world the *occupancy* is more of a continuous stochastic variable. As explained by Maier et al. (2012) and Hornung et al. (2013), which are focused on generic non-static environment, the probability $P(\mathbf{v}_i \,|^{map}\, \mathbf{p}_{1:t})$ that a voxel $\mathbf{v}_i$ is occupied at time t is recursively computed given all sensor measurement ${}^{map}\mathbf{p}_{1:t} = \{{}^{map}\mathbf{p}_1, ..., {}^{map}\mathbf{p}_t\}$. Consistent with considereations introduced by Moravec and Elfes (1985), we can transform the occupancy probability with a *logOdds* formulation translating this complex recursive computation in a frame-by-frame – faster – update function for each voxel as:

$$V(\mathbf{v}_i)_{1:t} = V(\mathbf{v}_i)_{1:t-1} + V(\mathbf{v}_i)_t \tag{1.7}$$

In this way each voxel is updatable over time, in a continuous manner, in order to react to the changes in the environment. Aside from the details of this probabilistic occupancy model, well explained in Hornung et al. (2013), we can focus on the voxel data. In case the occupancy probability can be represented with a continuous variable q, *e.g.* a *float* variable, and, as depicted in Figure 1.7, by taking into account only voxels with $q \geq q_{th}$, where q_{th} is a user-defined threshold, we can produce a graphical representation of the environment. In the aforementioned figure each voxel is painted with a color that does not depend on the q value but on the z component of the index $\mathbf{v}_i$: hotter color means higher voxel w.r.t. the ground. If, conversely, we want to store in each voxel the real color information, as depicted in Figure 1.8, we need a more complex data structure associated with each voxel and in addition a more sophisticated update strategy compared to the Equation 1.7. For example, the new voxel data could be (q,r,g,b) with q the usual occupancy probability and $r,g,b \in \mathbb{R}$ the three components of the color information, with a fourfold increase in the memory footprint. In this case, however, the update strategy is no longer straightforward: for example the color of the voxel at time $t+1$ cannot be computed as follows:

$$\begin{bmatrix} r & g & b \end{bmatrix}_{t+1} = \begin{bmatrix} r & g & b \end{bmatrix}_t + \begin{bmatrix} r & g & b \end{bmatrix}_{new} \tag{1.8}$$

since this is a divergent approach, which distorts color information. For this reason we need to provide our voxel with an addition information w that is called *weight* and hence the new voxel data is $< (q,r,g,b), w >$. In this case we can perform a weighted update of our data in this way (using $l = \begin{bmatrix} r & g & b \end{bmatrix}$):

$$\begin{aligned} l_{t+1} &= \frac{l_t w_t + l_{new} w_{new}}{w_t + w_{new}} \\ w_{t+1} &= w_t + w_{new} \end{aligned} \tag{1.9}$$

This weighted update can be applied to every data structure that needs to be updated in a conservative way and will be clearer in the Section 1.3.2.

To conclude, if we use 4 *bytes* for a generic *float* variable and we want to store a map composed by 10^8 voxels (*e.g.* a dense grid over a typical room using a resolution $r = 0.01m$) we need:

- $1 \times 10^8 = 100$ *MBytes* for a binary-occupancy grid;
- $4 \times 10^8 = 400$ *MBytes* for a continuous occupancy grid (q as voxel data);

- $20 \times 10^8 = 2$ *GBytes* for a continuous occupancy grid with color information ($< (q,r,g,b), w >$ as voxel data).

The latter example reveals that the memory footprint of a representation is directly proportional to the number of Voxels, for this reason choosing the correct data structure (Section 1.2) for modelling the space is a crucial design aspect when we deal with an high level of details.

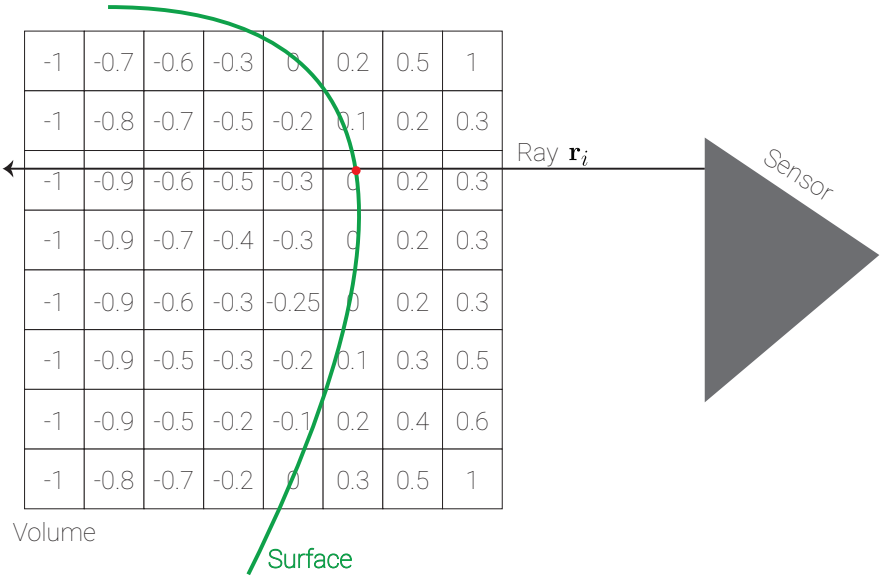


Fig. 1.9 This figure represent a bi-dimensional representation of a 3D grid which stores SDF values. A generic ray $\mathbf{r}_i$ crosses the surface, of a random object, running through the entire mapped volume. In each voxel crossed by the ray we store the metric distance between the center of that voxel and the nearest intersected surface, we can notice that voxels to the right, belonging to the free space, contain a positive growing values moving away from the surface; conversely the voxels to the left contains negative degressive values.

1.3.2 Truncated Signed Distance Function

Occupancy grid has been popular in robotics for several years – and it is still now – but it can be considered as a simple overlap of sensor measurements over time. Although the measurements were merged together through Bayesian updates to produce a continuous representation rather than a binary occupancy map, there is no a real smoothed representation of surfaces in the environment. Newcombe et al. (2011) for the first time use another approach to densely fuse RGB-D measurement (where RGB means a simple 2D color image and D means also a *depth* information of the scene for each pixel) in a 3D dense grid by exploiting a non-parametric representation of the environment named *Signed Distance Function* or briefly *SDF*. The SDF, introduced by Curless and Levoy (1996), can be described as follows:

SDF: looking at Figure 1.9 we can see that the SDF is, however, a grid-based representation where in each voxels we are going to amalgamate clues about *Surfaces* storing the distance of the target voxel w.r.t. to the nearest surface by using positive values to represent free space and negative values for occupied (or occlued) space. In the image a ray $\mathbf{r}_i$ (see Equation 1.5), starting from the sensor and running through the space, intersects the surface of an object represented as a simple green curve. Each voxels

along the ray's path stores the metric distance w.r.t. to the red intersection point. Same procedure will be repeated for each ray originated by the sensor.

As argued by Hernández et al. (2007), if our sensor produces rays $\mathbf{r}_i$ (and corresponding points $\mathbf{p}_i$) with a Gaussian noise model we can build the optimal surface reconstruction only by averaging the SDF representation of measurement over time. The averaging procedure is described by Equation 1.9 considering as voxel data d_s, that is the exact signed distance function computed during current measurement, associated with a weight w. Figure 1.10 shows the quality of reconstruction achieved with this simple procedure despite sensor noise.

The *Truncated Signed Distance Function* (or *TSDF*) is a variant of the SDF where the update of the voxels along the ray's path is truncated after a certain distance *beyond* the surface (and in many variants also before). This is done for many reason: first of all to save time by updating only a subset of the entire 3D Grid; the second and very important reason is that the voxels behind a surface surely reside into the object of belonging but it is highly likely that they could belong to something else because this portion of the space is considered *occluded* by the object itself. Conversely, this is not true for the voxels between the object and the sensor that are surely a portion of free space.

A disadvantage (or an advantage as stated by Newcombe et al. (2011)) is that the information about the environment is encoded in this generic continuous function $D(x,y,z) = D(\mathbf{v}_i) \in \mathbb{R}$ over the space that is obviously discretized over voxels coordinates, more formally is a *discrete scalar field*. Figure 1.11 depicts this function with a colormap: in this case we can easily see that the surface does not intersect, in most cases, the center of the voxels (black dots). Hence, to retrieve the real surface, we need to perform a sort of interpolation over discretized data, to be more precise we need to analyze the zero-crossing of this scalar field. One of the most used solution is *Marching Cube*, introduced by Lorensen and Cline (1987), which consists, as the name suggests, in a moving cube (enclosing 8 voxels) over the grid that analyzes zero-crossing events inside the corresponding portion of the field trying to produce the polygon (or multiple polygons) needed to best represent the part of the related isosurface (*i.e.* a surface over the field with constant value, in this case *zero*). The *Marching Cube* procedure is not fast but produces a solid mesh that can be used for robotic tasks such as: obstacle avoidance with potential fields (Khatib, 1986), path planning for welding/painting and stable objects grasp. For visualization purposes, instead, as depicted in Figure 1.10 we can produce a current view of the environment through a raycasting procedure that is obviously faster than Marching Cube but less descriptive in geometrical sense (a raycast is a 2.5D representation of the environment and not a full 3D reconstruction). The

Fig. 1.10 Samples from the work of Newcombe et al. (2011). The rightmost image represents outcome of a raycasting procedure over the SDF volume, it resembles a mesh but is only a 2.5D representation of the vantage point. In the middle image a colormap tints the surface according to the normals direction. The first image depicts a single noisy, and incomplete, output of the RGB-D sensor.

raycasting however is well exploited also for more complex tasks (camera localization (Newcombe et al., 2011)) thanks to the hidden capability of the SDF to represent also surface normals that can be directly derived analyzing the gradient of the SDF itself.

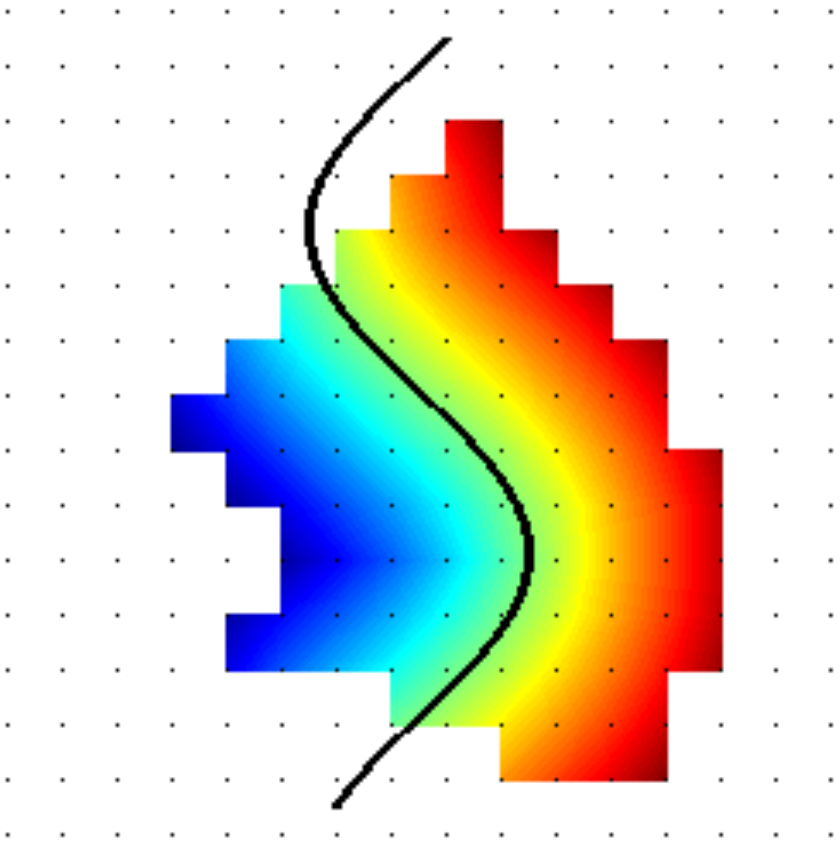


Fig. 1.11 A bi-dimensional representation of a SDF field interpolated in order to produce a continuous heatmap where hotter colors represent occupied volume and colder colors maps free space. The real surface, represented by the black curve, could lie in any point of the lattice despite the discretization introduced by the voxelization procedure.

Chapter 2

Small scale mapping for industrial robotic: RobotFusion

2.1 Rationale

Object recognition and 3D pose estimation are key tasks in industrial applications requiring autonomous robots to understand the surroundings and pursue grasping and manipulation (Papazov, Haddadin, Parusel, Krieger, and Burschka, 2012). Indeed, manipulation mandates estimation of the 6 Degree-Of-Freedom (6-DoF) pose (position and orientation) of the objects with respect to the base coordinate system of the robot. This pose estimation should be not only robust to clutter, occlusion and sensor noise, but also efficient to avoid slowing down the manipulation process.

Most object recognition and pose estimation algorithms rely on matching 2D or 3D features between off-line 3D models (either sets of 3D scans or CAD models) and scene measurements in the form of depth or RGB-D images. In particular, exploitation of color and depth cues from RGB-D images through suitable integration of texture-based and 3D features can yield quite remarkable performance across a variety of benchmark RGB-D datasets as shown by Aldoma et al. (2013) or Xie et al. (2013).

Although providing compelling results on standard benchmarks, the above mentioned multi-stage, multi-modal, feature-based pipelines turn out unsuited to practical real-time industrial applications due to exceedingly slow execution times. Furthermore, due to reliance on a single vantage point, these approaches may fail when the sought objects are captured under high levels of occlusion (Aldoma et al., 2011) and/or their shape and texture do not appear distinctive enough in the chosen view. Finally, an additional nuisance that may hinder performance of such approaches is represented by the high sensor noise affecting the depth frame provided as as input to the algorithms,

which tends to distort significantly the 3D surfaces and often cause holes and artifacts (Nguyen et al., 2012).

In this chapter, we investigate on the use of an RGB-D sensor mounted on top of a robotic arm to explore the surrounding environment in order to gather and process together RGB-D frames taken from different vantage points. This approach holds the potential to tackle the aforementioned issues inherent to perception from a single viewpoint, as also highlighted by the recent multi-view object recognition approach of Fäulhammer et al. (2015), who proved how multi-view information can increase recognition accuracy and robustness by the integration of features from different viewpoints. In our work, we aim at 3D object reconstruction and grasping within a typical industrial robotics environment, namely a robotic arm performing *pick-and-place* operations on objects laying on planar surfaces such as workbenches, conveyor belts or positioners. The only working assumptions underpinning our approach consist in the robot being endowed with a highly accurate and repeatable encoder system (as it is typically the case of industrial robotic arms) and in objects laying on a plane. In these settings, we leverage on the high accuracy of robot encoders to automatically fuse together different views. First, at each vantage point we collect and fuse together multiple frames within a voxelized 3D representation, so to smooth out 3D data without distorting the underlying object geometries. Then, the views taken from different vantage points are merged together into a Truncated Signed Distance Function (TSDF) representation: we dub our approach *Robot Fusion* on account of its affinity to the well-known Kinect Fusion system of Newcombe et al. (2011). Successively, we introduce a segmentation approach based on the planar assumption that allows for quick computation of grasping points without the need to carry out object recognition. Finally, the obtained grasping points are deployed to perform grasping via the robotic gripper mounted on the arm.

The novel contributions of this work thus concern: i) the *Robot Fusion* approach to reconstruct 3D objects by leveraging on robot encoders and TSDF representations, ii) an algorithm to segment objects out of a planar surface, iii) an algorithm to compute grasping points from surfaces without requiring any previous object recognition step. We report experimental results that demonstrate the effectiveness and accuracy of the proposed object reconstruction and segmentation stages, as well as the ability to attain good grasping points that enable to perform grasping successfully in cluttered scenes comprising several objects of diverse shapes.

2.2 Reconstruction and Recognition for Grasp

On-line fusion of the range images gathered by a moving sensor to achieve 3D reconstruction of the environment is a key task for RGB-D SLAM (Simultaneous Localization and Mapping) frameworks. Among prominent approaches, Kinect Fusion by Newcombe et al. (2011) or the work of Whelan et al. (2014) attain a highly accurate 3D reconstruction of the environment by fusing the depth measurements taken from different viewpoints alongside with camera movement into an occupancy map, i.e. a discrete voxel grid where each cell may either be void or contain the distance from the nearest surface via a Truncated Signed Distance Function (TSDF) (See Section 1.3.2).

Besides mapping, the other key task in SLAM frameworks consists in sensor localization. Purposely, the Iterative Closest Point (ICP) algorithm is widely deployed to align the current depth image with respect to either the previous one or the global map. However, due to the algorithm allowing to estimate successfully small motions only, ICP is prone to failure when the incoming frames get processed at a pace that turns out too slow compared to sensor movements. To overcome this issue, sparse matching of 2D/3D features or dense, direct matching of image intensities may be deployed to coarsely align the current depth image prior to running ICP (Engel et al., 2014). Several other methods, such as the one of Kehl et al. (2014), rely on optimization of a global cost function to estimate all sensor poses coherently. Although very effective, these approaches are unsuited to real-time industrial robotics due to excessive running times, such as e.g. several minutes (Kehl et al., 2014).

Differently, we address the SLAM localization task by relying on the high accuracy and repeatability of the robot encoders as well as on forward kinematics, as explained in details in Section 2.3.2. This leads to a fast and highly accurate 3D reconstruction step which is conducive to estimate a reliable set of grasping points on the reconstructed surfaces in absence of any information concerning the types of objects present in the scene. Indeed, estimating a set of contact point providing a stable grasp based only on 3D geometry mandates very accurate 3D reconstruction of surfaces, as figured out by Montana (1991).

As for approaches aimed at grasping and manipulation based on grasp point detection, methods that try to estimate grasp points on unknown objects from a single range image have been proposed by Yun Jiang and Moseson, S. and Saxena, A. (2011) and Popović et al. (2010). These works are based on edge analysis and estimate the approach position of the gripper with respect to the target object (i.e., no estimation of the individual grasp points). Unfortunately, these methods can deal with thin objects or thin object parts only in scenes with low occlusion levels, both constraint due to the inherent limitations of relying on a single viewpoint already described in Section 2.1.

Another issue with these approaches is that the grasp configuration is computed for grippers with a simple geometry, like parallel grippers. Indeed, as stated by Popović et al. (2010), using a complex gripper, like a five fingered anthropomorphic hand, requires shaping it as a parallel gripper to achieve successful grasp. Another possible approach consists in estimating directly the grasp points on the object surface and compute, via inverse kinematics, the position and orientation of the gripper. Saut and Sidobre (2012) describes how it can be hard to achieve a grasp configuration for a multi-finger anthropomorphic hand in real-time, and similar conclusions are drawn by Hang et al. (2014). Thus, these methods are suitable to autonomous grasping based on offline estimation of grasp points on known objects, an object recognition stage to detect object instances and associated poses, a final strategy to choose automatically the optimal grasp configuration querying a database.

2.3 Method description

This Section describes in detail the main stages of the proposed system. As already mentioned, our approach relies on typical assumptions for industrial manipulation and *pick-and-place*, i.e. the presence of a high precision industrial robotic arm and a planar workbench holding objects. In Section 2.3.1 we illustrate how forward and inverse kinematics can be employed to accurately compute the camera 6-DoF pose with respect to the robot main coordinate system regardless of the mounting point used on the robotic arm, this effectively replacing the visual data-based localization necessary to perform SLAM reconstruction of the scene. Then, we show how to attain accurate multi-view 3D reconstruction by fusing together range images within a voxelized TSDF representation. In Section 2.3.2, we propose a segmentation approach based on a novel plane extraction method, dubbed *HeightMap*, which estimates all scene planes orthogonal to a given *gravity vector*. This results in a small and predictable computational time, which compares favorably to RANSAC-based plane fitting and allows for segmenting effectively the individual objects from the previously obtained 3D reconstruction. Finally, in Section 2.3.3, we describe our proposed method for grasp points extraction from the 3D surface of segmented objects, which does not require an explicit object recognition stage. In particular, grasp points and grasp approach position are computed by leveraging on the object *Canonical Reference Frame* and so to avoid collisions between the robotic arm and the environment.

2.3.1 Multi-view reconstruction via RobotFusion

We assume to deploy an industrial robot with high accuracy and repeatability (i.e. $\approx 0.05mm$ ISO TC 184SC 2 Robots and robotic devices (2015)). High accuracy means that we can impose a 6-DoF end effector pose ${}^{0}\mathbf{T}_{EE}$ with high precision in a dexterous workspace:

$$ {}^{0}\mathbf{T}_{EE} = \begin{bmatrix} {}^{0}\mathbf{R}_{EE} & {}^{0}\mathbf{p}_{EE} \\ 0 & 1 \end{bmatrix} $$

In our system, the end effector is an RGB-D camera (*i.e.* an Asus Xtion) with a 3D-printed housing, this yielding an unknown transformation between the robot wrist and the 3D coordinate system attached to the camera (i.e. the camera coordinate system):

$$ {}^{0}\mathbf{T}_{EE} = {}^{0}\mathbf{T}_{W} \cdot {}^{W}\mathbf{T}_{EE} \tag{2.1} $$

While ${}^{0}\mathbf{T}_{W}$ is the result of a simple application of forward kinematic equations on the target robot, ${}^{W}\mathbf{T}_{EE}$ is estimated during an offline calibration stage by means of a fiducial marker having a known 3D pose, ${}^{0}\mathbf{T}_{M}$, in the main robot coordinate system. To accomplish this, we use an Augmented Reality Framework (Garrido-Jurado et al. (2014)) to estimate the pose of the fiducial marker w.r.t. the camera coordinate system: ${}^{Cam}\mathbf{T}_{M} = {}^{EE}\mathbf{T}_{M}$, thereby closing the loop in the equation

$$ {}^{0}\mathbf{T}_{W} \cdot {}^{W}\mathbf{T}_{EE} \cdot {}^{EE}\mathbf{T}_{M} = {}^{0}\mathbf{T}_{M} \tag{2.2} $$

in order to estimate the unknown transformation between the wrist and the camera

$$ {}^{W}\mathbf{T}_{EE} = {}^{W}\mathbf{T}_{0} \cdot {}^{0}\mathbf{T}_{M} \cdot {}^{M}\mathbf{T}_{EE} \tag{2.3} $$

Now, each point, ${}^{cam}\mathbf{p}_{i}$, belonging to the point cloud ${}^{cam}\mathscr{P} = \{p_1, ..., p_n\}$ acquired by the RGB-D sensor can be expressed as:

$$ {}^{0}\mathbf{p}_{i} = {}^{0}\mathbf{T}_{EE} \cdot {}^{EE}\mathbf{p}_{i} \tag{2.4} $$

Hence, for each new view gathered alongside robot movement, we can directly obtain a point cloud registered into the main robot coordinate system without the need to estimate any alignment transformation with respect to the previously acquired clouds.

A 3D object reconstruction algorithm needs multiple *range images* from different viewpoints to build an unified representation of the surface of the target object. State-of-the-art SLAM techniques pursuing accurate and dense scene reconstruction, such as *e.g.* Newcombe et al. (2011) and Bylow et al. (2013), first localize the camera

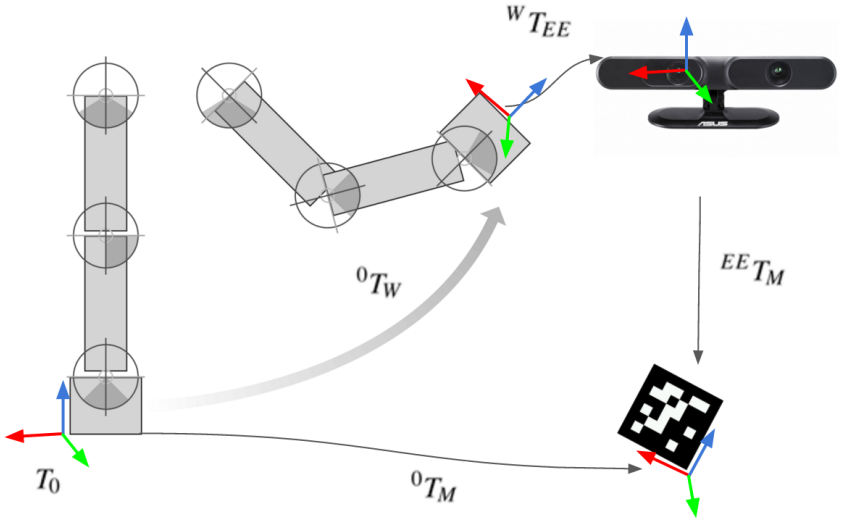


Fig. 2.1 Kinematics chain to compute the transformation between the wrist and the camera ($^{W}\mathbf{T}_{EE}$).

and then merge each newly acquired range image into a global representation of the environment based on a *Truncated Signed Distance Function* (TSDF) defined on a voxel grid. Therein, each voxel either is void or stores the signed distance from its center to the nearest surface.

We adopt the same TSDF representation. However, as described, we do not need to carry out an explicit camera localization step as we can seamlessly integrate depth measurements into a global voxel grid by deploying robot kinematics. It is worthwhile pointing out that the use of a TSDF-based representation allows our system to fuse together seamlessly not only acquisitions from diverse viewpoints, but multiple range images taken from the same vantage point alike. The latter process turns out highly effective to counteract the significant amount of noise that typically affects the RGB-D data acquired by consumer depth cameras.

To minimize the amount of calculations, we update only the voxels visible from the current vantage point (view frustum) rather than the entire grid [1]. Moreover, we rely on an *octree* representation of the voxel grid, this decreasing memory requirements significantly (Steinbrucker et al., 2013). Thanks to the reduced computational complexity, and unlike most previous works on dense volumetric mapping that leverage on GP-GPU acceleration, we can run efficiently our reconstruction system on the CPU, this being a beneficial trait as regards its potential implementation on the compact embedded platforms often deployed in industrial environments.

[1] https://github.com/sdmiller/cpu_tsdf

To assess quantitatively the accuracy provided by the proposed RobotFusion approach based on fusing depth measurements into a TSDF within a viewpoint and across viewpoints, we created a synthetic setup in the Gazebo simulator integrated in the ROS framework [2]. In particular, we simulated an industrial robot equipped with an RGB-D sensor and a table-top scene. As the simulation environment provides noiseless RGB-D data, we implemented a noise model similar to that proposed by Nguyen et al. (2012). In the experiments, we compared the reconstructions provided by RobotFusion to those that one would achieve by simply stitching together the point clouds from the different vantage points into the global coordinate system of the robot, which is also straightforwardly attainable by deploying robot kinematics. Figure 2.2a shows that the proposed approach can achieve smooth and accurate 3D reconstructions as well as how the devised fusion process provides a significantly higher accuracy with respect to simple stitching of the point clouds. Figure 2.2c pertains a reconstruction experiment carried out with real data: while RobotFusion can recover the object's surface accurately, the reconstruction created by stitching together the points clouds shows many gross errors due to sensor noise. Eventually, Figure 2.2b concerns en experiment aimed at evaluating the benefits brought in by fusing together multiple depth images at each viewpoint: the reconstruction error tends to decrease and then stabilize as more and more depth images get fused at a single viewing position. As such, it turns out beneficial to deploy a sufficient number of images (*e.g.* 10-15) to effectively smooth out noise and minimize the reconstruction error.

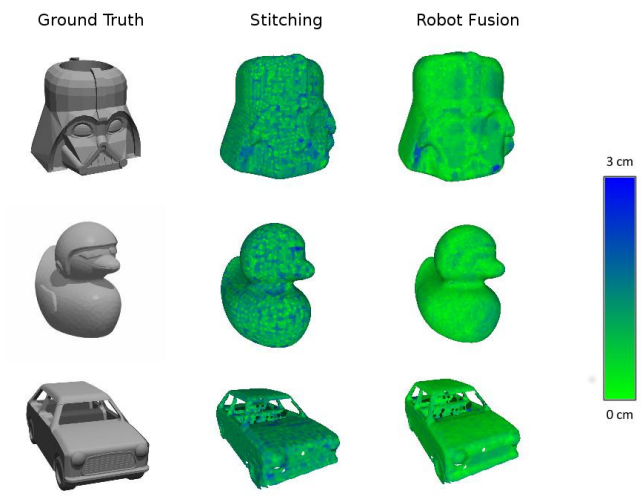
2.3.2 Plane-based segmentation

Industrial robots can be mounted only in three standard positions: *floor, wall, ceiling* (seldom at *45 degrees*). Thus, we can always calculate the relation between the *gravity vector* and the *z-axis* of the robot in the robot (or world) coordinate system. For example, in a *floor-mounted* robot the gravity vector is parallel and opposite to the *z-axis*: $\mathbf{g} = \begin{bmatrix} 0 & 0 & -\mathbf{z} \end{bmatrix}$.

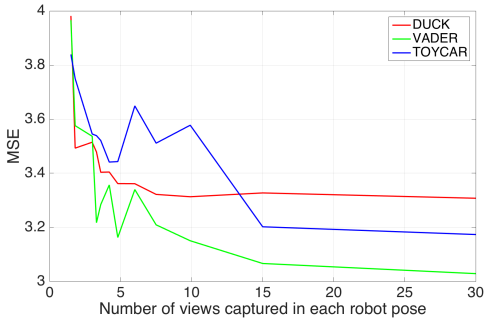
Hence, given a point cloud $\mathscr{P} = \{\mathbf{p_1}, ..., \mathbf{p_n}\}$ and the associated set of normals $\mathscr{N} = \{\mathbf{n_1}, ..., \mathbf{n_n}\}$, we extract the points belonging to planes orthogonal to the *gravity vector* by creating a *1-D* histogram $\mathscr{H} = \{S_1, .., S_k\}$ where each bin represents a subset (or *slice*) of points belonging approximately to the same plane. More precisely, assuming a *floor-mounted* robot and denoting the *slice-size* (*histogram bin-size*) as λ:

$$\mathbf{p_i} \in S_j \iff \mathbf{n_i} \cdot (-\mathbf{g}) < cos(\alpha), j = \lfloor \frac{p_{\mathbf{z},i}}{\lambda} \rfloor \tag{2.5}$$

[2] http://www.ros.org/



(a)



(b)

(c)

Fig. 2.2 (a) *Synthetic data*. Left: original 3D CAD model used as ground truth. Center: reconstruction by stitching point clouds. Right: TSDF-based reconstruction by RobotFusion. Colors encode the metric error w.r.t. the ground truth at reconstructed surface points. (b) Mean square error in function of the number of views for the 3 different models of the Figure 2.2c. The proposed TSDF-based approach obtains an increasing accuracy with a higher number of frames captured in each robot poses (in this case 12 robot poses around object). (c) *Real data*. Left: original 3D object. Center: reconstruction by stitching point clouds. Right: TSDF-based reconstruction by RobotFusion.

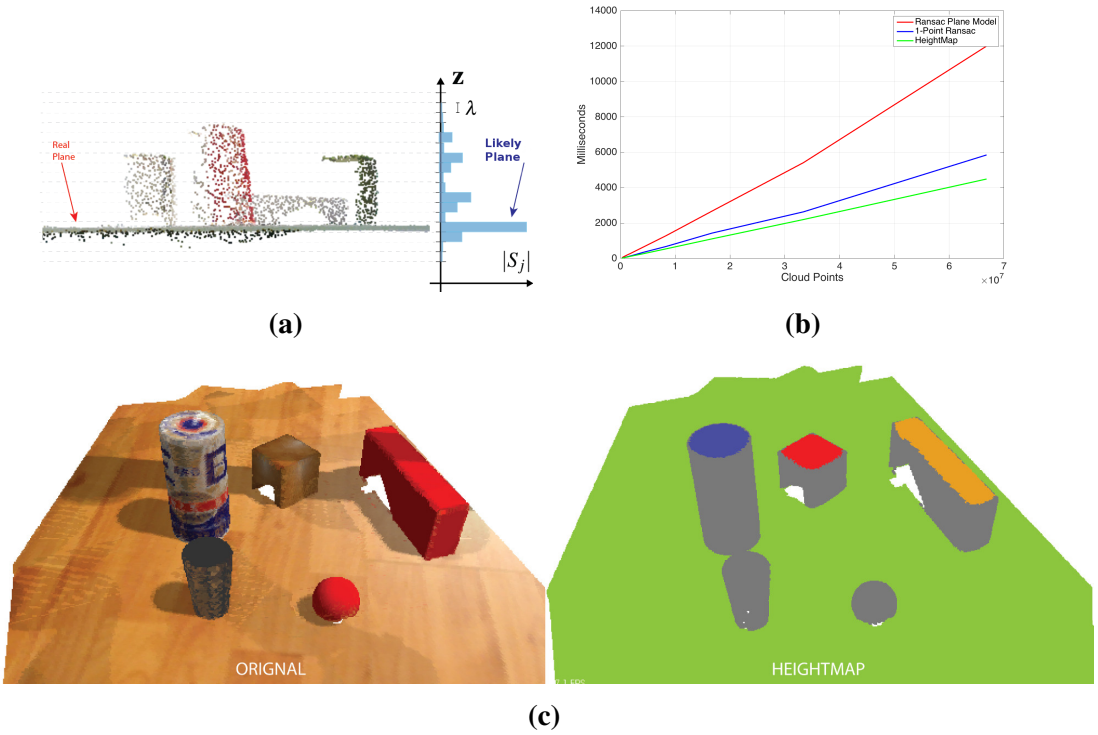


Fig. 2.3 (a) Side view of a point cloud representing a table-top scene, with four objects. Accordingly, the bin of the histogram reported on the right hand-side corresponding to the table is the highest one, while object surfaces tend to report much smaller bin values. (b) Execution times of plane extraction algorithms while increasing the point cloud size. (c) Example of horizontal planes extracted by a single run of the proposed *HeightMap* segmentation algorithm.

α being an arbitrary angular threshold used to define the maximum allowed plane inclination w.r.t the gravity vector and to withstand the presence of noise on the computed normals.

As illustrated in Figure 2.3a, for a *floor-mounted* robot histogram $\mathscr{H}$ highlights the position along the *gravity axis* of possible horizontal planes. By defining $S_{\mathbf{z_{min}},j}$,$S_{\mathbf{z_{max}},j}$ as, respectively, the minimum and maximum z coordinates in slice S_j, we can extract planes by selecting subsets of points belonging to those histogram bins that report a value over a pre-defined threshold. We call this approach as *HeightMap* segmentation.

Remarkably, plane extraction methods such as those based on *RANSAC* have no predictable execution time (Rahul Raguram, Jan-Michael Frahm, Marc Pollefeys, 2008) due to their intrinsic randomized-iterative nature. This is also the case of the *1-Point RANSAC* approach by Tombari et al. (2014), which enables to estimate dominant planes by means of a 1-point (rather than 3) RANSAC plane fitting by exploiting the normal associated to each point. Conversely, the proposed *HeightMap* approach enjoys a deterministic and fast execution time, due to its complexity being linear in the size of the point cloud.

Figure 2.3b reports the running times yielded by different plane extraction methods, namely *HeightMap*, standard RANSAC and 1-point RANSAC, in experiments carried out in the simulated working environment already described in Section 2.3.1. As we can see, *HeightMap* is the fastest approach and features a linear growth of the measured execution time with respect to the point cloud size. Figure 2.3c shows an example of the horizontal planes extracted by the *HeightMap* method from a point cloud. It is worth pointing out that a single run of the method allows to extract all the sought parallel planes, while getting an equivalent result by RANSAC-based approaches would require multiple iterations. On a broader level, we wish to point out that *HeightMap* may be applied to extract planes orthogonal to any arbitrarily oriented *gravity-vector*. Purposely, the point cloud may require a rotation to align one axis (e.g. the *z-axis*) to the considered *gravity-vector*.

Under the previously highlighted assumption of objects placed on a planar support, plane extraction provides a strong cue to segment the individual objects. Indeed, the largest plane extracted by *HeightMap* can be quite safely assumemed to represent the workbench and thus simply removed from the point cloud. Thereby, the objects become disconnected one to another and the points belonging to each object's surface can be identified straightforwardly by standard tools such as Euclidean Clustering [3].

2.3.3 Extraction of grasp points

The goal of the proposed grasp point extraction approach is to estimate directly grasp points on the object surface regardless of the gripper shape (or regardless of the number of fingers in case of a robotic hand) by means of an efficient iterative algorithm. Our main approach to reduce the complexity of grasp point estimation on a 3D surface is to reduce the solution space from 3D to 2D, so to be able then to apply well known *planar grasp* algorithms as studied by Ponce and Faverjon (1991), Ferrari and Canny (1992), Bicchi and Kumar (2000). Planar grasp algorithms require a 2D polygon as input: to deal with this, we reuse the *HeightMap* segmentation technique proposed in Section 2.3.2 applied on a single object cluster rather than on the point cloud of the entire 3D scene. Figure 2.4a shows the proposed pipeline to compute grasp points in the robot coordinate system, which is described in the following.

Principal axes To begin with, each 3D object cluster is sliced along its principal axes. To obtain these stable axes, we compute the EigenValue Decomposition (EVD) of the covariance matrix of each object cluster, as described by Salti et al. (2014), which yields three repeatable directions.

[3] http://www.pointclouds.org/documentation/tutorials/cluster_extraction.php

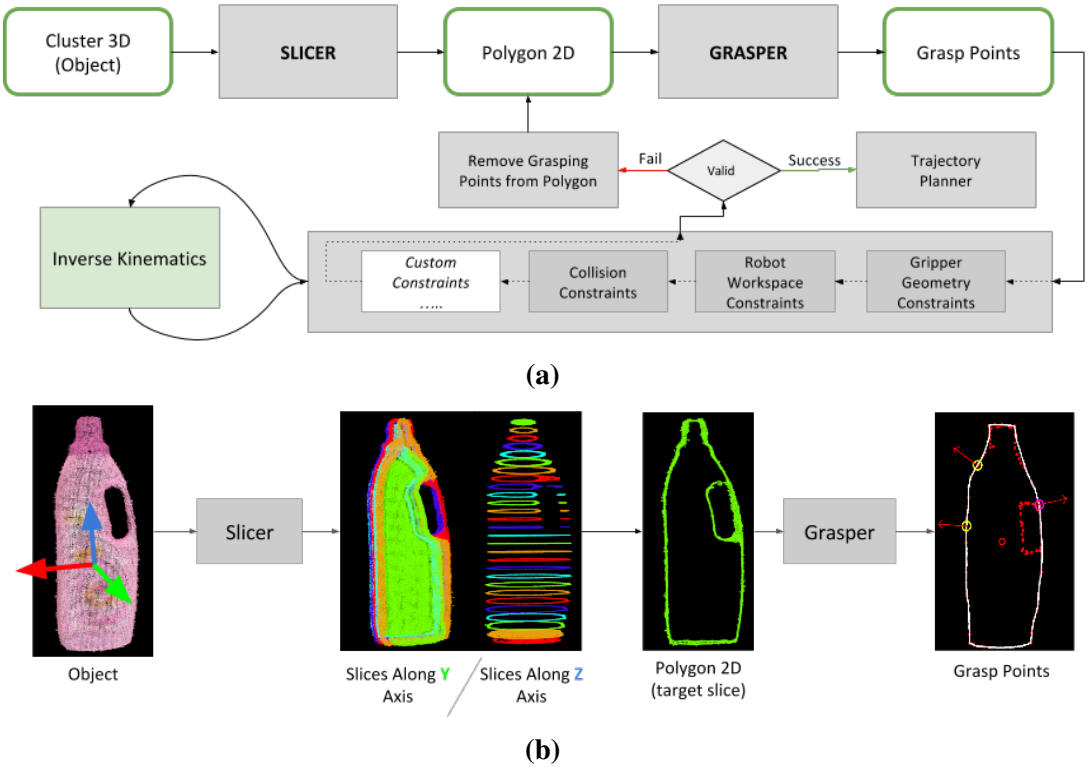


Fig. 2.4 (a) Proposed pipeline for grasp points extraction. Each subset of the polygon points (defined as Grasping Points) will be validated through N validation stages. In addition to fixed constraints, the user can choose custom constraints for a specific task. (b) Graphical representation of a 3D object passing through the previous pipeline until it reaches a valid grasp point configuration.

Slicer The algorithm selects a slice and projects the points onto a plane orthogonal to the slicing axis. To simplify the obtained shape, the system computes the concave hull of the slice, this yielding a 2D polygon suitable to the next stages.

Grasp Point Extraction At this stage, N points are randomly extracted from the polygon boundary (in practice, only the first points are randomly extracted, the remaining ones are chosen so to avoid useless point subsets, e.g. adjacent points).

Validation The extracted grasp points are then evaluated through several stages. The first stage checks for geometrical constraints: if the detected grasp points lie outside of the gripper (or robotic hand) workspace, the grasp points are removed. This can be carried out, e.g., by checking if the distance between 2 grasp points is larger than the maximum opening of a parallel gripper. The second stage checks for robot workspace constraints, by evaluating whether grasp points can be reached in a dexterous portion of the workspace. The third stage evaluates collision constraints, by checking if the gripper, with the 6-DOF pose relative to the grasp points, would collide with environment. In

addition, *custom* constraints can also be further added to refine grasp points in case of specific tasks or manipulators, by adding additional validation stages.

Grasp point removal If one of the validation stages fails, the set of grasp points is removed for the current 2D polygon. Conversely, if no validation stage fails, since the grasp points are already in the robot coordinate system , the system can directly compute the full inverse kinematic chain from the grasp points to the robot base, so that a trajectory planner can be instructed to easily allow the robot approach the object and carry out the grasp.

Of course, given the greedy and iterative nature of this algorithm, the determined solution may not be the optimal one, and the system may also fail to find a feasible solution if the required iterations exceed the maximum number of allowed iterations.

2.4 Grasping experiments

The entire pipeline comprising 3D reconstruction, plane-based segmentation and extraction of grasp points was tested in real grasping experiments dealing with 8 scenes created by placing several objects (between a set of 8) on a planar support, as illustrated in Figure 2.5b. The experimental setup consists of an Industrial Robot, *i.e.* a *Comau Smart Six* with six degrees of freedom and an accuracy and repeatability lesser than *0.05 mm*. The end effector, shown in Figure 2.5a, is a dual-use tool: the first part of the tool is an Asus Xtion RGB-D sensor, the second part a two finger Robotic Hand with three contact point providing a more stable grasp compared to a simple parallel gripper with only two contact points.

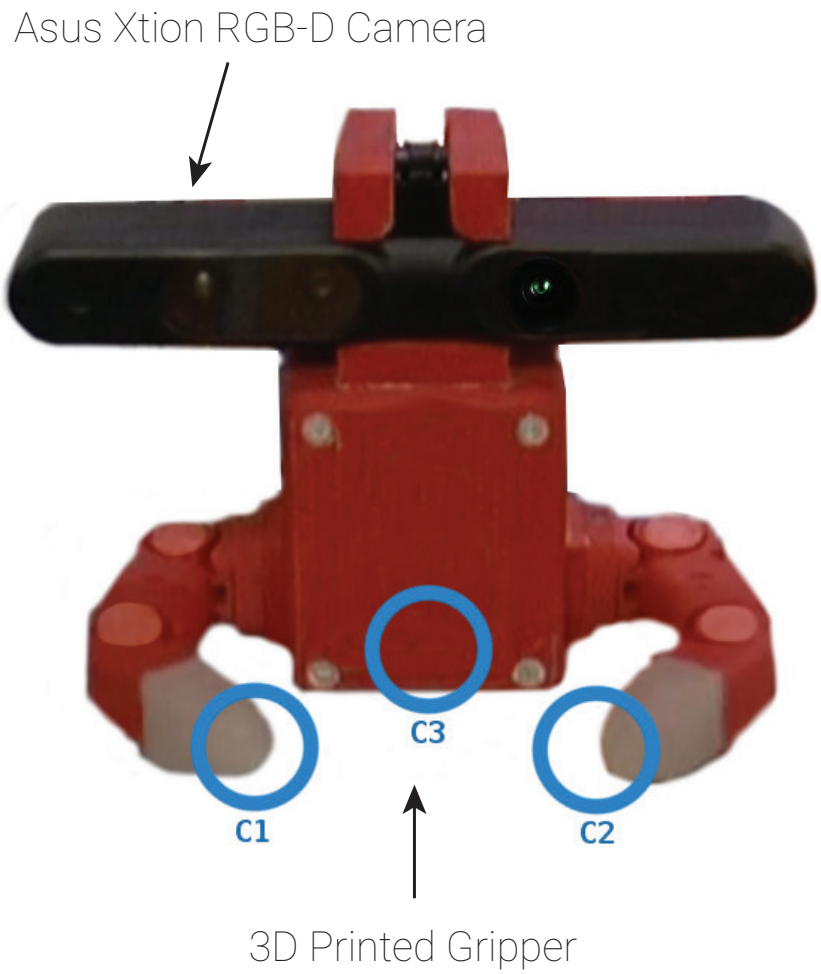
Each scene was tested 5 times and in each experiment the robot had to grasp all the objects present in the scene. Accordingly, a binary outcome (*Success* or *Failure*) was reported for each possible grasp depending on whether or not the robot succeeds in grasping the object and placing it into a bin. Thus, *Success* requires all the stages in our pipeline to work effectively, while a *Failure* may be ascribed to a variety of causes, such as missing an object because of a wrong segmentation or an unstable grasp (e.g. the robot does not picks up the object or the object falls while being moved towards the bin) due imprecise extraction of the grasping points.

Table 2.1 summarizes the results of our grasping experiments. In particular, for each of the 8 objects, the last two columns report the number of possible grasps within the 40 experiments (5 tests for each of the 8 scenes) as well as the percentage of successful ones. Each row depicts also two examples of grasps performed by the robot together with the associated contact points computed by the algorithm described in Section 2.3.3.

It is worth highlighting that, as each object was placed in the different scenes according to different poses (see the pictures in Table 2.1), those automatically determined by our algorithm are not simple caging grasps but, indeed, stable predetermined grasp configurations. The overall success rate over all the grasping experiments turned out to be *96,56 %*, with the failures mainly due to imprecise localization of the grasping points.

A qualitative evaluation of the system is presented in these two videos: Video1 [4] , Video2 [5] . The first footage presents the evaluation setup used in previous experiments using a 2.5-fingers gripper. In the second video, instead, a real scenario with a 3-fingers gripper shows the high repeatability of the proposed approach, also with a multi-shape gripping organ and two fixed cameras.

[4]https://www.youtube.com/watch?v=ECXamfIaPcQ
[5]https://www.youtube.com/watch?v=Y1FaBE54E5A

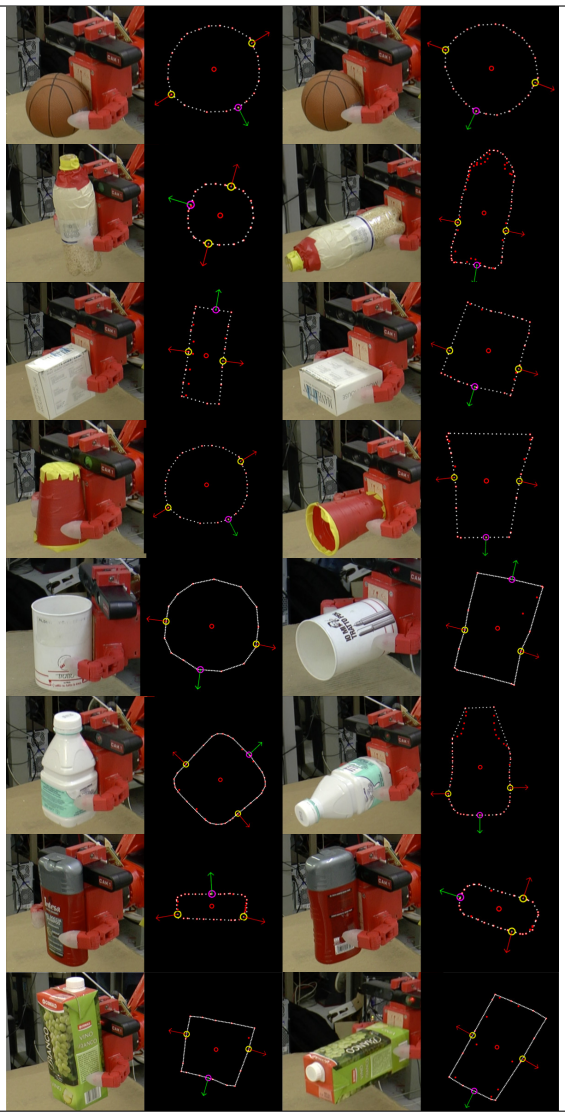


(a)

(b)

Fig. 2.5 (a) End Effector used during the experiments. The top part is an Asus Xtion RGB-D Sensor, the bottom part is the Gripper: a two finger robotic hand with three contacts points *(C1,C2,C3)*. (b) Two sample scenes of our grasping experiments.

Table 2.1 Grasping Results

Object Type	**Grasp carried by the robot and corresponding grasp configuration**	**Possible Trials**	**Success** (%)
Ball		20	100
Bottle		20	100
Box		20	100
Cup		30	95
Cylinder		20	100
Milk		30	100
Showergel		20	75
Winepack		30	100

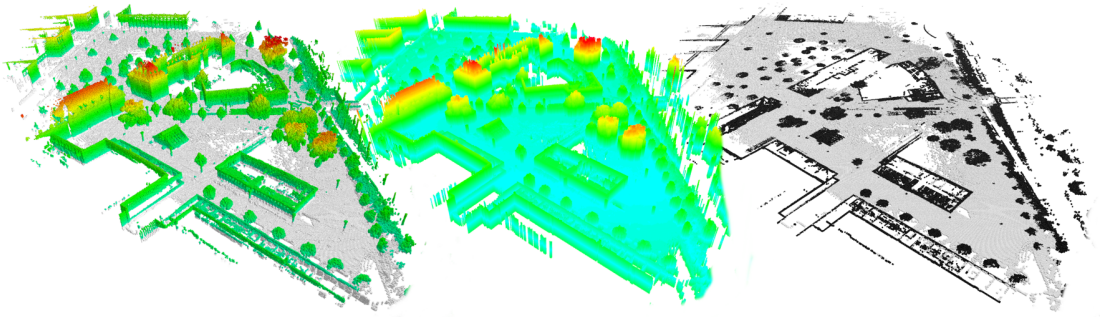


Fig. 3.1 SkiMap encodes seamlessly a full 3D reconstruction of the environment (left), a height map (center) and a 2D occupancy grid (right). The three representations can be delivered on-line with decreasing time complexity. The displayed maps have been obtained on the *Freiburg Campus* dataset.

Chapter 3

Large scale mapping for mobile robotic: SkiMap

3.1 Rationale

As stated in the Chapter 1 several work are focused on the mapping task and address issues such as memory efficiency, quite mandatory to enable navigation in large spaces, and time efficiency, which concerns creating on-line the representation required by the navigation system, such as a 2D occupancy grid to plan a path through the environment or a 3D reconstruction to avoid obstacles reliably while a robot moves around or a 2.5 (aka height) map to assess free space at the flight altitude of a MAV (Micro Aerial Vehicle). Along this latter research line, in this chapter we will focus on mapping and present a novel approach, dubbed *SkiMap*, introduced in Section 1.2.4, which is particularly time efficient and flexible enough to support seamlessly different kinds of representations that may be delivered on-line according to the application requirements, as depitcted in the Figure 3.1.

Another favorable trait of our mapping framework is the ability to erode and fuse back measurements in real-time upon receiving optimized poses from the sensor localization module in order to improve the accuracy of the map. Indeed, many recent sensor localization algorithms based on visual data can perform pose optimization on-line, e.g. upon detection of a loop closure, which holds the potential to continuously improve the map as long as sensor measurements may be injected therein according to the new optimized poses rather than the old ones.

Our framework has been implemented as a ready-to-use ROS (Quigley et al., 2009) package freely distributed for research and education purposes[1]. The package can be configured either to achieve mapping in conjunction with any external sensor localization module or as a full-fledged SLAM system like *Slamdunk*, developed by Fioraio and Di Stefano (2015), or *ORBSLAM2* from Mur-Artal and Tardós (2017).

3.2 Mapping Data Structure

In this section we explore the *SkiMap* algorithm in its entirety, describing the key data structure as well as how to carry out mapping differently from standard approaches like *Octree* or 3D Grid. Furthermore, we highlight the inherent parallelism of the proposed data structure, which is conducive to notably improved time efficiency in key tasks dealing with robot navigation.

3.2.1 Tree of SkipLists

SkiMap relies on the basic concept of grouping voxels within a tree as outlined in Figure 3.2. The actual voxels are nodes at depth *3*, which are grouped into nodes at depth *2* according to equal quantized (x,y) coordinates, the nodes at depth *2* in turn grouped into nodes at depth *1* according to equal quantized x coordinates. However, adopting a classical tree structure to realize the concept illustrated in Figure 3.2 would not be efficient because of the unbounded number of siblings at each depth level (unlike the *octree*, in turn, where each node has always *8* children). Indeed, should the children of each node be stored in a ordinary list, performing a random access would exhibit $\mathcal{O}(n)$ complexity. To overcome this efficiency issue, we adopted a rather uncommon data structure called *SkipList* and proposed by Pugh (1990). As shown in Figure 3.3 a *SkipList* is apparently similar to an *Ordered Linked List*, but the former also stashes a super-structure aimed at bringing the computational complexity associated with random access from $\mathcal{O}(n)$ down to $\mathcal{O}(\log n)$. In a *SkipList* elements are kept ordered, and thus,

[1] https://github.com/m4nh/skimap_ros

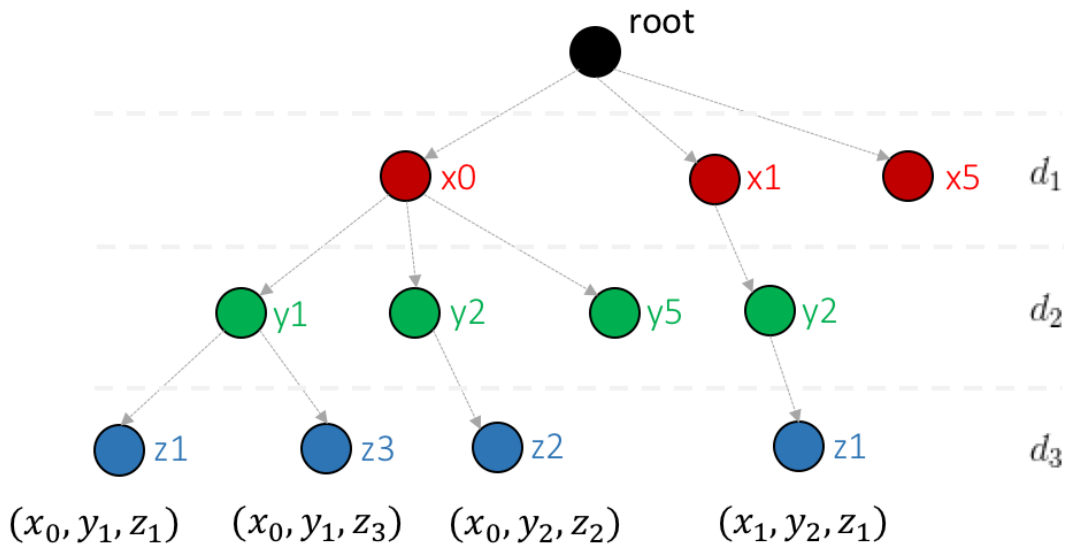


Fig. 3.2 Tree structure to group voxels according to their coordinates. The maximum depth of the tree is *3*, nodes with depth d_3 being voxels while those with depths d_1, d_2 being transient nodes. Nodes at depths d_1, d_2 store only integer numbers representing the associated quantized coordinate, while voxels (*blue nodes*) can be deployed to store user data, such as for example Occupancy Probability (Thrun et al., 2005).

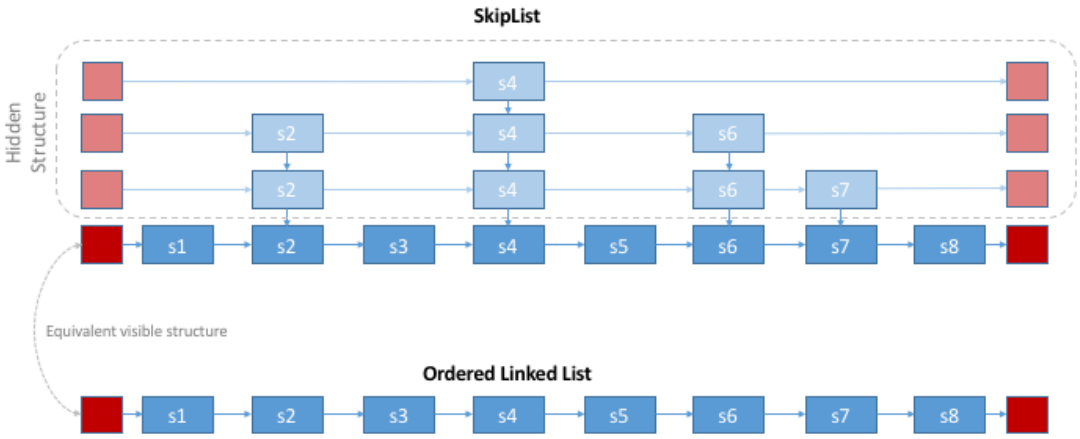


Fig. 3.3 The visible part of a **SkipList** is identical to a **LinkedList**. The hidden segment of a SkipList shall ensure a random access complexity of $\mathcal{O}(\log n)$ rather than $\mathcal{O}(n)$.

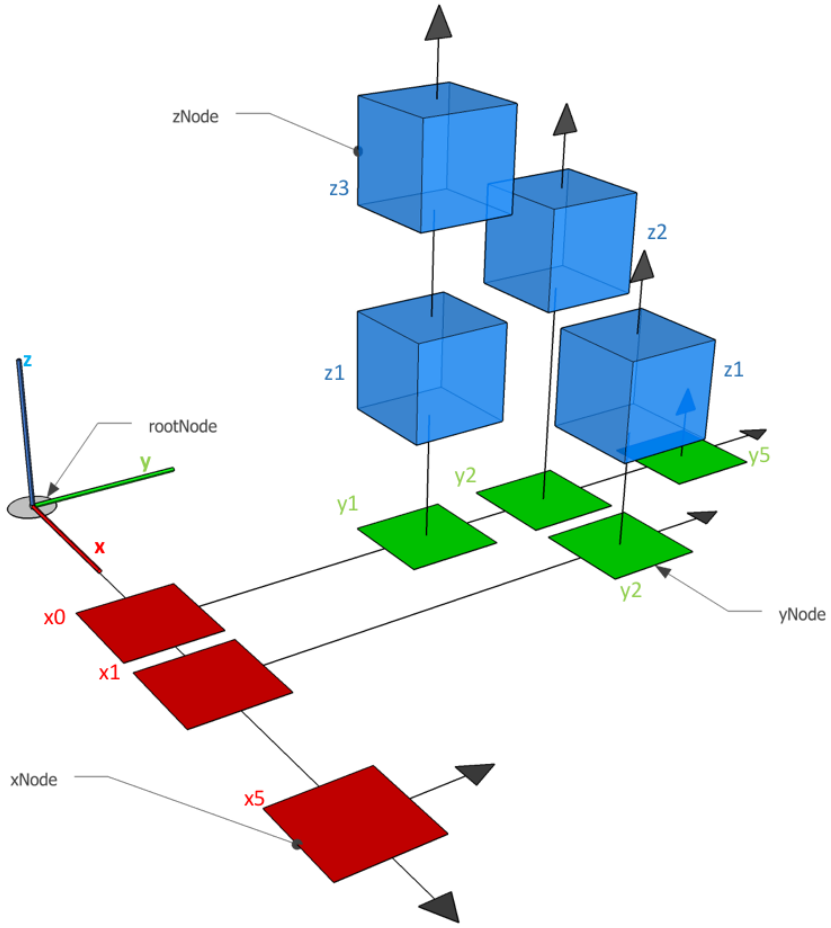


Fig. 3.4 Grouping voxels into a **Tree of SkipLists**. Each voxel (*blue box*) is linked to the *rootNode* by a *yNode* (green tile) which in turn is linked to a *xNode* (red tile).

compared to an ordinary list, insertion time grows from $\mathcal{O}(1)$ to $\mathcal{O}(\log n)$ due to each insertion requiring a search.

Figure 3.4 shows the actual realization of the concept illustrated in Figure 3.2. A first *SkipList* keeps track of quantized x coordinates, thereby realizing depth level *1* of Figure 3.2; the items of the first *SkipList* are referred to as *xNodes* and colored in red; each *xNode* is in turn a *SkipList* which keeps track of quantized y coordinates, thus implementing depth level *2* of Figure 3.2; the items in these nested *SkipLists* are dubbed *yNodes* (green); eventually, each *yNode* is a *SkipList* of *zNodes* (blue), which represent the actual voxels and provide the containers for any kind of user data. Therefore, the concept shown in Figure 3.2 is realized by a novel data structure that may be thought of as a **Tree of SkipLists**. It is worth pointing out that with the proposed data structure the coordinates of a voxel can be obtained by iterating through its predecessors and thus need not to be stored in the containers together with user data; for example for the voxel referred to as z_3 in Figure 3.4, iterating back through predecessors provides coordinates

SkipList Depth	Integration Time	Visiting Time	Memory
4	56 ms	215 ms	**432 MB**
8	**29 ms**	269 ms	588 MB
16	**29 ms**	**215 ms**	900 MB
32	32 ms	259 ms	1524 MB
64	33 ms	258 ms	2743 MB

Table 3.1 Analysis of SkipList depth: tests performed on **Freiburg Campus** dataset with a resolution of $0.05m$. The Table reports the average computation time to integrate new sensor measurements ($\sim$180k points), the average time for a full visit of the map and the memory footprint of the map.

(x_0, y_1, z_3). A similar technique is used in *Octomap* to avoid coordinates storage in the leaves of the octree as clarified by Hornung et al. (2013).

As a detailed description of the *SkipList* is outside the scope of this thesis and can be found in the work of Pugh (1990), we conclude this section with a brief review of its key concepts related to this topic. A *SkipList* is a multi-level linked-list in which the first level is a list containing all the elements ordered by a *Key* (each node being a pair $< Key, Value >$); level i contains about half elements of level $i-1$, still ordered by *Key*. Similarly to a *Binary Tree*, a search is performed starting from level $i = i_{max}$ down to $i = 1$ in $\mathcal{O}(\log n)$, at the expense of memory footprint (due to replicated elements).

Depth (*i.e.* number of levels) is a parameter of the *SkipList* to be chosen based on application settings. Indeed, there exists an upper limit beyond which one gets no further benefits in terms of timing performance while significantly increasing memory footprint. As reported in Table 3.1, this is vouched also by our experimental findings.

3.2.2 Voxel indexing

As each node of our data structure is addressable by a *Key*, we can use it to map real world coordinates to quantized indexes just as it would happen in a 3D Grid. Thus, extending the concepts introduced in Equation 1.3, to retrieve the voxel $\mathbf{v} = \begin{bmatrix} I_x & I_y & I_z \end{bmatrix} \in \mathbb{Z}$ corresponding to a 3D point $\mathbf{p} = \begin{bmatrix} x & y & z \end{bmatrix} \in \mathbb{R}^3$:

$$I_x = \left\lfloor \frac{x}{r} \right\rfloor, I_y = \left\lfloor \frac{y}{r} \right\rfloor, I_z = \left\lfloor \frac{z}{r} \right\rfloor \tag{3.1}$$

r denoting, as usual, voxel resolution. Unlike a 3D Grid, however, we can use also negative indexes as they represent *Key*s of a map rather than simple indexes of an array. This is important for mapping applications as, more often than not, the ground reference of the map (aka *Zero Reference Frame*) is not known a priori.

With our data structure, querying for a voxel $f(I_x, I_y, I_z) = \mathbf{v}$ consists in executing the iterative query $h(g(f(I_x), I_y), I_z) = \mathbf{v}$. Thus, with reference to Figure 3.4:

- $f(\bullet)$ retrieves a *red tile* / *xNode*
- $g(\bullet)$ retrieves a *green tile* / *yNode*
- $h(\bullet)$ retrieves a *blue box* / *zNode* / *Voxel*

Each of three query function $f(\bullet), g(\bullet), h(\bullet)$ can result in either a *Hit* or a *Miss*. Moreover, each generic function $\phi(f(I_x))$ may be performed concurrently because it involves separate branches of the SkipList Tree (see again Figure 3.4). In simple terms each sub-tree related to a *xNode* is completely separated by its siblings, for this reason each of them can be accessed concurrently.

3.2.3 Parallelization

As highlighted in the previous section, the proposed data structure inherently provides for a high degree of parallelization. Besides, even a single *SkipList* may enable a certain level of parallelization by using locks on nodes, as introduced by Pugh (1998). However, we decided to exploit only the high parallelism among voxel indexing operations enabled by our data structure while not deploying also the lock-based technique to further parallelize accessed within a *SkipList*, mainly to maintain a lean and simpler code and secondly due to lock-based algorithm being often unpredictable, which makes them unsuited to real-time tasks.

As already mentioned the operations involving separated branches of the first level of our SkipList Tree, that is $f(I_{x_i}) \neq f(I_{x_j}) \rightarrow x_i \neq x_j$, can be performed in parallel. We can classify all the possible operations on the data structure into two main categories:

- *Visiting Operations*: Visiting the whole tree (*i.e.* reaching each voxel of the map) consists in visiting all first-level nodes in parallel and collecting the results:

$$\pi(\Gamma) = \sum_{i=first}^{last} \pi(f(I_{x_i})) \tag{3.2}$$

- *Updating Operation*: upon performing a generic update operation we cannot know in advance whether it will produce a new allocation or a deallocation or it will just update the content of an existing voxel without performing further search. To ensure that an update operation is not concurrent over others we can reuse again the previous technique: we assume that two update operations do not conflict if they belongs to two separate first level branches. In a typical application scenario we are given a set of sensor measurement to be integrated into the map: a set of 3D points: $C = \{(x,y,z)\}$. Hence, we can group these points in subsets according to their first quantized coordinates

$$C = \bigcup_{i=min}^{max} C_{x_i} \mid C_{x_i} : \{ \left\lfloor \frac{x}{r} \right\rfloor = I_{x_i} \} \tag{3.3}$$

  so to perform the integration operation dealing with each of the subsets in parallel while ensuring no concurrent memory access.

This kind of parallelization is useful not only to improve timing performance but also in scenarios in which map updates may occur from separated sensors in separated chunks, for example in multi-agent localization and mapping Parker et al. (2013). For the record, parallelization across *yNode*s is also possible but may lead to a computational overhead due to only *xNode*s being extensive. Nonetheless, we plan to investigate on a possible deeper parallelization of the computation.

3.2.4 Radius search

In a *SkipList* the Nearest Neighbor Search is straightforward: when we search for a *Key* in the list we always know the previous and next *Key*s present in the set, even when the searching *Key* is missing. A Radius Search around a target index is performed collecting all the elements between two indexes I_{r^-}, I_{r^+} obtained starting from a center index and computing the boundaries with discrete radius dimension:

$$I_{r^+}, I_{r^-} = I \pm \left\lfloor \frac{radius}{resolution} \right\rfloor \tag{3.4}$$

as a *SkipList* is an ordered linked-list, iterating from I_{r^-} to I_{r^+} allows for executing the search with $\mathcal{O}(k)$ time complexity, k being the number of elements within the range (Range Search). We can extend this approach to each of the *SkipList*s present in our Tree, so to perform a Range Search along each x, y, z dimension and obtain a Box Search. Then, filtering all the points found within the Box based on the distance from the box center allows for fetching a Sphere and thus achieve a Radius Search. As it will be shown in Section 3.4, thanks to the parallelization approach enabled by our data structure and discussed in previous section, our method outperforms standard implementation of search algorithms such as the *KD-Tree* or *Octree*.

3.3 Salient features of Skimap

In this section we present some additional modules deployed in our mapping system not essential to the core *SkiMap* data structure itself but that make this mapping framework somewhat unique, especially as regards Section 3.3.1.

3.3.1 Ground tracking and 2D querying

Although our proposal may be considered a generic 3D Mapping framework, it has been conceived to address robot navigation scenarios. Therefore, we found it useful to endow the framework with a module dedicated to tracking the ground plane. Thus, upon activation of the ground tracking module, the camera mounted on-board the robot must get a shot of the ground plane in the very first frame. The main plane found in the first frame is treated as the ground plane, which allows for classifying easily all the 3D points sensed in the successive frames as either *ground* or *obstacle* points. This technique permits also to set the *Zero Reference Frame* of our map in the centroid of the first floor, thereby ensuring that z coordinates are zero near the ground. More generally, if the core *SkiMap* algorithm may be provided with measured points classified as *ground* or *obstacles*, they can be integrated in the map differently, and, in particular, so as to reduce the time complexity to integrate the former. Indeed, with reference to Figure 3.4, integrating a *ground point* boils down to allocating or updating only a *green tile*/*yNode* rather than a voxel, which implies reaching just depth level 2 of our SkipList Tree, whilst integrating *obstacle* points would require going deeper to reach level 3.

We can make the same point as visiting through the SkipList Tree: should we wish to retrieve only the information about ground in order to obtain a 2D Map we would need to visit the tree only up to depth level 2, thereby reducing time complexity dramatically as vouched by Figure 3.8 . The figure shows also that the ability to create extremely rapidly a 2D view of the 3D Map is peculiar to *SkiMap*, a classical approach

like the *Octree* being much slower due to the need to visit all voxels and project them on the ground in order to retrieve a 2D Map.

3.3.2 Map continuous update on pose graph optimization

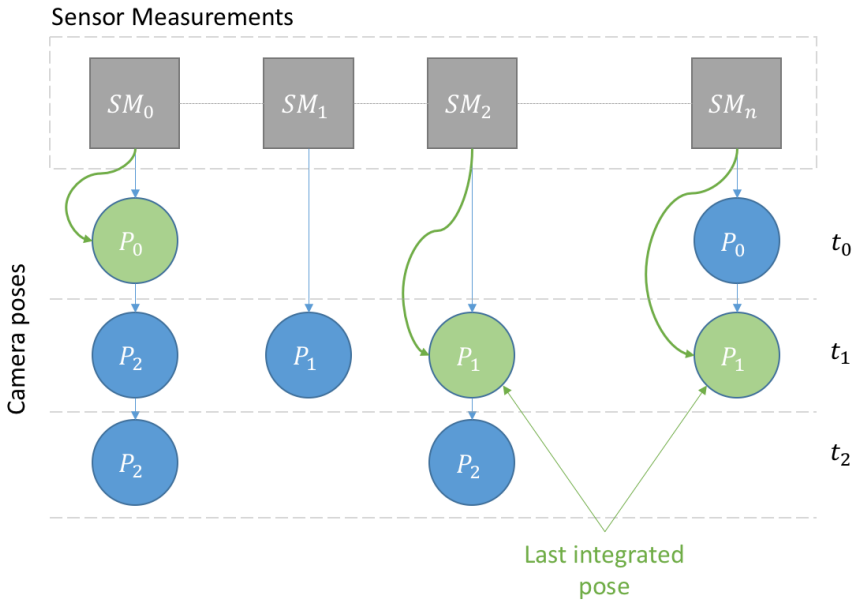


Fig. 3.5 The Pose History consists of a set of queues associated with Sensor Measurements (SM). This structure allows for linking diffent poses to any SM so to keep track of which pose has been used to integrate them into the map as well as of the existence of newer ones possibly produced by the on-line pose optimization process. For example, at time t_2 the history linked to SM_0 shows that the mesaurements have been fused into the map according to P_0 but there exists a newer pose, i.e. P_2: the Pose Integrator may choose to erode SM_0 from the map according to P_0 and fuse measurements back according to P_2, marking then the latter as the *last integrated pose* for SM_0. Conversely, the last pose and last fused pose associated with SM_n do coincide, so no action would be taken by Pose Integrator fot those measurements.

What we call *Map Update* is the procedure to fuse new sensor measurements in the model of the environment, i.e. the Map. In a real scenario the update procedure cannot rely on an unidirectional workflow, because for several reason we may need to remove from the Map past data. For the abovementioned reason, the *Map Update* cannot be a *Lossy* procedure but need to preserve temporal information in order to reverse any update.

The idea of *Erosion* of past sensor measurements and *Fusion* (or *Integration*) of new ones in a voxel grid was first introduced by Fioraio et al. (2015). The integration procedure, described by Curless and Levoy (1996), allows to *fuse* sensor measurement in a voxel grid according to a *weight*; for example, to integrate the occupancy probability:

$$P'(v) = \frac{P(v)W(v) + p_i(v)w_i(i)}{W(v) + w_i(v)}, W'(v) = W(v) + w_i(v) \tag{3.5}$$

where $P'(v), P(v)$ are the new and old occupancy probability of voxel v, respectively. $W'(v), W(v)$ the new and the old weight. As proposed in Fioraio et al. (2015), the *Erosion* process consists in just inverting the integration process:

$$P'(v) = \frac{P(v)W(v) - p_i(v)w_i(i)}{W(v) - w_i(v)}, W'(v) = W(v) - w_i(v) \tag{3.6}$$

Erosion and fusion of sensor measurements may be deployed in conjunction with any sensor localization module capable of delivering optimized poses, e.g. upon detection of a loop closure. Thereby, the map may be updated by removing sensor measurements according to old poses and fusing them back according to the new, optimized poses. Our mapping system supports this feature by overriding simple *plus* and *minus* operation for the generic data type associated with each voxel, which allows the user to handle any desired kind of measurement (Occupancy Probability, SDF, RGB ....) in order to implement Equation 3.5 and Equation 3.6, provided that the operation of integrating new measurement in the global map is reversible.

However, though a sensor tracker typically produces poses at a certain controlled and approximately fixed pace (*e.g.* at every new sensor measurement or a controlled subset of them), optimized poses are delivered asynchronously with respect to such a regular rhythm, *e.g.* because a loop closure has been detected, and may happen to compete with live tracking as concerns updating the map. Therefore, as illustrated in Figure 3.5, we have endowed *SkiMap* with a *Pose Manager* capable to create a *Pose History*: the system treats live poses and optimized poses seamlessly by inserting them in a set of queues, each associated with the sensor measurements (e.g. a depth image) taken at a certain time stamp; a *Pose Integrator* chooses from the *Pose History* a subset of poses and integrates the associated sensor measurements in the voxel map; if the pose that's about to be integrated is an optimized one, its predecessor will be *eroded* from the map first. The choice of the subset of poses to be integrated into the map occurs according to the following criteria:

- live poses must be integrated as soon as possible.
- among optimized poses, those spatially closer to the current live pose are picked first.
- the upper bound of the subset cardinality is fixed to ensure predictable computation time.

3.4 Experiments

3.4.1 Implementation details

SkiMap is implemented in C++ and wrapped in a ROS package, so to maximize its portability and usability in the robotics community. Thanks to widespread use of *C++ Generics*, the*SkiMap* data structure is contained in a couple of header files. Furthermore *C++ Generics* enable to chose *Data Type* to represent coordinates: for example, in our current implementation we have chosen *short* as index data type allowing values in range $[-32.768, 32.768]$, which results in a map of $655.36m$ along each dimension with a resolution of $0.01m$, according to Equation 1.3. Also voxels are *templetized* so to allow the user to store whatever information therein, provided that the data structure, representing custom voxel content, respects the assumption described in Section 3.3.2.

3.4.2 Results

The *SkiMap* mapping framework has been evaluated using some heterogeneous datasets categorized as follows:

- Medium-sized datasets captured with RGB-D sensors (Sturm et al., 2012).
- Public large-sized datasets captured with laser scanners mounted on pan-tilt units (*Freiburg Campus* [2], *New College* by Smith et al. (2009)).
- Small and Medium-sized datasets captured in our Lab through RGB-D sensor on mobile robots (Figure 3.11).

The public datasets are endowed with ground truth camera poses, while in the experiments concerning our datasets we deploy *Slamdunk* (Fioraio and Di Stefano, 2015) to track the camera. Thus, the quantitative evaluation reported in Figure 3.6, Figure 3.7, Figure 3.8, Figure 3.9 deals with the first two categories only - because of the availability of ground poses - and concerns a comparison between *SkiMap* and the *Octree*[3] that is, to the best of our knowledge, the foremost mapping solution, provided with spatial queries, in terms of memory efficiency. To attain a more comprehensive assessment, for each dataset we have considered multiple map resolutions,*i.e.* $0.05m$, $0.1m$ and $0.2m$. In Figure 3.9 we have considered also the *kd-tree*[4] because of its wide adoption

[2]Courtesy of B. Steder, available at http://ais.informatik.uni-freiburg.de/projects/datasets/fr360/

[3]version used: https://github.com/OctoMap/octomap

[4]version used: http://pointclouds.org/

in spatial search tasks such as *radius search*. All the experiments have been run on a 5th generation Intel Core i7.

First we have assessed basic tasks like *"Integrating New Measurements"* (Figure 3.6) and *"Visiting the Map"* (Figure 3.7), finding out that *SkiMap* is almost always more efficient than the *Octree*. Figure 3.8 highlights how the *2D Query* feature introduced in Section 3.3.1 enables to outperform the *Octree* in obtaining a similar representation. A qualitative example of the *2D Query* feature can also be seen in Figure 3.10, with the ground correctly reconstructed; it is worthwhile pointing out here that, as vouched by Figure 3.8, obtaining this kind of representation by performing per-voxel projection to ground would imply a significantly higher time complexity. Finally, Figure 3.9 is about the timing performance of the *radius search* task, quite relevant, e.g., for the sake of avoiding obstacles while navigating within the workspace under reconstruction Figure 3.9 points out the much higher efficiency of *SkiMap* with respect to both *Octree* and *kd-tree*, even without considering the initialization time to build the spatial index required by the *kd-tree* which is not accounted for in the aforementioned figure. As for memory occupancy, Table 3.2 highlights how *SkiMap* tend to be almost as efficient as the *Octree* in case of large environments while providing less memory savings with smaller workspaces.

As for the experiments dealing with datasets taken in our Lab, we used two mobile robots, namely *Youbot* (Bischoff et al., 2011) and *Tiago* [5], equipped with a Asus Xtion RGB-D sensor and, rather than relying on ground truth information, deployed *SlamDunk* (Fioraio and Di Stefano, 2015) to track the robot/camera 6-DOF pose and fuse sensor measurements into the map according to the estimated poses. Furthermore, leveraging on the *Pose Optimization* module offered by SlamDunk, we can realize the *Map Update* feature of *SkiMap* (see Section 3.3.2). Both robots were operated manually, in small (*Youbot*) and large (*Tiago*) sized environments within our Lab, so to collect and fuse together multiple sensor measurements in order to reconstruct a map of the explored workspace. Figure 3.11 depicts examples of reconstructed maps with and without the *Map Update* process enabled by SlamDunk's *Pose Optimization* module. It is worthwhile pointing out that with our approach the optimized maps are not attained off-line within a post-processing step but built in real-time as described in Section 3.3.2.

This Video [6] presents a qualitative evaluation of all the *SkiMap* relevant parts, by exploiting real mobile robots to perform medium-large scale environment reconstruction.

[5] http://tiago.pal-robotics.com/
[6] https://www.youtube.com/watch?v=MverWmFAgkg

3.5 Skimap extensions

So far we have described a novel mapping approach mainly devoted to robot navigation. The primary objective was to provide an efficient mapping framework suitable to real-time applications in embedded robotics platforms. Thus, unlike approaches that focus on dense and accurate 3D reconstruction, such as e.g. Dai et al. (2016), our method is aimed at building as efficiently as possible the kinds of representation required to support robot navigation effectively. Moreover, the framework can also provide some basic form of semantic information, such as telling apart *ground* and *obstacles*. In the next chapter we describe how to enrich the degree of semantic perception accommodated by *SkiMap* by incorporating detection of certain object instances Fioraio and Di Stefano (2013), e.g. items to be picked or manipulated by the robot.

Dataset	**Type**	**Memory Saving wrt 3D Grid**		
		Resolutions		
		0.05m	**0.1m**	**0.2m**
Freiburg Campus $292 \times 167 \times 28m^3$				
	octree	98.75%	96.21%	90.61%
	skimap	98.52%	94.28%	83.33%
New Dataset College $250 \times 161 \times 33m^3$				
	octree	99.74%	99.00%	96.76%
	skimap	99.77%	98.84%	95.30%
Freiburg Long Office $23 \times 25 \times 10m^3$				
	octree	90.50%	84.71%	74.91%
	skimap	82.62%	71.30%	54.63%

Table 3.2 Percentage of memory savings with respect to a full 3D grid.

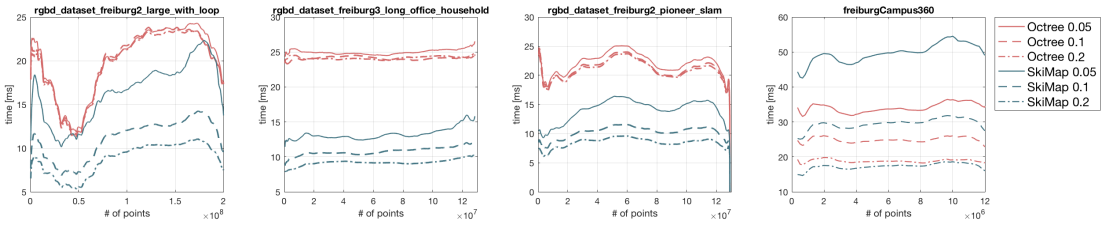


Fig. 3.6 Time to integrate new measurements into the map with increasing number of total points. The first three datasets deal with RGB-D sensors ($\sim$ 320k points per scan) while the last one was acquired by a Laser Scanner mounted on Pan-Tilt unit ($\sim$ 180k points per scan). SkiMap provides inferior performance in the last dataset due to the scans featuring very spread and distant points (up to 50m).

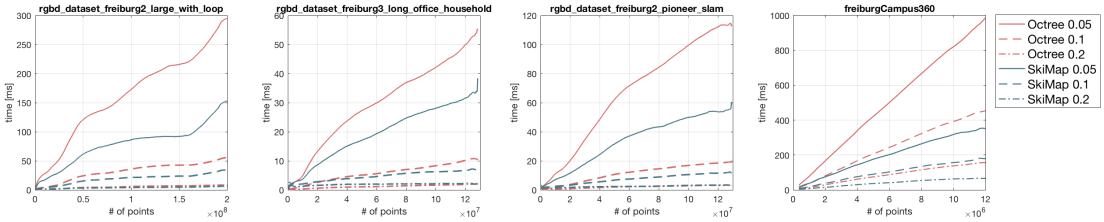


Fig. 3.7 Time to visit the whole map. Visiting in this case means the retrieval of the whole voxels set *e.g.* for visualization purposes or for global navigation path planning.

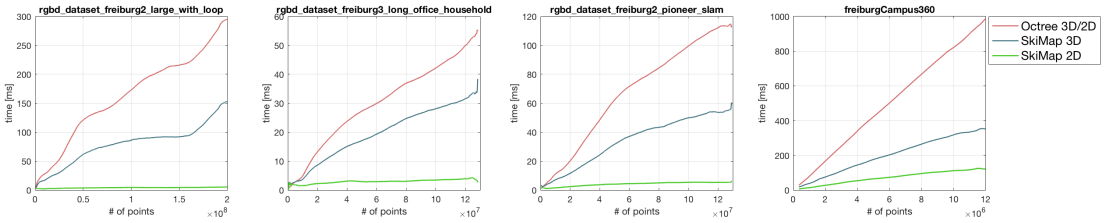


Fig. 3.8 Comparison between 3D and 2D reconstructions. The *Octree* requires the same time to perform a full 3D or a 2D reconstruction because in both cases it needs to iterate over all the 3D points. *SkiMap*, instead, turns out faster than the *Octree* in obtaining a 3D map as well as much faster in creating a 2D map thanks to the *2D Query* feature.

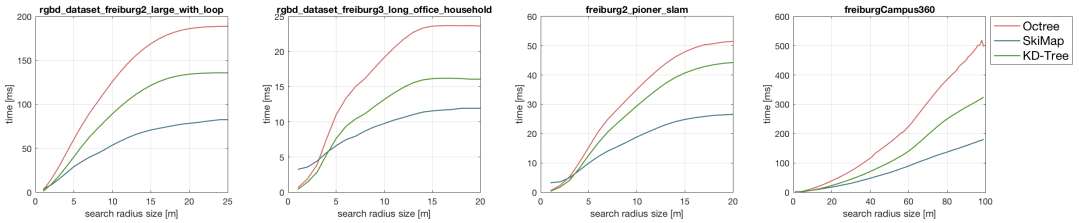


Fig. 3.9 Time to perform a radius search with increasing of radius size. *SkiMap* outperforms both the *Octree* and the *kd-tree* on all datasets.

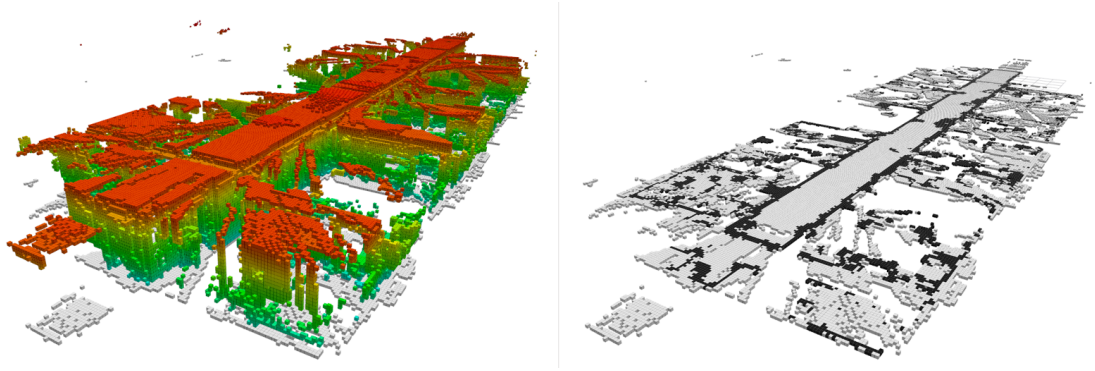


Fig. 3.10 A Map built from Corridor Dataset collected in *Octomap* (Hornung et al., 2013). *SkiMap* allows for efficiently detecting the ground and, without further computational cost, discard higher obstacles like the roof (red voxels in the left image) and labeling the ground voxels as *navigable* (white regions in the right image).

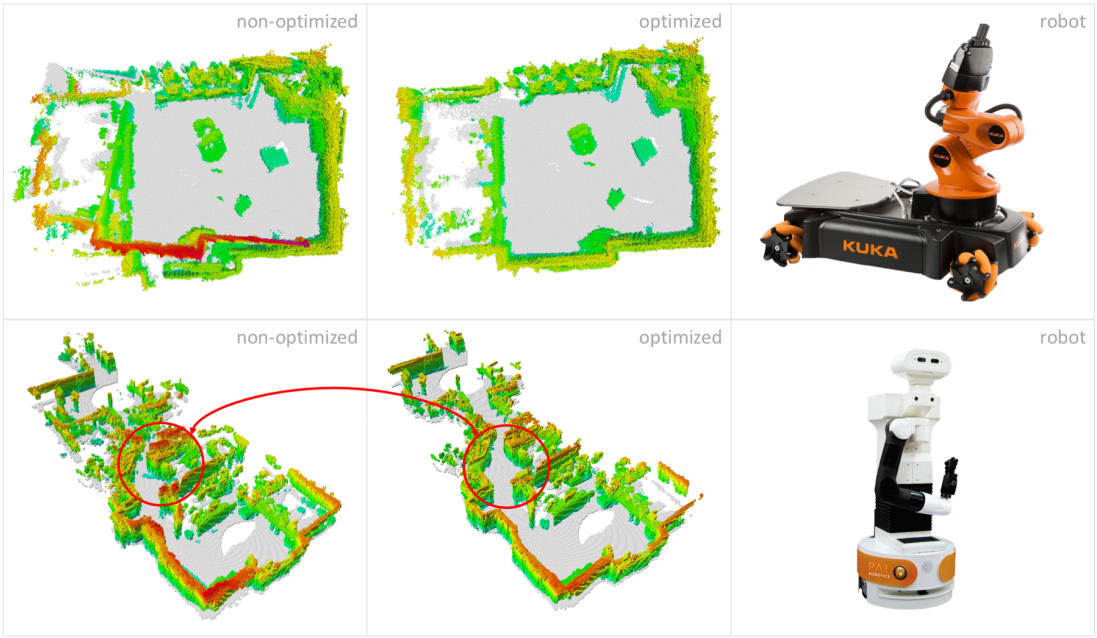


Fig. 3.11 The first row concerns a small room ($5m \times 4m \times 3m$) reconstructed by *Youbot* in *eye-on-hand* configuration. The second row represents a medium-size environment ($8m \times 35m \times 3m$) reconstructed by *Tiago* through an RGB-D camera mounted on the head. The middle column highlights the significant improvement in reconstruction accuracy provided by the real-time map optimization process.

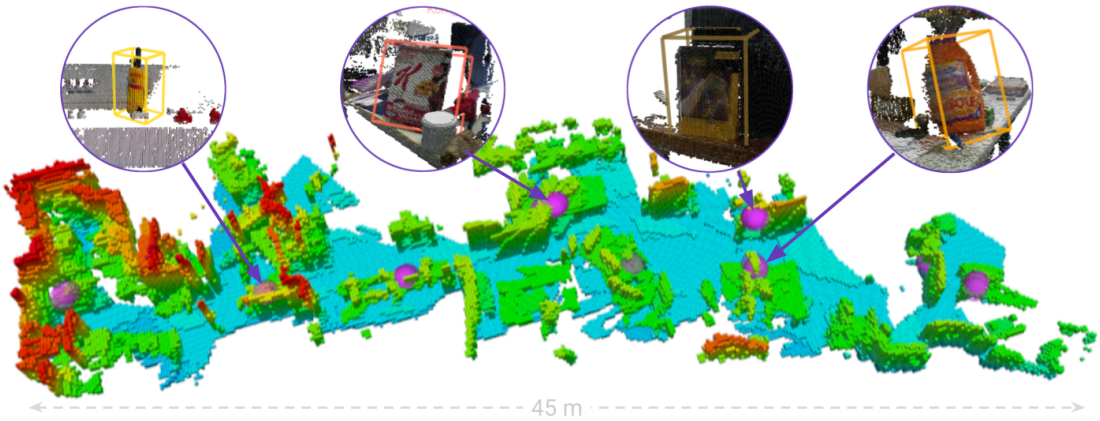


Fig. 4.1 Large-scale map reconstructed online by SkiMap++ through a mobile robot equipped with an head-mounted RGB-D camera. Purple spheres represent areas found alongside with reconstruction which are likely to contain object instances. Magnified circles represent outcomes of the final *Instance Estimation Algorithm*, which is performed in the aforementioned areas only. The whole map is acquired by relying on the robot's own odometry in order to track camera poses over time.

Chapter 4

Sparse semantic mapping for robotic manipulation: SkiMap++

4.1 Introduction and related works

In previous chapters we have discussed how autonomous robots may rely on suitable mapping modules to navigate within an unknown space (Chapter 3) or to manipulate unknown objects (Chapter 2). In this chapter we are concerned with extending *SkiMap* (described in Chapter 3) to enrich the representation to the level of semantics, i.e. so as to go beyond pure geometric mapping and incorporate information to enable detection of certain objects as well as estimation of their 6-DoF poses in the world space. This novel type of semantic mapping might be useful, e.g., to support an autonomous robot that would navigate within an unknown environment while seeking for certain objects

that, if found, should be picked, as it might occur, for example, in some service or rescue scenarios.

In literature, many works tackled the problem of 3D object detection and pose estimation from RGB-D images. Such techniques can be split between those relying on a single view to perform detection like Wohlhart and Lepetit (2015), Chi Li et al. (2016) or Brachmann et al. (2016); and those techniques deploying multiple images from several vantage points to ascertain whether an object is located in the observed scene and compute its pose. Multi-view object detection pipelines have been presented Thomas et al. (2006), who track feature points across views to determine the location and pose of the objects of interest jointly. Collet and Srinivasa (2010) propose to handle each view independently and then perform a global refinement. Civera et al. (2011) detect object instances in a sequence of images by means of SURF correspondences (Bay et al., 2006) and insert such objects into a map refined by a SLAM (Simultaneous Localization and Mapping) algorithm. This work introduced the idea of integrating the object detection into a SLAM pipeline to increase resilience of object localization with respect to the partial occlusions occurring in a single view: by reconstructing a consistent model of the 3D scene, and performing the detection therein, one can robustly identify the instances of interest by exploiting -possibly partial- evidence accumulated over time.

Object detection and 6-DoF pose estimation from 3D data may be achieved by detecting 3D keypoints, then computing and matching 3D descriptors between the current scene and a set of 3D models as described in Aldoma et al. (2013, 2012); Rusu et al. (2010); Salti et al. (2014). A different approach is due to Lai et al. (2014, 2012), who project per-pixel object probabilities from RGB-D frames onto voxels to obtain a semantic labeling of the scene, though in this work object poses are not estimated.

Alternatively, one can augment a SLAM pipeline to account for the task of object instance detection: the works of Salas-Moreno et al. (2013) and Fioraio and Di Stefano (2013) were among the first papers to propose leveraging on recognized object instances as a means to improve the consistency of the SLAM process and vice-versa. Tateno et al. (2016) propose to rely on a framework that simultaneously deploys a SLAM algorithm (used to obtain a reconstruction of the scene), a segmentation algorithm and an object recognition algorithm, so as to match descriptors to such segments to provide accurate and stable 6-DoF poses for the objects of interest. Li et al. (2016) also rely on the idea of synergistic exploitation of SLAM reconstructions for object detection, in order to improve scene understanding. This task is accomplished by fusing object hypotheses from single frames (possibly depicting partially-occluded instances) into a Global Semantic Map, as introduced by Tateno et al. (2015).

This chapter follows the above-mentioned line of work by presenting a novel framework, referred to as SkiMap++, which allows for simultaneous recognition of objects and reconstruction of the environment as explored by a mobile agent. By accumulating evidence for objects into an extensible, real-time, mapping system, *SkiMap++* can detect the presence of objects of interest in a 3D reconstruction of the scene and estimate their 6-DoF pose. The name is – obviously – inherited from the mapping framework introduced in Chapter 3.

This chapter is divided essentially in two main sections: the first section describes how to train our system with N generic target objects (*Offline Pipeline* Section 4.2); then the second section describes how the system employs the outcome from the first training procedure to perform object recognition and pose estimation during real-time operation (*Online Pipeline* Section 4.3).

Why SkiMap? The *SkiMap++* framework is based on the SkiMap structure, which we introduced in Chapter 3 to realize efficient real-time mapping of large scale environments. Among the key features of this proposal are: Suited for large scale environments, thanks to a low memory footprint; Fast random voxel access $O(\log n)$; Equipped with several components implementing the Fusion/Erosion technique from Fioraio et al. (2015), so as to optimize the map on-line alongside with reconstruction; Ability to perform radius-based search with better performance than *Octree*(Meagher, 1982) and *Kd-Tree* (Bentley, 1975). The flexibility of the SkiMap data structure allow us to adopt it in the *SkiMap++* pipeline in order to store not only the map of the explored workspace but also the kind of data instrumental to the Object Recognition task, *i.e.*, in our proposal, *2D Features*, *Object Hypotheses* and *Guessed Instances*. These additional data-types need to be queried and updated in real-time and at the same time they need to be stored in a map as large as the mapped environment. Thus, *SkiMap++* relies on continuous update of these heterogeneous maps and schedules queries on them so as to speculate on 6-DOF Objects Poses.

4.2 Offline pipeline

The *SkiMap++* object recognition approach is based on detection of *sparse 2D features*, that are then matched against a pre-trained Object Database, to achieve full 6-DoF pose estimation of target objects alongside with reconstruction of the environment. The matching component is built upon a Random Forest Classification system able to predict, given a 2D feature descriptor, to which object the feature belongs to together with the coordinates of the voxel – in the object reference frame – in which the said feature was found during the training phase. Brachmann et al. (2014) investigated the

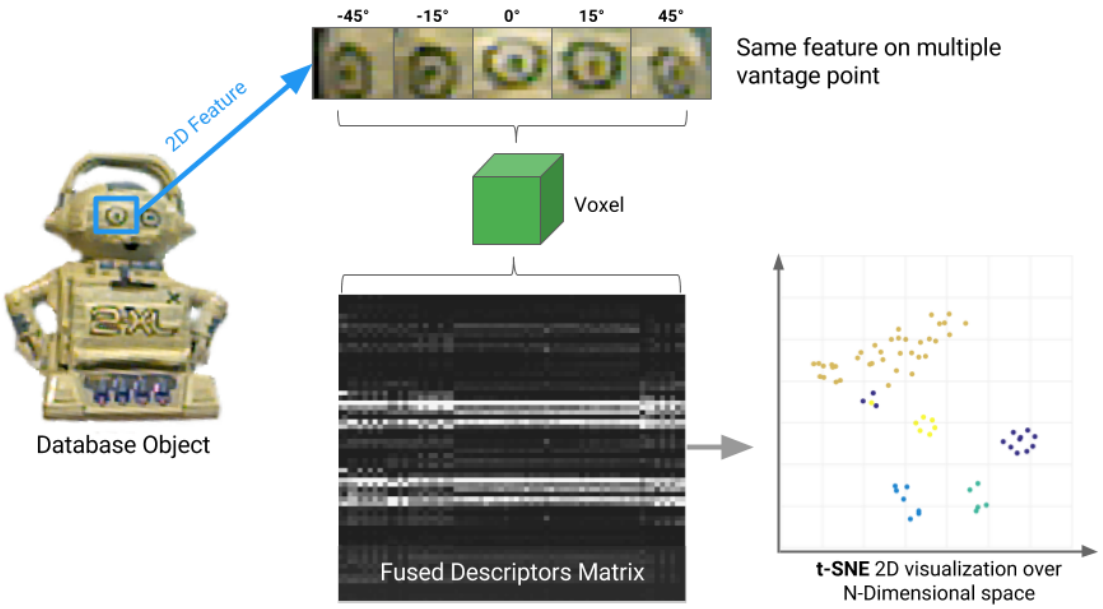


Fig. 4.2 Each object feature looks differently depending on the vantage point. Experimental results show that fusing together, inside the same voxel, multiple descriptors computed from different viewpoints yields a *Descriptor Matrix* representing a multi-modal distribution in the descriptors space $\mathbb{R}^n$. A 2D visualization of descriptors obtained by *t-SNE* (Maaten and Hinton, 2008) highlights how these different descriptors tend to concentrate into a few clusters.

Decision Forest approaches to predict, from a given feature, the object class and its position in the model reference frame. Formally:

$$p(c|d) \qquad p(\mathbf{y}|c) \tag{4.1}$$

namely the decision forest predicts the class $c \in C$ – given a feature $d \in \mathbb{R}^n$, the n-dimensional feature space – as well as the probability of object point $\mathbf{y}$ – in the object coordinate system – given class c. Indeed, the prediction $p(\mathbf{y}|c)$ in the second step is achieved by storing, at training time, all feature positions in the leaf of the decision trees, filling a multi-modal distribution in $\mathbb{R}^3$ discretized over a $5 \times 5 \times 5$ fixed grid. Furthermore, by adopting a *dense feature* approach, the technique introduced by Brachmann et al. (2014) can predict during the online phase, the eligible object class and its internal point given a generic *pixel* (input images are densely described).

In *SkiMap++* we exploited a similar approach, but focused on the analysis of the multi-modal distribution in the descriptor space $\mathbb{R}^n$, a distribution that grows during the database acquisition of each region of the target object model. As can be seen in Figure 4.2, if we observe a target object from different vantage points, the same keypoint (*e.g.* the one belonging to the *eye* of the *Toy Robot*) is likely to show different appearances depending on the point of view. Multiple appearances in the *descriptor* space yield different descriptor vectors $d \in \mathbb{R}^n$ modeled as a *Mixture of Multi-Variate*

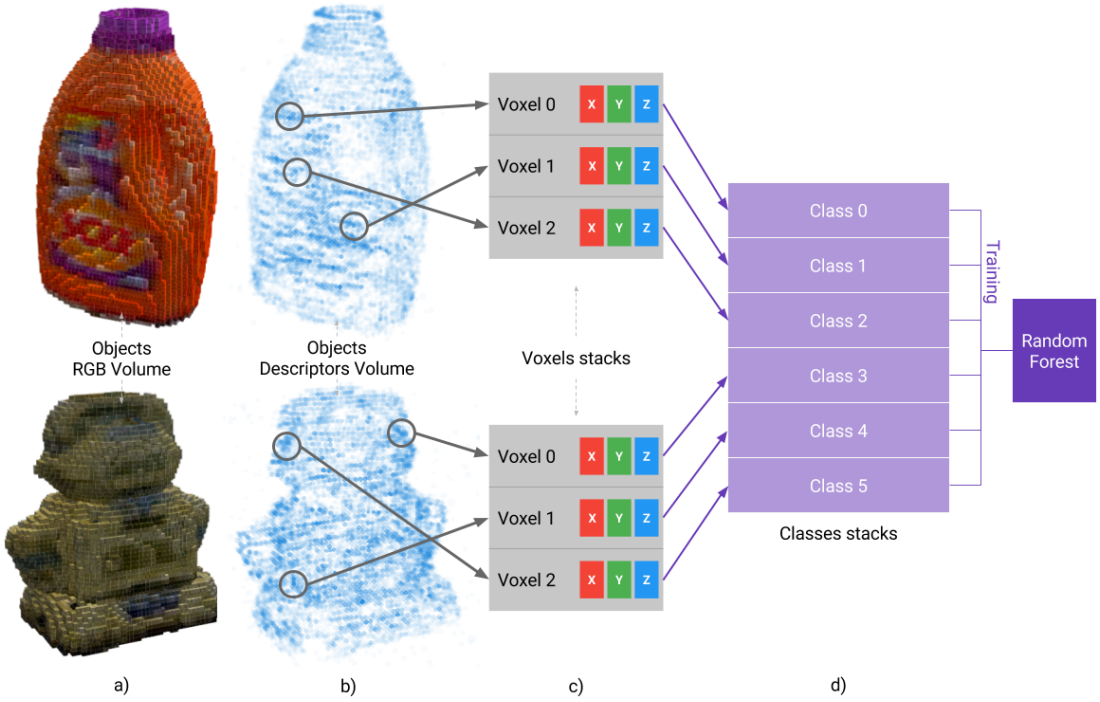


Fig. 4.3 The stacking procedure used in *SkiMap++* to create the Object Dataset and train the associated Classifier which can then be used on-line to perform object recognition. Column *a)* shows the reconstructed RGB Volumes of two objects (A *Bottle of Detergent* and a *Toy Robot* respectively). Column *b)* depicts the Descriptors Voxels Volume containing descriptions of multiple appearances as Figure 4.2. Column *c)* shows equally sized voxels stacks, ordered by cardinality, for each object. Finally, in *d)*, voxels stacks are merged into a global Classes Stack that will represent the prediction *target* for the forest training process.

distributions over $\mathbb{R}^n$. In the experiment outlined in Figure 4.2 we use SURF features (Bay et al., 2006) to detect and describe keypoints, resulting in a $\mathbb{R}^{64}$ descriptor space. As shown in in Figure 4.2, given an object feature we employed the *t-SNE* technique, presented by Maaten and Hinton (2008), to analyze the distribution of descriptors dealing with different vantage points and found that these tend to form a small number of clusters. In this way is possible to display a the *Descriptor Matrix* (*i.e.* a matrix $M \in \mathbb{R}^{n \times m}$ whose columns are m samples belonging to the descriptor space $\mathbb{R}^n$) to prove that descriptors are condensed around a bunch of means.

We deploy a classifier trained to predict from keypoint descriptors the 3D voxel – in the object coordinate system – wherein the 3D feature originating is likely to falls within. To achieve this, we exploit SkiMap also during the database creation phase, so to build multiple voxel maps for each object: one fuses RGB data (to obtain an user-friendly object representation, see column *a)* in Figure 4.3) while another one fuses together descriptors in such a way that each voxel stores a *Descriptor Matrix* (we will use interchangeably the terms *Descriptor Voxel* and *Descriptor Matrix Voxel*, see column *b)* in Figure 4.3). Figure 4.6 illustrates this dualism between RGB voxels

and the corresponding Descriptor Voxels. We denote a *class* $c \in C$ for each Descriptor Voxel, determining a mapping function that from c allows us to easily compute the corresponding point in the object coordinate frame. We then train a Random Decision Forest to predict, given a target 2D feature, the object class as well as the voxel containing it:

$$p(c|d) \qquad v(c) = \mathbf{v}_j \tag{4.2}$$

we replace the second part of Equation 4.1 with a deterministic function to compute the exact voxel given a predicted class c. To avoid confusion we need to define the difference between a predicted *class* and the *labels* used in the semantic labeling procedure: the *label* is $l \in \{0, 1, ...m-1\}$ where m is the number of training objects; a *class*, instead, predicts both the object as well as the voxel therein, $c \in \{0, 1, ..., m*k-1\}$ where k is the size of subset of Descriptor Voxels choosen among all object's voxels. We can easily convert a *class* in the corresponding *label*:

$$l = \lfloor \frac{c}{k} \rfloor \tag{4.3}$$

but not vice-versa because the $\lfloor \cdot \rfloor$ operator makes it a lossy procedure. As the number of Descriptor Voxels could be very large we need to choose k carefully: in our system we choose to stack the Descriptor Voxels of each object ordered by their cardinality, then keeping those containing more cues. Figure 4.3 shows a sample object database (containing only two models) to clarify the process to convert each voxel into a class and vice-versa:

$$v(c) = \mathbf{v}_j \qquad o(\mathbf{v_j}) = o_i = l \tag{4.4}$$

with o_i the index of the object containing voxel $\mathbf{v_j}$, in other terms: the *label*. Interestingly, this process might be thought of as a Local-to-Global indexing conversion. Having obtained a set of classes, each of which originated by a $d \in \mathbb{R}^n$ vector, we can train a decision forest classifier according to the standard procedure described by Criminisi and Shotton (2013). Without any loss of information, we reduce the memory footprint of Descriptor Voxels by computing the means of the relative mixture of gaussian (*e.g.* centroids in $\mathbb{R}^n$ of each eligible cluster, as highlighted in the *t-SNE* representation shown in Figure 4.2). As the number of clusters is not known a-priori, we adopt the *Mean Shift* clustering approach by Cheng (1995).

The aforementioned compression procedure is very useful also to validate the prediction of the Random Forest. If we define $\{\mathbf{m}_0, ..., \mathbf{m}_s\}$ as the set of clusters computed with *Mean Shift*, with $\mathbf{m}_i \in \mathbb{R}^n$, and we have the Classifier prediction c

originated by the feature $d \in \mathbb{R}^n$, we can easily retrieve the corresponding set of clusters associated with the voxel $\mathbf{v}_j = v(c)$ (Equation 4.4). With:

$$\delta_{min} = \min_{\mathbf{m}_i} \delta(d, \mathbf{m}_i) \tag{4.5}$$

we can compute the cluster $\mathbf{m}_i$ with the minimum distance δ_{min}, defining a generic distance function $\delta(\cdot)$, to the starting descriptor d. Choosing a custom threshold δ_{th}, we can discard every prediction with $\delta_{min} > \delta_{th}$ by assuming that the prediction is wrong because the descriptor d belongs to an *alien* cluster $\mathbf{m}_i$ not present into the targer voxel. The study of the correct distance function $\delta(\cdot)$ is not present in this work but we can say that the best function may be the *Mahalanobis distance*, described by Mahalanobis (1936), which, however, requires the storage of the covariance matrix of hundreds of samples, at the detriment of the compression; in our experiments we used the simple $L2$ distance which however improve classification performances.

It is worth pointing out that the proposed *SkiMap++* framework is detector-agnostic. In fact, in our implementation the adopted 2D feature detector-descriptor is just a parameter of the system, as k the number of *classes* per object. In Section 4.4 we show some results while varying these parameters.

4.2.1 Built-in Model Database Compression

In object recognition pipelines based on 2D feature detection, the recognition rates turn out quite negatively affected by discrepancies between off-line and on-line conditions such as different light sources. So to strengthen the algorithm, we deemed useful to provide more evidence into the database: in our experiments, for example, we evaluated the performance and memory footprint of whole system while adding more amd more evidence. Intuitively, in *SkiMap++* more visual cues only leads to increasing cardinality of each voxel in the Descriptor Volume without increasing – meaningfully – the memory footprint (shown in Figure 4.4), which would be the case if we stored each single RGB-D frame of the object's scan. Thanks to *Mean Shift* clusters analysis we can further compress the amount of stored information by saving only clusters centers.

4.3 Online pipeline

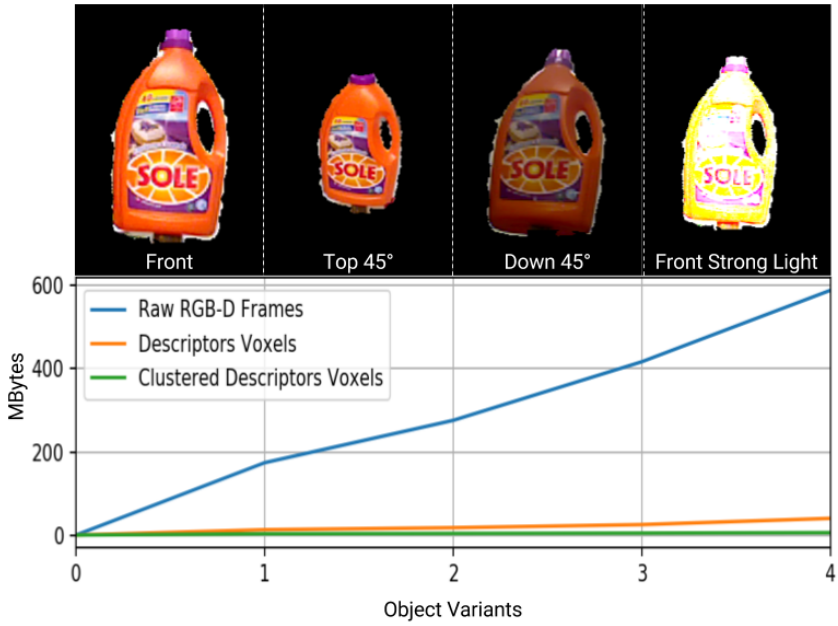


Fig. 4.4 For an object in the dataset many variants may be acquired. In the figure each variant is intended as a full rotation around the object with the RGB-D camera in different conditions, *e.g.* in this figure from the *Front* or from the *Top* with an angle of 45°, and so on. Each variant enriches the object description by filling Descriptor Voxels with additional evidences. Clustering the descriptors ensures the further decrease of memory footprint compared to the usage of all descriptors computed from the raw RGB-D frames. Storing the clustered representation of Descriptors is necessary to verify Classifier prediction as described in Section 4.2,Equation 4.5.

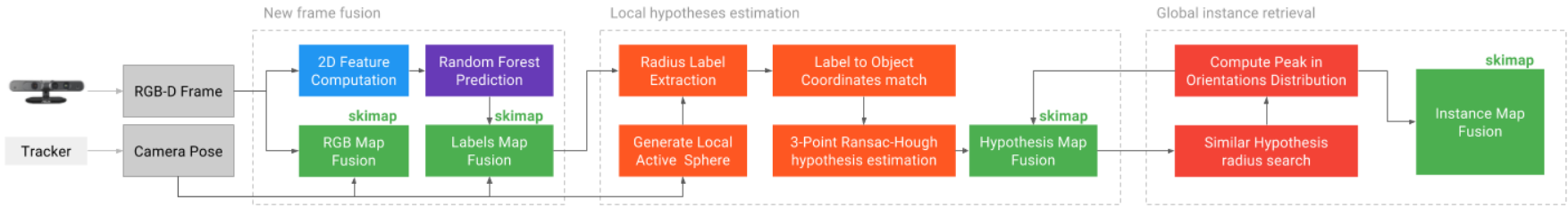


Fig. 4.5 *SkiMap++* online pipeline. First, new frames are integrated in two separate maps, one for RGB data and the other for labels. A *local active sphere* is generated according to the current camera pose, this sphere is used to query the label map to obtain objects matches and compute local hypotheses. Such hypotheses will be fused into another map. Given a target region inside the Hypotheses Map, the last phase of the pipeline entails the identification of the hypothesis with highest score, performing a radius search of similar hypotheses and merging them together to estimate and refine the final 6-DOF pose. Refined hypotheses shall be considered as object *Instances* and will be fused again in a global instances map.

In this section we examine the *SkiMap++* pipeline from acquisition of a the new frame to the final object instance recognition and pose-estimation. As depicted in Figure 4.5, the work-flow is subdivided into three macro blocks each of which affecting a different SkiMap volume. In the next subsections each block will be explained in detail. To summarize: the input data for the *SkiMap++* recognition pipeline is a generic pair *(RGB-D frame, camera pose)*, regardless of how they are generated; RGB-D data are integrated into a *SkiMap* RGB Volume according to the associated pose in order to reconstruct the environment. Simultaneously, sparse *2D features* are extracted from the current frame and processed by the *Random Forest Classifier*, built with the procedure described in Section 4.2, in order to predict *Labels* which will be fused into an associated semantic SkiMap Volume (Section 4.3.1): Figure 4.6 illustrated the dualism between RGB and Labels volumes.

For each *Camera Pose* an *Active Sphere* can be derived, so as to outline a local area of interest, over *labels* volume, on which formulate hypotheses about objects, whose hypotheses will in turn be fused into a further SkiMap (Section 4.3.2). Finally we can perform a 3D query on the last Hypotheses Map to retrieve final object instances, resulting from the aggregation of Hypotheses in the neighbourhood of the queried point.

4.3.1 Frame Integration Module

The first stage of *SkiMap++* is the *Frame Integration* block: this sub-component of the system is responsible of two main, independent, mapping tasks.

The first task is to build the RGB Volume, just as in the original SkiMap approach (Chapter 3). Accordingly, colour information is fused into a *RGBWeightedVoxel* data structure implementing weighted *sum/subtraction* operations. As mentioned in the related chapter, this is a peculiar trait of SkiMap, which allows the system to *integrate* new sensor measurements or *de-integrate* past data marked as invalid.

The second – and more important for the purposes of this chapter – task is the computation of a semantic map, which leverages again on the SkiMap data structure. This semantic map is fed with *Labels* predicted trough the Random Forest described in Section 4.2, using as input data the 2D Sparse Features detected in the current frame. Figure 4.6 illustrates a portion of the semantic map in which voxels are coloured according to the predominant *Label* within. In this case, in fact, unlike a *RGBWeightedVoxel* that could be merged by mixing colours, labels cannot be naively aggregated because of their categorical nature. Thus, the semantic map adopts as base element a *MultiLabelVoxel* implementing the *sum/subtraction* operations, as follows:

$$V_i = \{\langle l, w_i \rangle : l \in L, w_i \in \mathbb{Z}_0^+\} \tag{4.6}$$

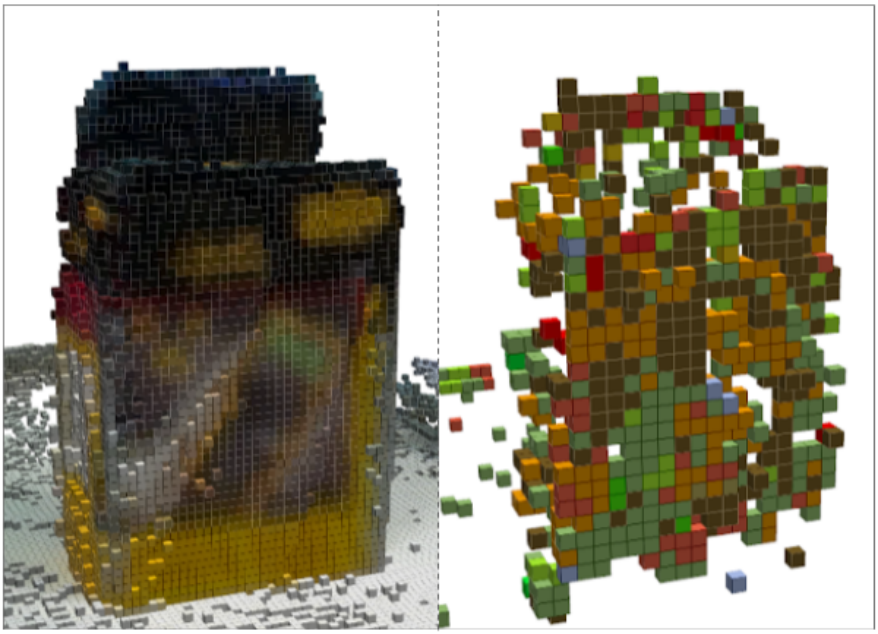


Fig. 4.6 The left part depicts a portion of environment reconstructed through *SkiMap++* containing the *multimeter* object (the first object in Figure 4.12), fusing RGB-D Frames captured from multiple vantage points. On the right, the corresponding semantic map obtained by fusing *Labels* is shown instead. For visualisation purposes, voxels in the latter representation are coloured depending on object to which they belong. In this case, *brown* voxels are those belonging to the *multimeter*.

$$V_1 \pm V_2 = V_3 \rightarrow V_3 = \{\langle l, w_1 \pm w_2 \rangle\} \tag{4.7}$$

Equation 4.6 describes a generic *MultiLabelVoxel* V_i as a *MultiSet*, namely a *Set* with repetitions, of classes $l \in L$ with cardinality w_i (*i.e.* repetition counter). This notation is necessary to allow a Label to be fused into a Voxel repeatedly, without losing the cardinality information; furthermore we allow two Voxels to be merged together through the lossless procedure described by Equation 4.7). Such kind of Voxel shall be equipped with a method to retrieve the maximal ordered pair: $\langle l, w_i \rangle$ in order to fetch the largest Label and its weight during the matching phase.

The main purpose of the subsystem described above is to provide a – searchable – semantic map through which we can retrieve all labels belonging to a given object. In the next section we will detail how to speculate on object 6-DoF pose hypotheses starting from a local map of labels.

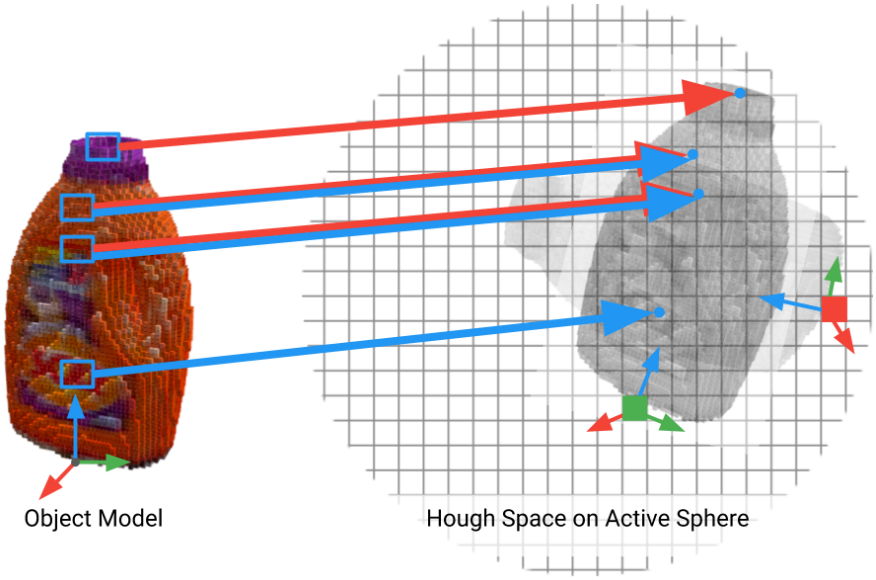


Fig. 4.7 The right part of the image depicts the *Active Sphere* on which we build the 3D Hough space. For each three random correspondences we can project an Object Reference Frame ${}^{obj}\mathbf{T}_i$ inside the sphere computing the relative Reference Frame ${}^{H}\mathbf{T}_{obj}$ in the Hough coordinate space and cast votes for the object base in the relative bin: in the figure the first match (blue arrows) and the second match (red arrows) project the object base in the same bin (green square). The other bin (red box) represents an hypotheses brought in by a false positive. Had the three hypotheses voted for their own centroid instead of the base, the associated bin would have accounted a distorted number of votes taking into account as inliers the relatively rotated matches, coming from the false positive hypotheses.

4.3.2 Local Hypotheses Estimation Module

The second module of *SkiMap++* depends on the *Camera Pose* from which it derives an *Active Sphere*. In particular, the goal is to compute a reference frame ${}^{M}\mathbf{T}_S$ (**M**ap to **S**phere) relative to the camera frame ${}^{M}\mathbf{T}_C$ (**M**ap to **C**amera) falling into its frustum:

$$ {}^{M}\mathbf{T}_S = {}^{M}\mathbf{T}_C \cdot {}^{C}\mathbf{T}_{t_z} \tag{4.8} $$

where ${}^{C}\mathbf{T}_{t_z}$ is a translation along the z-axis. Such translation could be fixed (for example in our experiments to $1m$) or could be computed by analysing scene conditions (*e.g.* by computing the nearest point in depth image). This global reference frame ${}^{M}\mathbf{T}_S$ is used to perform a radius search on the *Labels Map*, allowing the system to build a *per-frame Local Space* on which to perform Hypotheses Estimation, rather than attempt this task on the whole map, thus ensuring bounded time complexity for subsequent stages of the process. The outcome of a radius search on the Labels Map is a *set* of Voxels of type *MultiLabelVoxels*, described in Equation 4.6. SkiMap allows, during a search, to enrich *User Data* stored in Voxels with their coordinates, by obtaining a subset:

$$H = \bigcup_{C_i \subset C} H_i = \{(c_i, \mathbf{p}) : c_i \in C_i\} \tag{4.9}$$

where H is the set of pairs $(c, \mathbf{p})$ containing a predicted class c and a point $\mathbf{p}$ representing the center of the associated voxel. As it can be seen from Equation 4.9, the set H can be split into many similar subsets, one for each object, with C_i containing only the predicted classes from the *i-th* object. For each pair $(c, \mathbf{p})$, we can apply Equation 4.4 to retrieve:

$$o(v(c)) \rightarrow (o_i, \mathbf{v}_j) \tag{4.10}$$

generating the new pair $(o_i, \mathbf{v}_j)$, where $\mathbf{v}_j \in \mathbb{R}^3$ represents the center of the j-th voxel of the i-th object. By iterating this approach, we estimate a set of 3D matches $\{(c, \mathbf{p}), (o_i, \mathbf{v}_j)\} \rightarrow \{(\mathbf{p}, \mathbf{v}_j)\}$ suited to 6-DoF pose estimation of the target object within the Local Space. In simple terms on the current Labels map, reasoning about a single object, for each voxel we have the coordinates of the voxel's center and by means of the associated *class* we can retrieve the coordinates of the center of the voxel, in the original model, as close as to the scene voxel (*close* in terms of the 2D descriptor).

We employ a fast and robust technique to estimate 6-DoF pose of the objects under occlusion and clutter. This problem was addressed also by Tombari and Di Stefano (2012) with promising results in comparison with other competitors involved in 3D free-form object recognition problem. The most important feature in this approach is to reduce the Hough-problem complexity from 6 to 3 dimensions, discarding the rotational part of pose estimation implicitly covered by the voting process. Unfortunately the scheme of Tombari and Di Stefano (2012) requires the estimation of a Local Reference Frame per feature, since each feature should cast a vote independently from others. The estimate of a Local RF is a computationally onerous procedure, not suited to real-time application, but at the same time the approach has been proven valid. We thus built a new, hybrid, approach drawing inspiration also from the work of Den Hollander and Hanjalic (2007). This approach relies on the 3-Point Ransac/Hough-voting scheme described as follows:

3-Point Ransac/Hough-voting scheme: given $M_i = \{(\mathbf{p}, \mathbf{v}_j)\}$, the set of matches involving object i in the current Local Space, we pick m random subsets $R_{M_i} \subset M_i$ with $|R_{M_i}| = 3$. Each set of such 3 points $\mathbf{p} \in R_{M_i}$ outlines a Reference Frame through which we can project an instance of object i in the Local Space (i.e. in the **M**ap reference frame) ${}^{M}\mathbf{T}_{obj}$ and cast a vote for its centroid (or any rigid point attached to it, for example its base) in the associated bin of a *3D Hough Space* built right on the

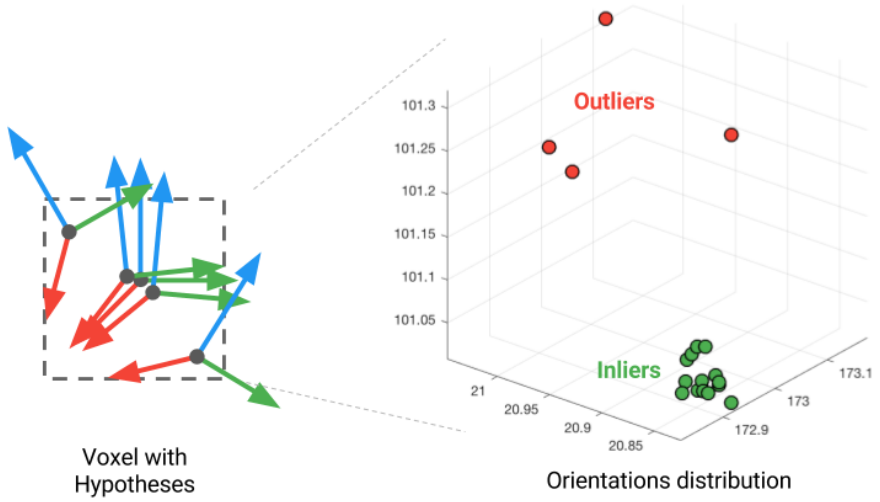


Fig. 4.8 On the left, a sample of a classical *Hypotheses Voxel* containing many computed Reference Frames. On the right, the distribution of their orientations represented with *axix angle* notation in a $SO(3)$ group. With high probability inliers will be grouped together in a cluster, the centroid of which can be inferred as the best candidate for the final resulting orientation of the instance.

Active Sphere. To increase the performance of this voting scheme, differently from the approach of Tombari and Di Stefano (2012), we do not vote for the centroid of the projection of the object in the scene, but for its base, thus ensuring that every false positive with inferred centroid near the real object pose will score far from the correct bin. In Figure 4.7 the voting scheme procedure is depicted graphically.

As mentioned above, the final aim of the *Local Hypotheses Estimation* component is to fuse hypotheses in a new map, so as to store spatial information about these guesses. The base unit of this new SkiMap structure is a *HypothesesVoxel* consisting of a simple set $V = \{(\mathbf{t}_h, \mathbf{R}_h, o_i)\}$ of tuples where $\mathbf{t}_h$ represents the position of the hypothesis in map reference frame, $\mathbf{R}_h$ its rotation in the same space and o_i the identifier of the object being considered. More precisely, a 6-DoF Hypothesis should be stored at least in a $\mathbb{R}^6$ space, but following the approach of Tombari and Di Stefano (2012) we store them in a $\mathbb{R}^3$ space via their translational component, preserving each orientation $\mathbf{R}_h$ as is, to deploy them in a further refinement step.

4.3.3 Global Instance Retrieval Module

The last module of *SkiMap++* is mainly dedicated to refining the hypotheses coming out from the previous stage. Starting from the ${}^M\mathbf{T}_S = ({}^M\mathbf{R}_S, {}^M\mathbf{p}_S)$ *Active Sphere* reference

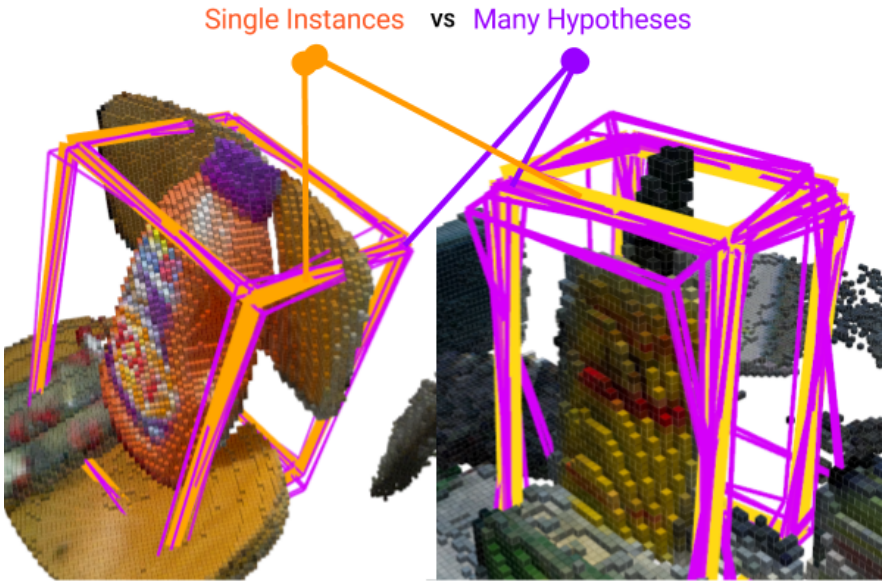


Fig. 4.9 This figure portrays a frame taken from a real-time scan of one scene of the dataset. RGB Reconstruction is carried out with two different resolutions: $0.005m$ for left object and $0.01m$ for right object. *Hypotheses* (purple boxes) are contained with *HypothesesVoxels* of $0.03m$ instead. Finally the *Instances* (orange boxes) are grouped in $0.05m$ *InstancesVoxels*.

frame, we can perform again a neighborhood search on *Hypotheses Map* based on a target object o_i:

$$r({}^{M}\mathbf{p}_S, o_i) = \{(\mathbf{t}_{h_k}, \mathbf{R}_{h_k}, o_i), k \in [0, m)\} \tag{4.11}$$

where the result of the search operation $r(\cdot)$ is a set of m hypotheses in the neighborhood of the point ${}^{M}\mathbf{p}_S$. Since this resulting subset may contain some outliers, we cannot simply average the outcome in a $\mathbb{R}^6$ space. Therefore, once again, we can adopt a statistical approach analyzing the orientation distribution of these hypotheses in the $SO(3)$ group of the *axis angle* representation of them (Figure 4.8 clarifies this issue). To infer the final *Instance* $(\mathbf{t}_I, \mathbf{R}_I, o_i)$ of the object o_i in the scene we can choose as orientation $\mathbf{R}_I$ the centroid of the largest cluster in the orientation space and by simply averaging positions $\mathbf{t}_I = \frac{1}{m} \sum_{k=1}^{m} \mathbf{t}_{h_k}$ of the corresponding hypotheses (dubbed *inliers*).

Finally, also the found *Instances* will be integrated into a further SkiMap structure in units called *InstancesVoxels*, a simple extension of *HypothesesVoxels*. Figure 4.9 illustrates graphically, by means of a real frame captured from *SkiMap++* execution, the difference between *Hypotheses* and *Instances*.

4.4 Experimental results

In this section we first describe the dataset used in our experiments. Next we show some quantitative results obtained from the aforementioned dataset while varying some key parameters involved in *SkiMap++* pipeline. Finally we present a qualitative evaluation of the algorithm.

4.4.1 SK17: a new dataset for multi-view Object Recognition

To test the whole *SkiMap++* framework we have tried to find suitable public dataset without success. The same type of search performed by Li et al. (2016) ended up in the same conclusion. The main reasons are the absence of 6-DoF ground truth of objects or the low frame density. Also the proposed dataset in the latter study is not still fully available to be used with our approach (missing RGB-D frames for objects). For this reason we developed a brand new dataset comprising 12 Objects and 5 Medium/Large Scale Scenes. The Camera, an Asus Xtion, is tracked by the Vicon Motion Capture System *(Vicon Motion Systems, LA, USA)* which provides a precision of the order of magnitude of *1cm*. The 6-DoF ground truth of the objects in each scene was manually labelled in a global reference frame centered on the floor of each scene (Precision of ground truth has the same resolution of the smallest RGB map attainable by the SkiMap approach (*i.e.* about $0.005m$).

Table 4.1 Here a complete list of all objects in our dataset. The table shows a comparison between two best sets of parameters, tuned during our tests of *Skimap++* running over the *SK17* dataset, varying the 2D Descriptor. The percentages represent the recall after a complete round within every scene.

	Boxgreen	Boxmouse	Boxred	Chamo	Cupgreen	Cupred	Detergent	Glue	Korn	Multimeter	Robot	Talc	Fps
SIFT	**33%**	**100%**	35%	**100%**	**75%**	**75%**	**100%**	**100%**	**100%**	**100%**	**78%**	**75%**	6
SURF64	0%	50%	**87%**	**100%**	0%	0%	**100%**	**100%**	**100%**	**100%**	75%	**75%**	**20**

4.4.2 Quantitative results

In this subsection we examine the performance of *SkiMap++* in terms of precision/recall computed on all the 5 scenes of the dataset by taking into account all the objects instances present within the environment and by counting a *True Positive* when we do find the instance with a maximum translational error under $0.03m$ and maximum rotational error of 10° in comparison with their ground truth. Figure 4.10 shows the accuracy of the system by varying some key parameter. As observed from the first plot, by increasing the number of *classes* trained per-object (*i.e.* the number of Descriptors

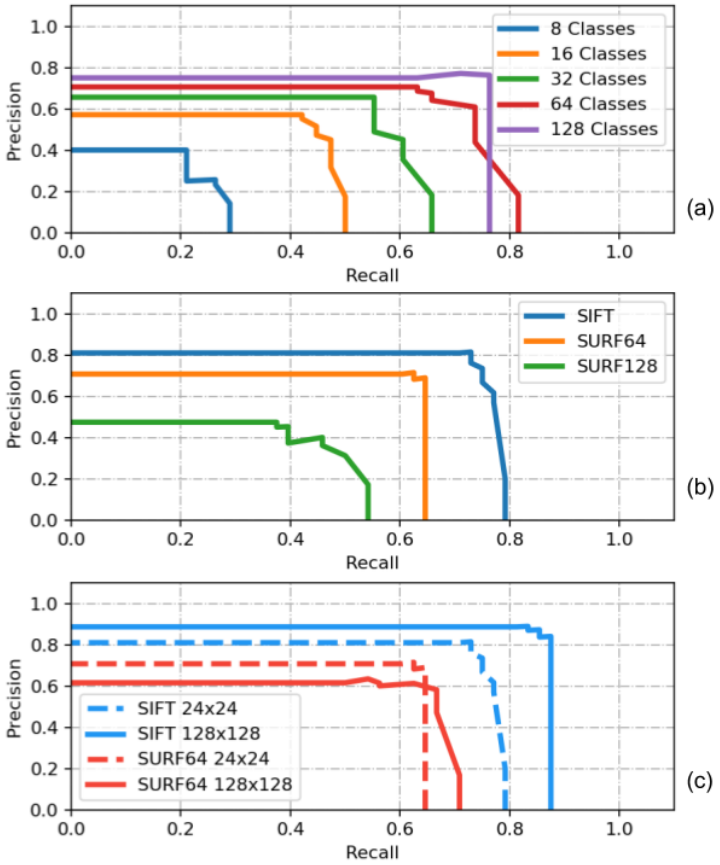


Fig. 4.10 These plots show precision/recall while varying some key parameters like: (a) Number of trained classes per object; (b) differences between 2D Feature detectors; (c) size of the random forest while using the same descriptor. The overall result shown here is over 80.0/80.0 precision/recall index.

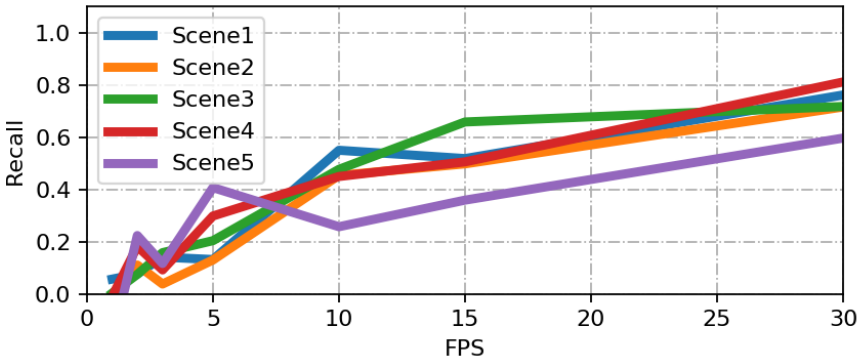


Fig. 4.11 This plot shows the importance of Multi-View based Object Recognition. Decreasing the number of vantage points(*i.e.* decreasing fps and so the number of frames) the accuracy falls down to zero.

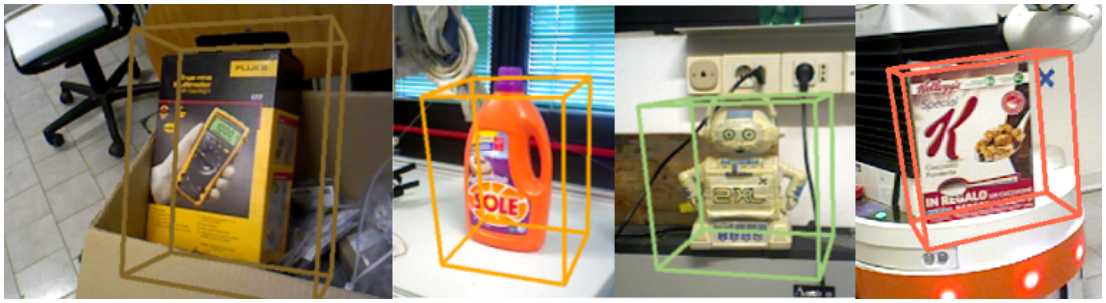


Fig. 4.12 Real-time Augmented Reality enabled by stable 6-DOF pose.

Voxels taken into account to grow decision trees) we can achieve higher performance, but what also stands out is that after a certain number of classes the accuracy starts to decrease. Intuitively this is due to the under-fitting problem in the Random Forest that would need to grow in size to learn more labels to be classified. The second plot shows a simple comparison between 2D Features that reflects their intrinsic characteristics. The last plot addresses the inverse problem seen in the first graph: the curves show that keeping the detector fixed and increasing the size of the Random Forest (*e.g.* 24x24 means a forest of 24 trees with depth 24) we can slightly increase accuracy at the expense of prediction time. Overall, Figure 4.10 shows a precision/recall index over 80/80 in the best position within parameters space. Moreover Figure 4.11 highlights the key feature of this Multi-View Object Recognition approach: the higher is the density of frames (and therefore the number of vantage points), the larger is the accuracy of the system. The latter result is justified by a series of important factors, such as: an high frame rate helps to merge an higher number of evidences; many vantage points can be lost if the frame rate is low when camera movements are fast and without allowing for the fact that the multi-view approach is the only solution for occlusion. Finally, the Table 4.1 lists an overview on accuracy based on single objects among all scenes; it should be noted that the score of the object *Boxgreen* is very low because it is the only object with an overall dimension comparable to Voxels Resolutions (about *5cm* on the longest side), which renders pose estimation very challenging.

4.4.3 Qualitative results

We provide here also some qualitative results to show that *SkiMap++* is suited to real settings. Figure 4.1 shows a large scale environment fully reconstructed by means of a mobile robot by relying on its odometry system to track the camera pose. In this environment we placed some objects, taken from our dataset, and the robot successfully found them while mapping the workspace. Other qualitative samples are shown in .Figure 4.13 and Figure 4.12: the latter illustrates also the interesting Augmented Reality

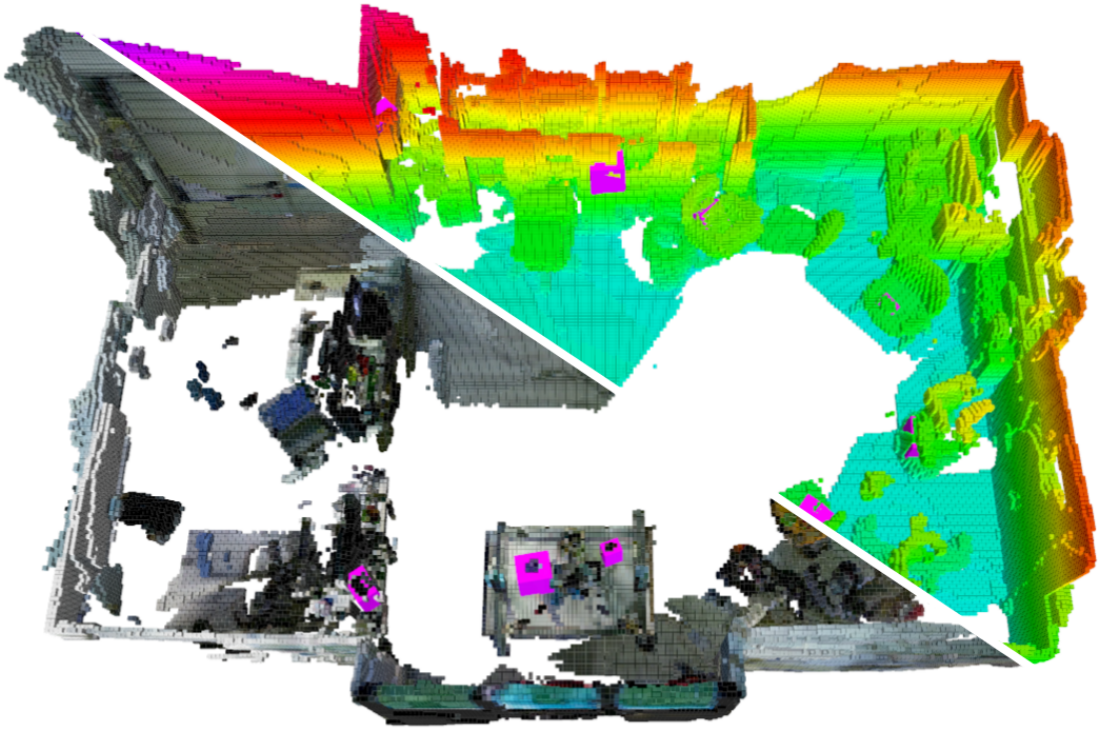


Fig. 4.13 Large scale map (dataset scene) fully reconstructed by *SkiMap++* with Object Instances identified by purple bounding boxes.

attainable thanks to the high stability of the object hypotheses in the global reference frame yielded by *SkiMap++*. This Video [1] presents multiple run-time executions of *SkiMap++* in a real scenario with both a Mobile Robot and a Motion Tracking System to estimate camera pose.

[1] https://www.youtube.com/watch?v=ki_Lbl4lEIY

Part II

Machine Teaching made easy

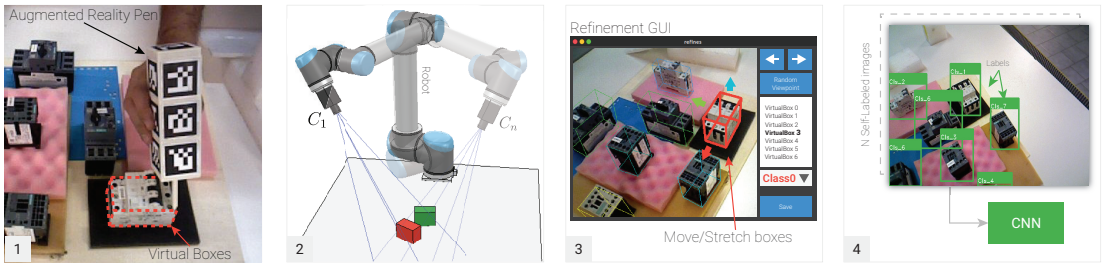


Fig. 5.1 The *ROARS* labeling pipeline: 1) Draw virtual boxes around the target objects. 2) Perform a scan with robot. 3) Refine the virtual boxes looking at few frames. 4) Generate an arbitrary number of self-labeled images.

Chapter 5

Using robot to train Deep Networks: ROARS

5.1 Train a Deep Network for Object Detection

As mentioned in Chapter 2 and Chapter 4, multiple tasks in robotics require some sort of understanding of the environment surrounding the robots, for example, a picking task would at least require the automatic identification of category or instances of known objects. In this chapter we will focus over this field, that in computer vision is commonly referred as *object detection* (or even *instance detection*), and, as many others, has seen a great development in recent years mainly due to the introduction of deep learning methods like Redmon and Farhadi (2017) or Huang et al. (2017) that offers incredibly good real time performance, but requires a long training phase using huge dataset of images annotated with bounding box. However, the manual annotation of images is long, tedious and prone to errors often producing too small dataset or noisy annotations, especially when performed by non-professionals. Even if big dataset containing various categories of objects are publicly available in literature (Calli et al., 2017; Lin et al., 2014; Rennie et al., 2016), they usually concern generic classes of objects (e.g. person,

car, cat etc.), but for many applications, especially industrial ones, the recognition of a specific set of objects, which often change over time, is required. To deploy effectively deep learning solution in such scenario the user would need to retrain or adapt the system each time the requirements change. In practice, each new object will require thousand of new annotated images, which translate in tens of human work hours (the same things happens due to any change in light conditions). At large scale, manual annotations became an impractical and long job. Instead, we propose a user-friendly tool that let users automatically collect huge datasets of images annotated with the bounding box of the visible objects. To achieve this task we deploy augmented reality techniques together with the high repeatability of an industrial robot (similar scenario as in Chapter 2) that is called *ROARS* which stands for *RObot for Augmented-Reality Self-labeling*. With our tools, the time needed to create a training dataset drops to the mere acquisition and processing time, more importantly we can be assured that all the annotation are error free, leading to a small gain in performance of the final detector compared to manual annotation on the same dataset, as we will show in Section 5.3.

An alternative approach proposed recently is the use of synthetic images either rendered, like Mayer et al. (2016),Ros et al. (2016), Movshovitz-Attias et al. (2016) and Carlucci et al. (2016), or grabbed from realistic videogames as in Richter et al. (2016) or Johnson-Roberson et al. (2016). These techniques allow the creation of potentially infinite perfectly annotated images with zero or minimal human effort, however, the variability is inherently limited by the rendering setup and can not represent all the real world environments. Moreover, this approach may still need lots of hours of specialized human work for the creation of the synthetic object and scenes along with a lot of computational power to achieve photo realistic rendering. Even worse, as reported by Movshovitz-Attias et al. (2016) and Carlucci et al. (2016) using only synthetic images alone does not grant good performance on real data due to the inherent differences between ideal and real images; usually, an additional fine-tuning for domain adaptation with few real data is still needed to maximize the performance. Alternatively, Georgakis et al. (2017) proposes a hybrid approach where an object detection system is trained on synthetic rendered 3D objects superimposed on real scenes. Even in this case, the blend between synthetic and real images is not perfect and an additional fine-tuning on the real dataset is still needed. The huge gap between synthetic and real image is also testified by Shrivastava et al. (2017) were the authors try to learn a domain specific generative network to transform perfect synthetic images and make them more realistic. In contrast, our proposal directly semi-automatically annotates real images taken in the deployment scenario, thereby avoiding any gap between the training and test conditions.

More specifically for robotic tasks, Zeng et al. (2016) proposes a semi automatic techniques for acquiring a training dataset for object segmentation and Mitash et al.

(2017) extends the idea for object detection including in the creation process some physical simulation to create realistic object arrangement. However, both those solution require depth information from the sensor and can only work for objects paired with reliable 3D models. This is a significant limitation for tasks where we are interested in recognizing categories of products, like oranges, apples, etc., for which a 3D model can not be defined since each instance is different from others. Our proposal, instead, does not need any clue on the 3D shape of the object and does not even need depth information neither at training nor testing time.

We developed a ROS package[1] implementing all the tools needed to reproduce the *ROARS* labeling pipeline for whoever has a robot with an hand-mounted RGB camera. The ROS package also includes two wrappers for the popular object detector systems Yolo (Redmon and Farhadi, 2017) and SSD (Huang et al., 2017), so to allow ROS users to easily include them in their applications.

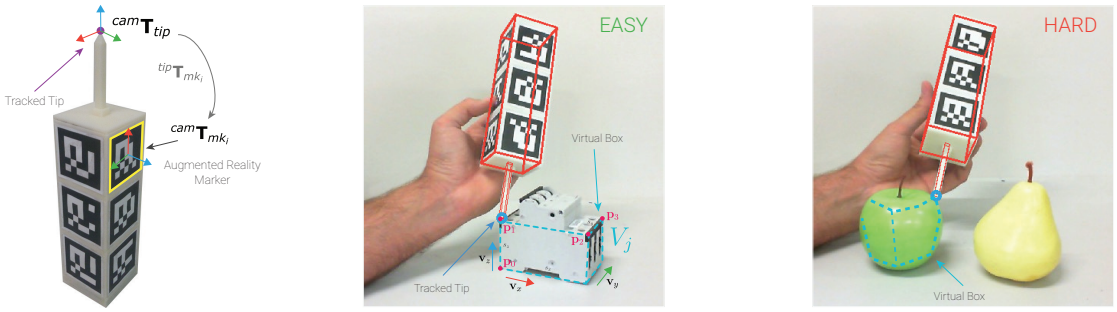
5.2 Method description

Our approach can be briefly described as follow: if we have a set of images for each of which we have the exact 6-DoF pose of the camera tracked by the robot and, in addition, we have a set of objects in the environment with a known pose w.r.t. the camera, we can project in each image the simplied representation of these items through augmented reality, from where we can extrapolate several labels (e.g. a 2D box surrounding the target object) useful to train an object detector. The *ROARS* labeling pipeline can be summarized in these few steps (also depicted in Figure 5.1):

0. *Arrange the objects randomly in a scene;*
1. Draw virtual boxes around the target objects;
2. Scan the environment with the robot;
3. Refine virtual boxes analyzing the outcome of the scan;
4. Generate a training dataset.

In this chapter we will describe in detail all these steps: Section 5.2.2 describes the physical tool used to interact directly with the environment to draw virtual boxes through augmented reality around the target objects. This step is not mandatory for the rest of the pipeline but it is useful to create easily an initial guess; Section 5.2.1 designs formally a generic input dataset in such a way it can be reproduced easily by anyone;

[1] https://github.com/m4nh/ars



(a) ARP **(b)** Drawing on geometrical object **(c)** Drawing on rounded object

Fig. 5.2 (a) The *Augmented Reality Pen* (ARP) used to draw virtual boxes. The pen is composed of several Augmented Reality Markers with a known pose ${}^{tip}\mathbf{T}_{mk_i}$ w.r.t. the tip. (b) Tracking the tip we can easily draw a virtual box around a target object by touching its edges. (c) Conversely, its not simple to draw a virtual box around a rounded object not having reference points.

Section 5.2.3 presents the technique to refine (or even create from scratch) the virtual boxes by only exploiting the outcome of the environment scan; in Section 5.2.4 we will describe the straightforward procedure to generate labels starting from virtual boxes. Further in Section 5.2 we briefly introduce some notations used in this work.

Common Notations: We will use the notation m_i to indicate a generic image and the symbol $b = \{x_b, y_b, w_b, h_b, c_b\}$ to indicate a square region (box) therein, where x_b, y_b are the coordinates of the center and w_b, h_b the *width* and *height* respectively; optionally the box b could have the parameter c_b indicating the *class* (or the *label*) of that box. This attribute is useful in our context because we typically deal with CNNs able to predict not only a region around our target obejcts, but also their class of belonging.

5.2.1 The input dataset

The mandatory step of the *ROARS* labeling pipeline is to have a dataset described formally here:

$$\begin{aligned} \mathbb{D} &= \mathbb{F} \cup \mathbb{I} \\ \mathbb{F} &= \{F_i = \{{}^0C_i, m_i\}, i \in [0, \ldots, n]\} \\ \mathbb{I} &= \{V_j = \{{}^0\mathbf{T}_j, s_j, c_j\}, j \in [0, \ldots, k]\} \end{aligned} \tag{5.1}$$

the dataset $\mathbb{D}$ is composed by two independent subsets $\mathbb{F}$ and $\mathbb{I}$. $\mathbb{F}$ is the set of n pairs $F_i = \{{}^0C_i, m_i\}$ (we will call it *frame*) where m_i is the generic i-th image which

corresponds to a camera matrix ${}^0C_i = \hat{A} \cdot {}^0\mathbf{T}_{cam_i}$ that is a composition of *intrinsics* ($\hat{A} \in \mathbb{R}^{4\times4}$) and *extrinsics* (${}^0\mathbf{T}_{cam_i} \in \mathbb{R}^{4\times4}$):

$$\hat{A} = \begin{bmatrix} f_x & 0 & c_x & 0 \\ 0 & f_y & c_y & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, {}^0\mathbf{T}_{cam_i} = \begin{bmatrix} {}^0\mathbf{R}_{cam_i} & {}^0\mathbf{t}_{cam_i} \\ 0 & 1 \end{bmatrix} \tag{5.2}$$

where $\mathbf{R}_{cam_i} \in \mathbb{R}^{3\times3}$,${}^0\mathbf{t}_{cam_i} \in \mathbb{R}^3$ are the orientation and the position of the i-th viewpoint. The $\mathbb{I}$ set, instead, is the collection of k object instances on the scene; each instance V_j (*e.g.* a 3D **V**irtual box like in Figure 5.2(b)) is represented by the tuple $\{{}^0\mathbf{T}_j, \mathbf{s}_j, c_j\}$ where ${}^0\mathbf{T}_j$ is its 6-DoF pose, $\mathbf{s}_j \in \mathbb{R}^3$ is the size and $c_j \in \mathbb{N}^0$ is the *class* of the object. Thus, the main step in the *ROARS* pipeline is to create this dataset; more specifically we can build separately $\mathbb{F}$ and $\mathbb{I}$. The straightforward methods to build the set of *frames* $\mathbb{F}$ have been already addressed in the Chapter 2. To build instead the instance set $\mathbb{I}$, we will describe in Section 5.2.2 an augmented reality tool able to interact directly with the environment; as an alternative, in Section 5.2.3, we will see how to compute an initial guess of our instances without interacting with the real scene but looking only at a pair among the m_i images presents in $\mathbb{F}$.

5.2.2 The augmented reality pen

The second part of the input dataset is $\mathbb{I}$ which collects all the virtual boxes $V_{0,\dots,k}$ arranged around our target objects. As depicted in Figure 5.2, we developed a physical tool wallpapered with Augmented Reality Markers, in short: Markers (the same as Chapter 2), that we dubbed *ARP* (Augmented Reality Pen), useful to perform an *online* labeling procedure by interacting directly with the environment. Thus, we design a 3D printed artifact – resembling a pen – placing several Markers on it in a known pose ${}^{tip}\mathbf{T}_{mk_i}$ w.r.t. the tip of this tool (the placement is CAD-driven). Given that the Marker Detector (also available in the OpenCV library (Bradski and Kaehler, 2008)) estimates the pose of each of the markers w.r.t. the camera ${}^{cam}\mathbf{T}_{mk_i}$, we can compute easily the position of the tip w.r.t. the camera:

Fig. 5.3 In this picture are shown four random *frames* captured after the robot scan: the first row is referred to the Industrial and the displayed virtual boxes are drawn with ARP; the second one is related to the Fruits with annotations created offline with the technique shown in Figure 5.4). In the first row is clear how some virtual box is not perfect aligned with the real object.

$$^{cam}\mathbf{T}_{tip} = ^{cam}\mathbf{T}_{mk_i}(^{tip}\mathbf{T}_{mk_i})^{-1} = \begin{bmatrix} ^{cam}\mathbf{R}_{tip} & ^{cam}\mathbf{t}_{tip} \\ 0 & 1 \end{bmatrix} \tag{5.3}$$

as shown, for each marker we can compute an hypothesis about the pose of the tip $^{cam}\mathbf{t}_{tip}$; by averaging the hypotheses of the all visible markers we can obtain a more accurate estimation. A detailed explanation of this approach is described by by Jiawei *et.al.* Jiawei et al. (2010) who developed a similar tool as a Human-Computer interface.

Thus, given the opportunity to estimate the pose of any point in the scene in front of the camera, by means of the ARP, as shown in Figure 5.2(b), we can easily draw a virtual box V_j around a geometrical object by touching all its edges (or a subset of them). More generally with the ARP we can collect a set of 3D points $\hat{P}_{V_j} = \{^{cam}\mathbf{t}_{tip_e} = \mathbf{p}_e, e \in [0,...,n]\}$, corresponding to the frame-by-frame position of the tip while touching several edges, from which we can build the corresponding virtual object $V_j = \{^0\mathbf{T}_j, s_j, c_j\} = \sigma(\hat{P}_{V_j})$ by means of a generic function σ of this set of points. Notwithstanding many function σ can be designed to this purpose, we propose a simple one through which we can compute V_j starting from only four points: with

reference to the Figure 5.2(b) if we collect four points p_0, p_1, p_2, p_3 we can build the tuple $\{{}^0\mathbf{T}_j, s_j, c_j\}$ in this way:

$$
\begin{aligned}
{}^{cam}\mathbf{T}_j &= \begin{bmatrix} \mathbf{v}_x & \mathbf{v}_z \times \mathbf{v}_x & \mathbf{v}_z & \mathbf{p_0} \\ 0 & 0 & 0 & 1 \end{bmatrix} \\
\mathbf{v}_z &= \frac{\mathbf{p}_1 - \mathbf{p}_0}{\|\mathbf{p}_1 - \mathbf{p}_0\|}, \mathbf{v}_x = \frac{\mathbf{p}_2 - \mathbf{p}_1}{\|\mathbf{p}_2 - \mathbf{p}_1\|} \\
s_j = \begin{bmatrix} s_x & s_y & s_z \end{bmatrix} &= \begin{bmatrix} \|\mathbf{p}_2 - \mathbf{p}_1\| & \|\mathbf{p}_3 - \mathbf{p}_2\| & \|\mathbf{p}_1 - \mathbf{p}_0\| \end{bmatrix}
\end{aligned}
\tag{5.4}
$$

whereas, for the object's class c_j is relative to the object type and is not related to geometric data. It should be noted that in Equation 5.4 we have built the RF of the virtual object ${}^{cam}\mathbf{T}_j$ referred to camera coordinate system, but we can easily transform it in the robot RF by knowing the current camera pose ${}^0\mathbf{T}_j = {}^0\mathbf{T}_{cam} \cdot {}^{cam}\mathbf{T}_j$. With this simple algorithm we can collect as many virtual boxes V_j as the target objects within the scene, provided that they are reachable from the user. Even though this approach is quite straightforward with geometrical objects, as depicted in Figure 5.2(c), it is hardly applicable with rounded objects. To this end, in the next section we will propose also an offline procedure to sketch our virtual boxes V_j only by labeling a pair of *frames* produced during the robot scan.

5.2.3 Objects Pose refinement

Once we have performed the scan of the environment by means of the robot, we have collected the set of frames $\mathbb{F}$ and, if we have exploited also the ARP tool, also the set of instances $\mathbb{I}$ is available. Therefore, as shown in Figure 5.3, we can display – for debugging purposes – a random *frame* F_i of our scene reprojecting into the image the visible Virtual Boxes. The reprojection technique is straightforward: for each virtual box $V_j = \{{}^0\mathbf{T}_j, s_j, c_j\}$ we can build the set of 3D points in the matrix form ${}^0P_{V_j} \in \mathbb{R}^{4\times 8}$ (each column of this matrix corresponds to a single 3D point in homogeneous coordinates) collecting the eight edges of the 3D box. The same point set can be computed in the camera reference frame ${}^{cam}P_{V_j} = {}^{cam_i}\mathbf{T}_0 \cdot {}^0P_{V_j}$ where ${}^{cam_i}\mathbf{T}_0$ is the inverse of the camera pose in F_i. The corresponding set of 2D points H_{V_j} can be computed using the perspective transformation:

$$
\begin{bmatrix} \lambda H_{V_j} & 1 \end{bmatrix}^\intercal = \hat{A} \cdot {}^{cam}P_{V_j} \tag{5.5}
$$

where λ is the scale factor. Obviously, this reasoning can also be applied if our virtual object is composed by a different number of points, without compromising the rest of the labeling pipeline (*e.g.* instead of Virtual Boxes one can use Virtual Squares made by only 4 points if we are dealing with planar objects).

Again in Figure 5.3 we can see many examples of 2D Virtual Boxes H_{V_j} (We will use H_{V_j} to indicate the 2D reprojection of a virtual box V_j) drawn over 4 *frames* picked randomly from our two datasets (it is clear now that we can produce this 2D representation for every *frame* in our dataset without any additional information). Within the above image, we can notice (top-right image) how the H_{V_j} produced with the ARP tool is not perfect due to several factors (*e.g.* caused by user hand-shake during labeling). Thus at this stage we can – optionally – correct the pose of the corresponding virtual box V_j: we developed a simple GUI, shown in Figure 5.1(3), to regulate these imperfections. Despite the developed graphical interface, we can describe these adjustments formally saying that if we consider a virtual box $V_j = \{{}^0\mathbf{T}_j, s_j, c_j\}$ and we modify its pose ${}^0\mathbf{T}_{j^{(2)}} = {}^0\mathbf{T}_j {}^j\mathbf{T}_\varepsilon$ by a small roto-translation ${}^j\mathbf{T}_\varepsilon$ (or even changing its size $s_j^{(2)} = s_j + s_\varepsilon$) we can immediately see what happens in all the *frames* of the dataset. Anyway, we have empirically found that a perfect adjustment is possible looking simultaneously to four *frames* only (this test is reflected in our GUI with a dedicated tool). Needless to say that it is not possible to perform this adjustment looking only at a single *frame* because with a single view point we do not have any 3D information of the environment, so we could be dealing with perspective illusions.

However, as mentioned in the previous section, what happens if we cannot use the ARP tool (For example we want to apply the *ROARS* pipeline to a dataset created earlier)? Figure 5.4 reports a simple algorithm, which can be applied on every set of frames similar to $\mathbb{F}$, to produce a set of instance $\mathbb{I}$: starting from a random pairs of frame F_1, F_2 we can draw two 2D boxes b_1, b_2 around the considered object (the red apple in the figure); given the two camera matrices ${}^0C_1, {}^0C_2$ we can compute two views frustum which intersect – likely – in the center of mass of the real object; we can easily compute the intersection of the two rays r_1, r_2, focusing in the center of each frustum, to compute a single 3D point where to position our new virtual box ${}^0\mathbf{T}_j$ (its rotational part cannot obviously be computed with this approach, but we can choose the canonical base $I \in \mathbb{R}^{3\times3}$ for it), and use the size of the far plane of one of the two frustums to compute a coarse size s_j. Choosing a custom class c_j we have built all the parts of our virtual object V_j, or at least its initial guess. Then, with the procedure described at the beginning of this chapter, we can refine its appearance to fit the real object at best. All the Fruits Dataset was built with this method with good results, as shown in Figure 5.3.

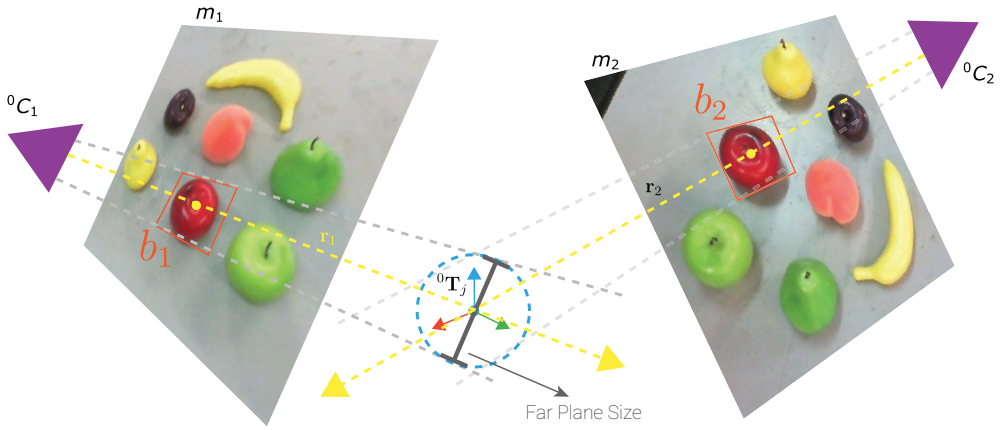


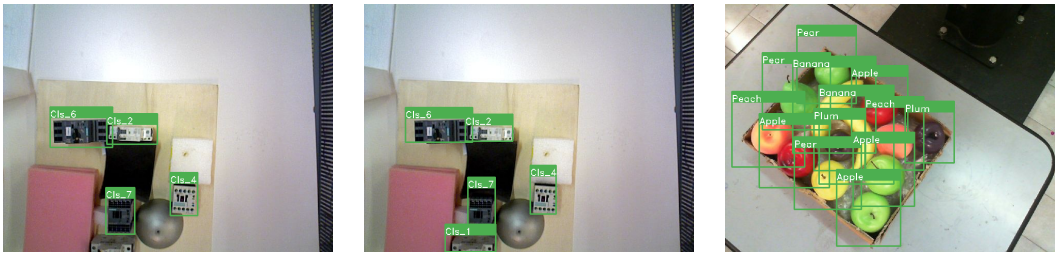
Fig. 5.4 Graphic representation of the algorithm to estimate the coarse 6-DoF pose of an object only by exploiting a pair of *frames* $\{{}^0C_1, m_1\}, \{{}^0C_2, m_2\}$ and a pair of boxes b_1, b_2 drawn over m_1 and m_2.

5.2.4 Generate Training Data

Once we have built the whole dataset $\mathbb{D} = \mathbb{F} \cup \mathbb{I}$ we can use it to build a real Training Dataset to train a CNN-based object detector, such as e.g. Yolo by Redmon and Farhadi (2017) and SSD by Huang et al. (2017). This kind of training dataset $\mathbb{T} = \tau(\mathbb{D}) = \{B_i = \{b_j\}, i \in [0, \ldots, n]\}$, namely is the collection of sets B_i of boxes $\{b_j\}$ related to each image m_i belonging to the frame F_i. So, in short, we need to associate to each image m_i the 2D boxes b_j surrounding our target objects; thus, since we have for each frame the 2D reprojection of our virtual boxes (*i.e.* H_{V_j}) in the current image, our function τ needs to transform each H_{V_j} in the corresponding box b_j. The function that we use is the simplest one, the *minimum bounding box* enclosing all the 2D points in H_{V_j}. See Figure 5.3 (bottom-right).

5.3 Experimental evaluation

To experimentally validate *ROARS* we mainly performed three sets of tests on two novel datasets that we are going to introduce in Section 5.3.1. The first tests, described in Section 5.3.2, present a careful analysis of our automatic labels against manual ones to prove that our method produces equivalent or better annotations. The second tests, reported in Section 5.3.3, deals with training two kinds of CNN based detector on our auto-labeled training sets as well as on an equivalent manually annotated one to check which option yields better final performance. Finally, we introduce in Section 5.3.4 a new and interesting way of analyzing datasets, made possible by the creation using



(a) Manual Industrial **(b)** Auto Industrial **(c)** Auto Fruits

Fig. 5.5 Samples from the dataset that we have used for our experiments, the green rectangles display the annotated bounding boxes, the white text over each box is the class mapped to that box. (a) Electromechanical components dataset manually annotated, (b) Electromechanical components dataset with our automatic labels, (c) Fruit dataset with our automatic annotations.

ROARS. A qualitative evaluation of the trained detectors is shown in Figure 5.8 as in this Video [2] , which also features a live demo of the *ROARS* labeling procedure.

5.3.1 Datasets and evaluation metrics

To validate our proposal we choose as test beds two different object detection tasks: one concerning the recognition of 7 types of electromechanical components (*Industrial*), the other with 5 classes of fruits (*Fruits*); some samples from the two datasets are depicted in Figure 5.5. The objects of interest in the first task display low intra and inter class variability: the appearance of each component remains always the same across all the images, but different components are remarkably similar. The second task, instead, deals with categories of objects showing high intra and inter class variability as each fruit is quite different from the other and even fruits belonging to the same class can have different appearances, e.g. in our acquisition we have two kinds of apple and pear showing different peel colors. The first dataset is composed of 9 acquisition ($\sim$ 36000 frames), the second by 8 shorter ones ($\sim$ 7500 frames). Both datasets were built by means of an industrial manipulator, a COMAU Smart Six, with a position repeatability lesser than 0.05*mm* (same setup as in Chapter 2).

We chose one sequence from the *Industrial* dataset and two from the *Fruits* dataset to be used as test sets for the trained object detectors, we will refer to them as Industrial_Test and Fruits_Test respectively. The other sequences are randomly sampled to create sets with increasing number of samples, each set is then splitted in 80% train and 20% validation. We will use the following notation to indicate one of such set: ⟨`dataset name`⟩_⟨`number of samples`⟩, *e.g.* Industrial_1000 identifies 1000 sample

[2] https://www.youtube.com/watch?v=tj_h_N1Bylo

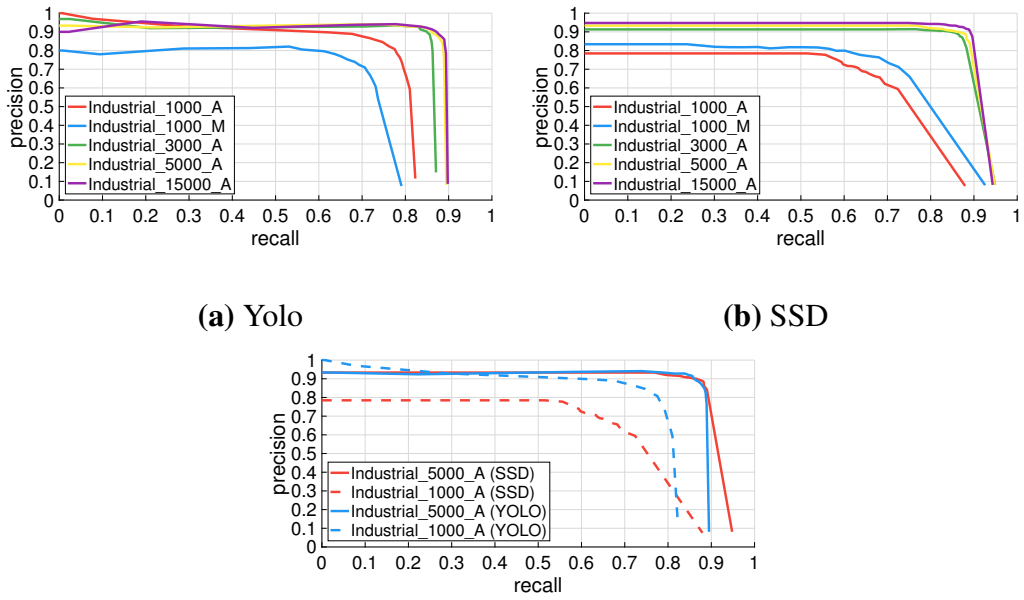
from the training sequences of Industrial that will be split into 800 samples for training and 200 for validation. All the dataset have been automatically annotated with *ROARS*, but for further tests we enriched Industrial_1000 with manual annotation as well. We will use a "_M" suffix for manually annotated dataset (e.g. Industrial_1000_M) and "_A" suffix for those annotated using *ROARS* (e.g. Industrial_1000_A). For comparison the manual annotation process of 1000 frames took us slightly more than 10 hours, while with *ROARS* in less than an hour we were able to annotate all the 9 sequence of the Industrial dataset ($\sim$ 35000 frames) with most of the time spent adjusting the coarse 6-DoF pose in the refine step, done just once for each sequence.

For the next tests we will use the standard object detection metrics defined for the PASCAL VOC challenge Everingham et al. (2010). Given a prediction b_j^p and the corresponding ground truth box b_i, we consider b_j correct if they have the same class and $IOU(b_j^p, b_i) > IOU_{th}$ with IOU intersection over union of the boxes and IOU_{th} a threshold parameters. Given the set of correct predictions we can measure: *Precision*,*Recall* and *average intersection over union for correct predictions (avgIOU)*. Usually a detector produces quite a lot of b_j^p each one associated with a certain confidence value $t_j^p \in [0,1]$, by thresholding the minimum confidence allowed we can tune the behaviour of the system. We represent the global performance with different confidence threshold using Precision/Recall curves and, synthetically, with the **mean average precision** (*mAP*), defined as the approximation of the area under the precision recall curve.

5.3.2 Annotation Study

Looking at the annotation time alone, *ROARS* would be preferable from a user perspective as long as the labels produced are at least equivalent to the manual ones. To verify this claim we have considered our manually defined boxes as the output of an ideal object detector and tested its performance on the dataset considering the automatically obtained annotations as ground truth.

Using $IOU_{th} = 0.3$ we obtain the following result: Precision= 98.49%, Recall= 95.02% and $avgIOU = 0.7035$. To understand why we did not obtain perfect scores we visually examined the difference between the two sets of annotations. We found out that the missing 1.5% Precision can be explained by class mistakes made by the human annotator during the labeling process, *ROARS*, instead, cannot wrongly label the class associated with a *b* as long as the refined initial 3D virtual box alignment is correctly labeled. The 5% missing recall is instead due to situations like that depicted in Figure 5.5(a-b) where the visible portion of an object (the bottom object *Cls_1*) is too small to allow the human annotator to understand what is looking at, *ROARS* once again does not suffer from this issue and can correctly generate useful annotations even in



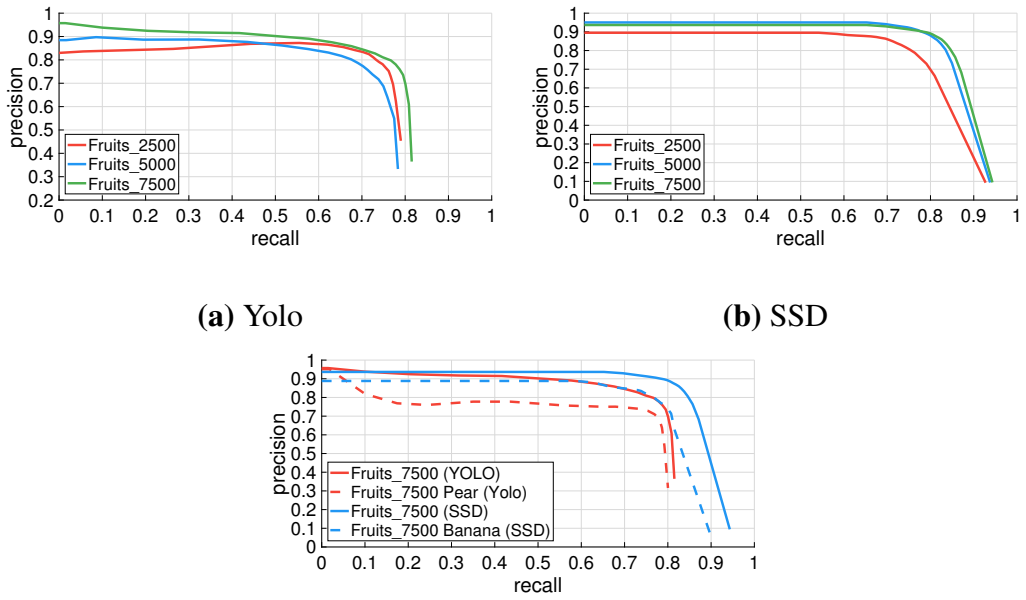
(a) Yolo **(b)** SSD

(c) Comparison

Fig. 5.6 Precision/Recall curves for the two type of detector trained on different subsets of the Industrial dataset. (a) and (b) report the results for Yolo and SSD respectively; (c) instead displays a comparison between them.

this challenging images. Finally, the relatively low *avgIOU* highlights a key difference between *ROARS* and a human annotator, the former always produces a b big enough to enclose the whole object as side effect of the reprojection of the virtual 3D box, while the latter usually encloses in b only the portion of object visible in the current image (See the Figure 5.3 top-right image where a virtual box is drawn also where the object is occluded). As a result, the manual and auto b does not always have the exact same shape, especially in cluttered environments, and this cause the relatively low *avgIOU*. An example of slightly different shapes between the two type of annotation can be observed in Figure 5.5 (a-b) on the box tagged "Cls_4" that in the manual case tightly enclose the top of the object, while in the auto case is slightly bigger than needed.

Nevertheless, as we will prove in the following paragraphs, the annotation automatically produced by *ROARS* can effectively be used to train and validate any machine learning based object detector obtaining performance comparable with a manually annotated dataset, or slightly better if we take into account the mistakes that may occur in the manual annotation process.



(a) Yolo **(b)** SSD

(c) Worst Class Comparison

Fig. 5.7 Precision/Recall curves for the two type of detector trained on different subsets of the Fruits dataset. (a) and (b) report the results for Yolo and SSD respectively; (c) instead displays a comparison between SSD and Yolo globally and considering the worst class specific precision/recall curve for each one: SSD experiences difficulties with *banana*, while YOLO with *pear*.

5.3.3 Object Detector Test

Once assessed that with *ROARS* we are able to obtain an automatically annotated dataset equivalent or better than the manual one, we want to test the performance of a state of the art object detector system when trained on this kind of data. We choose as detectors Yolo_v2 (Redmon and Farhadi, 2017) (shortened in Yolo) and ssd_inception_v2 (Huang et al., 2017) (shortened in SSD), using for both the original author's implementation and public pre-trained networks as starting points for the fine tuning on our datasets. The first tests were conducted for the Industrial task and we use as test bench 126 images randomly picked from Industrial_Test plus 20 photo of the same objects acquired in a different scene with a smartphone camera, all the images were carefully manually annotated. We will call Industrial_Test+ this dataset.

We know from the literature that usually machine learning systems work better with big training sets. However, we are interested in quantifying which is the real performance difference from using few hundreds to few thousands images for a relatively small number of classes as the one of our tasks, and with *ROARS* we can do that very easily. To actually measure the performance boost we define four different

	mAP		*avgIOU*	
Training set	Yolo	SSD	Yolo	SSD
Industrial_1000_M	0.589	0.619	0.7479	**0.795**
Industrial_1000_A	0.731	0.562	0.719	0.728
Industrial_3000_A	0.799	0.809	0.713	0.720
Industrial_5000_A	0.828	0.831	0.705	0.729
Industrial_15000_A	0.834	**0.851**	0.709	0.732

Table 5.1 Mean average precision (*mAP*) and average intersection over union (*avgIOU*) on the Industrial_Test+ for Yolo and SSD trained using 5 different training sets with increasing number of images. The best result for each of the two metrics are highlighted in bold.

sets with increasing number of images, respectively Industrial_1000, Industrial_3000, Industrial_5000 and Industrial_15000. The samples for the Industrial_1000 have both automatic and manual annotations, so we can test if the final detection performance change or not according to the label used, we are going to use the suffix _M (for manual) and _A (for auto) to refer to the two sub-types.

Given the five different training sets, we trained both detectors on them for 100000 step with `batch_size=24` using the hyperparameter recommended by the authors. The results are ten slightly different detectors that we tested on Industrial_Test+, we report in Figure 5.6 the precision and recall curves obtained and in Table 5.1 the *mAP* and *avgIOU*. Looking at the result we can see that for both detectors the performance increases, as expected, together with the size of the training set used, vouching for a method to ease and speed up the creation of training data. Looking at the results using Industrial_1000_A or Industrial_1000_M we can see that there is no clear difference between using manual annotations or the one obtained with *ROARS*: Yolo works distinctly better with the automatic annotation (+0.14 *mAP*), SSD instead slightly worse with the automatic (-0.057 *mAP*) but within the range of error that could be caused by random factors in the training process. Scaling up the training set the best results are obtained by SSD on Industrial_15000_A with 0.851 *mAP*; looking at the corresponding curve in Figure 5.6 we can see how the detector can maintain a precision above 95% for recall $\sim$ 95%. Looking at the *avgIOU* obtained by the detectors we can see how the best performing methods are, unsurprisingly, the two trained on the manually annotated

	mAP		***avgIOU***	
Training set	Yolo	SSD	Yolo	SSD
Fruits_2500_A	0.666	0.720	0.710	0.744
Fruits_5000_A	0.660	0.804	0.6818	0.749
Fruits_7500_A	0.722	**0.810**	0.734	**0.756**

Table 5.2 Mean average precision (*mAP*) and average intersection over union (*avgIOU*) on Fruits_Test for Yolo and SSD trained using 3 different training sets with increasing number of images. "A" suffix marks training sets with annotations produced by *ROARS*. The best result for each of the two metrics are highlighted in bold.

Industrial_1000_M. This behaviour highlighting once again the small difference in shape between the bounding box produced by *ROARS* and those made by a human annotator: an algorithm trained on human annotations learn to reproduce them, thus obtaining higher IOU when tested on the human annotated test set.

We repeat similar experiments on the more challenging Fruits dataset, we annotated all the eight sequences using *ROARS*, then we use six of them to create three different training and validation sets with increasing number of samples and sequences, respectively Fruits_2500 (2 sequences), Fruits_5000 (4 sequences) and Fruits_7500 (6 sequences). The remaining two are used as the test bench for the detector creating a huge test set of 2600 images (refered as Fruits_Test), unlike before we use automatic annotations both for training and tests sets. To create a more difficult task we used for training sequences where the fruits are always inside a box with various backgrounds, while one of the two test sequences shows fruits in a box but with a new background and the other fruits widely placed on a table, thus a completely unseen environment.

Given the six training sets, we trained both SSD and Yolo for 40000 steps with `batch_size=24` and keeping the hyperparameter fixed as in the previous training. We tested the six different resulting detectors on Fruits_Test and report in Figure 5.7 and Table 5.2 the results. Once again all the performance indexes are really good and increase proportionally with the size of the training sets used. Even in this case, the best absolute performance is obtained by SDD using the Fruits_7500 dataset, with a *mAP* as high as 0.81, compared to the best Yolo result with 0.72. In Figure 5.7(c) we report a comparison between the two best performing Yolo and SSD detectors showing the

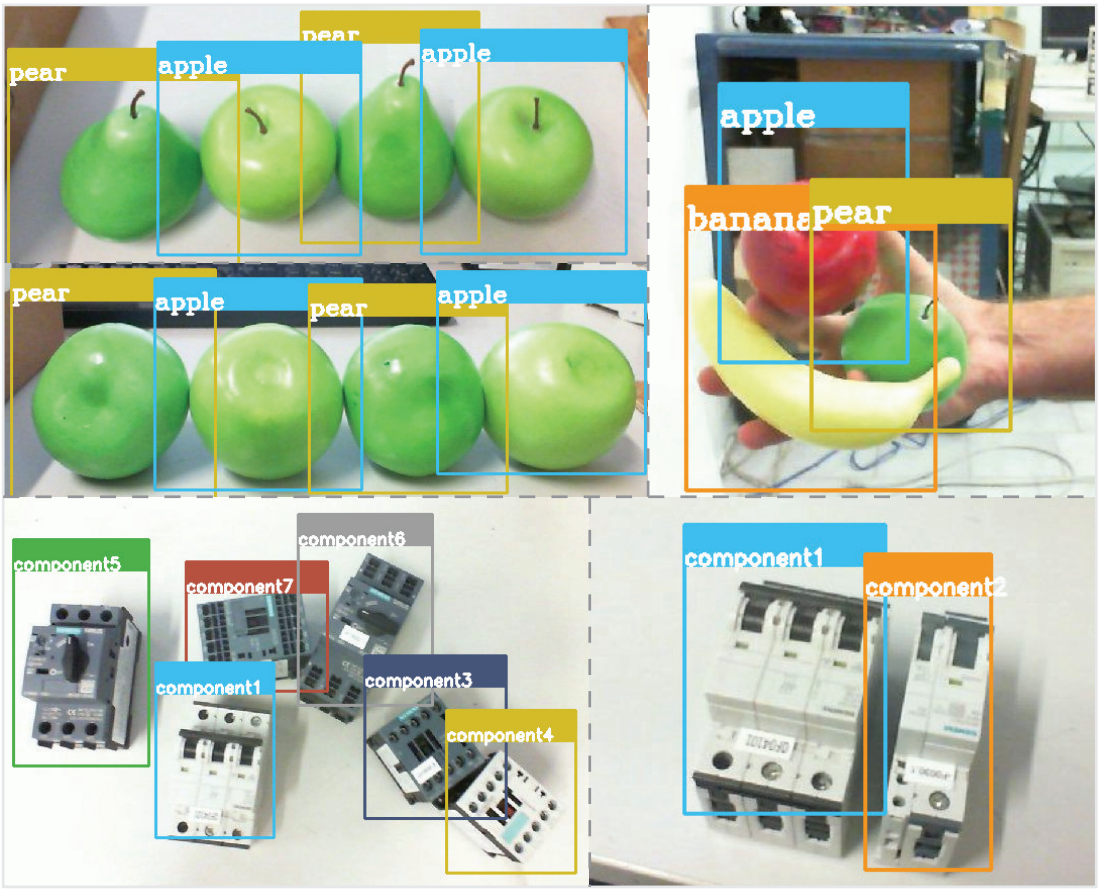
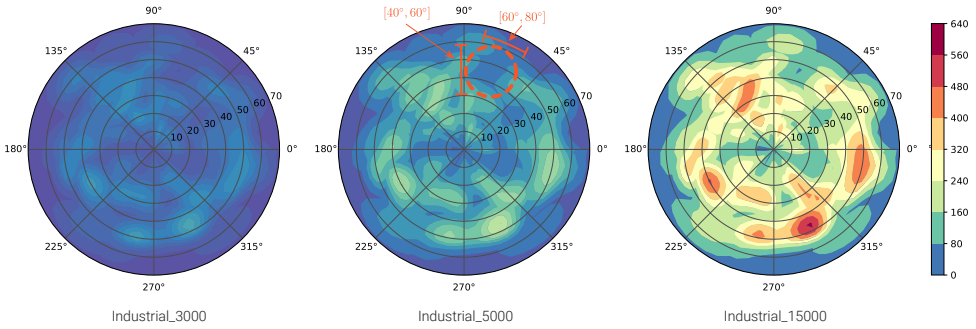


Fig. 5.8 Detectors output (all correct) computed over the stream of a usb camera. Its surprising how the detector is able to distinguish *apple*s and *pear*s only looking at their lower side.

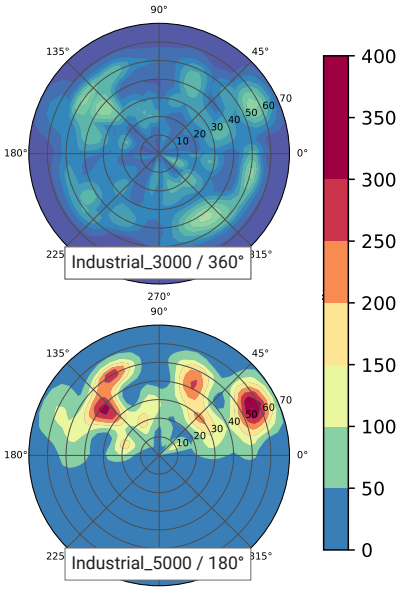
worst class specific precision recall curves (i.e. the performances for the hardest class for each detector).

5.3.4 Viewpoint Coverage

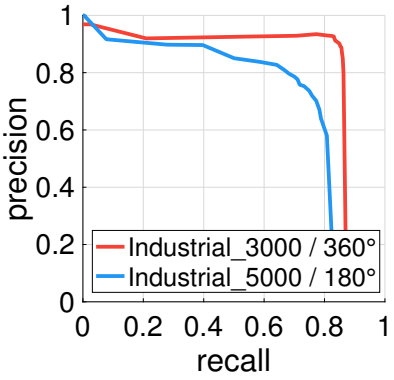
All the results reported so far show that *ROARS* is fast and effective for dataset creation. However, it has one additional useful side effect. For each image in the dataset we know the position of the camera with respect to each object in the scene: for each Virtual Box $V_j = \{{}^0\mathbf{T}_j, s_j, c_j\}$ we can compute the position of the i-th camera w.r.t. that object ${}^j\mathbf{T}_{cam_i} = ({}^0\mathbf{T}_j)^{-1} \cdot {}^0\mathbf{T}_{cam_i}$. If we consider only the position of the camera ${}^j\mathbf{p}_{cam_i}$ expressed in polar coordinates ${}^j(r,\theta,\phi)_{cam_i}$ (*i.e. radial*, *azimuthal*, *polar*) in the Virtual Box reference frame, we can build a 2D histogram by aggregating only (θ,ϕ) into bins. For each frame F_i we vote for the bin corresponding to ${}^j(r,\theta,\phi)_{cam_i}$. This kind of histogram tries to show the *coverage* of *viewpoints* of an object, thus we refer to this kind of metric as *Viewpoint Coverage* (VC) and Figure 5.9 (a) shows a heat



(a)



(b)



(c)

Fig. 5.9 (a) The Viewpoint Coverage computed for object of class 0 for the Industrial_3000/5000/15000 datasets. The ColorBar maps the color with the number of views voting for corresponding polar bin. (b) Shows the VC over Industrial_3000 and Industrial_5000, but the latter is filtered removing half the viewpoints. (c) Depicts the benefits of a better coverage despite the lower number of training samples.

map visualization of the histograms for three of the Industrial training sets presented above, hotter colors correspond to more image taken from that point of view. To clarify the meaning of the VC we can look at the same abovementioned image (the middle histogram) and we can see a dashed circle focusing on a region with a low score; that means the the object related to the histogram has never been seen within that region of (θ, ϕ). Such metric is particularly useful to spot potential flaws in the training set, i.e. cold portions of the histogram correspond to viewpoints where the detector will likely fail having almost never seen the objects from that position during training. In this way, Viewpoint Coverage can also be used to guide the user in the creation of the best training set possible, e.g. guide the acquisition of new sequences given the Viewpoint Coverage of the already acquired one. With this metric we can answer to the question "More images = Better detector?": from the Figure 5.9(b)(c) it would appear not true because a training set of 3000 samples (with a coverage of 360°) performs better than the one with 5000 samples (with a coverage of 180° only).

5.4 Extension of ROARS

We have demonstrated through careful tests and validation that *ROARS* can be used to create quickly and effortlessly high-quality datasets for object recognition and obtain excellent performance. In the future, we plan to experiment using the same technique to automatically generate more complex annotations like, for example, pixel level semantic segmentation maps. One of the main limitations of *ROARS* is that it can only be used for relatively small objects given the limited camera movement allowed due to the constraint imposed by the robot arm. Our method, however, can be generalized to any acquisition settings as long as it is possible to track the camera pose with good accuracy. For example, we plan to test *ROARS* using the VICON motion capture system or the SLAM engine integrated is some recent smartphones (e.g. the augmented reality toolkits made available by both Apple and Google).

Part III

Conclusions

Conclusions and future work

This thesis has been focused on the synergy between Computer Vision and Robotics. In Chapter 3 and Chapter 4 Computer Vision helps Robotics to understand the environment in order to perform tasks like Navigation or Object Grasp. In Chapter 2 and Chapter 5, instead, Robotics enables Computer Vision tasks by exploiting the high reliability and repeatability of the pose of a robotic agent. During the study of this synergy, several results have been achieved. Chapter 3 describes a novel 3D sparse (CPU-based) data structure, that turned out to be one of the first – faster – alternative to *Octree* in the field of Mobile Robotics. The possibility to exploit Mapping to perform Object Recognition in real-time has been explored in Chapter 4 by developing a lightweight CPU-based framework designed for Object Recognition on low-power robotic platforms or industrial controllers not holding a GPGPU hardware. Chapter 2, instead, was one of the first work to propose the use of the robot as a Camera Tracker proving that an interactive high-precision camera trajectory is mandatory for several industrial application which cannot suffer from the localization inaccuracies typical of a classical SLAM system. Finally, Chapter 5 describes a simple yet quite amazing technique to train a Deep Neural Network, again by deploying the high repeatability of an Industrial (or Collaborative) robot, in order to achieve a *near-perfect* Object Detector.

With this work we may make several assumptions: 1) It is possible to perform a large-scale 3D mapping in real-time on CPU with arbitrary Voxel Data; 2) It is possible to train a *near-perfect* Object Detector by exploiting a Robot; 3) The robot is almost crucial, in industrial applications, to perform a *Multi-View Object Detection* which has been proved to be state-of-the-art. Given these premises the natural evolution of this thesis on which we are already working on, is to build a general purpose Object Detector Robotic System dedicated to industrial applications like *Pick&Place* or *BinPicking* oriented towards *Industry 4.0*, where Robots and Humans collaborates to achieve the final goal. Following the intuitions of Chapter 5, an Human can teach a Robot to detect a set of generic objects with minimal effort (Figure 5.8). Hence, the basic idea is to exploit this simple procedure to produce a dense 3D reconstruction of the environment preserving semantic information about the target items, like in Figure 5.10. This kind of representation is key to perform grasps, as highlighted in Chapter 2. We conducted a

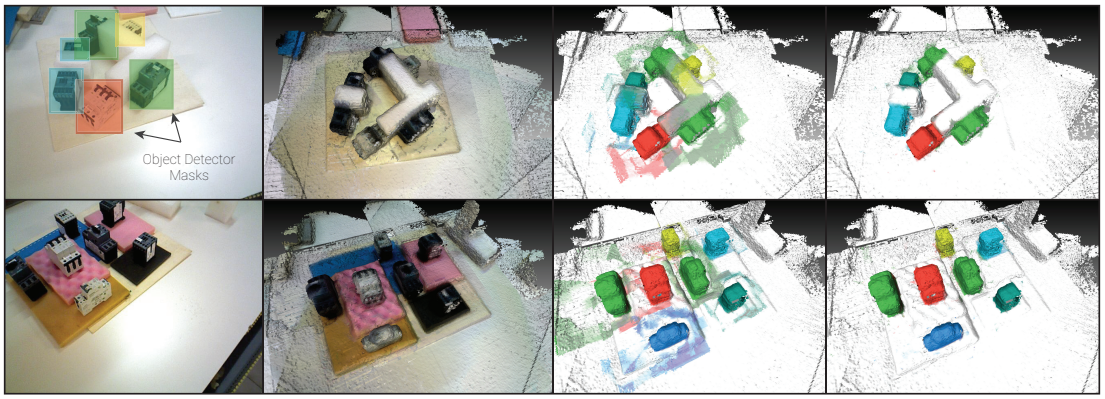


Fig. 5.10 The *Object Detection Fusion* approach: the first column represents frames coming from an RGB-D camera with superimposed object detection masks; in the second column the frames are fused together in a TSDF Volume (i.e. RGB Volume) where each voxel stores real RGB information; in the third one, the labels coming from the Object Detector are fused together in another TSDF Volume (i.e. Label Volume) where each voxel stores labels information; the last column depicts the Label Volume after background refinement (see Table 5.3). In the figure we are using a *color metaphor* to represent labels (one for each object) allowing a graphical human-interpretable representation through which is possible to *blend* labels (merging related color) in order to understand what happens behind the scenes.

preliminary study of this *Object Detection Fusion* approach over the *RGB-D Scenes Dataset v2* (Lai et al., 2014), with some promising results, as reported in Table 5.3. The *Object Detection Fusion* concept is similar to that described in Chapter 4: it stores in each Voxel a Multi-modal Distribution of Object Labels such that by analyzing – ex-post – the voxel map is possible to infer the pose of target objects. Though, our novel approach seems to be faster and simpler than similar solutions (Cavallari and Di Stefano, 2015; Lai et al., 2014; Li et al., 2016; McCormac et al., 2017; Tateno et al., 2016; Xiang and Fox, 2017).

Overall, this thesis endorsed the link between Robotics and Computer Vision by providing theoretical and practical [3] [4] instruments ready to be integrated in a real applications. The hope is that these intuitions will be further developed, especially with the advent of *Industry 4.0* and *Collaborative Robotics*, where Humans and Robots will ever-increasingly coexist.

[3] https://github.com/m4nh/skimap_ros
[4] https://github.com/m4nh/roars

Table 5.3 The 3D *Intersection over Union* for the five object categories in the RGB-D Scenes Dataset v2 (Lai et al., 2014) with the *Object Detection Fusion* approach. Since the 6-DoF Ground Truth for these object is near impossible, due to high symmetry, we choose to compare the 3D IoU of the minimum bounding box of the ground truth objects with the estimates. In this table several approaches was compared: **HT** means *High Threshold* used to distinguish real object voxels from noisy region; **LT** means *Low Threshold* and **WB** means *Without Background removal*.

	IoU 3D (HT)	IoU 3D (LT)	IoU 3D (WB)
Bowl	85.14	77.68	54.32
Cap	80.62	75.96	52.07
Cereal Box	87.44	77.66	58.23
Coffe Mug	84.68	65.13	51.11
Soda Can	87.44	78.46	52.06
overall	85.06	74.98	53.56

List of papers

De Gregorio, D., Tombari, F., & Di Stefano, L. (2016, October). RobotFusion: Grasping with a Robotic Manipulator via Multi-view Reconstruction. In Computer Vision–ECCV 2016 Workshops (pp. 634-647). Springer International Publishing.

Meattini R., Benatti S., Scarcia U., **De Gregorio, D.**, Benini L., Melchiorri C. (2016). EMG-based Grasp Proportional Control and Pattern Recognition for Human-Like Control of Robotic Hands. International Workshop on Human-Friendly Robotics, September 28-29, 2016, Genova.

De Gregorio, D., & Di Stefano, L. (2017). SkiMap: An efficient mapping framework for robot navigation. In Proceedings of IEEE International Conference on Robotics and Automation (ICRA), Singapore, 2017, pp. 2569-2576. doi: 10.1109/ICRA.2017.7989299

De Gregorio, D., Cavallari, T., & Di Stefano, L. (2017). SkiMap++: Real-Time Mapping and Object Recognition for Robotics. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (pp. 660-668).

M. Busi, A. Cirillo, **De Gregorio, D.**, M. Indovini, G. De Maria, C. Melchiorri, C. Natale, G. Palli, S. Pirozzi. (2017). The WIRES Experiment: Tools and Strategies for Robotized Switchgear Cabling, In Procedia Manufacturing, Volume 11, 2017, Pages 355-363, ISSN 2351-9789, https://doi.org/10.1016/j.promfg.2017.07.118.

Meattini, R., Benatti, S., Scarcia, U., **De Gregorio, D.**, Benini, L., and Melchiorri, C. (2018) A sEMG-Based Human-Robot Interface for Robotic Hands Using Machine Learning and Synergies. IEEE Transactions on Components, Packaging and Manufacturing Technology. doi: 10.1109/TCPMT.2018.2799987

De Gregorio, D., Zanella, R., Palli, G., Pirozzi, S., Melchiorri, C. (2018). Integration of Robotic Vision and Tactile Sensing for Wire-Terminal Insertion Tasks. IEEE Transactions on Automation Science and Engineering *[SUBMITTED]*

De Gregorio, D., Tonioni, A., Palli, G., Di Stefano, L. (2018). Semi-Automatic Labeling for Deep Learning in Robotics. IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), Madrid, 2018 *[SUBMITTED]*

List of Supplementary Video Material

Robot Fusion Video: https://www.youtube.com/watch?v=ECXamfIaPcQ This video, related to Chapter 2, shows the live execution of the Recognition-Grasping pipeline on unknown objects with a low-cost 2.5-finger gripper.

Robot Fusion Video 2: https://www.youtube.com/watch?v=Y1FaBE54E5A This video, related to Chapter 2, shows the live execution of the Recognition-Grasping pipeline on unknown objects with an industrial 3-finger gripper.

SkiMap Video: https://www.youtube.com/watch?v=MverWmFAgkg This video, related to Chapter 3, shows several runs of a mobile robot while mapping the environment by means of the SkiMap framework.

SkiMap++ Video: https://www.youtube.com/watch?v=ki_Lbl4lEIY This video, related to Chapter 4, shows the Recognition Pipeline of SkiMap++ with both a Mobile Robot and a Motion Tracking System.

Roars Video: https://www.youtube.com/watch?v=tj_h_N1Bylo This video, related to Chapter 5, shows the proposed labeling pipeline in a real industrial scenario.

List of figures

List of tables

References

Aldoma, A., Tombari, F., Prankl, J., Richtsfeld, A., Di Stefano, L., and Vincze, M. (2013). Multimodal cue integration through Hypotheses Verification for RGB-D object recognition and 6DOF pose estimation. *Proceedings - IEEE International Conference on Robotics and Automation*, pages 2104–2111.

Aldoma, A., Tombari, F., Rusu, R. B., and Vincze, M. (2012). OUR-CVFH: Oriented, Unique and Repeatable Clustered Viewpoint Feature Histogram for Object Recognition and 6DOF Pose Estimation. *Pattern Recognition*, 7476:113–122.

Aldoma, A., Vincze, M., Blodow, N., Gossow, D., Gedikli, S., Rusu, R. B., and Bradski, G. (2011). Cad-model recognition and 6dof pose estimation using 3d cues. In *Computer Vision Workshops (ICCV Workshops), 2011 IEEE International Conference on*, pages 585–592.

Arora, S. and Barak, B. (2009). *Computational complexity: a modern approach*. Cambridge University Press.

Bay, H., Tuytelaars, T., and Van Gool, L. (2006). Surf: Speeded up robust features. In *European conference on computer vision*, pages 404–417. Springer.

Bellare, M. and Rogaway, P. (1993). Random oracles are practical: A paradigm for designing efficient protocols. In *Proceedings of the 1st ACM conference on Computer and communications security*, pages 62–73. ACM.

Bentley, J. L. (1975). Multidimensional binary search trees used for associative searching. *Communications of the ACM*, 18(9):509–517.

Bicchi, A. and Kumar, V. (2000). Robotic grasping and contact: a review. In *Robotics and Automation, 2000. Proceedings. ICRA '00. IEEE International Conference on*, volume 1, pages 348–353 vol.1.

Bischoff, R., Huggenberger, U., and Prassler, E. (2011). Kuka youbot-a mobile manipulator for research and education. In *Robotics and Automation (ICRA), 2011 IEEE International Conference on*, pages 1–4. IEEE.

Brachmann, E., Krull, A., Michel, F., Gumhold, S., Shotton, J., and Rother, C. (2014). *Learning 6D Object Pose Estimation Using 3D Object Coordinates*, pages 536–551. Springer International Publishing, Cham.

Brachmann, E., Michel, F., Krull, A., Yang, M. Y., Gumhold, S., and Rother, C. (2016). Uncertainty-Driven 6D Pose Estimation of Objects and Scenes from a Single RGB Image. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 3364–3372. IEEE.

Bradski, G. and Kaehler, A. (2008). *Learning OpenCV: Computer vision with the OpenCV library*. " O'Reilly Media, Inc.".

Bylow, E., Sturm, J., Kerl, C., Kahl, F., and Cremers, D. (2013). Real-time camera tracking and 3d reconstruction using signed distance functions. In *Robotics: Science and Systems (RSS), Online Proceedings*.

Calli, B., Singh, A., Bruce, J., Walsman, A., Konolige, K., Srinivasa, S., Abbeel, P., and Dollar, A. M. (2017). Yale-CMU-Berkeley dataset for robotic manipulation research. *The International Journal of Robotics Research*, page 027836491770071.

Carlucci, F. M., Russo, P., and Caputo, B. (2016). A deep representation for depth images from synthetic data. pages 1362–1369.

Cavallari, T. and Di Stefano, L. (2015). Volume-based semantic labeling with signed distance functions. In *Pacific-Rim Symposium on Image and Video Technology*, pages 544–556. Springer International Publishing.

Chaumette, F. and Hutchinson, S. (2007). Visual servo control, part ii: Advanced approaches. *IEEE Robotics and Automation Magazine*, 14(1):109–118.

Cheng, Y. (1995). Mean shift, mode seeking, and clustering. *IEEE transactions on pattern analysis and machine intelligence*, 17(8):790–799.

Chi Li, Bohren, J., Carlson, E., and Hager, G. D. (2016). Hierarchical semantic parsing for object pose estimation in densely cluttered scenes. In *2016 IEEE International Conference on Robotics and Automation (ICRA)*, pages 5068–5075. IEEE.

Civera, J., Galvez-Lopez, D., Riazuelo, L., Tardos, J. D., and Montiel, J. M. M. (2011). Towards semantic SLAM using a monocular camera. In *2011 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 1277–1284. IEEE.

Collet, A. and Srinivasa, S. S. (2010). Efficient multi-view object recognition and full pose estimation. In *2010 IEEE International Conference on Robotics and Automation*, volume 2, pages 2050–2055. IEEE.

Conway, J. (1970). The game of life. *Scientific American*, 223(4):4.

Criminisi, A. and Shotton, J. (2013). *Decision forests for computer vision and medical image analysis*. Springer Science & Business Media.

Curless, B. and Levoy, M. (1996). A volumetric method for building complex models from range images. In *Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*, pages 303–312. ACM.

Dai, A., Nießner, M., Zollhöfer, M., Izadi, S., and Theobalt, C. (2016). Bundlefusion: Real-time globally consistent 3d reconstruction using online surface re-integratio. *arXiv preprint arXiv:1604.01093*.

De Gregorio, D. and Di Stefano, L. (2017). Skimap: An efficient mapping framework for robot navigation. In *2017 IEEE International Conference on Robotics and Automation (ICRA)*, pages 2569–2576.

Den Hollander, R. J. and Hanjalic, A. (2007). A combined ransac-hough transform algorithm for fundamental matrix estimation. In *BMVC*, pages 1–10.

Endres, F., Hess, J., Sturm, J., Cremers, D., and Burgard, W. (2014). 3-d mapping with an rgb-d camera. *IEEE Transactions on Robotics*, 30(1):177–187.

Engel, J., Schöps, T., and Cremers, D. (2014). LSD-SLAM: Large-scale direct monocular SLAM. In *European Conference on Computer Vision (ECCV)*.

Everingham, M., Van Gool, L., Williams, C. K. I., Winn, J., and Zisserman, A. (2010). The pascal visual object classes (voc) challenge. *International Journal of Computer Vision*, 88(2):303–338.

Ferrari, C. and Canny, J. (1992). Planning optimal grasps. In *Robotics and Automation, 1992. Proceedings., 1992 IEEE International Conference on*, pages 2290–2295 vol.3.

Fioraio, N. and Di Stefano, L. (2013). Joint detection, tracking and mapping by semantic bundle adjustment. *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 1538–1545.

Fioraio, N. and Di Stefano, L. (2015). Slamdunk: Affordable real-time rgb-d slam. In *Computer Vision - ECCV 2014 Workshops: Zurich, Switzerland, September 6-7 and 12, 2014, Proceedings, Part I*, pages 401–414.

Fioraio, N., Taylor, J., Fitzgibbon, A., Stefano, L. D., and Izadi, S. (2015). Large-scale and drift-free surface reconstruction using online subvolume registration. In *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 4475–4483.

Fridovich-Keil, D., Nelson, E., and Zakhor, A. (2017). Atommap: A probabilistic amorphous 3d map representation for robotics and surface reconstruction. In *Robotics and Automation (ICRA), 2017 IEEE International Conference on*, pages 3110–3117. IEEE.

Fäulhammer, T., Aldoma, A., Zillich, M., and Vincze, M. (2015). Temporal integration of feature correspondences for enhanced recognition in cluttered and dynamic environments. In *Int. Conf. Robotics and Automation (ICRA)*, pages 3003–3009.

García, I., Lefebvre, S., Hornus, S., and Lasram, A. (2011). Coherent parallel hashing. In *ACM Transactions on Graphics (TOG)*, volume 30, page 161. ACM.

Garrido-Jurado, S., Muñoz-Salinas, R., Madrid-Cuevas, F. J., and Marín-Jiménez, M. J. (2014). Automatic generation and detection of highly reliable fiducial markers under occlusion. *Pattern Recognition*, 47(6):2280–2292.

Georgakis, G., Mousavian, A., Berg, A. C., and Kosecka, J. (2017). Synthesizing Training Data for Object Detection in Indoor Scenes.

Grisetti, G., Stachniss, C., and Burgard, W. (2007). Improved techniques for grid mapping with rao-blackwellized particle filters. *IEEE Transactions on Robotics*, 23(1):34–46.

Hang, K., Stork, J., and Kragic, D. (2014). Hierarchical fingertip space for multi-fingered precision grasping. In *Intelligent Robots and Systems (IROS 2014), 2014 IEEE/RSJ International Conference on*, pages 1641–1648.

Hernández, C., Vogiatzis, G., and Cipolla, R. (2007). Probabilistic visibility for multi-view stereo. In *Computer Vision and Pattern Recognition, 2007. CVPR'07. IEEE Conference on*, pages 1–8. IEEE.

Hornung, A., Wurm, K. M., Bennewitz, M., Stachniss, C., and Burgard, W. (2013). Octomap: an efficient probabilistic 3d mapping framework based on octrees. *Autonomous Robots*, 34(3):189–206.

Huang, J., Rathod, V., Sun, C., Zhu, M., Korattikara, A., Fathi, A., Fischer, I., Wojna, Z., Song, Y., Guadarrama, S., et al. (2017). Speed/accuracy trade-offs for modern convolutional object detectors.

ISO TC 184SC 2 Robots and robotic devices (2015). ISO 9283. Manipulating industrial robots – Performance criteria and related test methods. *International Organization for Standardization, Geneva, Switzerland*.

Jiawei, W., Li, Y., Tao, L., and Yuan, Y. (2010). Three-dimensional interactive pen based on augmented reality. In *Image Analysis and Signal Processing (IASP), 2010 International Conference on*, pages 7–11. IEEE.

Johnson-Roberson, M., Barto, C., Mehta, R., Sridhar, S. N., Rosaen, K., and Vasudevan, R. (2016). Driving in the Matrix: Can Virtual Worlds Replace Human-Generated Annotations for Real World Tasks?

Kehl, W., Navab, N., and Ilic, S. (2014). Coloured signed distance fields for full 3d object reconstruction. In *Proceedings of the British Machine Vision Conference*. BMVA Press.

Khatib, O. (1986). Real-time obstacle avoidance for manipulators and mobile robots. *The international journal of robotics research*, 5(1):90–98.

Lai, K., Bo, L., and Fox, D. (2014). Unsupervised feature learning for 3D scene labeling. In *2014 IEEE International Conference on Robotics and Automation (ICRA)*, pages 3050–3057. IEEE.

Lai, K., Bo, L., Ren, X., and Fox, D. (2012). Detection-based object labeling in 3D scenes. In *2012 IEEE International Conference on Robotics and Automation*, pages 1330–1337. IEEE.

Land, M. F. and Nilsson, D.-E. (2012). *Animal eyes*. Oxford University Press.

Li, C., Xiao, H., Tateno, K., Tombari, F., Navab, N., and Hager, G. D. (2016). Incremental scene understanding on dense SLAM. In *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 574–581. IEEE.

Lin, T.-Y., Maire, M., Belongie, S., Hays, J., Perona, P., Ramanan, D., Dollár, P., and Zitnick, C. L. (2014). Microsoft coco: Common objects in context. In *European conference on computer vision*, pages 740–755. Springer.

Lorensen, W. E. and Cline, H. E. (1987). Marching cubes: A high resolution 3d surface construction algorithm. In *ACM siggraph computer graphics*, volume 21, pages 163–169. ACM.

Maaten, L. v. d. and Hinton, G. (2008). Visualizing data using t-sne. *Journal of Machine Learning Research*, 9(Nov):2579–2605.

Mahalanobis, P. C. (1936). On the generalised distance in statistics. *Proceedings of the National Institute of Sciences of India, 1936*, pages 49–55.

Maier, D., Hornung, A., and Bennewitz, M. (2012). Real-time navigation in 3d environments based on depth camera data. In *Humanoid Robots (Humanoids), 2012 12th IEEE-RAS International Conference on*, pages 692–697. IEEE.

Mayer, N., Ilg, E., Hausser, P., Fischer, P., Cremers, D., Dosovitskiy, A., and Brox, T. (2016). A large dataset to train convolutional networks for disparity, optical flow, and scene flow estimation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4040–4048.

McCormac, J., Handa, A., Davison, A., and Leutenegger, S. (2017). Semanticfusion: Dense 3d semantic mapping with convolutional neural networks. In *Robotics and Automation (ICRA), 2017 IEEE International Conference on*, pages 4628–4635. IEEE.

Meagher, D. J. (1982). Geometric modeling using octree encoding. In *Computer Graphics and Image Processing*, pages 129–147.

Mitash, C., Bekris, K. E., and Boularias, A. (2017). A self-supervised learning system for object detection using physics simulation and multi-view pose estimation. *arXiv preprint arXiv:1703.03347*.

Montana, D. (1991). The condition for contact grasp stability. In *IEEE Intemational Conference on Robotics and Automation*.

Moravec, H. and Elfes, A. (1985). High resolution maps from wide angle sonar. In *Robotics and Automation. Proceedings. 1985 IEEE International Conference on*, volume 2, pages 116–121. IEEE.

Movshovitz-Attias, Y., Kanade, T., and Sheikh, Y. (2016). How useful is photo-realistic rendering for visual learning? In *Computer Vision–ECCV 2016 Workshops*, pages 202–217. Springer.

Muis, A. and Ohnishi, K. (2005). Eye-to-hand approach on eye-in-hand configuration within real-time visual servoing. *IEEE/ASME transactions on Mechatronics*, 10(4):404–410.

Mur-Artal, R. and Tardós, J. D. (2017). Orb-slam2: An open-source slam system for monocular, stereo, and rgb-d cameras. *IEEE Transactions on Robotics*, 33(5):1255–1262.

Needham, J. (1974). *Science and Civilisation in China: Historical Survey, from Cinnabar Elixirs to Synthetic Insulin. Vol. 5, Chemistry and chemical technology; Pt. 3, Spagyrical discovery and invention*, volume 3. Cambridge University Press.

Newcombe, R. a., Davison, A. J., Izadi, S., Kohli, P., Hilliges, O., Shotton, J., Molyneaux, D., Hodges, S., Kim, D., and Fitzgibbon, A. (2011). KinectFusion: Real-time dense surface mapping and tracking. *10th IEEE International Symposium on Mixed and Augmented Reality*, pages 127–136.

Nguyen, C., Izadi, S., and Lovell, D. (2012). Modeling kinect sensor noise for improved 3d reconstruction and tracking. In *3D Imaging, Modeling, Processing, Visualization and Transmission (3DIMPVT), 2012 Second International Conference on*, pages 524–530.

Nießner, M., Zollhöfer, M., Izadi, S., and Stamminger, M. (2013). Real-time 3d reconstruction at scale using voxel hashing. *ACM Transactions on Graphics (TOG)*.

Papazov, Haddadin, Parusel, Krieger, and Burschka (2012). Rigid 3d geometry matching for grasping of known objects in cluttered scenes. *The International Journal of Robotics Research*.

Parker, L. E., Fregene, K., Guo, Y., and Madhavan, R. (2013). Multi-robot localization, mapping, and path planning. In *Multi-Robot Systems: From Swarms to Intelligent Automata: Proceedings from the 2002 NRL Workshop on Multi-Robot Systems*, page 21. Springer Science & Business Media.

Ponce, J. and Faverjon, B. (1991). On computing three-finger force-closure grasps of polygonal objects. In *Advanced Robotics, 1991. 'Robots in Unstructured Environments', 91 ICAR., Fifth International Conference on*, pages 1018–1023 vol.2.

Popović, M., Kraft, D., Bodenhagen, L., Başeski, E., Pugeault, N., Kragic, D., Asfour, T., and Krüger, N. (2010). A strategy for grasping unknown objects based on co-planarity and colour information. *Robotics and Autonomous Systems*, 58(5):551 – 565.

Pugh, W. (1990). Skip lists: A probabilistic alternative to balanced trees. *Commun. ACM*, pages 668–676.

Pugh, W. (1998). Concurrent maintenance of skip lists. Technical report.

Quigley, M., Conley, K., Gerkey, B. P., Faust, J., Foote, T., Leibs, J., Wheeler, R., and Ng, A. Y. (2009). Ros: an open-source robot operating system. In *ICRA Workshop on Open Source Software*.

Rahul Raguram, Jan-Michael Frahm, Marc Pollefeys (2008). A Comparative Analysis of RANSAC Techniques Leading to Adaptive Real-Time Random Sample Consensus.

Redmon, J. and Farhadi, A. (2017). Yolo9000: better, faster, stronger.

Rennie, C., Shome, R., Bekris, K. E., and De Souza, A. F. (2016). A Dataset for Improved RGBD-Based Object Detection and Pose Estimation for Warehouse Pick-and-Place. *IEEE Robotics and Automation Letters*, 1(2):1179–1185.

Richter, S. R., Vineet, V., Roth, S., and Koltun, V. (2016). Playing for data: Ground truth from computer games. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 9906 LNCS:102–118.

Ros, G., Sellart, L., Materzynska, J., Vazquez, D., and Lopez, A. (2016). The SYNTHIA Dataset: A large collection of synthetic images for semantic segmentation of urban scenes. In *CVPR*.

Roth-Tabak, Y. and Jain, R. (1989). Building an environment model using depth information. *Computer*, 22(6):85–90.

Rusu, R. B., Bradski, G., Thibaux, R., and Hsu, J. (2010). Fast 3D recognition and pose using the Viewpoint Feature Histogram. In *2010 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 2155–2162. IEEE.

Ryde, J. and Brünig, M. (2009). Non-cubic occupied voxel lists for robot maps. In *Intelligent Robots and Systems, 2009. IROS 2009. IEEE/RSJ International Conference on*, pages 4771–4776. IEEE.

Salas-Moreno, R. F., Newcombe, R. a., Strasdat, H., Kelly, P. H., and Davison, A. J. (2013). SLAM++: Simultaneous Localisation and Mapping at the Level of Objects. *2013 IEEE Conference on Computer Vision and Pattern Recognition*, pages 1352–1359.

Salti, S., Tombari, F., and Stefano, L. D. (2014). Shot: Unique signatures of histograms for surface and texture description. *Computer Vision and Image Understanding*, 125:251 – 264.

Saut, J.-P. and Sidobre, D. (2012). Efficient models for grasp planning with a multi-fingered hand. *Robotics and Autonomous Systems*, 60(3):347 – 357. Autonomous Grasping.

Shrivastava, A., Pfister, T., Tuzel, O., Susskind, J., Wang, W., and Webb, R. (2017). Learning from simulated and unsupervised images through adversarial training.

Simard, P. Y., Amershi, S., Chickering, D. M., Pelton, A. E., Ghorashi, S., Meek, C., Ramos, G., Suh, J., Verwey, J., Wang, M., et al. (2017). Machine teaching: A new paradigm for building machine learning systems. *arXiv preprint arXiv:1707.06742*.

Smith, M., Baldwin, I., Churchill, W., Paul, R., and Newman, P. (2009). The new college vision and laser data set. *The International Journal of Robotics Research*, 28(5):595–599.

Steinbrucker, F., Kerl, C., and Cremers, D. (2013). Large-scale multi-resolution surface reconstruction from rgb-d sequences. In *The IEEE International Conference on Computer Vision (ICCV)*.

Sturm, J., Engelhard, N., Endres, F., Burgard, W., and Cremers, D. (2012). A benchmark for the evaluation of rgb-d slam systems. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 573–580. IEEE.

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

Tateno, K., Tombari, F., and Navab, N. (2016). When 2.5D is not enough: Simultaneous reconstruction, segmentation and recognition on dense SLAM. *Proceedings - IEEE International Conference on Robotics and Automation*, pages 2295–2302.

Teschner, M., Heidelberger, B., Müller, M., Pomerantes, D., and Gross, M. H. (2003). Optimized spatial hashing for collision detection of deformable objects. In *Vmv*, volume 3, pages 47–54.

Thomas, A., Ferrar, V., Leibe, B., Tuytelaars, T., Schiel, B., and Van Gool, L. (2006). Towards Multi-View Object Class Detection. In *2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition - Volume 2 (CVPR'06)*, volume 2, pages 1589–1596. IEEE.

Thrun, S., Burgard, W., and Fox, D. (2005). *Probabilistic Robotics (Intelligent Robotics and Autonomous Agents)*. The MIT Press.

Tombari, F. and Di Stefano, L. (2012). Hough voting for 3d object recognition under occlusion and clutter. *IPSJ Transactions on Computer Vision and Applications*, 4:20–29.

Tombari, F., Fioraio, N., Cavallari, T., Salti, S., Petrelli, A., and Stefano, L. D. (2014). Automatic detection of pole-like structures in 3d urban environments. In *Intelligent Robots and Systems (IROS 2014), 2014 IEEE/RSJ International Conference on*, pages 4922–4929.

Whelan, T., Kaess, M., Johannsson, H., Fallon, M., Leonard, J. J., and Mcdonald, J. (2014). Real-time large scale dense RGB-D SLAM with volumetric fusion. *Intl. J. of Robotics Research, IJRR*.

Wilhelms, J. and Van Gelder, A. (1992). Octrees for faster isosurface generation. *ACM Transactions on Graphics (TOG)*, 11(3):201–227.

Wohlhart, P. and Lepetit, V. (2015). Learning descriptors for object recognition and 3D pose estimation. *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, (1):3109–3118.

Xiang, Y. and Fox, D. (2017). Da-rnn: Semantic mapping with data associated recurrent neural networks. *arXiv preprint arXiv:1703.03098*.

Xie, Z., Singh, A., Uang, J., Narayan, K. S., and Abbeel, P. (2013). Multimodal blending for high-accuracy instance recognition. In *Intelligent Robots and Systems (IROS), 2013 IEEE/RSJ International Conference on*, pages 2214–2221.

Yun Jiang and Moseson, S. and Saxena, A. (2011). Efficient grasping from RGBD images: Learning using a new rectangle representation. In *Robotics and Automation (ICRA), 2011 IEEE International Conference on*.

Zeng, A., Yu, K.-T., Song, S., Suo, D., Walker, E., Rodriguez, A., and Xiao, J. (2016). Multi-view Self-supervised Deep Learning for 6D Pose Estimation in the Amazon Picking Challenge. pages 1386–1393.