Scene Representation for Object Monitoring

Albert Plà
E-mail: apla@eia.udg.edu
Supervisor: Danica Kragic
Examiner: Danica Kragic
Computer Science

July 28, 2009

Abstract

In robotics, having a 3D representation of the environment where a robot is working can be very useful. In real-life scenarios, this environment is constantly changing for example by human interaction, external agents or by the robot itself. Thus, the representation needs to be constantly updated and extended to account for these dynamic scene changes.

In this work we face the problem of representing the scene where a robot is acting. Moreover, we ought to improve this representation by reusing the information obtained in previous scenes. Our goal is to build a method to represent a scene and to update it while changes are produced. In order to achieve that, different aspects of computer vision such as space representation or feature tracking are discussed.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Nowadays, robotics and computer vision are becoming important issues in our society. Everyday robots are involved in many different activities (Figure 1.1) such as manufacturing processes, cleaning tasks [41] or rescue activities [25] in natural disasters. All these tasks require one or both of the following skills: navigation in an environment and object manipulation. To succeed in these activities often robots require to obtain a representation of the scenario in which they are acting. Those are not static scenarios as objects can be moved by the robot or an external agent. In this thesis we propose a method to represent these scenes. And allow these representation to be updated after changes of the environment.

Figure 1.1: a) Robotic arm with a hand. (source: HiTech Gadgets Pro) b) American mine rescue robot. (source: U.S. Department of Labor) c) ROMI, a Korean cleaning robot (source: Korean Electronics and Telecommunications Research Institut)

1.1 Problem Statement

As already stated above, we face the problem of representing and updating a dynamic scene. Our aim is to find a way to represent a 3D scenario and update its information after changes. This can be done by comparing the current scene with the previous ones and by combining the 3D information with other 2D features, such as the motion produced in the scene. For that purpose we need to find a way to describe a 3-dimensional environment and a method to detect when something is moving in it.

The main goal of this thesis is to establish the bases of a framework to represent the information of a scene and to store the changes that took place in it. We want to obtain a model of a scene which could be updated combining the prior and the new information when an alteration occurs. To reach this objective we divided our main goal in smaller aims:

- Study different methods of space representation in order to evaluate how a scene could be represented.
- Analyze an optical flow method able to track image features.
- Implement a system able to represent a scene using the output of a stereo camera.
- Implement a system to detect if a scene is static or not and to track the objects that are moving in it.
- Implement a method to compare different scenes with the purpose of updating the newer one using the older information.

1.2 Outline of the Thesis

The thesis is organized in the following chapters:

- **Chapter 2: Related Work** presents some articles and investigations that inspired and helped during the realization of this work.
- **Chapter 3: Methodology** describes the foundations of the different methods used in this project. It analyzes different space representations and describes the Lucas-Kanade optical flow.
- **Chapter 4: Implementation** presents the built system, its parts and its behavior.

- **Chapter 5: Experiments** shows some of the performed experiments and their results.
- **Chapter 6: Conclusions** concludes this thesis and proposes some future work.

Chapter 2

Related Work

This thesis is mainly related to computer vision and robotics. So, it was necessary to read several articles and papers to get into the topic. There is a vast amount of literature in these areas so it was necessary to find some documentation which gave a greater overview of the area. A good starting point was to read *Towards Grasp-Oriented Visual Perception for Humanoid Robots* by Bohg *et al.* [18]. In this paper, the problem of designing a vision system for the purpose of object grasping in everyday environments is studied. It is a recommendable reading because it gives notions about attention systems, 3D reconstructing, grasp approximation and other related topics.

Point Clouds and Features

One of the essential needs of this work is to represent a scene from a point cloud. In consequence the process of building a point cloud was studied: Peasley [27] proposes an algorithm able to build robust point clouds using different stereo image pairs. Different ways of representing 3-dimensional objects and stereo images were also analyzed. Murray and Little [24] present a representation system consisting of small pieces of the image which they call *patchlets*; *patchlets* content information about the position, the size, the colors and the surface normal of the corresponding part of the image. Quigley *et al.* [22] propose a high-resolution 3D scanning system in order to improve the performance of object detection and representation. However, this paper was not directly related with the thesis as our work is focused in stereo cameras.

Volume Representations

Our project principally focuses on two different approaches: octree structures and hierarchical bounding boxes. Willhelms *et al.* [42] propose to encode solid renderings using the octree hierarchy; this work was also evaluated by Velasco and Torres [39]. A good resume of the octree and similar representations bases can be found in the

12^{th} chapter of Foley and VanDam's book *Computer Graphics* [13]. The second approach, the hierarchical bounding box, is presented and discussed by Huebner *et al.* [17]; it is explained in a very simple and comprehensive way by Haverkort [15] in the paper *Introduction to bounding volume hierarchies*, an abstract of his PHD thesis. As seen in Larsson and Akenine-Möller's work [21] bounding volumes can be used for other applications such as collision detections; it also can be used to implement ray tracers, as Wald *et al.* [40] discuss in their projects.

Motion Tracking

Motion tracking is another relevant part of the thesis. Currently there are several feature trackers developed. For example, an efficient way to track objects in a scene is to use a particle filter [2, 23]. As Okuma *et al.* [26] and also Sarkka *et al.* [34] show in their work, the particle filter can be used to track more than just one target. The feature tracker that has been chosen for this thesis is the Kanade-Lucas-Tomasi [5] tracker which is based on the Lucas-Kanade [38] optical flow method. This tracker has been used in several applications such as face tracking (Bourel *et al.* [8]) or in line and broken space tracking [11].

Grasping

Although grasping and segmentation are not intervening directly in this work, the result of this thesis could be useful to improve or to help to develop different segmentation and grasping techniques. In consequence, some grasping related reading has been done before starting the thesis. It provided a general idea about how objects can be grasped by a robotic arm and also some information about segmentation. Nowadays there are different ways of determining which is the best grasping configuration for an object. Bohg *et al.* [6] present a method of grasp selection based on the 2D shape of an object; considering that this thesis can provide information about the shape of an object, it could help to determine which is the best grasp point of an object. Another interesting grasping method is the one presented by Bergström and Kragic [3]: from the segmented results of a scene, they create a partial 3D reconstruction of the target object in order to determine the best grasp. Borst *et al.* [7] analyze different grasping types and discuss what features determine if a grasp point is appropriate or not. Some more relevant information about segmentation can be found in Willimon [43] and Taylor and Kleeman's [36] work.

Attribute Representation

The work by Huebner *et al.* [16]. Their work is oriented towards the idea of developing cognitive capabilities in artificial systems through Object Action Complexes [14]. They try to identify which properties make an object being that object. Huebner *et al.* propose to describe objects with as many attributes as possible (visual attributes, shape attributes, manipulation attributes, etc.). This work is interesting

for this thesis as our *properties database* could be linked with the ActionTable in the future.

Finally, Welke *et al.* [19] present a system which identifies and stores locations of objects, encountered from different angles of view, in a search process. This work shares some similarities with our thesis, both of them try to represent a scene from the data obtained using stereo cameras. However, they consider that the scene does not change while they are building the scene and they focus on achieving the most detailed representation as possible; in contrast, our project focuses on updating the scene when something changes in it and trying to understand the change. Another relevant aspect of Welke's work is the usage of Ego-Spheres as environment representation. Ego-Spheres are a short-term memory for a robot in an egocentric way; it stores information about the robot's location inside a spherical space taking the robot as reference: the closer to the sphere limit the objects are, the further from the robot they stay. More about this concept can be found in the article published by Achim *et al.* [12].

Chapter 3

Methodology

In this chapter we present the used methods in the thesis. First we discuss about 3D scene representations, introducing the Octree structure and the Hierarchical Bounding Boxes. Then we describe a method to track movement in image sequences so changes in a scene can be detected.

3.1 Space Representation

One of the goals of this project is to study and analyze different space representations. For a robot is important to know where it is moving; if it is interacting in a household environment is useful to have a map of the place in order to avoid obstacles like tables, doors, etc. In the same way, when it has to manipulate objects it would be a good idea to represent positions and additional information such as e.g. size and shape of these objects to facilitate grasping.

3.1.1 Spatial-Partitioning Representations

One of the most common kinds of representation is the spatial-partitioning (SPR); here, solid objects are decomposed into groups of smaller non-intersecting parts in order to simplify more complex items. The primitives, sizes, orientation and other parameters of the decompositions may vary depending on the technique which is used. In these representations the smaller the decompositions are, the better is the obtained accuracy. There are many ways to decompose a volume: using one or various primitives, sequential or hierarchical structures, etc. In the following paragraphs some of these methods will be commented:

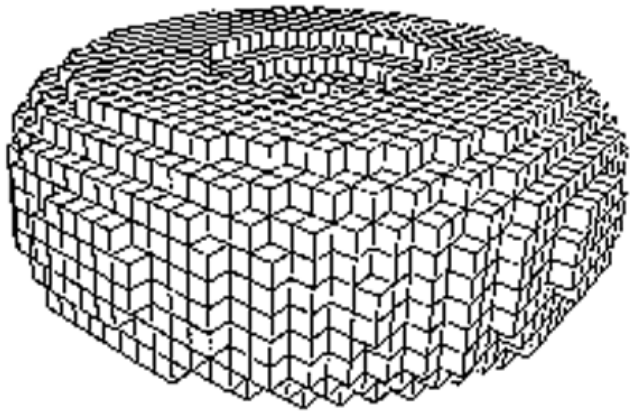

Figure 3.1: Torus representation using spatial-occupancy enumeration

Cell Decomposition

Cell decomposition is one of the simplest SPRs. From a predefined set of parameterized primitive geometric figures, many times curved, the scene is decomposed using them. Cell decomposition uses the computational brute force to fit the primitives in the best way possible. The fact that the different primitives cannot overlap constraints the search for good representations. Another disadvantage of this technique is that one object can have many representations, making the task of comparing scenes more difficult.

Spatial-Occupancy Enumeration

Spatial-Occupancy Enumeration (SOe) is a special case of Cell Decomposition. It uses only one primitive, usually cubes, and the decompositions are arranged in a fixed regular grid; the different cells are usually called voxels. It is based on an occupancy system: that means that one cell can either be occupied or empty. All cells occupied by the solid are labeled. As we can see in Figure 3.1, using this method a unique and unambiguous modeling of the volume is always found. Another advantage is that simple operations such as finding adjacencies, split, merge, and Boolean operators (OR, AND, XOR, etc.) are simple to implement; However, some weaknesses appears when we try to represent curved or non-rectangular objects, they can only be approximated. For example, if we are using cubes as voxels only the shapes which are parallel to the cubes' faces can be represented accurately. Obviously, decreasing the size of the cells improves the quality of the representation but it also increases the computational time need to build it: the complexity of this representation is n^3.

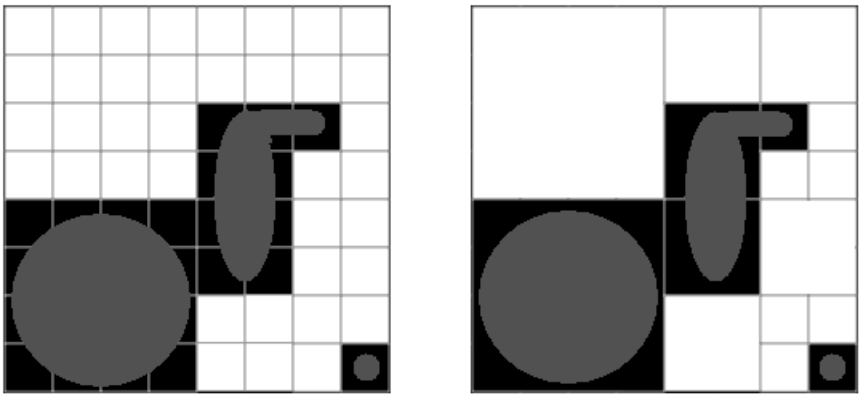

Figure 3.2: a)2D image encoded using a regular grid. b) 2D image encoded using a quadtree.

Octree

The octree representation is the hierarchical variant of Spatial-Occupancy Enumeration. It was developed to improve the efficiency of SOE and to minimize storage requirements. This representation was inspired by quadtrees (Figure 3.2): a system to encode 2D images [20]. Both methods use the 'Divide and Conquer' principle, employing the binary subdivision.

The quadtree divides a 2D plane in both dimensions generating 4 rectangles, it then fills the resulting regions using white (completely empty), black (completely full) or gray (partially full). This operation is recursively repeated on the gray zones until they disappear or until a certain precision is reached. As seen in Figure 3.3, the whole image can be represented with a 4-child tree where each node represents a subdivision of the image.

Octrees follow the same mechanism but adding an additional dimension. This results in 8 nodes instead of 4; those are usually referred by numbers 0 to 7 or for their position (up, down, front, back, left, right): LUF,LUB,LDF,LDB,RUF,RUB,RDF,RDB. Usually, the number of nodes needed to represent a solid figure is proportional to its surface [35]; this happens due to the fact that when we analyze a solid figure we cannot see its inside so only the boundary of the object is represented. In consequence the cost of building an octree is proportional to the surface of the solid.

Octree Operations

- Boolean set operations: Union and Intersection

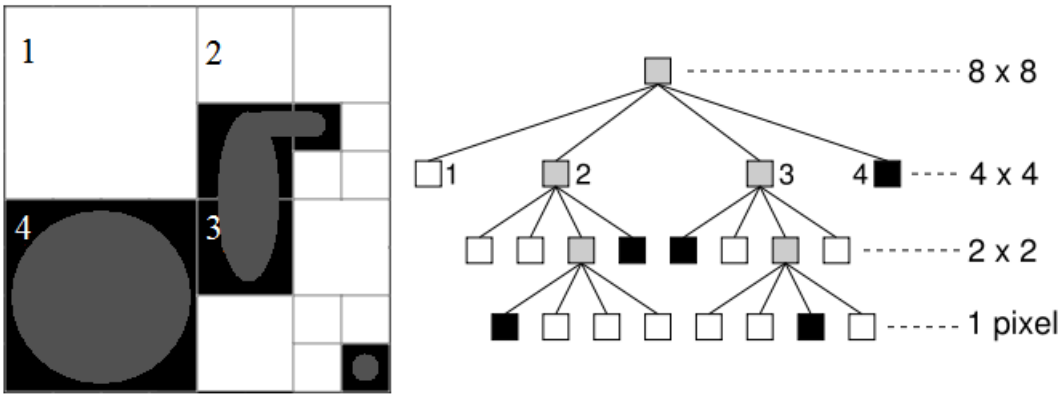

Figure 3.3: Tree representation for a quadtree

Boolean operations in quadtrees and octrees are not difficult to implement and they are not computationally expensive *(o(n))*. Union (∪) and Intersection (∩) shares the same mechanism: Having two octrees (A and B), we traverse both of them from the top to the leaves. Then, every pair of nodes is compared: In both operations a gray node is added to the new octree if the two nodes are gray; if the union is being performed, we add a new black node to the resulting octree if one of the pair's nodes is black, a white node is added if both of them are white; in the intersection case, the black node is only added if both are black. At the end of the operations it is important to check that no gray node has all its children black or white, if that happens the node must be filled with the corresponding color.

- Neighbor finding

  When a solid figure is represented by nodes, it can be very useful to identify its neighbors. There are several algorithms suitable to find the neighbor of a node in a concrete direction. One of them is the 'opposite direction algorithm'. This algorithm works both in octrees and quadtrees. Once a node and a direction for the neighbor has been chosen (e.g. North), the tree must be traversed bottom up until a node is reached from an opposite way node (e.g. South West or South East). When that happens, the inverse path must be followed (e.g. if the bottom-top path was SW - SE, the top-bottom should be NE - SW).

3.1.2 Bounding Boxes

Another way of representing 3D objects is to fit their shape into a polyhedron. The most common shape is a box. As seen in Figure 3.4 a bounding box is a cuboid which fits all the points of Objects. A minimum Bounding Box is the smallest box which

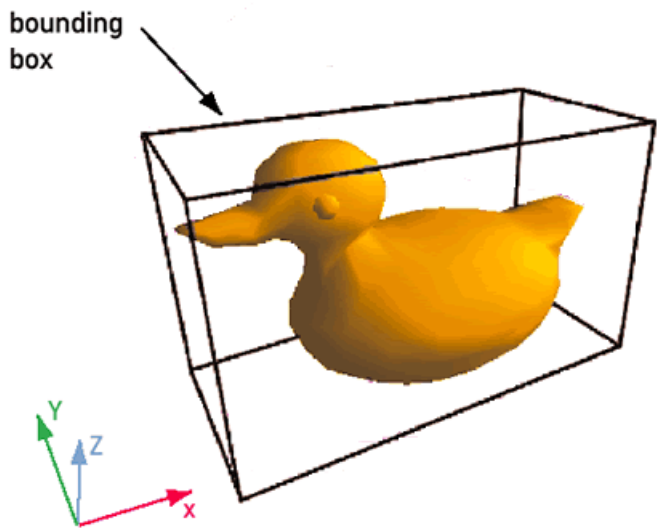

Figure 3.4: Bounding box and bounding sphere for a 3D object

encloses an arbitrary point set. Opposing the methods mentioned in Section 3.1.1, the Bounding Boxes can overlap. This representation does not give an accurate model of the object's shape; however, it simplifies its modeling and gives a general idea of its position, orientation and occupancy.

Hierarchical Bounding Boxes

The Hierarchical Bounding Boxes is a method which represents a 3D object by decomposing it into smaller boxes and organizing them in a hierarchical structure. To achieve that, the point cloud corresponding to the 3D object can be split in smaller data sets. By fitting the new points into smaller boxes and repeating this action recursively we obtain a more precise rendering of the solid object. The output of this process can be structured as a tree, where the first box is the root node and the successive cuboids are its children (see Figure 3.5). This is a less rigid representation than the octree since the cuboids do not follow a default orientation or size; in addition, the weight of this system is very light because not many boxes are needed to get a good approach to the solid figure. However, performing operations with different trees (e.g. union or intersection) can be very difficult because nodes cannot be compared directly and it may require a rebuilding of the structure.

3.2 Feature Trackers

With our approach we cannot build representation of a solid if this or the cameras are moving as it would be instantly obsolete; therefore we need a way to detect if

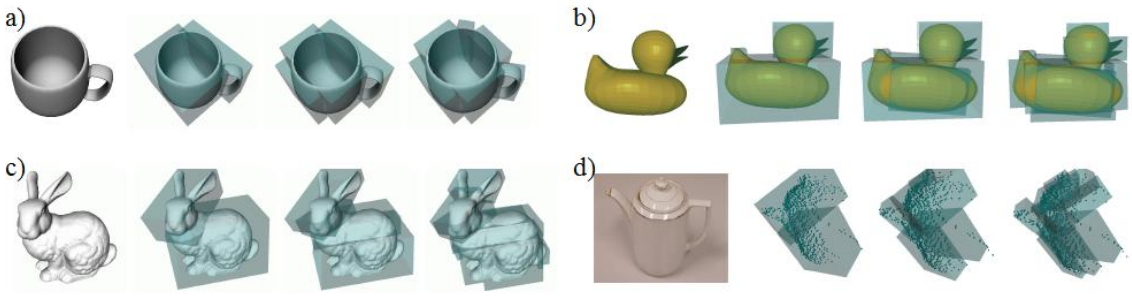

Figure 3.5: a) Hierarchical bounding boxes for a synthetic mug. b) HBB for a synthetic duck. c) Representation for a synthetic rabbit. d) HBB splits for a teapot.

there is motion in a scene. In addition, this information can be used later for a better comprehension of the scene.

3.2.1 Lucas-Kanade Optical Flow Method

Lucas and Kanade [38] developed a two frame differential method for optical flow estimation. It is based on the assumption that the flow in a local neighborhood around the central pixel is constant in a given time. Initially this method was developed to track objects while the camera was moving, however, it can also be used to follow objects.

When the camera moves the information stored on an image can vary significantly due to position, illumination and angle of vision changes. However, if two images are recorded in short time intervals they do not differ very much from each other because they are describing the same scene from very nearby viewpoints. A sequence of images can be defined as *F(x,y,t)* where x and y are the space and t is the time variable. The variation of the space variables in a short period of time ($k+\tau$) is known as displacement [33] and can be obtained through the following equation where ε,η are the displacement parameters:

$$F(x, y, t + \tau) = F(x + \varepsilon, y + \eta, t) \tag{3.1}$$

However, due to illumination changes, occlusions and other situations, Equation 3.1 is not always satisfied; some points appear or disappear, others are modified, etc. Despite this fact, Equation 3.1 can still be applied on various zones of an image, especially in those which are textured. These zones are enough to track the motion during an image sequence.

Kanade and Tomasi [38] point out that a single pixel was not enough to be tracked; still having a significant special brightness, it could be confused with adjacent pixels or loose its brightness because of lighting changes. So they propose that the best

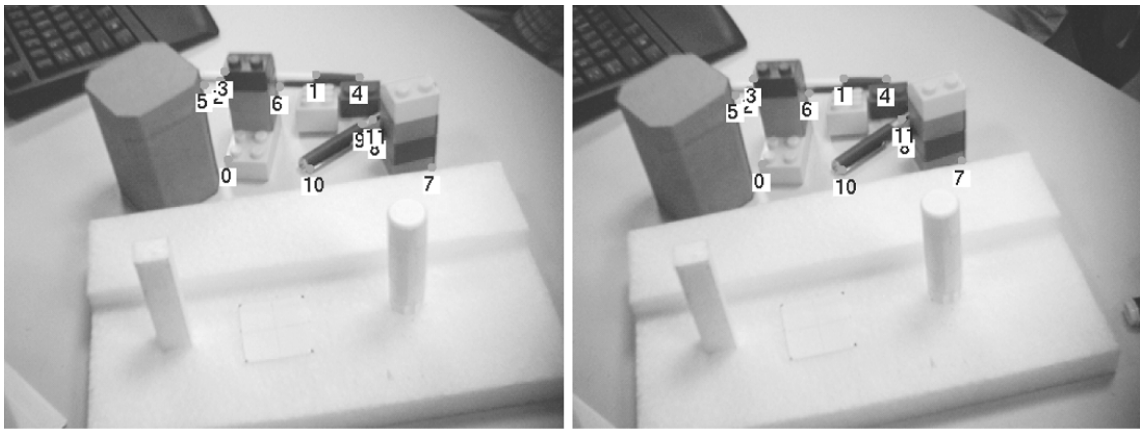

Figure 3.6: Tracked features using the Kanade-Lucas-Tomasi tracker

solution is to track a set of pixels: a textured window. Nevertheless the content of a window could change over time, and its pixels could move at different speeds; so, how do we know if we are following the same window? And how can we calculate the displacement vector if pixels does not necessarily move at the same speed?

To solve the first problem they propose to evaluate the variation of the different windows, i.e. if between two scenes a window changed too much they discard it. This decision improves the robustness of the tracking although it makes it sensitive to radical and fast movements. The second problem is solved by describing the movement of the windows by more complex transformations (e.g. affine maps [28]) instead of describing them just with translations.

The Lucas-Kanade optical flow is the base for the KLT (Kanade-Lucas-Tomasi) tracker [5]. Using the techniques described previously the KLT defines a measure of dissimilarity to quantify the change of a feature between an image and its predecessor and to search for affine changes [10]. In addition, a translation model of motion is used to track the best features in the sequence. The tracker can be configured to find new features every time one is lost, allowing the use of a constant number of features. It is used in many fields: object tracking (Figure 3.6), motion detection, stereo matching, etc.

3.3 Summary

Being conscious of the aims of the project we have discussed different ways to represent a 3-dimensional object. On the one hand, the octree representation offers a unique hierarchical representation that facilitates the comparison between dif-

ferent structures, it also offers intuitive information about the density and space distribution; moreover, if accuracy is not a requirement, the octree structure is computationally light. On the other hand, the hierarchicalbounding volumes structure offers a more dynamic scene representation which can be better for some robotic tasks such as grasping. However, the output models of that system could not be unique. In the project both methods were used so we can compare the results.

Regarding the motion tracker needed in this thesis to detect changes in a scene, it was built using Lucas-Kanade optical flow and the KLT tracker. These methods are quite strong and, as seen in the literature [38, 33], they have been tested many times so we can assume that they work properly.

Chapter 4

Implementation

In this chapter we will present and describe the proposed system. The different parts of the framework will be explained and, the different libraries and programs that were used will be commented.

In Figure 4.1 the system is outlined: First the data is acquired from the stereo head using the 3D-Frost [30] program in order to build an initial representation of the environment; this model will be stored in the scene memory for further usage. Then the *motion detection* module will detect any movement in the scene. When the scene becomes static again the trajectory of the movement will be stored and the *update process* will be called. After detecting some movement or detecting a viewpoint change, the scenario will be rebuilt, updated or extended using the representation stored in the *scene memory*. Finally the information of the model will be saved as an *XML* file so it can be used and reloaded in the future.

4.1 Acquisition Process

The input data needed for the system are point clouds. A point cloud is a set of vertices which represents a 3D scene in a three-dimensional coordinate system. To obtain the data we use the 3D-Frost program. From stereo cameras, it receives the disparity image which is used to get a point cloud of a scene. Moreover, the program makes a prior segmentation of the different objects in the scene based on t a projection of the point cloud on the detected table plane; nevertheless this segmentation can be imprecise when objects are too close or overlapping due to noise.

3D-Frost can run in different modes, one of them is to work with background differencing [44]. When this option is selected, a mask is established from image differencing [29] in order to roughly remove the background of the image and to work only with the objects placed in the scene. That mask improves the resulting

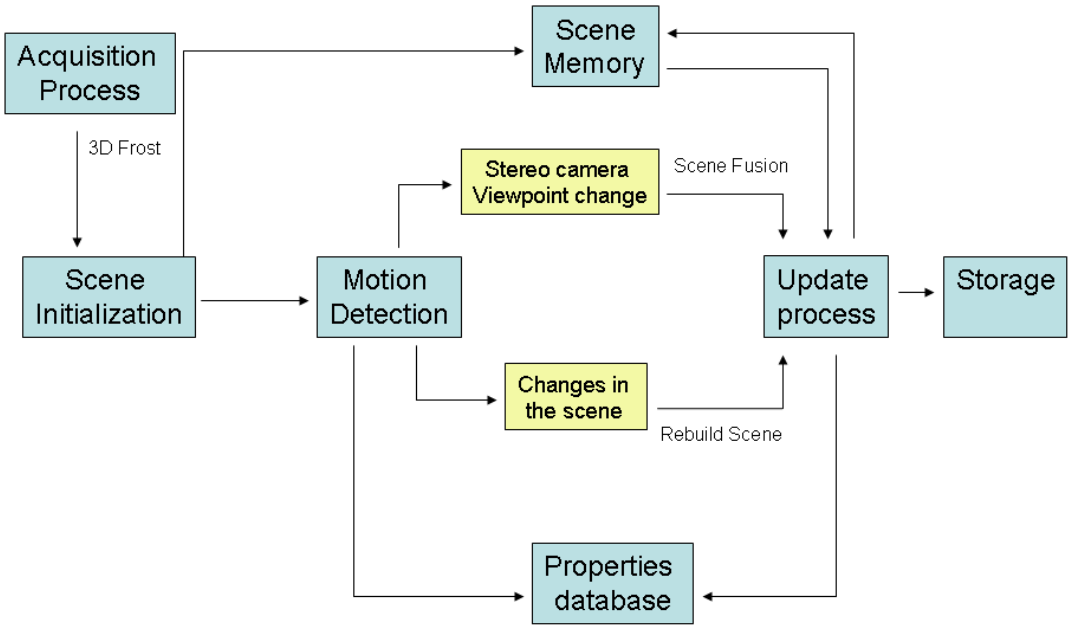

Figure 4.1: Overview of the processes in the proposed system. The acquisition process will generate a point cloud from a stereo camera and the system model will be initialized. The motion detection block will perceive movements in the scene in order to decide if the system must be updated. The scene memory will store the last representation of the scenario in order to compare the difference to the new scene. The properties database will store information about the different regions in the scene. The update process will add or remove the new information depending on the kind of movement detected (stereo cameras viewpoint change or changes in the scenario). Finally the storage module will save the information on a XML file.

disparity image, removing a great amount of noise. As the goal of the project is to find a good way to represent different objects in a scene, we activate the background differencing option so our results will be more clean and accurate.

When 3D-Frost is executed, different files are obtained:

- Various point clouds containing the information in different coordinate systems (Figure 4.2 a).
- The rectified images (Figure 4.2 c/d) .
- A segmented image of the scene (Figure 4.2 b) .

3D-Frost segments the point cloud using a projection to the table plane, the output is a segmented image. From these 2D segments we calculate the segmented point cloud.

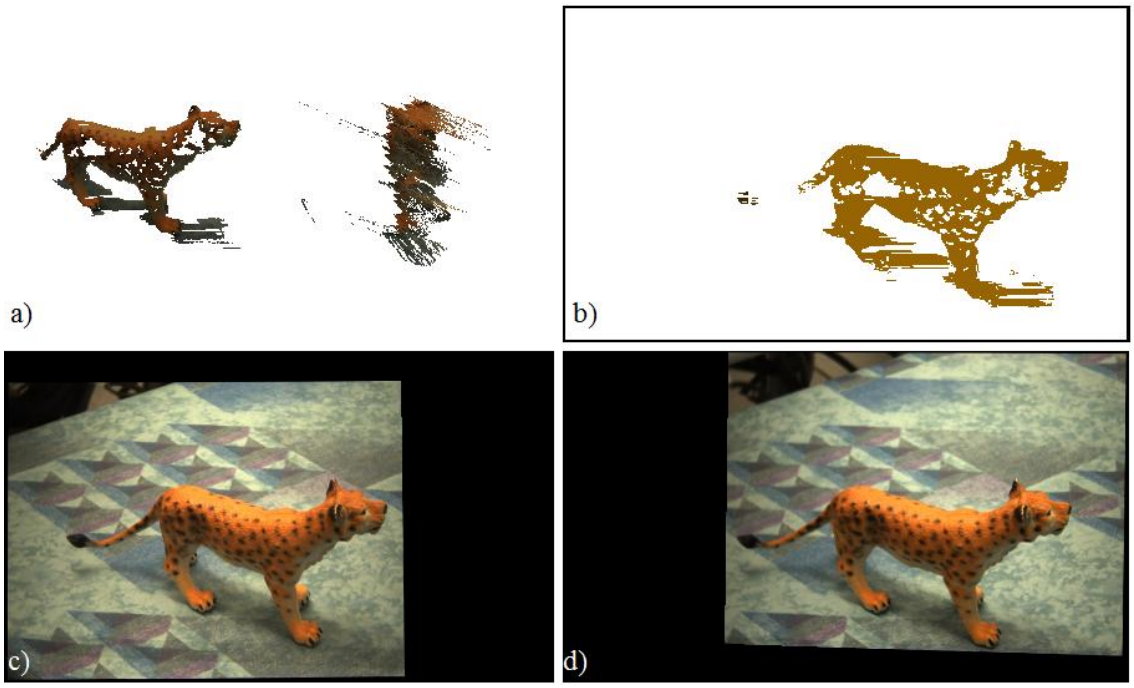

Figure 4.2: Outputs of the 3D-Frost program: a) Point cloud seen from the camera and from the top of the table. b) Segmented image. c) Right camera rectified image. d) Left camera rectified image.

The point cloud files, besides the coordinate points, also store some information about the source image such as the position of the pixels they are referring to and their color. As the segmented image (Figure 4.2 b) is directly drawn on the camera image, we can use that picture to relate each pixel of the point cloud with its segment.

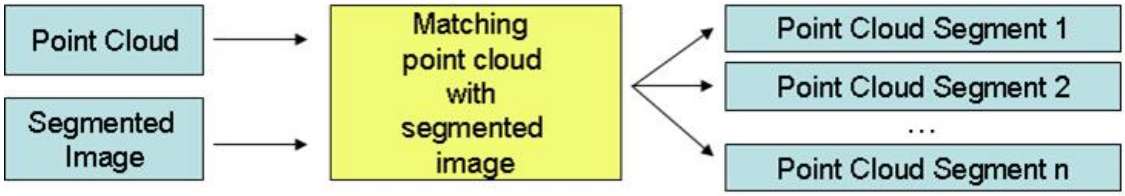

Figure 4.3: Segmenting point clouds diagram

By this we can construct new individual point clouds for every segment (Figure 4.3). Those segmented point clouds will later be used to construct the representation of the scene.

4.2 Scene Initialization

The scene initialization stage builds the first representation of the scene. In Section 3.1 we proposed two different structures for solid object rendering: the octree and the hierarchical bounding boxes. We decided to implement both in order to compare their performance. Both of them are built using the resulting point clouds of the 3D-Frost (every point cloud is considered a different segment).

4.2.1 Octree

To build the octree we decided to use the tree structure representation instead of the sequential one due to its more comprehensive and intuitive design. We designed an *Octree* class that holds the point cloud of the output and the root node of the tree. Also a *Node* class was created; it stores the location of the node, a boolean that indicates if the node is a leaf node, a pointer to its children and, if the node is a leaf, a string that indicates the segment to which it belongs and the points that it contains.

Split and Merge

To build the octree from the point cloud we use the Split and Merge method [9]. An imaginary cube which fits all the points of the point cloud is created (the root node), and is then divided in 8 parts (its children). Next, the system checks if any point lies in the child's space; if that happens, the process is recursively repeated. The process could be repeated infinitely because there will always be children containing points; in consequence it is necessary to set a maximum depth parameter which indicates when to stop the process. When this limit is reached, the child regions are tagged with the segment of the containing points and the nodes are set as

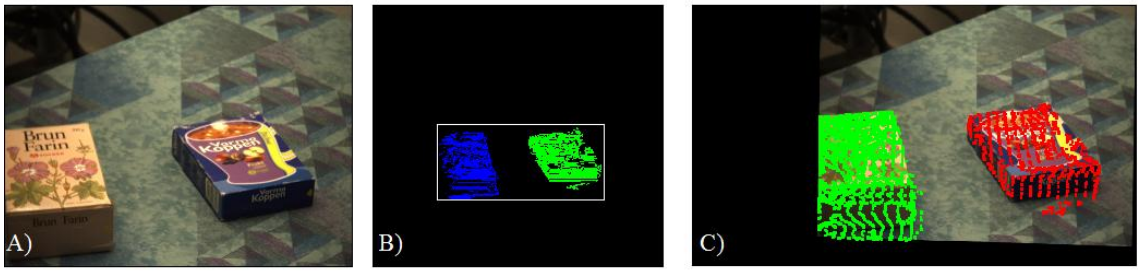

Figure 4.4: a) Original frame from the left stereo camera. b) 3D octree rendering (Depth = 7). c) Back projection of the octree to the left image (Depth = 5).

leaves. Finally, all the leaves are compared with their siblings to avoid unnecessary splits: if they all share the same tag then they are merged into the same node.

The depth parameter is critical to this algorithm because it determines the required time to build the octree and the maximum number of nodes needed (8^{depth}). Moreover it defines the accuracy of the representation. Experimental evaluation (see Section 5.2) will show that a maxium depth of 5 or 6 is the best compromise between accuracy and complexity. Although the representations obtained with this depth look less precise than the one obtained with 7 or 8, it was enough to notice the changes in a scene. Furthermore, the time needed to build the representation was much shorter.

Output visualization

A way to visualise the built structure was needed in order to verify the proper functioning of the system and to evaluate its accuracy. We implemented two visualization methods: one shows a 3D representation of the octree and the other draws the back projection of the octree in the original stereo images.

The 3D visualization tool (Figure 4.4 b) was build using the OpenGL [32] library and the GLUT utilities [31]. It draws all the leaf nodes of the octree into an OpenGL environment using a different color for every segment. This visualization allows to rotate and translate the represented octree. It is a good way to check the quality of the octree and to understand where the objects are placed.

The second visualization (Figure 4.4 c) is oriented to evaluate the segmentation of the different objects of the scene. The visualization corresponds to the octree back projection on the left stereo camera image. It draws a square to the octree corresponding input image region; the color of the drawing depends on the segment to which the node belongs.

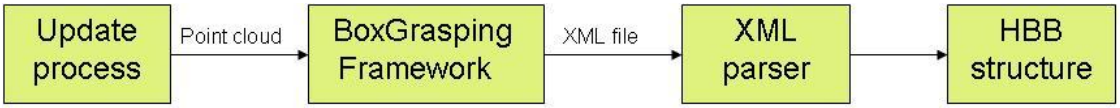

Figure 4.5: Different parts of the building the HBB process

4.2.2 Hierarchical Bounding Boxes

To implement the hierarchical bounding boxes (HBB) representation we build a similar structure to the octree. We created a class called *HBB* and another one called *Node*. The HBB class holds the point clouds and the root node; The *Node* class stores the size, the position, the segment of the cuboid and a pointer to its children.

We used an implementation of the Minimum Volume Bounding Box approach: the BoxGrasping framework [17]. BGF provides the box approximation of a point cloud by a set of boxes for the purpose of grasping. The framework was specially developed to obtain good grasps for a robot; We can use it to get the HBB as one of its options is to build an XML file containing the HBB of an object. As can be seen in Figure 4.5, the XML file is read by an XML parser [37] and then the HBB tree is build.

This tree was visualized similar to the octree as described in Section 4.2.1.

After performing a few tests we realized that this method is very sensitive to the high noise in the point clouds, making the bounding boxes too big or giving them a bad orientation (see Section 5.2). This fact, in addition to the difficulty of comparing different HBB tree structures, made us discard this representation.

4.3 Motion Detection

The main purpose of this module, as mentioned above, is to detect movement in a scene and to detect when it remains static again. Moreover, this module can be used to determine the motion of the objects in the scene, so tat it can be stored on the properties database for a later usage (e.g. to track the movement of the objects or to determine how they can be moved) . In consequence, the two principal functions of this module will be the 'motion detector' and the 'feature tracker'.

For this purpose the Clemson's University KLT Implementation [4] was used. The KLT is a free open source implementation of Kanade and Lucas' work [38]. It was implemented in C so a few modifications were required in order to make it fully compatible with our system, which is build in C++. The Clemson's University KLT

Implementation provides a framework able to select suitable texture-based features on a gray scale image for further tracking. Once features is selected, it can track them along an image sequence; in case of feature loss due to occlusions or image changes, it is prepared to replace them with new features.

4.3.1 Motion Detector

Given an image sequence of a scene, the goal of this function is to determine when the scene is static or if something is moving in it. To do this, an arbitrary number of features are selected in the first image. Those features are tracked in every image of the sequence, selecting new features every time one is lost. We assume that when features are lost there is some kind of motion in the scene. When no new features are selected we calculate the Euclidean distance d (Equation 4.1) between the feature's position in the current (x_1, y_1) and the previous (x_2, y_2) image:

$$d = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2} \tag{4.1}$$

Then we perform the summation of all the distances; if the result is smaller than a constant (*MaxDifference*) we consider that there is no motion between the two snapshots. If this happens in three consecutive frames we consider that the scene is static. The absence of movement is defined in Equation 4.2 where n represents the frame number and i the index of the ith feature.

$$\begin{aligned} Stopped = (&\sum_{i=0}^{nfeat} \sqrt{(x_{n_i} - x_{(n-1)_i})^2 + (y_{n_i} - y_{(n-1)_i})^2} < MaxDifference) \\ \wedge(&\sum_{i=0}^{nfeat} \sqrt{(x_{(n-1)_i} - x_{(n-2)_i})^2 + (y_{(n-1)_i} - y_{(n-2)_i})^2} < MaxDifference) \\ \wedge(&\sum_{i=0}^{nfeat} \sqrt{(x_{(n-2)_i} - x_{(n-3)_i})^2 + (y_{(n-2)_i} - y_{(n-3)_i})^2} < MaxDifference). \end{aligned} \tag{4.2}$$

In Algorithm 1 the pseudo code for this function is shown.

4.3.2 Feature Tracking

The aim of this function is to store the movement of the different features along an image sequence in plain text file. Given an image sequence, the function selects an arbitrary number of features from the first image. Then features are tracked in the different images and, finally, they are stored in a vector along with their position and the frame number where they appear. Every time a feature is lost another one is selected so there is always a constant number of tracked features.

The different vectors are stored in a plain text file following this schema:

Algorithm 1 Static scene detection

1: $motion \leftarrow$ **false**
2: **while** $motion =$ **true do**
3: $oldFeatures \leftarrow Features$
4: $Features \leftarrow obtainFeatures$
5: $dist2 \leftarrow dist1$
6: $dist1 \leftarrow dist0$
7: $dist0 \leftarrow summationDistance(oldFeatures, Features)$
8: **if** dist0 < MaxDifference **and** dist1 < MaxDifference **and** dist2 < MaxDifference **then**
9: $motion \leftarrow$ **true**
10: **end if**
11: **end while**
12: **return** STOPPED

```
Frame number ( positionX | positionY ) --- ...
```

Example:

```
Feature 0:
0 ( 517.000000 | 287.000000 ) ---  1 ( 516.986145 | 286.998535 ) ---
2 ( 516.854004 | 286.962128 ) ---  3 ( 516.703369 | 286.951660 ) ---
4 ( 517.176208 | 287.225830 ) ---  5 ( 513.131714 | 286.160919 ) ---
Feature 1:
0 ( 423.000000 | 269.000000 ) ---  1 ( 421.031799 | 263.996277 ) ---
Feature 2:
0 ( 475.000000 | 268.000000 ) ---  1 ( 474.983704 | 267.979004 ) ---
2 ( 473.535492 | 266.834412 ) ---
```

This file can later be read to visualize the trajectories of the features. Some of the features can have small position variations due to change of the lighting or to the image's noise. In order to avoid displaying these overlays the distance between the initial and the last point of a movement is compared with a threshold (overlay parameter); only those features whose movement is longer than the threshold are visualized. Experimental evaluation (see Section 5.3) showed that a good threshold is 25 (Figures 4.6 and 4.7).

4.4 Update Process

The update process is responsible for retrieving the previous scene representation, to compare it with the after motion information and to update it: that can improve the

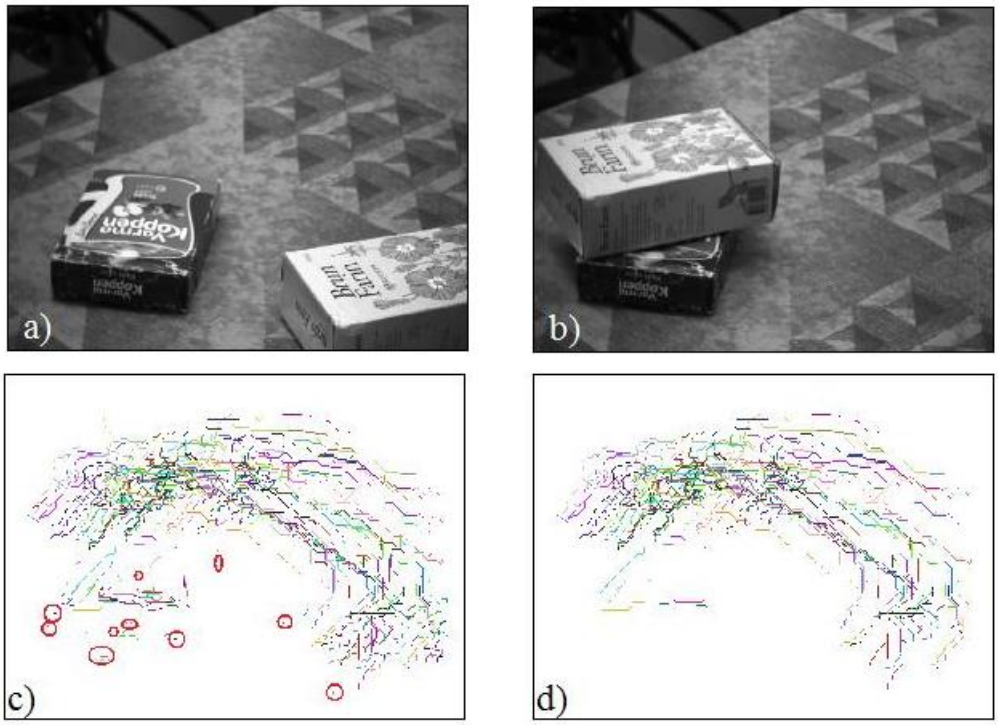

Figure 4.6: a) First frame of the image sequence. b) Last frame ofthe image sequence. c) Trajectories of the features without using a threshold (outliers marked in red) . d) Trajectories of the features using a threshold of 50.

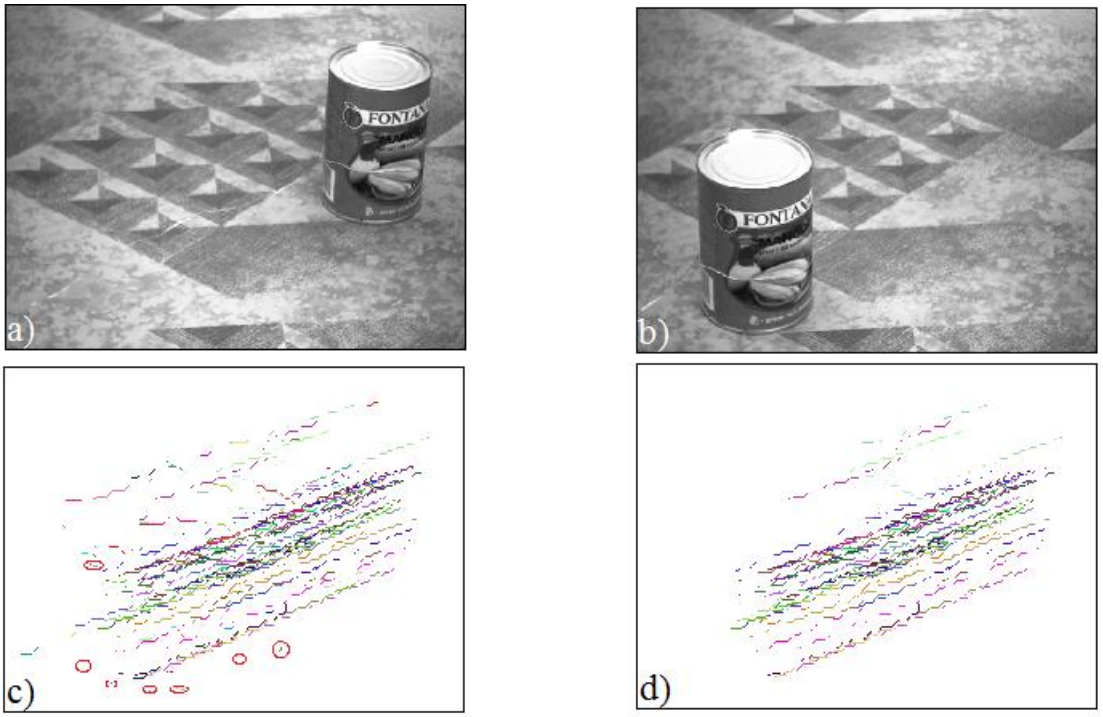

Figure 4.7: a)First snapshot of the image sequence. b) Last capture from the image sequence. c) Trajectries of the features without using a threshold (overlays marked in red) . d) Trajectories of the features after removing outliers.

given segmentation of the acquisition process, to add new properties to the database or merge the representation with the one stored in the scene memory.

4.4.1 Updating The Scene

When a change is detected in the scene, the representation of the scene must be updated. In order to achieve that, new data is required from the 3D-Frost (a new point cloud and a new prior segmentation). This information is used to build a completely new octree which is compared with the one stored in the scene memory. This comparison allows us to determine what was in the scene before, what is new and what has moved. Comparing both octrees we are able to realize what was in the scene before and what is new. This information can help to segment the different objects of the new scene in a better way and to keep track of previously segmented objects.

Comparing Two Octrees

One simple way of comparing two octrees is to use an XOR comparison. The system analyzes which regions of the old octree are overlapping the new one so we know what was in the scene before. However, when a scene changes it can produce or erase shadows; also the illumination conditions may vary. This can produce a significant change in the obtained point cloud causing alterations in the octree (even for objects that remained static) and making the node-by-node XOR comparison not suitable (e.g. the coordinates of the root node may change, making it impossible to directly equate both trees).

The comparison method we chose is a variation of the XOR function. Instead of comparing directly the node of an octree with its homonym, we check if the corresponding cube of a leaf node overlaps any part of the old octree. If that occurs it is considered that the node was already there before and its tagged with the same segment; otherwise we suppose that the node is new and we tag it with the segment detected by the 3D-Frost. An example of this method can be seen at Figure 4.8. To make the system more flexible and to reduce the noise influence we introduced a tolerance parameter into the comparison. As can be seen in Figure 4.9 this parameter increases (tolerance > 0) or decreases (tolerance < 0) the size of the nodes in order to consider that very close nodes are overlapping or to discard nodes that only share a small space of its volume.

4.4.2 Merging Scenes

Another possibility to update the scene is to merge different point clouds. If the stereo cameras are moved, resulting in a translation or a rotation, new information

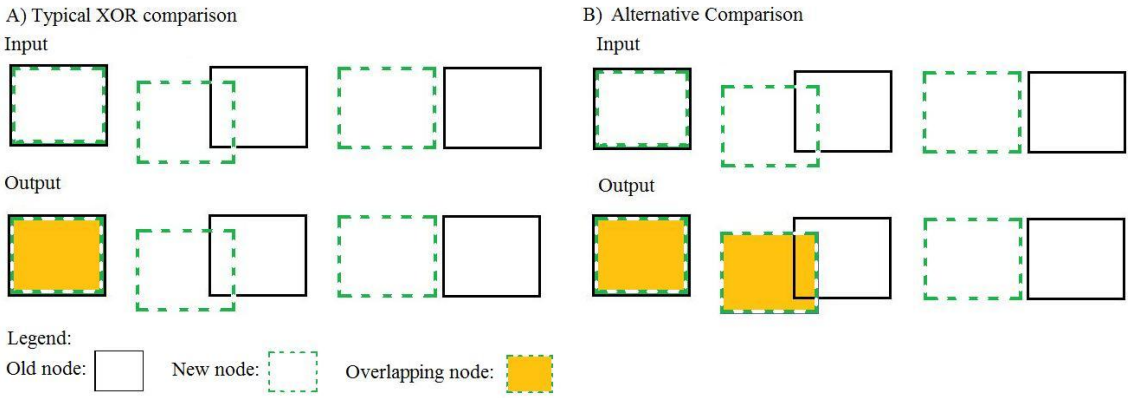

Figure 4.8: Comparison between the XOR (a) and its alternative (b)

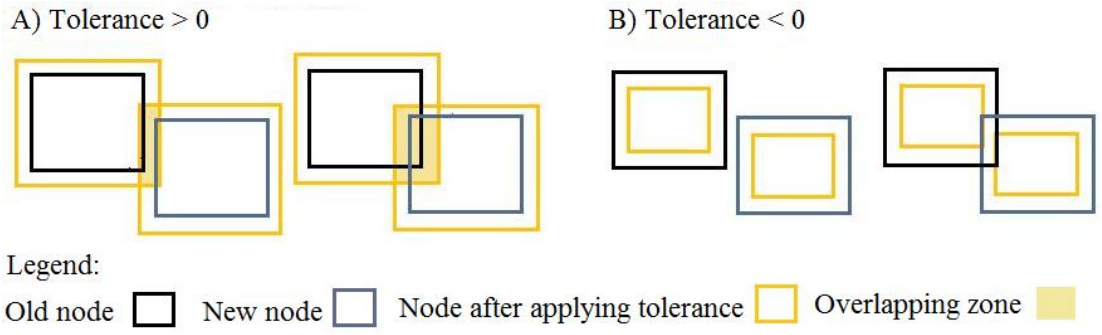

Figure 4.9: Effect of the tolerance parameter in the comparison algorithm. a) Tolerance parameter > 0. b) Tolerance parameter < 0.

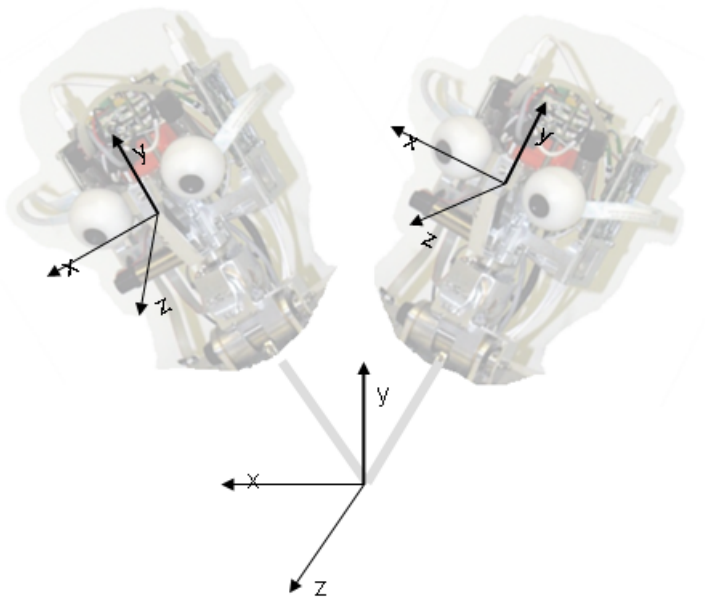

Figure 4.10: Different coordinate systems used by 3D-Frost. There is a coordinate system beetween the stereo cameras which varies depending on the head position. A static coordinate system is placed at the base of the head.

about the existing objects can be found and also new objects can appear in the scene. To simplify the problem, we assume that the scene is static while the robot is moving.

To merge two different scenes the most important thing to consider is the coordinate system of the point clouds. If both clouds share the same coordinate system the clouds can be merged as a single point cloud where the octree can be build from. However, if they use different systems it is necessary to change those systems to the same one so the octree can be build. The kinematics of the robot [1] are good enough to control its position and to estimate the movements that they have made. That allows us to know where the robot (and the cameras) are at any given moment facilitating the task of converting the coordinate systems.

3D-Frost can give a fixed coordinate system independently from the position of the robotic head. In consequence, we do not have to transform the point clouds obtained from the program. In Figure 4.10 we can observe the shared coordinate system. However, given the transformation between the coordinate frames in which two points clouds have been obtained, our system can transform them into a common coordinate frame. Depending on the kind of movement one or both of the following operations must be applied to every point of the point cloud:

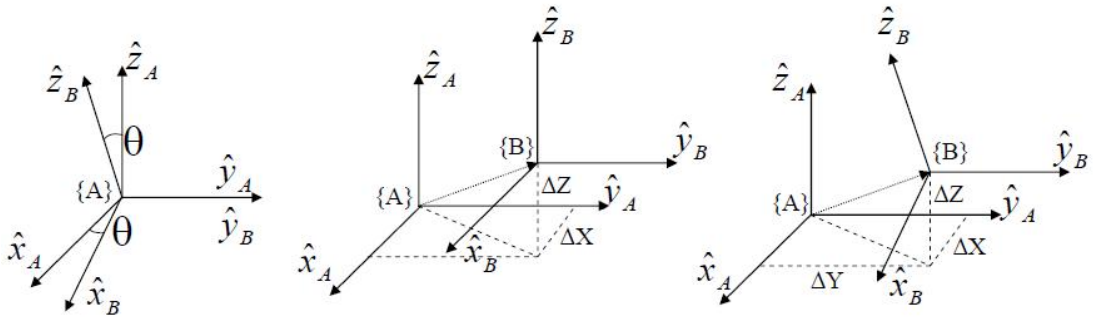

Figure 4.11: a)Rotation on the Y axis. b) Translation using the vector B. c) Combination of both operations. (source: Universitat de Girona)

Rotation

X axis rotation:

$$\begin{pmatrix} x' & y' & z' \end{pmatrix} = \begin{pmatrix} x & y & z \end{pmatrix} \times \begin{pmatrix} 1 & 0 & 0 \\ 0 & cos\theta & -sin\theta \\ 0 & sin\theta & cos\theta \end{pmatrix}$$

Y axis rotation (Figure 4.11 a):

$$\begin{pmatrix} x' & y' & z' \end{pmatrix} = \begin{pmatrix} x & y & z \end{pmatrix} \times \begin{pmatrix} cos\theta & 0 & sin\theta \\ 0 & 1 & 0 \\ -sin\theta & 0 & cos\theta \end{pmatrix}$$

Z axis rotation:

$$\begin{pmatrix} x' & y' & z' \end{pmatrix} = \begin{pmatrix} x & y & z \end{pmatrix} \times \begin{pmatrix} cos\theta & -sin\theta & 0 \\ sin\theta & cos\theta & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

Translation

Translation from point A to point B (Figure 4.11 b) :

$$\begin{pmatrix} {}^{A}p_x \\ {}^{A}p_y \\ {}^{A}p_z \end{pmatrix} = \begin{pmatrix} {}^{B}p_x \\ {}^{B}p_y \\ {}^{B}p_z \end{pmatrix} + \begin{pmatrix} {}^{A}O_x \\ {}^{A}O_y \\ {}^{A}O_z \end{pmatrix}$$

4.5 Properties Database

The *Properties Database* is in charge of storing different features for every segment. It is composed of a C++ class which stores the features that can be extracted from the point cloud and the octree. However, it is expected that in the future it will be replaced or linked in with the Attribute Action [16] software.

Besides the motion information, currently it is storing two kinds of features: color information and the volume of the segment.

Considering that every leaf node has a pointer to its correspondent points, color information can be obtained directly from the input point clouds. 3 vectors for every segment are created in order to store the RGB histograms.

The volume information is calculated from the Octree structure: the volume of all the nodes corresponding to a segment is added so an approximate value of the segment volume is obtained.

The different features can be consulted at any time as they are also stored in the XML file which describes the octree.

4.6 Storage

The main goal of the storage module is to keep the structure of the octree for later usage. To achieve that aim we decided to use an XML structure.

In the XML structure the path of the different point clouds is saved. The different features stored in the property database are also saved in the XML. Finally the octree is stored recursively under the 'Node' tag; inside this tag all its properties (position, segment, etc.) are stored and, if it is not a leaf node, eight new 'Node' tags saving the information of the children are created. In the following code a sample of the obtained XML file is shown:

```
<OCTREE>
  <PC>/media/Multimedia/ImgSequences/leopardCanStop/BMP/0.crd</PC>
  <PC>/media/Multimedia/ImgSequences/leopardCanStop/BMP/1.crd</PC>
  <Node>
    <Region>--</Region>
    <Name>Dad</Name>
    <Size>
      <P0>-81.500420,-143.672073,-1260.933838</P0>
      <P1>172.100540,1.023830,-891.022400</P1>
    </Size>
    <Leaf>false</Leaf>
    <Node>
```

```
      <Region>--</Region>
      <Name>DadChild0</Name>
      <Size>
        <P0>-81.500420,-143.672073,-1260.933838</P0>
        <P1>45.300060,-71.324120,-1075.978149</P1>
      </Size>
      <Leaf>false</Leaf>
      <Node>
        ...
      </Node>
    </Node>
    <Node>
      ...
    </Node>
    ...
  </Node>
  <Properties>
    <colors>
      <Segment>
        <ID>0</ID>
        <histoR>12, 43, 34, 34, 43,73, 13</histoR>
        <histoG>12, 10, 42, 11, 34,48, 13</histoG>
        <histoB>12, 0, 48, 45, 43,73, 13</histoB>
      </Segment>
      <Segment>
        <ID>1</ID>
        ...
      </Segment>
      ...
    </colors>
    <Volume>
      <Segment>
        <ID>0</ID>
        <Volume>230</Volume>
      </Segment>
      ...
    </Volume>
  </Properties>
</OCTREE>
```

Chapter 5

Experiments and Results

In this chapter different experiments are carried out to test the behavior of the parts of the system given different parameter setting. After the applied hardware is presented in the first section, the representation itself, the motion tracking and the update of the model are evaluated.

5.1 Used Hardware

All the tests have been done using a Dell laptop; however, the acquisition process has been made using a special robot head [1] endowed with stereo cameras. Laptop characteristics can be found on Table 5.1.

Computer Model	Dell XPS m1330
Processor	Intel Core2 Duo T8300 @2.40 GHZ
Hard disk	160 GB
RAM memory	3GB
Graphic card	GeForce 8400M 512MB
Operating system	Ubuntu 8.10

Table 5.1: Features of the used computer

Characteristics of the Kinematic head:

- Seven degrees of freedom: four in the neck (neck pan, neck tilt, neck yaw and head tilt) and thre in the eyes (a common tilt and an independent pan for each eye).
- Baseline: 46.5 * 2 mm

- Two stereo cameras: one foveal and one peripheral.
- Right foveal camera focal length: 2226.56 pixels.
- Left foveal camera focal length: 2224.85 pixels.
- Right peripheral camera focal length: 527.6 pixels.
- Left peripheral camera focal length: 525.663 pixels.

5.2 Representation Tests

In this section the representation system has been checked. In order to achieve that, different kinds of scenes have been represented. First of all we have tested artificial and clean point clouds so we could have a first impression about how the system works in an ideal environment. Then experiments were done using real data, this involves that some noise appears on the images and external factors, such as light, affect their quality.

5.2.1 Building Synthetic Scenes

To test the performance of the representation process and to evaluate which parameters are the best to represent a point cloud, experiments with artificial point clouds have been realized. These point clouds have no fuzzy zones or noise so its representation should be very accurate. Apart of the accuracy, also the needed time to build the model has been evaluated.

Homer Model

The first point cloud that has been tested is a 3D representation of the cartoon character Homer Simpson (see Figure 5.1). This point cloud is composed of 5616 points, which is a sufficient amount of points to represent an object; however, in real images this amount is usually higher so we could say that this is a light point cloud.

Octree

To evaluate the performance of the octree a model of the point cloud was built using different depth parameters (from 3 to 8). Figure 5.2 shows the output of the experiment; Table 5.2 displays the required time to build the representation.

In this experiment we can see that, as we expected, the octrees with less deepth have less accuracy. However, it is interesting to notice that due to the low density

Figure 5.1: Homer Simpson's point cloud

of the point cloud when a too high depth parameter is chosen (Figure 5.2 e/f) the cubes of the octree become too small and a big part of the solid is not represented by the octree. In consequence, we could say that the best representations considering the accuracy is the one with a depth of 6 (Figure 5.2 d) and the best considering the space occupancy is the one with a depth of 5 (Figure 5.2 c). Referring to process time, obviously, the tests with a lower deepth need less time to be built. Although any model needed more than 4 seconds to be built, it is important to remember that the point cloud used was very light.

Depth parameter	Time needed (ms)	Depth parameter	Time needed (ms)
3	55	6	1280
4	196	7	2180
5	584	8	3112

Table 5.2: Time execution for the Homer Simpson model using an octree representation

Hierarchical Bounding Boxes

The same experiment has been done for the alternative representation method: Hierarchical Bounding Boxes. Unlike the first experiment, there are no parameters to check so just one test has been done. To evaluate the processing time we recorded

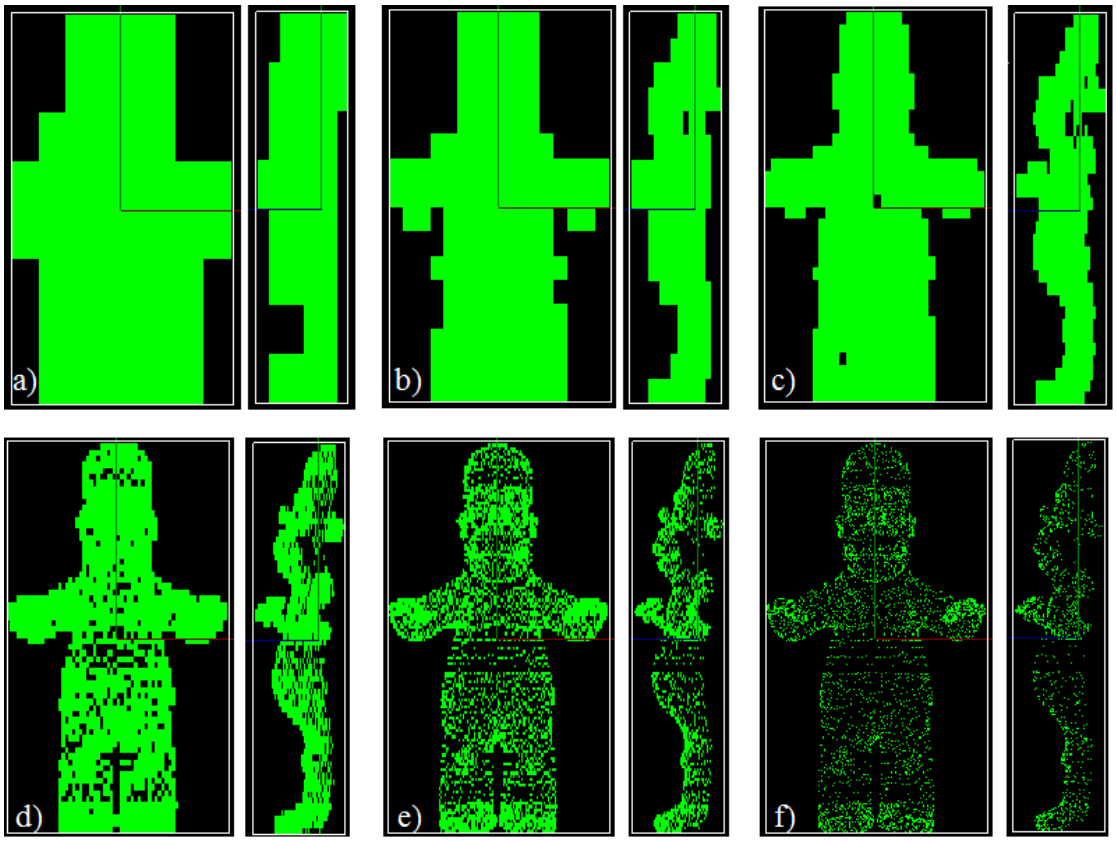

Figure 5.2: Octree representation of the Homer Simpson's point cloud using different depth parameters a) 3 b) 4 c) 5 d) 6 e) 7 f) 8

the period needed for the BoxGrasping framework [17] to create the XML describing the HBB and then recording the time needed for our system to load it.

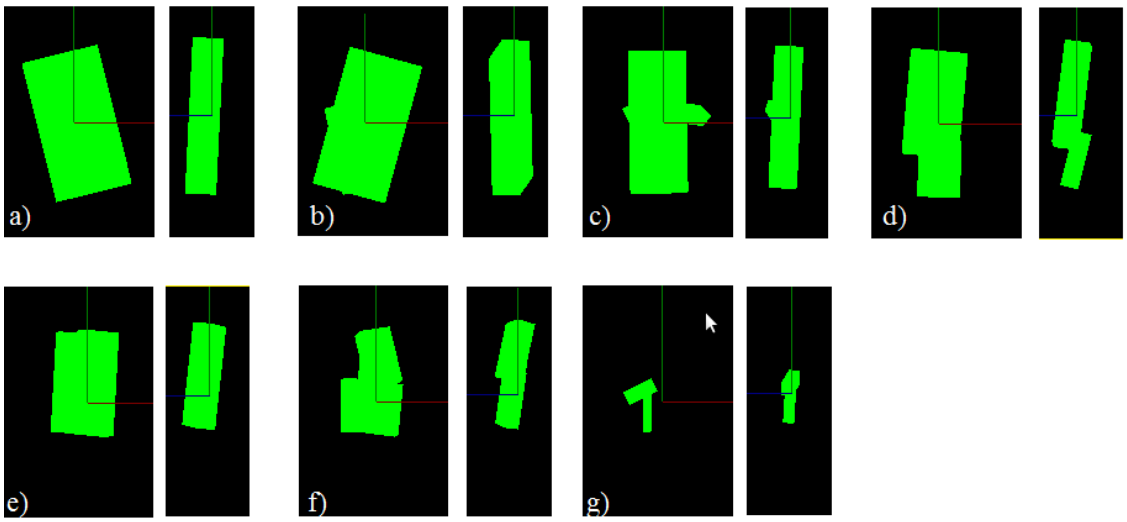

Figure 5.3: HBB representation for the Homer Simpson's point cloud. Every figure represents a level in the hierarchy. a) Root level b) Level 1. c) Level 2. d) Level 3. e) Level 4. f) Level 5. g) Level 6.

Box decomposing time (ms)	XML loading time (ms)	Total time (ms)
31411	205	31616

Table 5.3: Time execution for the Homer Simpson model using an HBB representation

Looking at the results shown at Figure 5.3 the first thing we notice is that the representation is not as accurate that the one obtained with the alternative representation method: e.g. at Figure 5.3 f we can identify the box that corresponds to the head, however, the representation of the head is not as precise as the one seen at the top of Figure 5.2 d. Regarding the execution time we can observe in Table 5.3 that it is much higher than the time need by the slowest octree representation (31616 milliseconds against 3112).

Bird Model

The second point cloud that has been tested is a 3D representation of a bird (see Figure 5.4). In contrast to the Homer point cloud, this one has a great quantity of points: 57987. This number is closer to the ones obtained by the stereo cameras so this test gives us an idea about the time needed to represent a real object.

Octree

As with the Homer model, the bird model has been evaluated in terms of depth and time. The results are shown in Table 5.4 and in Figure 5.5.

Similar to the previous test, the representation with the highest depth had some unrepresented regions due to the smaller size of the cubes; however, this only happens with a depth of 8 due to the higher density of the point cloud. For the same reason we can observe that the time needed to build the representations increased significantly making the representations of a level higher than 5 too slow. In consequence we could say that the best performance of this test has been achieved with a depth of 5 (Figure 5.5 c) as the accuracy achieved is quite similar to the highest ones but with much less time needed (less than 7 seconds).

Depth parameter	Time needed (ms)	Depth parameter	Time needed (ms)
3	696	6	25414
4	2176	7	91586
5	6869	8	200312

Table 5.4: Time execution for the bird model using an octree representation

Hierarchydal Bounding Boxes

The representation of the bird point cloud using HBB can be seen in Figure 5.6. As happened before, the accuracy of the model is worse than the one obtained with the octree. Nevertheless, we can observe that in the current experiment the representation is more similar to the real object than the one obtained with the Homer model. This is probably due simple structure of the figure.

We can observe in Table 5.5 that the execution time is quite similar to the previous one (see Section 5.2.1); considering that the number of points in the point cloud is significantly higher than in the previous one, we can assume that the time needed to build this representation is not strongly related with the size of the point cloud. However, it is still slower than the octree representation using a deepness of 6.

Box decomposing time	XML loading time	Total time
31012	196	31108

Table 5.5: Time execution for the bird model using an HBB representation

5.2.2 Building Real Scenes

In this section the representation system is tested with a real point cloud. In the previous experiments we saw what the system's behavior in an ideal environment

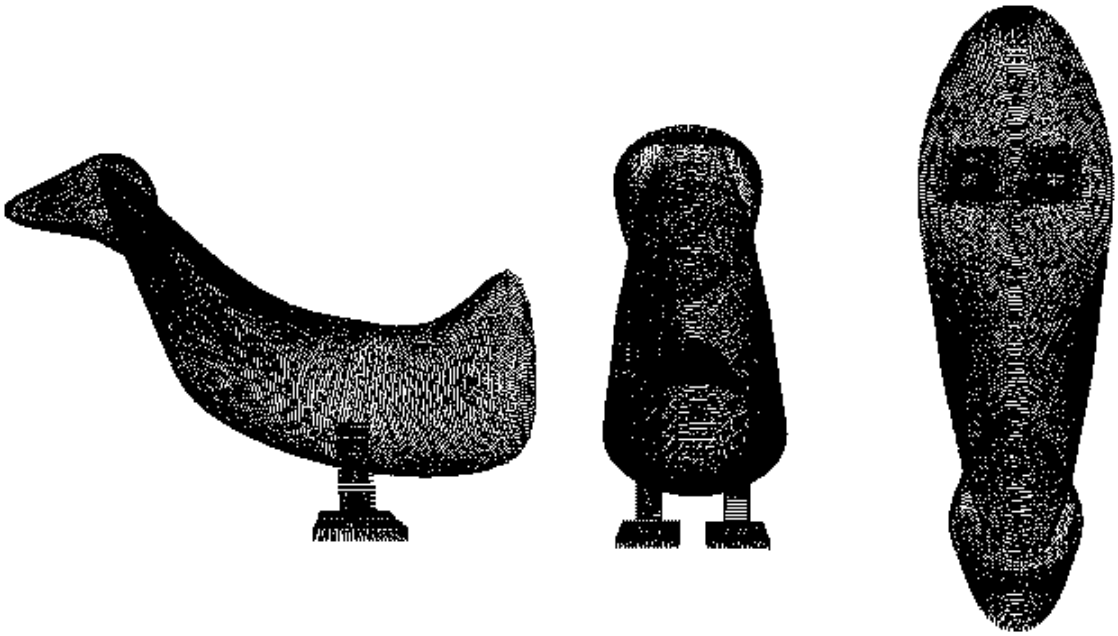

Figure 5.4: Bird's point cloud

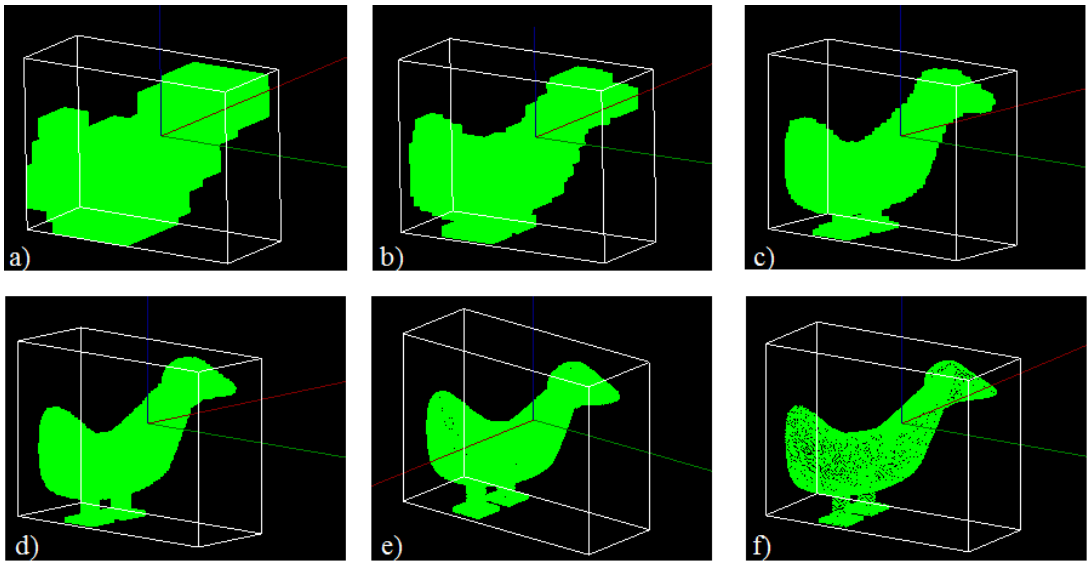

Figure 5.5: Octree representation for the bird's point cloud using different depth parameters a) 3 b) 4 c) 5 d) 6 e) 7 f) 8

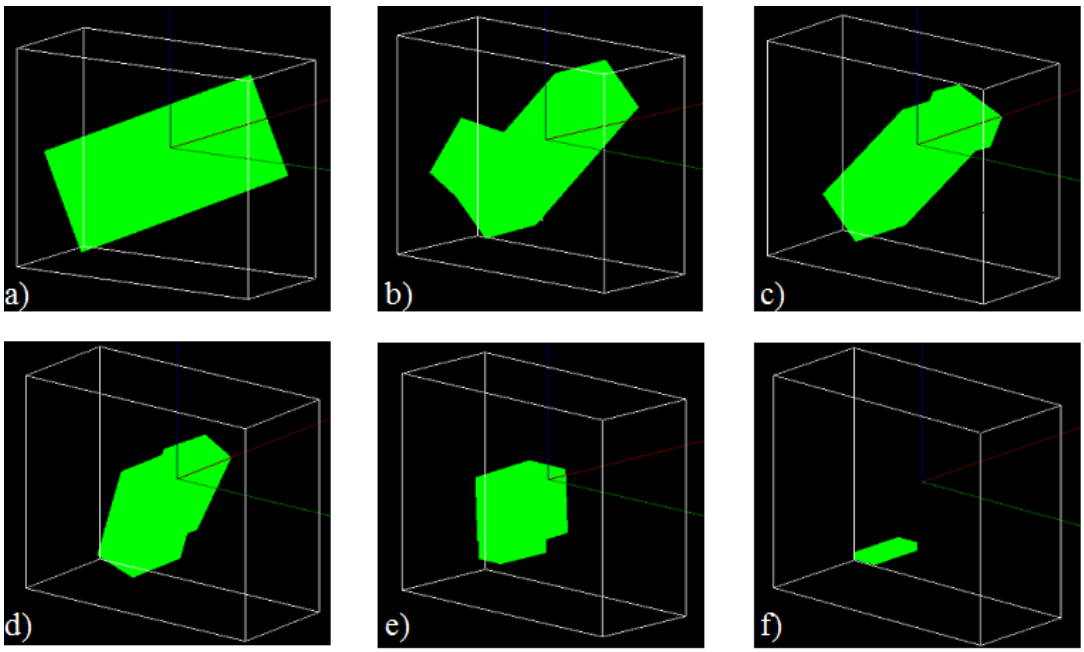

Figure 5.6: HBB representation for the bird's point cloud. Every figure represents a level in the hierarchy. a) Root level b) Level 1. c) Level 2. d) Level 3. e) Level 4. f) Level 5.

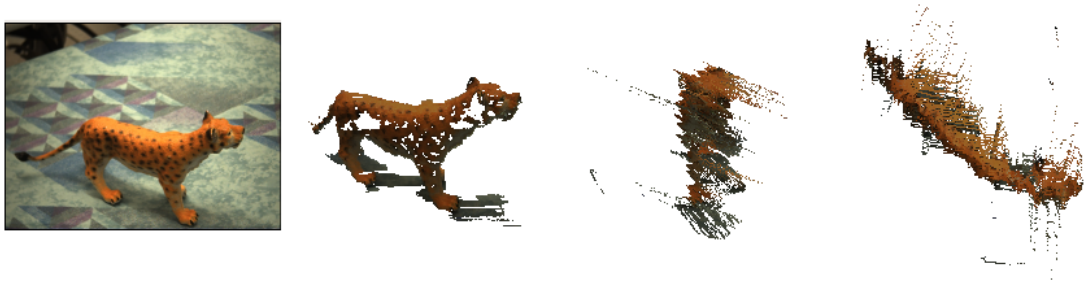

Figure 5.7: Point cloud obtained from a toy leopard seen from different viewpoints

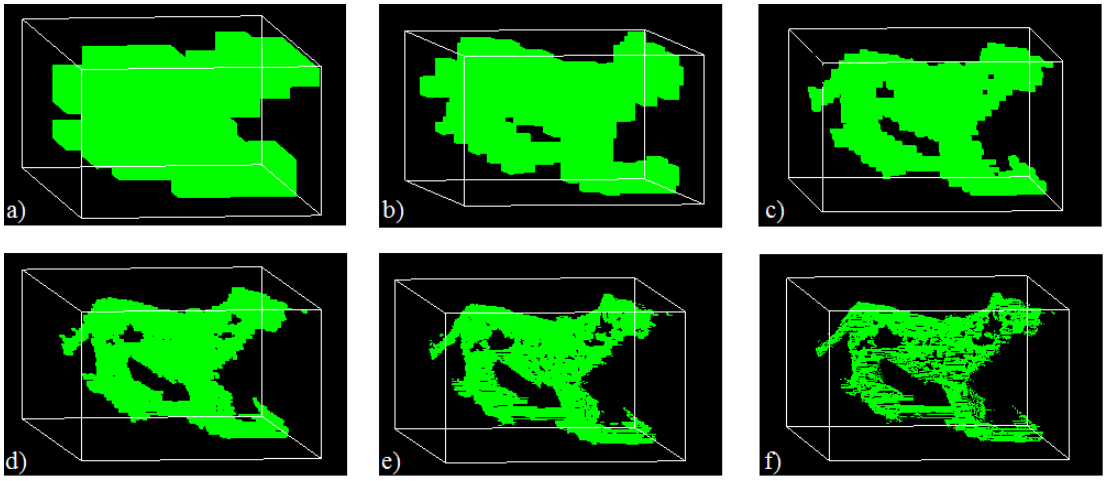

Figure 5.8: Octree representation for the leopard point cloud using different depth parameters a) 3 b) 4 c) 5 d) 6 e) 7 f) 8

is; in contrast, in the following set we can check the performance of the system with real scenes. The images used to do those experiments had some noise and not a perfect illumination. As was done above, the results are evaluated in terms of time and accuracy.

Tiger Model

The point cloud used in this test is the representation of a leopard toy. It was obtained from the stereo head using the 3D-Frost [30] program and it consists of approximately 40000 points. In Figure 5.7 different points of view of the point cloud can be found.

Octree

As seen in the previous experiments, the octree representation offers a good model of the scene using almost any depth level (Figures 5.8 c-f) . However, due to the lack of information in certain regions of the point cloud, some holes appear in the structure. This does not happen using a depth of 4 (Figure 5.8 b): the bigger size of the boxes makes the representation less precise, however, it fills the mismatching region and the silhouette of the leopard is still recognizable. The time needed to build the model by the different depths can be found in Table 5.6.

Depth parameter	Time needed (ms)	Depth parameter	Time needed (ms)
3	456	6	9737
4	1068	7	29746
5	2988	8	72621

Table 5.6: Time execution for the leopard model using an octree representation

Hierarchical Bounding Boxes

Comparing the output of this experiment with the previous HBB representations we can see that the obtained model looks more like the real scene. Nevertheless, is still far from being as comprehensive as the octree; although we can clearly distinguish parts of the leopard such as the head (Figure 5.9 f) or the front legs (Figure 5.9 d), other parts are impossible to perceive (e.g. the tail). Regarding the execution

Box decomposing time (ms)	XML loading time (ms)	Total time(ms)
29052	220	29328

Table 5.7: Time execution for the bird model using an HBB representation

time, again we can see that it does not vary with the size of the point cloud. Moreover, as in the other experiments, the execution time is higher than the needed by the octree with a depth level of 6.

5.2.3 Summary

In this section we tested the performance of both the octree and the HBB representation systems. Concerning the accuracy the best performance has been achieved with the octree using depths parameters between 4 and 6; using a higher deepness results in a lack of information in some zones of the model due to empty regions in point clouds. Using a lower depth level makes the model too imprecise and not recognizable. The representation of the octree is much more precise and simple than the one obtained with the HBB.

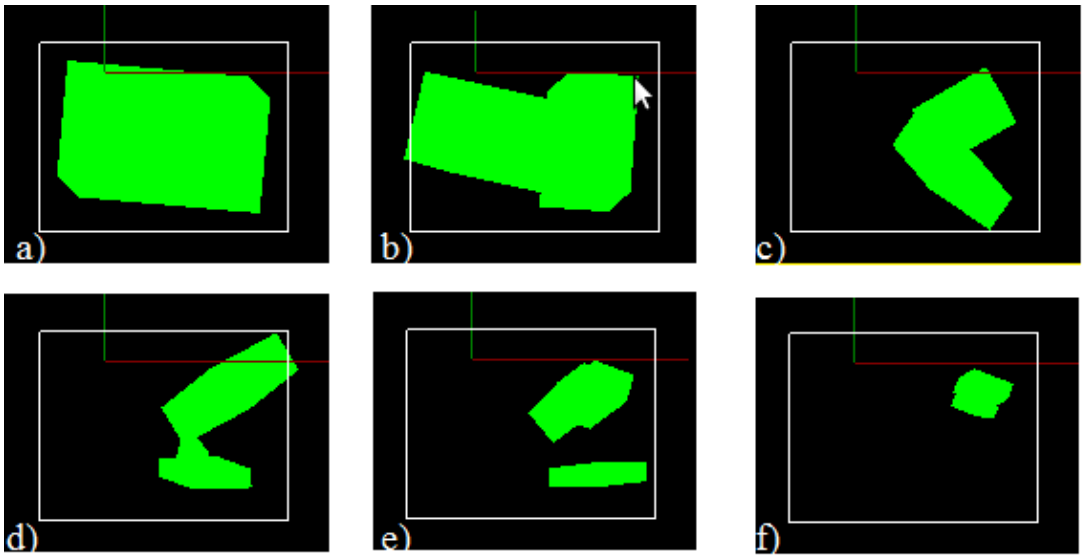

Figure 5.9: HBB representation for the Leopard point cloud by levels. a) Root b) Level 1 c) Level 2 d) Level 3 e) Level 4 f) Level 5

Referring to execution time evaluation, we can see that the time needed for the HBB does not depend on the size of the point cloud while the time needed by the octree depends on the depth parameter and on the size of the point cloud. However, in all the realized experiments the time needed for octrees with a depth lever lower than 7 has been lower than the time needed by the HBB.

Therefore, analyzing the results of the previous experiments and some others that have not been included on the report, we estimated that the best representation method for our purpose is the octree with a depth level between 4 and 6. In consequence, this is the system that has been used to represent scenes and update its information.

5.3 Motion Tracking Tests

In this section, we evaluate the motion tracker. Both the feature tracker and the motion detection system have been tested. In the experiments only the overlay parameter can be varied (see Section 4.3.2), so it is also discussed which is the best value for this parameter.

5.3.1 Sequence 1: Simple Movement

This is the simplest experiment we have done because there is just one object in the scene. The image sequence is composed of 28 snapshots, showing a can moved by a wire. The can starts to move in the first frame and it stops in the 30^{th}. In Figure 5.10 some snapshots of the sequence can be seen. The experiment has been repeated with 4 different values for the overlay parameter: 0, 25, 50 and 100. In Figure 5.11 the results are displayed.

The result of the experiment is very good as the trajectory of the can is clearly registered by the feature tracker. Moreover, the motion detector detected correctly that the can stops at the 30^{th} frame. In Figure 5.11 a we can observe the output without filtering the overlays while in Figure 5.11 b-d overlays disappeared.

5.3.2 Sequence 2: Multiple Objects Moving

In this experiment, as done above, a can is moved with a steel string. However, unlike the first one, there are more objects in the scene and one of them is moved due to the translation of the can (Figure 5.12). Opposing the previous experiment, in this one there is motion during the whole scene.

The motion detector detected that there is movement during the entire scene. Regarding the overlay parameter: in Figure 5.13 b the movement and the rotation of the second can be easily seen; however, this movement cannot be appreciated in Figures 5.13 c-d because the value of the parameter is too high.

5.3.3 Sequence 3: Fast Moving Objects

In this experiment a box is moved with hands. Unlike the previous tests, here an external object appears into the scene: a human hand. In this sequence the movement stops at the 20^{th} frame.

Although the detector noticed correctly that the scene becomes static at the 20^{th} snapshot, the results of this experiment are quite bad. The output of the tracking gives a very poor information about what happened in the scene; in Figure 5.15 is possible to understand that something has been moved in the left and in the right part of the image but there is no way to see the trajectory of the objects. If we observe Figure 5.14 we can notice that the box was moved too fast since when the box is moved its textures become blurred and the features are lost. This loss makes impossible to track the objects.

5.3.4 Sequence 4: Slow Moving Objects

The last experiment of this section is similar to the third one. As was done in the previous experiment, a box is moved with hands but this time slower. As happened before human hands appear in the scene. In Figure 5.19 we can see that this time the image remains clear so the features do not get.

In Figure 5.17 we can observe how this time the box has been tracked satisfactorily as the features did not disappear when the object was moved. It is also possible to see that the hands have not been tracked since they are not textured enough to be followed. Regarding the overlays parameter we observe that a value of 25th is enough to eliminate the overlays.

5.3.5 Summary

This set of experiments allowed us to make sure that the feature tracker and the motion detector are working correctly. Moreover it has been shown that to obtain good information about the trajectory of the objects, they have to move slowly to avoid motion blur. We also tested the overlay parameter, the results show that a value of 25 is enough to eliminate outliers from the output image. However, as seen in Section 5.3.2, if this value is too high (50 or 100) we can loose important information about the trajectory of the objects.

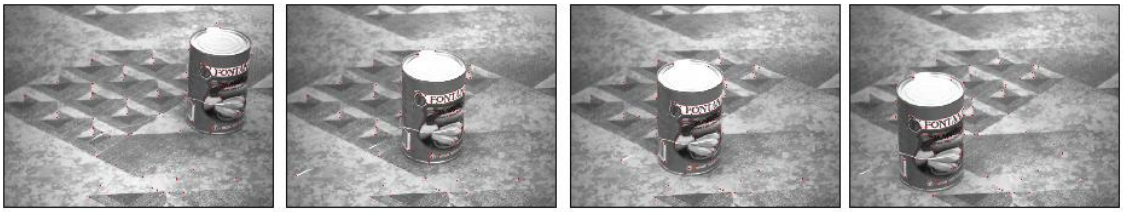

Figure 5.10: Image sequence used in experiment 1.

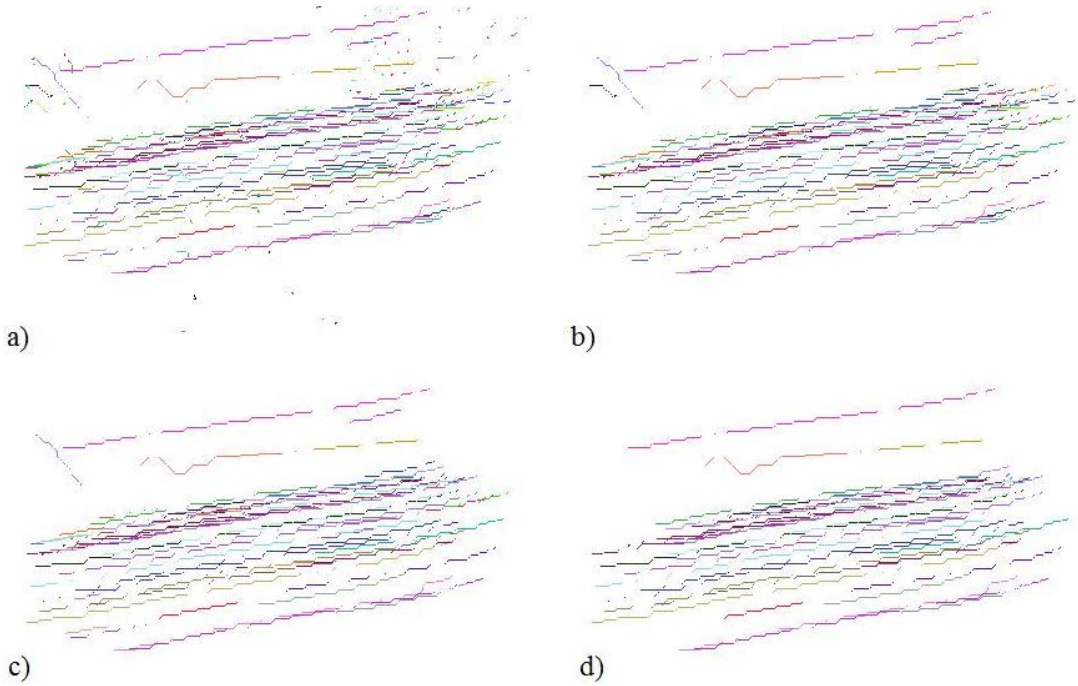

Figure 5.11: First tracking experiment with a) Overlay parameter = 0. b) Overlay parameter = 25. c) Overlay parameter = 50. d) Overlay parameter = 100.

Figure 5.12: Image sequence used in experiment 2.

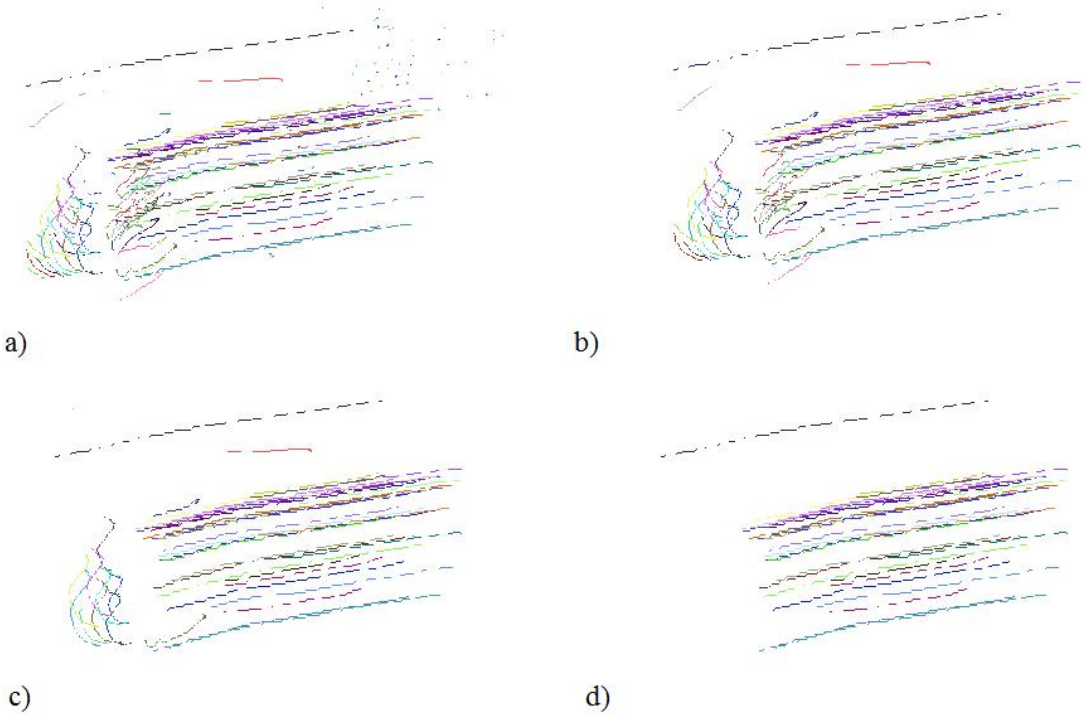

Figure 5.13: Second tracking experiment a) Overlay parameter = 0. b) Overlay parameter = 25. c) Overlay parameter = 50. d) Overlay parameter = 100.

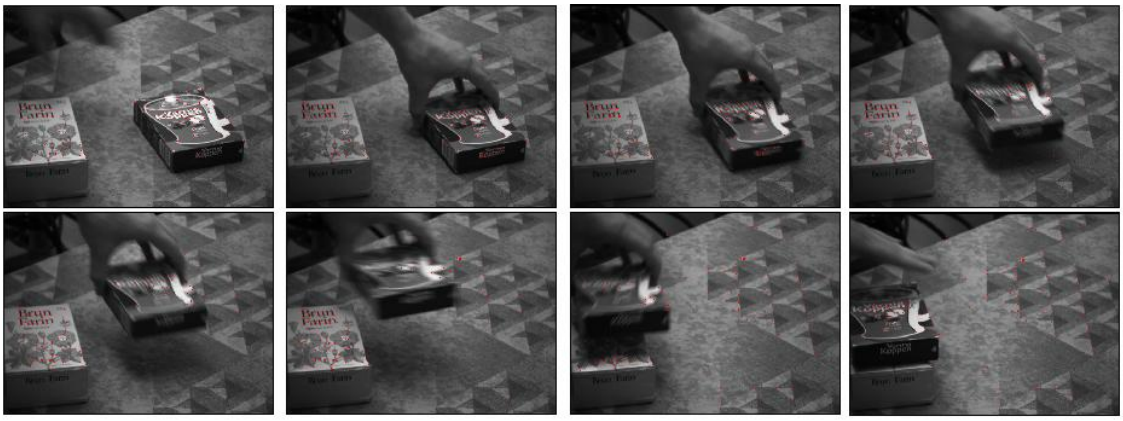

Figure 5.14: Image sequence used in experiment 3.

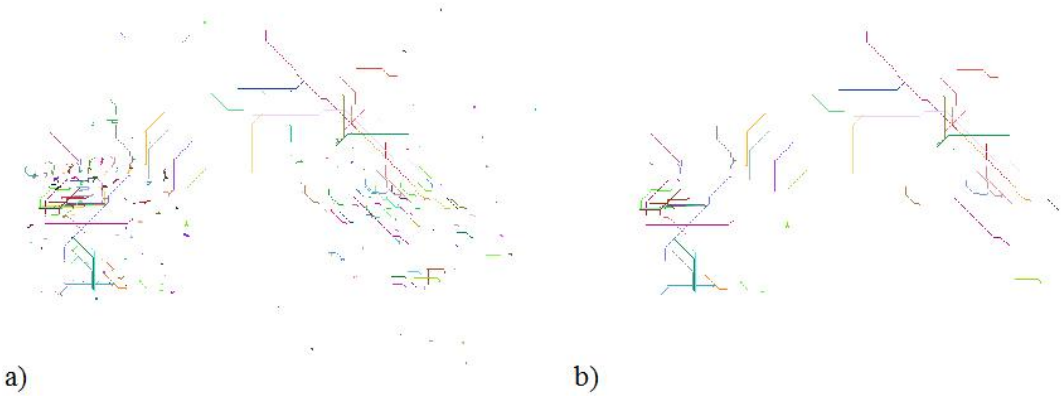

Figure 5.15: Third tracking experiment a) Overlay parameter = 0. b) Overlay parameter = 25.

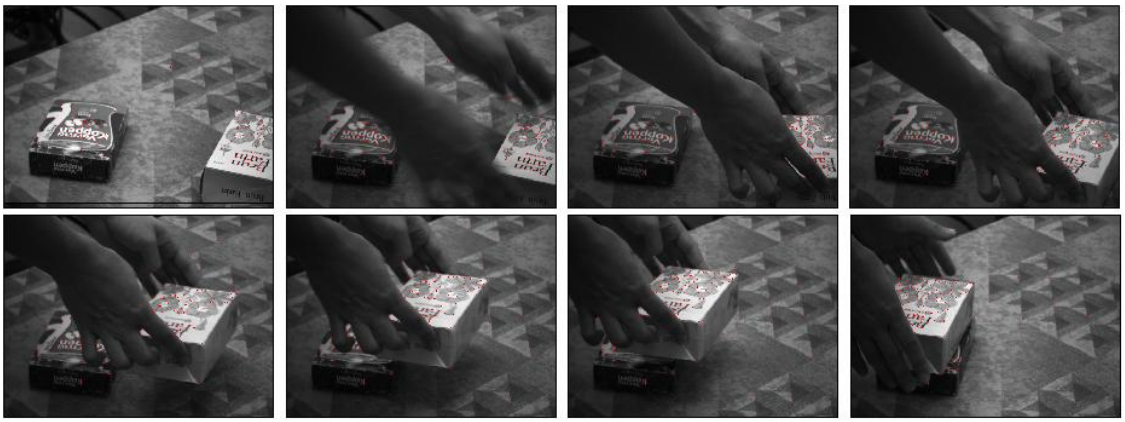

Figure 5.16: Image sequence used in experiment 4.

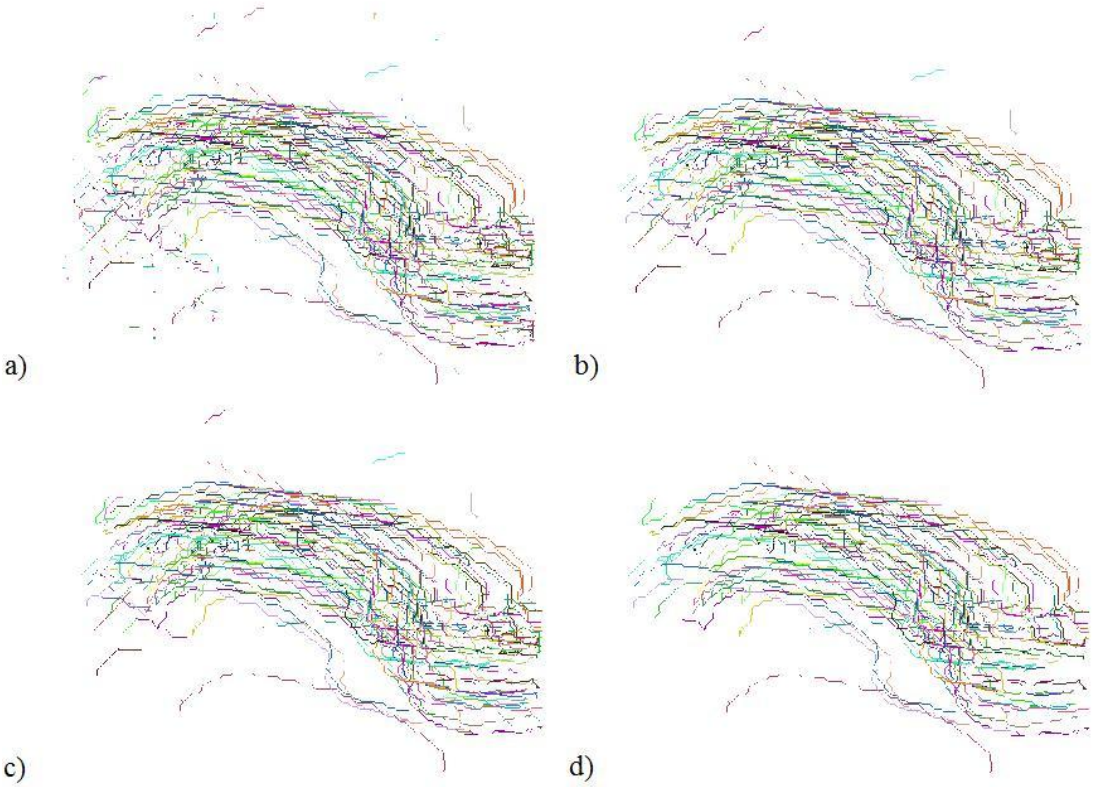

Figure 5.17: Fourth tracking experiment a) Overlay parameter = 0. b) Overlay parameter = 25. c) Overlay parameter = 50. d) Overlay parameter = 100.

5.4 Updating the Scene

In this section the scene updating is tested. Different experiments are performed in order to evaluate each lind of update: changes in the scene and changes in the camera viewpoint.

5.4.1 Changes in the Scene

As discussed on Section 4.4.1, when changes are detected in a scene its octree representation must be updated. In this section, three different scenes are analyzed and their parameters are tested in order to evaluate them and decide which are the best ones.

Boxes and Can Experiment

In the first experiment a can placed above a box (Figure 5.18 a) is moved on top of another box (Figure 5.18 b) . As can bee seen in Figure 5.18 c, the prior segmentation we obtain from the 3D-Frost detects two elements on the scene while there are three. We performed the experiment using values between 4 and 6 for the depth parameter

Figure 5.18: a) Initial position of the scene. b) Final position of the scene. c) Prior segmentation (Best seen in Color)

and values 0, 5 and -2 for the tolerance.

The results of this experiments shows that with the tolerance parameter set to zero the system is able to separate the can from the box in its segmentation independently from the depth (Figures 5.19 a and 5.20 a) although the experiment with a large depth detected more false positives. It is also interesting to notice that in the second box the region where the can was placed initially is detected as a new segment. This happens because this region was not visible in the first image. We also observe that the performance of the segmentation is better with a depth parameter of four and five, probably due to the smaller size of the octree nodes when a high precision is used.

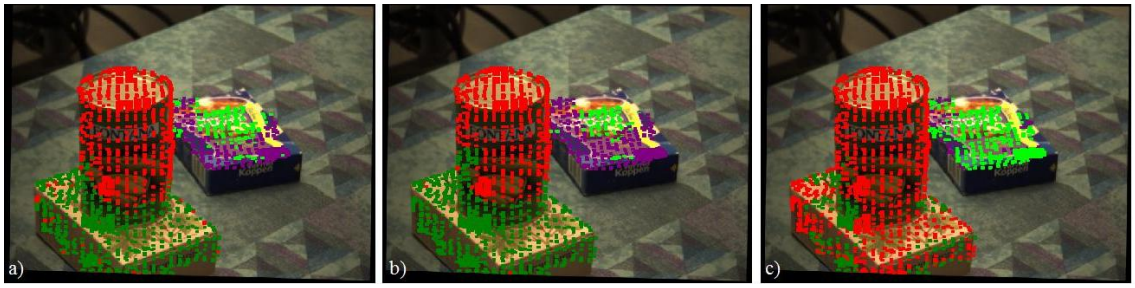

Figure 5.19: Boxes and Can experiment: depth 4. a) Tolerance = 0. b) Tolerance = 5. c) Tolerance = -2. (Best seen in Color)

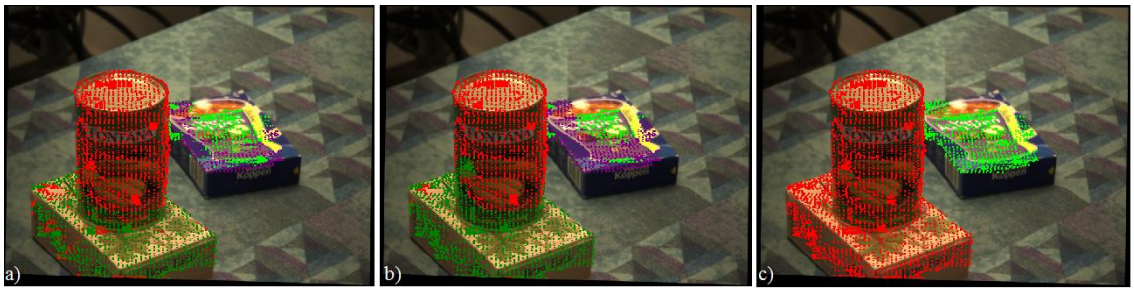

Figure 5.20: Boxes and Can experiment: depth 5. a) Tolerance = 0. b) Tolerance = 5. c) Tolerance = -2. (Best seen in Color)

Figure 5.21: Boxes and Can experiment: depth 6. a) Tolerance = 0. b) Tolerance = 5. c) Tolerance = -2. (Best seen in Color)

Figure 5.22: a) Initial position of the scene. b) final position of the scene. c) Prior segmentation

Regarding the tolerance parameter we can see at Figures 5.19 b, 5.20 b and 5.21 b tat the use of a positive tolerance improves the detection of the objects that where in the scene previously (the yellow box is better defined), in contrast we can notice that the border of new objects (e.g. the can) are confused with the older ones. In the other side, when a negative tolerance is used (Figures 5.19 c, 5.20 c and 5.21 c) the system is not able to distinguish between the new and the old objects; this phenomenon is more accentuated when the depth parameter is higher.

Can and Leopard Experiment

In the second experiment of the section a can is moved closer to a leopard toy (Figure 5.22 a, b). The prior segmentation we obtain just detects one object in the scene (Figure 5.22 c). As above, the experiment is executed with depth parameters of 4, 5 and 6 and with tolerance values of 0, 5 and -2.

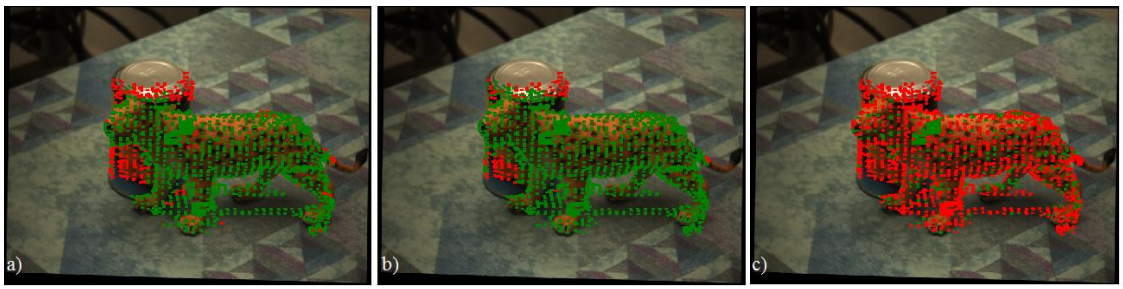

Figure 5.23: Can and Leopard experiment: depth 4. a) Tolerance = 0. b) Tolerance = 5. c) Tolerance = -2. (Best seen in Color)

The results we obtain in this experiment are quite similar to the previous ones. With a tolerance value of 0 the can and the leopard are well segmented with a depth of 4 and 5 (Figures 5.23 a and 5.24 a) while the segmentation with a depth of 6

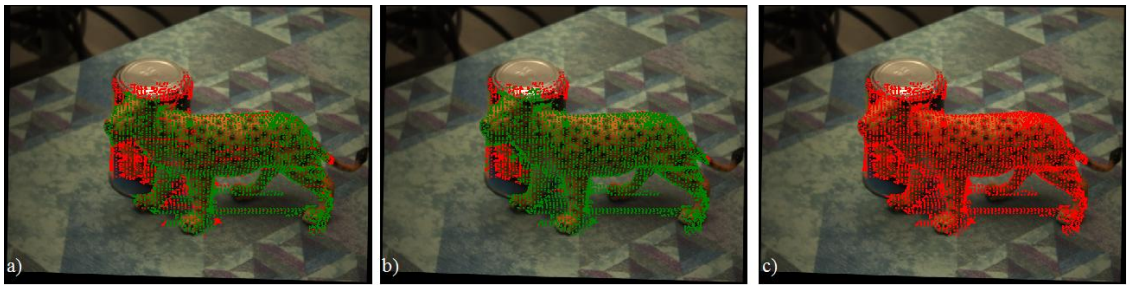

Figure 5.24: Can and Leopard experiment: depth 5. a) Tolerance = 0. b) Tolerance = 5. c) Tolerance = -2. (Best seen in Color)

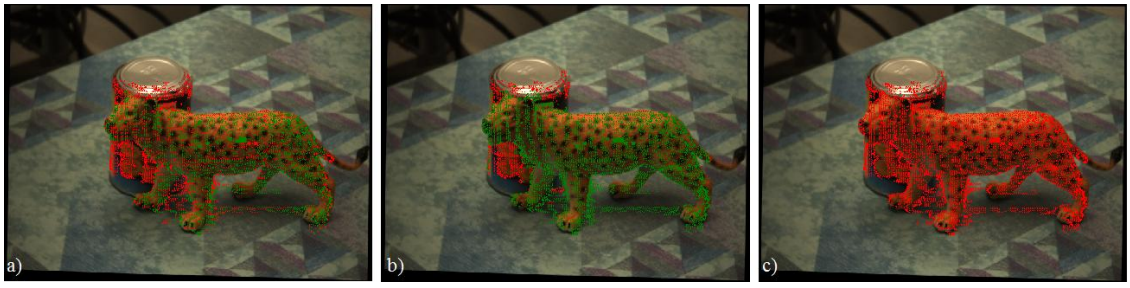

Figure 5.25: Can and Leopard experiment: depth 6. a) Tolerance = 0. b) Tolerance = 5. c) Tolerance = -2. (Best seen in Color)

(Figure 5.25 a) is a bit fuzzy and classifies some parts of the leopard in the same segment as the can. Moreover, when the tolerance is set to -2 (Figures 5.23, 5.24, 5.25 c) the scene is not well segmented.

Unlike in Experiment 5.4.1, when the tolerance value is 5 the performance of the segmentation is improved; this can be clearly seen in Figure 5.25 b. Probably this is because the leopard was slightly moved when the can was put behind the toy, causing a small variation in the leopard representation that is compensated with the tolerance margin.

Boxes Experiment

Figure 5.26: a) Initial position of the scene. b) final position of the scene. c) Prior segmentation

In the last experiment of the section we put one box on top of another one (Figure 5.26). The initial segmentation from 4D-Frost detects 5 segments, however, it groups both boxes under the same segment; the reason why there are 5 segments is the high number of textures that the yellow box has. As in the previous experiments, the depth parameter has been set to 4, 5 and 6 and the tolerance to 0, 5 and -2.

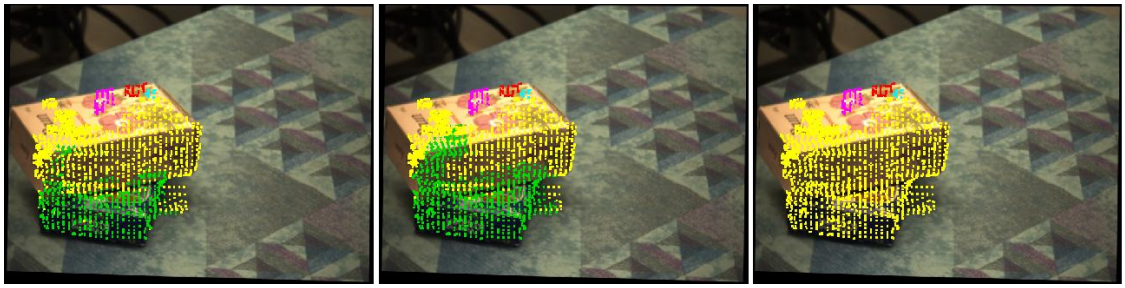

Figure 5.27: Boxes experiment: depth 4. a) Tolerance = 0. b) Tolerance = 5. c) Tolerance = -2. (Best seen in Color)

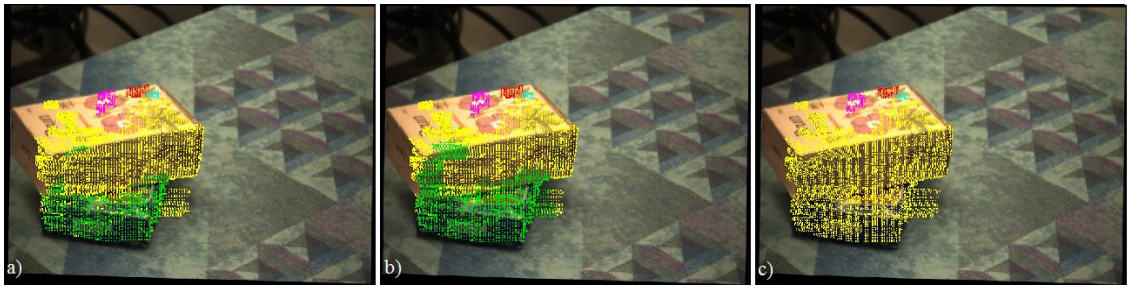

Figure 5.28: Boxes experiment: depth 5. a) Tolerance = 0. b) Tolerance = 5. c) Tolerance = -2. (Best seen in Color)

Figure 5.29: Boxes experiment: depth 6. a) Tolerance = 0. b) Tolerance = 5. c) Tolerance = -2. (Best seen in Color)

As expected, regardless of the depth parameter, both boxes have been segmented in different groups (Figures 5.27, 5.28 and 5.29 a); it is remarkable that in this experiment even with the depth parameter set to 6 the boxes have been well separated. However, when the tolerance parameter is 5 some parts of the yellow box are grouped in the same segment as the blue one. As happened in all the previous experiments, when the tolerance parameter is negative (Figures 5.27, 5.28 and 5.29 c) the segmentation fails.

Summary

This section showed the performance of the scene representation update after changes inside the scene. We noticed that the best depth parameters are 4 and 5. When the parameter exceeds these values the boxes of the octree become smaller and they need a higher tolerance to segment the objects, causing more inaccuracies.

We also observed that, usually, the tolerance parameter is not necessary; however, as can be seen at Experiment 5.4.1, if the results of the segmentation are not good an increase of the tolerance parameter can improve the results of the segmentation. In addition, we detected in all the experiments that when the tolerance is negative the updating system does not obtain good results, identifying different objects as the same one.

5.4.2 Viewpoint Changes

In this section we evaluate the performance of the system when the stereo cameras change their position. When the robotic head moves, the obtained scene is merged with the previous one. We present two different experiments, one where there is one object in every scene and another where an object appears in both scenes. As mentioned above, 3D-Frost offers a point cloud based on a coordinate system placed on the base of the robotic head; this means that we do not need to transform the point clouds and we can directly merge the scenes.

Different Objects in Different Scenes

In this experiment we merge two scenes which contain two different objects: a mango can and a salt pot. In Figure 5.30 we can observe the two inputs of the system. The experiment has been realized using depth parameters between 4 and 7.

In Figure 5.31 we can see the results of the experiment. We can observe that both objects have been succesfully represented and that their position in the representation corresponds with the scenes. Another point to notice is that the depth parameter have not influenced the results.

Same Object in Different Scenes

In this experiment we represent an object which appears in two different scenes (Figure 5.32) in order to determine if the representation merging can join them.

In Figure 5.33 we can see that the two different parts of the leopard toy match in space so the silhouette of the leopard is easily recognizable. We also notice that the depth parameter influences the accuracy of the image but not the matching.

Summary

In the last experiments we checked the performance of the system when the stereo cameras were moved and the different scenes were merged. We could see that the scene merging works properly as the figure of the leopard can be clearly distinguished even when the parts of it are divided in different scenes. We also could check that the depth parameter does not affect the performance of the merging and that the distribution of the space in the representation corresponds to the real scene.

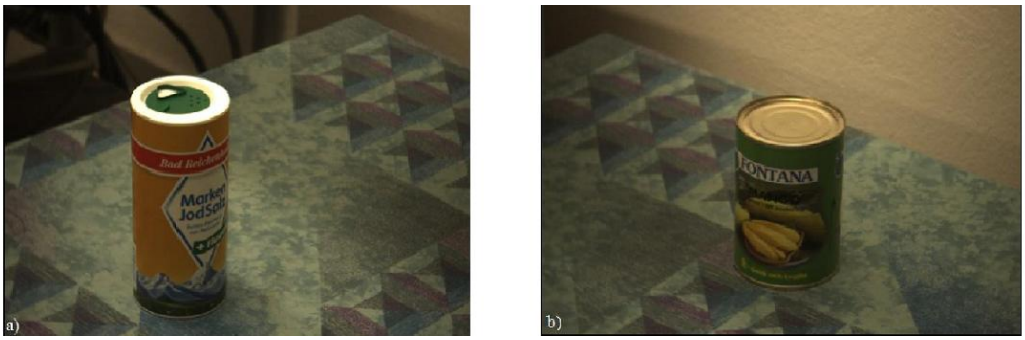

Figure 5.30: Scene seen from different viewpoints

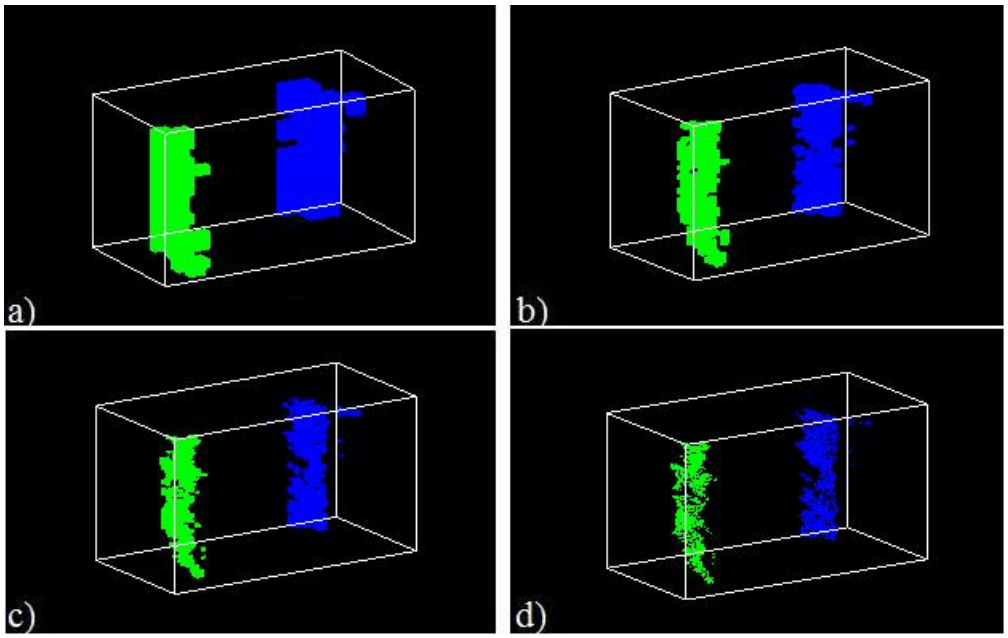

Figure 5.31: a) Scene merging results with a depth of 4. b) Scene merging results with a depth of 5. c) Scene merging results with a depth of 6. d) Scene merging results with a depth of 7.

Figure 5.32: Scene seen from different viewpoints

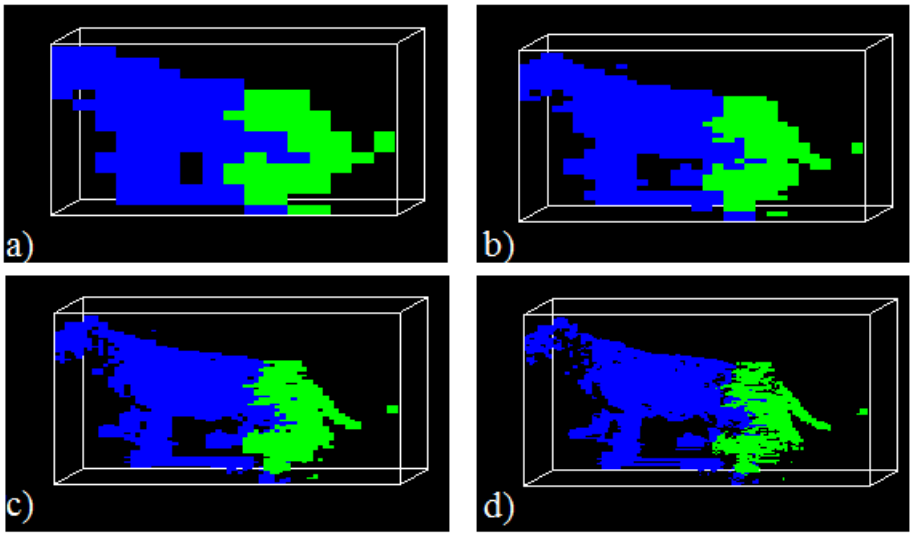

Figure 5.33: a) Scene merging results with a depth of 4. b) Scene merging results with a depth of 5. c) Scene merging results with a depth of 4. d) Scene merging results with a depth of 5.

Chapter 6

Conclusions

6.1 Conclusions

In this thesis we faced the problem of representing a scene and updating its information while changes are produced in the environment.

To achieve that we first discussed about different ways of representing 3D objects; specially we focused on the Octree and the Hierarchical Bounding Boxes methods. The first one has been implemented and tested, and the second tested in order to evaluate which one was more suitable for our purpose. After a set of experiments (see Section 5.2) we choose the Octree method which fulfilled our requirements best due to its higher accuracy in the representation.

We also discussed how to track objects in a scene in order to be able to detect motion in the scene and to store information about the changes that this motion produces. For that purpose the Kanade-Lucas [38] optical flow method was analyzed and a motion tracker was implemented using the KLT tracker [5].

The implemented system also incorporates a simple database which stores some extra information about the scene such as color information or objects' volumes. However, this point has not been specially developed as the thesis focused on the updating phase. As has been shown in Section 5.4 the implemented program is able to detect the changes produced on the scene and to improve the segmentation realized with external programs such as 3D-Frost [30]. It is also able to update the representation with the information it obtains when the cameras change their point of view.

The system has not been tested in real time application due to the difficulty of obtaining good and reliable point clouds online; however, if good point clouds could be obtained online, we think that linking the different modules of the presented work, the system could work in real time although some optimizations should be made.

6.2 Future Work

A good way to continue this project would be to improve its eficiency. As optimization has not been a priority during the development of the thesis. Also the test of new feature trackers such as particle filters would be a good contribution.

Moreover, the linking between the *properties database* and the Attribute Action [16] software would be a good improvement for the system. Finally another good idea would be to adapt the system to help in tasks such as grasping or navigation.

6.3 Acknowledgements

This thesis is the result of an Erasmus Exchange Program between the Kungliga Tekniska Högskolan and the Universitat de Girona. This experience allowed me to know a different culture and to improve my English level. It has been also interesting as it improved my knowledge in the Computer Vision and Robotics subjects. Apart from the academic improvements I made during this 6 months, I evaluate this exchange as a very positive experience as I have met many new and interesting people and discovered an amazing country as Sweden.

Bibliography

[1] T. Asfour, K. Regenstein, P. Azad, J. Schröder, A. Bierbaumand N. Vahrenkamp, and R. Dillmann. Armar-iii: An integrated humanoid platform for sensory-motor control. *6th IEEE-RAS International Conference on Humanoid Robots*, page 169175, 2006.

[2] T. Bando, T. Shibata, K. Doya, and S. Ishii. Switching particle filters for efficient real-time visual tracking. *Pattern Recognition, International Conference on*, 2:720–723, 2004.

[3] N. Bergström, , J. Bohg, and D. Kragic. Integration of visual cues for grasping. *International Conference on Computer Vision Systems. To appear.*, 2009.

[4] S. Birchfield. Klt: An implementation of the kanade-lucas-tomasi feature tracker. http://www.ces.clemson.edu/ stb/klt/, last checked: june 29th, 2009.

[5] S. Birchfield. Klt: An implementation of the Kanade-Lucas-Tomasi feature tracker. http://www.ces.clemson.edu/stb/klt/, 2007.

[6] J. Bohg and D. Kragic. Grasping unknown objects using shape context. *International Conference on Advanced Robotics*, 2009.

[7] Ch. Borst, M. Fischer, and G. Hirzinger. Grasping the dice by dicing the grasp. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems. (IROS 2003).*, volume 4, pages 3692–3697, 2003.

[8] F. Bourel, C. Chibelushi, and A. Low. Robust facial feature tracking. *British Machine Vision Conference*, 2000.

[9] S. Chen, W. Lin, and C. Chen. Split-and-merge image segmentation based on localized feature analysis and statistical tests. *CVGIP: Graph. Models Image Process.*, 53(5):457–475, 1991.

[10] D. Chetverikov, M. Nagy, and J. Verestóy. Comparison of tracking techniques applied to digital piv. *Pattern Recognition, International Conference on*, 4:4619, 2000.

[11] N. Chiba and T. Kanade. A tracker for broken and closely-spaced lines. In *In ISPRS Int. Society for Photogrammetry and Remote Sensing Conf., pages 676 to 683., Hakodate,Japan*, pages 676–683, 1997.

[12] K. Achim Fleming, R. Peters II, and R. Bodenheimer. Image mapping and visual attention on a sensory ego-sphere. In *Intelligent Robots and Systems*, pages 241–246. IEEE, 2006.

[13] J. Foley, A. van Dam, S. Feiner, and J. Hughes. *Computer Graphics: Principles and Practice in C*. Pearson, 2nd edition, 1990.

[14] C Geib, K Mourao, R Petrick, N Pugeault, M Steedman, N Krueger, and F Wörgötter. Object action complexes as an interface for planning and robot control. *IEEE RAS*, Int Conf. Humanoid Robots(Genova):Dec. 4–6, 2006, 2006.

[15] H.J. Havekort. *Results on Geometric Networks and Data Structures*. PhD, Utrecht University, 2004.

[16] K. Huebner, M. Björkman, B. Rasolzadeh, M. Schmidt, and D. Kragic. Integration of visual and shape attributes for object action complexes. volume Computer Vision Systems, pages 13–22. 2008.

[17] K. Huebner, S. Ruthotto, and D. Kragic. Minimum volume bounding box decomposition for shape approximation in robot grasping. In *International Conference on Robotics and Automation*, pages 1628–1633, 2008.

[18] K. Huebner M. Ralph B. Rasolzadeh D. Song J. Bohg, C. Barck-Holst and D. Kragic. Towards grasp-oriented visual perception for humanoid robots. *International Journal of Humanoid Robotics'09, Special Issue on Active Vision.*, 2009.

[19] R. Dillmann K. Welke, A. Tamim. Active multi-view object search on a humanoid head. *International Conference on Robotics and Automation*, 2009.

[20] A. Klinger. Patterns and search statistics. pages 303–339, 1972.

[21] T. Larsson and T. Akenine-Möller. A dynamic bounding volume hierarchy for generalized collision detection. *Computers & Graphics*, 30(3):451–460, June 2006.

[22] S. Gould E. Klingbeil Q. Le A. Wellman M. Quigley, S. Batra and A. Y. Ng. High-accuracy 3d sensing for mobile manipulation: Improving object detection and door opening. *Proc. International Conference on Robotics and Automation (ICRA)*, 2009.

[23] A. Mullins, A. Bowen, R. Wilson, and N. Rajpoot. Multiresolution particle filters in image processing. *Pattern Recognition, International Conference on*, 2006.

[24] D. Murray and J. Little. Patchlets: Representing stereo vision data with surface elements. *Applications of Computer Vision and the IEEE Workshop on Motion and Video Computing, IEEE Workshop on*, 1:192–199, 2005.

[25] I. R. Nourbakhsh, K. Sycara, M. Koes, M. Yong, M. Lewis, and S. Burion. Human-robot teaming for search and rescue. *Pervasive Computing, IEEE*, 4(1):72–79, 2005.

[26] K. Okuma, A. Taleghani, N. de Freitas, J. Little, and X. Lurton. A boosted particle filter: Multitarget detection and tracking. In Anonymous, editor, *Computer Vision - ECCV 2004*, pages 28–39. 2004.

[27] B. Peasley. 3d point cloud construction from stereo images. *Robotics and Automation*, 2008.

[28] A. Walker R. Fisher, S. Perkins and E. Wolfart. Geometric operations - affine transformation, 2003. http://homepages.inf.ed.ac.uk/rbf/HIPR2/affine.htm, last checked: june 24th, 2009.

[29] P. Rosin and T. Ellis. Image difference threshold strategies and shadow detection. *British Machine Vision Conf*, pages 347–356, 1995.

[30] M. Schmidt. Segmentation and attribution of 3d-objects with a stereo camera system. *Project Report at KTH/nada*, November 2007.

[31] SGI. Glut - the opengl utility toolkit. GLUT - The OpenGL Utility Toolkit, last checked: July 9th, 2009.

[32] SGI. Opengl. http://www.opengl.org/, last checked: July 9th, 2009.

[33] J. Shi and C. Tomasi. Good features to track. *IEEE Conference on Computer Vision and Pattern Recognition*, pages 593–600, 1994.

[34] S. Särkkä, A. Vehtari, and J. Lampinen. Rao-blackwellized particle filter for multiple target tracking. *Information Fusion Journal*, 8:2007, 2005.

[35] Z. Tang. Octree encoding: A new technique for the representation, manipulation and display of arbitrary 3d objects. *Journal of Computer Science and Technology*, 7:29–38, 1972.

[36] G. Taylor and L. Kleeman. Robust range data segmentation using geometric primitives for robotic applications. In *SIP*, pages 467–472, 2003.

[37] L. Thomason. Tinyxml. http://www.grinninglizard.com/tinyxml/, last checked: june 29th, 2009.

[38] C. Tomasi and T. Kanade. Detection and tracking of point features. *Carnegie Mellon University Technical Report*, pages 91–132, April 1991.

[39] F. Velasco and J. Torres. Cell octree: A new data structure for volume modeling and visualization, 2001.

[40] I. Wald, S. Boulos, and P. Shirley. Ray tracing deformable scenes using dynamic bounding volume hierarchies. *ACM Trans. Graph.*, 26(1), January 2007.

[41] M. Wang and J. Liu. Fuzzy logic-based real-time robot navigation in unknown environment with dead ends. *Robot Autonomous Systems*, 56(7):625–643, 2008.

[42] J. Wilhelms and A. Van Gelder. Octrees for faster isosurface generation. *ACM Trans. Graph.*, 11(3):201–227, 1992.

[43] B. Willimon, S. Birchfield, and I. Walker. interactive perception for cluttered environments. *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2009.

[44] T Yoshida. Image difference threshold strategies and shadow detection. *Image Processing 5*, pages 3487– 3490, 2004.