University of Bonn
Master of Science
Rheinische Friedrich-Wilhelms-Universitaet Bonn

Master of Science Thesis

Semantic Mapping Using Object-Class Segmentation of RGB-D Images

by

Nenad Biresev

Supervisor: Prof. Sven Behnke

Bonn, 2012.

Abstract

For task planning and execution of the complex tasks in an unstructured environments, an autonomous robot needs the ability to recognize and localize relevant objects. When this information is stored persistently in a semantic map, it can be used, e. g., to interact with humans. In this master thesis, I present a novel approach to learning semantic maps. This approach relies on measurements of RGB-D Kinect camera by means of simultaneous localization and mapping. I use random decision forests to segment object classes in images and exploit dense depth measurements of the Kinect camera to obtain scale-invariance. My object recognition method integrates shape and texture seamlessly. The probabilistic segmentation from multiple views is filtered in a voxel-based 3D map using a Bayesian framework. I report on the quality of the object-class segmentation method and demonstrate the benefits in performance when fusing multiple views in a semantic 3D map.

Contents

Acknowledgements

My profound gratitude goes to Professor Sven Behnke and Joerg Stueckler, my mentor and research adviusor, for guidance, contribution of ideas, and encouragement. I salute Joerg's and Sven's genuine passion for science, discovery and understanding, superior mentoring skills, and unparalleled availability. The research for this thesis was done at the Computer Science Institute of the Rheinische Friedrich-Wilhelms-Universitaet Bonn. I am grateful for the opportunity to work in such a stimulating environment, embedded in the exciting research context of Bonn. Special thanks go to Fraunhofer IAIS Sankt Augustin, who enabled me to perform the experiments and helped me with the thesis.

Finally, I wish to thank my family for their support. My parents have always encouraged and guided me to independence, never trying to limit my aspirations. Most importantly, I thank Jelena, my girlfriend, for showing untiring patience and moral support, reminding me of my priorities and keeping things in perspective.

Chapter 1

Introduction

1.1 Importance of Semantic Maps

Autonomous robots need semantic information about their environment in order to plan and execute complex tasks or to interact with human users on a semantic level, since the human may refer to an object. In order to gain such world knowledge, a robot not only needs the capability to recognize and localize objects, but also to represent this information persistently. For the task planning, the autonomous robot requires environment knowledge, which is knowledge about the objects in the world, their properties and relations between the objects. For example, given the task to pour a milk from the bottle to the cup, a robot should perform a sequence of actions like "Go to fridge, open it, find the milk in the fridge, grasp it, open it, pour it in the glass". To perform these sequence of operations, the robot needs to know that the milk is in the fridge, and to recognize and localize milk.

Knowledge about the structure and the current state of the environment is mostly stored in form of a map. The approach of how to represent, construct, and maintain maps has been one of the most active areas of research in robotics in the last decades. Typical robot maps are represented usually in 2D, in 3D, topologically and, additional sensor-relevant information such as specific features, or texture [29]. The autonomous robotics has an increasing interest in the semantic maps, which integrate semantic information into traditional robot maps. The importance of the semantic maps is that they provide an autonomous robot with deduction abilities (beside basic skills like navigation, localization, etc.) to infer information from its environment even when it has not been completely sensed. Using the semantic knowledge from the semantic maps, enables robot to perform more intelligent and autonomous. For example, if the robot is searching for a computer mouse, and it finds a monitor and a keyboard on a desk, then it can

deduce that a computer mouse is close from the keyboard.

1.2 Problems of Semantic Maps

The main problems of semantic maps are how these maps can be automatically acquired, how the semantic knowledge can be integrated with other types of knowledge in the maps (metric, topological, etc.), and how it can help robot to plan and execute tasks.

Autonomous robot needs to recognize and localize objects in the maps and store the semantic information in the map. The approach in which the robot uses RGB-D camera to recognize and localize objects, has certain limitations. The RGB-D cameras have valid depth measures only indoors, which constraints the robot to build semantic maps only in the indoor environment. The problem of recognizing objects are: speed and quality of object recognition. RGB-D Cameras work usually at 30Hz, 640x480 resolution images, thus they produce large amounts of data. Although the processing speed and storage capabilities of computers increased significantly in the last decades, the processing high-resolution images is today a very challenging task. Limited computational power constrains object recognition algorithms much more for real-time applications then for off-line applications. The hardware is developing more and more over years, and this makes the prediction possible that limited computational power constraints will relax within the next years. Another difficult problem is invariance to object transformations. Many object recognition algorithms require normalization of common variances, such as position, size, and pose of an object. Here, the normalization parameters cannot be estimated without reliable segmentation. Another challenging problem is to recognize ambiguous objects.

1.3 Contributions

In this master thesis, I propose a novel approach for learning semantic 3D maps containing object information. I combine object recognition in RGB-D images with simultaneous localization and mapping.

For object recognition, I apply random decision forests to classify images pixel-wise. The random decision forests classifier provides the probability over class labels for each pixel (see figure 1.1). Using depth information from RGB-D camera for the object-class segmentation algorithm, I obtain a scale-invariant classifier that incorporates shape and texture cues. In my approach, the random decision forest classifier is using color and depth features, which improves the classification performance comparing to approach where only color or only depth features are used.

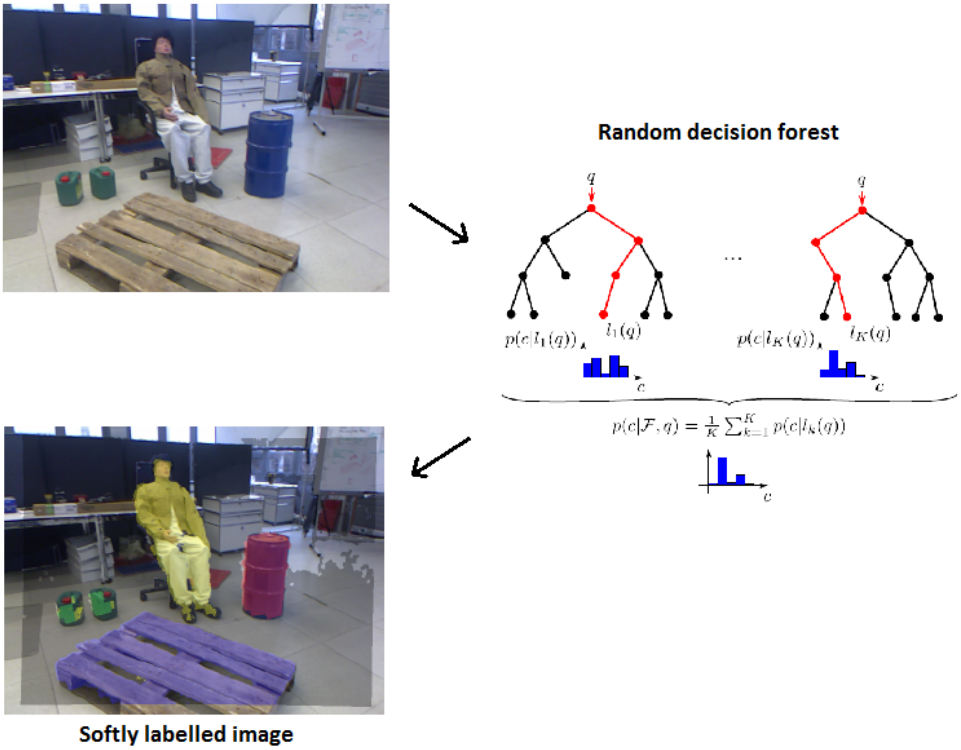

Figure 1.1: I apply random decision forests to classify images pixel-wise. The random decision forests classifier provides the probability over class labels for each pixel. Using depth information from RGB-D camera for the object-class segmentation algorithm, I obtain a scale-invariant classifier that incorporates shape and texture cues.

Given the camera trajectory estimate of an RGB-D SLAM method, I filter the soft labelling provided by random decision forests in a voxel-based 3D map using a Bayesian framework (see figure 1.2). This enables me to fuse classification evidence from several views and improve the robustness of the method for classification errors. This approach results in 3D maps augmented with voxel-wise object class information. In experiments, I evaluate the performance of the object recognition method and demonstrate the benefits of fusing recognition information from multiple views in a 3D map.

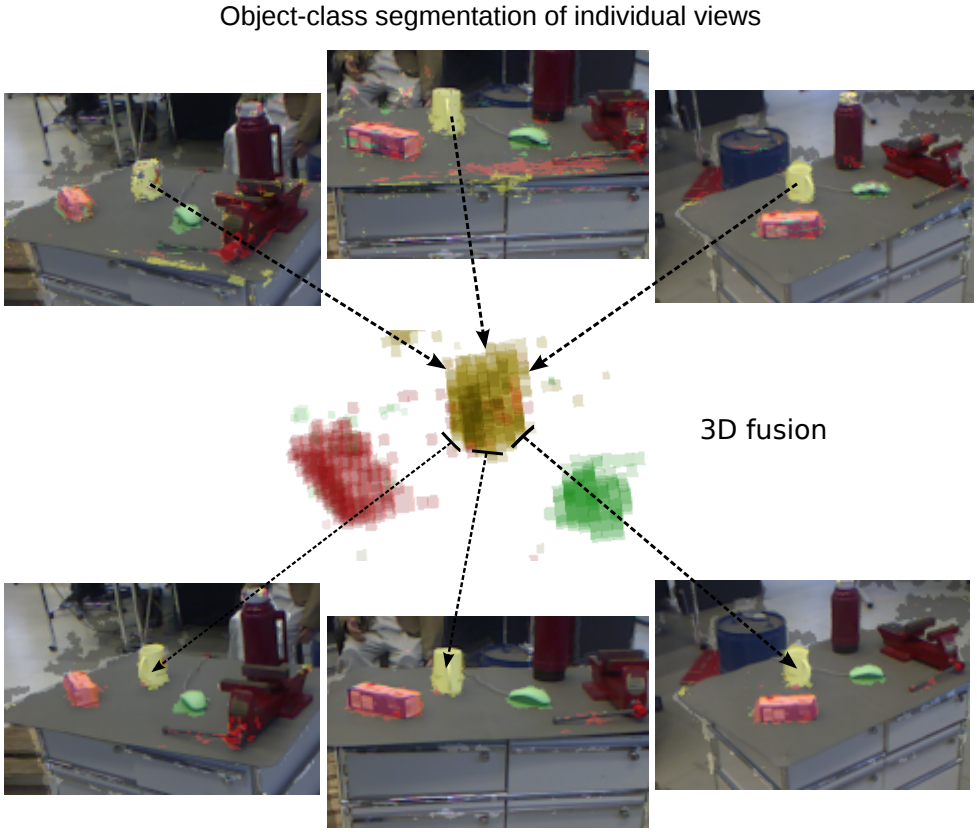

Figure 1.2: I fuse learned object-class segmentations of various views in 3D in a Bayesian framework.I not only obtain 3D object-class maps: Filtering in 3D from multiple views also reduces false positives and improves segmentation quality significantly. This reflects in the crisp back-projection of the 3D object-class map into the images.

1.4 Organization of the Thesis

The master thesis is organized as follows:

Chapter 2: Related work is discussed in the chapter 2, focusing on the fields semantic mapping, object class segmentation methods, and RGB-D SLAM methods.

Chapter 3: Basic knowledge is presented in this chapter, explaining the fundamentals of SLAM approach, VSLAM, Graph-SLAM, followed by object recognition approach using random decision forests. At the end of this chapter, the basics of semantic mapping are described.

Chapter 4: Methods which I was using in my thesis are explained here. Annotation tool, which is used for annotating the objects in RGB-D images in a semi-automatic way is described in the beginning of the chapter, followed by random decision forests used to solve the task object recognition, and at the end semantic mapping is described, using the trajectory from RGB-D SLAM and softly labelled images from random decision forest.

Chapter 5: Evaluation is described in this chapter, showing the performance of the object recognition method and demonstrate the benefits of fusing recognition information from multiple views in a 3D map

Chapter 2

Related Work

2.1 RGB-D SLAM

Many mapping approaches construct geometric representations of the environment. They have been using different sensors for this in the past, such as 2D and 3D laser scanners, monocular cameras, and stereo cameras. Recently, several methods were proposed that acquire full 3D maps from RGB-D images. For example, Henry et al. [9] present RGB-D mapping, which is a full 3D mapping system which applies graph-optimization algorithm combining visual features and shape-based alignment to obtain an accurate map. In Henry's approach, visual and depth information are also combined for view-based loop closure detection, which is followed by pose optimization to achieve globally consistent maps. Engelhard et al. [7] extract SURF features from the input color images taken from Kinect camera, and they match those features with features extracted from the previous images. They estimate the relative transformation between the camera frames using RANSAC algorithm. The pose refinement is done using ICP algorithm. As the last step, they optimize the resulting pose graph using a HOGMAN pose graph optimization. In the RGB-D SLAM method which I used in my work, they apply rapid registration of RGB-D images [26] and graph optimization to learn multi-resolution surfel maps. Such approaches do not incorporate valuable semantic knowledge like place or object labels into the 3D map.

2.2 Semantic Mapping

Some systems have been proposed that do semantic mapping. While most approaches use SLAM as a front-end to estimate a sensor trajectory [30, 28, 16, 17, 4, 5], some methods also incorporate the spatial relation of objects into SLAM. Tomono et al. [27], for example, builds a structured map, which consists of ob-

jects like furniture. They represent a map as a graph, where a node represents an object and an arc represents a relative pose between objects. SLAM is done by using odometry readings and sensor data obtained by object recognition. Loops are utilized as geometric constraints to correct the map distortion caused by errors in the data. They detect polyhedral object models in images and perform SLAM in 2D maps using the detected objects as landmarks. In contrast to the RGB-D SLAM approach which I use in my work [26], this method is restricted to objects with clearly visual linear edges. Zender et al. [30] presented an approach for creating conceptual representations of human-made indoor environments using mobile robots. Their approach refers to spatial and functional properties of typical indoor environments. Their model is composed of layers representing maps at different levels of abstraction. They apply SLAM in 2D maps using laser scanners, recognize objects using SIFT features, and map their locations in the 2D map. In addition to SIFT-based recognition, Vasudevan et al. [28] also detect doors by analysing laser scans, since they are important topologic objects that connect rooms. He presents a hierarchical probabilistic representation of space that is based on objects, and proposes a global topological representation of places with object graphs as local maps.

Meger et al. [16] combine semantic 2D mapping of objects with attention mechanisms. He presents a system, which performs robust object recognition in a realistic scenario, where a mobile robot moving through an environment must use the images acquired from its camera to recognise objects. The system in this work can be trained to localize the objects of interest in robots environment, and subsequently construct a spatial-semantic map of the region. They combine techniques like a peripheral-foveal vision system, an attention system combining bottom-up visual saliency with structure from stereo, and a localisation and mapping technique.

Nuechter et al. [17] apply ICP, plane segmentation, and reasoning to label planar segments in 3D maps that they acquire using 3D laser scanners. In their approach, they present integrated robot system for semantic mapping using a 3D laser scanner. They use 6D SLAM to register individual scans into a coherent 3D geometry map. Semantic labelling determines scene features. Trained classifier is detecting objects and localizing them. They visualize the semantic maps for human inspection. AdaBoost is applied on Haar wavelets and SVM classifiers on contour descriptions to detect objects and persons in the 3D maps. In my approach, I segment the original image data and fuse segmentation evidence from multiple views.

Castle et al. [4] and Civera et al. [5] propose purely vision-based means to acquire 3D maps with object labellings. In both approaches, SLAM is solved with feature-based monocular EKF-SLAM formulations. Objects are recognized using SIFT features and persistently maintained in the 3D feature map. Castle

et al presented approach where objects locations are incorporated as additional 3D measurements into a monocular SLAM process, which is using images to acquire and maintain a map of the environment, irrespective of whether objects are present or not. Civera et al. proposes a semantic SLAM process that merges in the estimated map traditional meaningless points with known objects. Here, using only the information extracted from a monocular image sequence the non-annotated map is built, and using the sparse set of images the known object models are automatically computed. When an object is recognized, its measured position is inserted in the 3D map and later refined by the SLAM algorithm in subsequent frames.

The approach of Ranganathan and Dellaert [18] learns 3D constellation models of places composed of objects using SIFT features. Their model is a 3D extension of the constellation object model, in which the objects are modelled by their appearance and shape. In this work, in a coordinate frame local to the place, the 3D location of each object is maintained. Using supervised learning, the object models are learned using roughly segmented and labelled training images. They use stereo range data to estimate 3D locations of the objects. They use the Swendsen-Wang algorithm, a cluster MCMC method, to solve the correspondence problem between image features and objects. The map consists of a set of places with associated models. The aforementioned approaches, however, do not build 3D maps with dense object information.

In my work, I integrate image-based object-class segmentation with SLAM from RGB-D images into a semantic 3D mapping framework. Each image is segmented pixel-wise into object classes and irrelevant background. Based on the SLAM trajectory estimate, this information is then projected into 3D to fuse object recognition results from multiple views into a consistent 3D map. This approach provides 3D segmentations of objects, and improves significantly classification performance.

2.3 Object-Class Segmentation Using Random Decision Forests

Object-class image segmentation is a challenging, actively researched problem in computer vision [24, 13, 10, 21]. One branch of research applies variants of random decision forests (RF, [3]). RFs are efficient classifiers for multi-class problems. They ensemble multiple random decision trees and achieve lower generalization error than single decision trees alone. RFs have been demonstrated to achieve comparable performance to SVMs [2]. Their major advantage is their high computational efficiency during recall. Implemented on GPU, training can be performed on massive datasets [23].

Shotton et al. [24] propose semantic texton forests, which are efficient and

powerful low-level features. It consists of collection of decision trees, which are directly applied on image pixels, thus they do not need the expensive computation of filter-bank responses or local descriptors. They are extremely fast to both train and test. The nodes in the decision trees provide an implicit hierarchical clustering into semantic textons, and an explicit local classification estimate. In their work, they use bag of semantic textons, which combines a histogram of semantic textons over an image region with a region prior category distribution. Then, over the whole image for categorization, the bag of semantic textons is computed, and over local rectangular regions for segmentation. They include both histogram and region prior which allows their segmentation algorithm to exploit both textural and semantic context. Semantic Texton Forests use simple features of luminance and color at single pixels or comparisons between two pixels in a RF classifier. Using image-level priors and a second stage of RFs, local and scene context is incorporated into the classification framework.

Schroff et al. [20] propose random decision forests for class based pixel-wise segmentation of images. They connect random decision forest classifiers using local features to classifiers using nearest neighbour matching class models. In this work, it has shown that quite different classifiers, such as nearest neighbour matching and texton class histograms, can be mapped onto a random decision forest. They improve classification performance of random decision forests by incorporating the spatial context and discriminative learning. In their work, they are using textons, colour, filterbanks, and Histograms of Oriented Gradients [6] features simultaneously, and it is shown that the combination of such multiple features increases the classification performance of random decision forests. They extends the node function of the random decision forest to incorporate global knowledge about the object classes. Single-histogram class models SHCMs are used to segment a test image into classes by a sliding window classifier. SHCMs are effectively mapped within random forests classifier.

Ladicky et al. [13] propose approach, where a hierarchical random field model, which is allowing integration of features computed at different levels of the quantisation hierarchy. In this work, using powerful graph cut based move making algorithms, they performe MAP inference in this model efficiently. A segment based conditional random field (CRF) is defined over the image, and inference is performed to estimate the label of each segment. The image quantisation enables the computation of powerful region-based features which are partially invariant to scale.

Schulz et al. [21] proposed approach in which object-class segmentation method is using a convolutional neural networks. It is assuming that the object of interest is centered and at a fixed scale, i.e that the segmentation problem has been solved. It is overperforming the challenging INRIA-Graz02 [15] dataset with regards to accuracy and the speed. It is using a convolutional architecture

with a multi-scale input. The method is using a HOG, and color image as a input, intermediate outputs, and epsilon-insensitive loss error function. First they compute HOG features at twice the map resolution and then they are subsampled. These operations are performed at three scales, where the resolution decreases by the factor of two. The teacher of the network is an image, where each pixel is marked with the class it belongs to, and is split into one map per class where the pixels are 1 when they are belonging to the class and are 0 when they don't belong to the class. In the end, the teacher image is smoothed and downsampled for each scale. They update the weights with the accumulated errors after each epoch using the RPROP [19]. It avoids the need to cross-validate a learning rate. It is using a fast GPU implementation, achiving a framerate of about 10fps during a recall. All the operations except preprocessing are performed on GPU.

Ion et al. [10] presented an approach a statistical model that collects larger scope image interpretations by selecting subsets of hypotheses from a bag of multiple figure-ground segments, based on mid-level scene constraints. They define a problem of image segmentation as optimization over sets of maximal cliques, which are sampled from a graph that connects all nonoverlapping figure-ground image segments. Here, the clique is a possible image segmentation composed of subsets of the figure-ground segments in the bag. They design and learn clique potentials, which are encoding both intrinsic, unary Gestalt segment properties and pairwise spatial compatibilities that account for plausible configurations of neighboring, spatially non-overlapping segments. By using the cliques, they are able to eliminate many implausible image segments and tilings that can't arise from the projection of surfaces in structured scenes. They based the learning the model parameters on maximum likelihood, which is alternating between sampling image tilings and optimizing their potential function parameters.

Recently, Shotton et al. [23] propose method which rapidly and accurately predicts 3D positions of body joints from a single depth image, without using temporal information. Their object recognition approach designs an intermediate body parts representation that maps the difficult pose estimation task into a simpler per-pixel classification task. Their object-class segmentation classifier estimates body parts invariant to pose, body shape, clothing, etc. by using the large and highly varied training dataset. They used confidence-scored 3D proposals of body joints and they reproject the classification result and find local modes. They show performance of 200 frames per second, and high accuracy on test set. They propose to normalize feature queries with the available depth to obtain scale-invariant recognition.

Stueckler et al. [25] presented an approach where object-class segmentation combines depth and color cues. They use Time-of-Flight (ToF) camera, which provides color images with depth information, to improve classification performance under scale and viewpoint changes. They normalize color and depth image

features with regard to scale and viewpoint by using depth and local surface orientation which are extracted from the ToF image. Local 3D shape features is incorporated into the classifier. Random decision forest classifier makes an efficient combination of depth and texture features.

In my work, I extend RF classification by incorporating both depth and color features. In contrast to Stueckler et al. [25], I use simple region features in color and depth and only normalize for scale changes to gain an efficient classifier for RGB-D images.

Chapter 3

Basics

3.1 SLAM

SLAM addresses one of the most fundamental problems in robotics. It is an acronym for Simultaneous Localization And Mapping, originally comes from Hugh Durrant-Whyte and John J. Leonard [14]. SLAM has the task to build a map of an unknown environment by a mobile robot (without a priori knowledge) or to update a map within a known environment (with a priori knowledge from a given map), and while doing that, at the same time it is keeping track of the its current location. SLAM is applicable for both 2D and 3D motion.

Maps are usually an answer to: "What does the world look like?". Maps are used to determine a robots location in an environment and to depict an environment for planning and navigation tasks. Maps represent in general the state of the environment at the time when the map is created. Localization has the task to estimate the place (and pose) of the robot relative to a map. It is giving the answer to the question: "Where am I?".

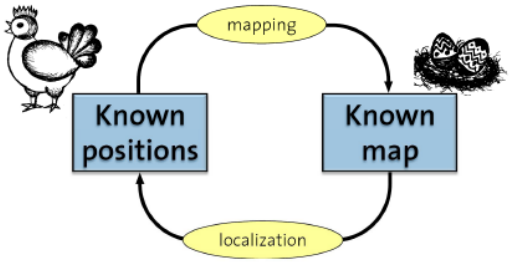

Figure 3.1: SLAM as a chicken and an egg problem. A map is needed for localizing a robot, and a good pose estimate is needed to build the map.

SLAM problems arise when the robot doesn't know its environment nor its

own position. All the robot knows are given measurements $z_{1:t}$ and controls $u_{1:t}$. In SLAM process the robot builds a map of its environment while simultaneously localizing itself relative to the current map. SLAM is one of the most difficult problems in the robotics community. SLAM is more difficult than the task of localization in the known map, because in SLAM the map is unknown. It is also more difficult task than mapping with known poses, because in SLAM the poses are also unknown.

The SLAM problem can be divided in two categories from the probabilistic perspective:

1. Online SLAM problem: evolves estimating the posterior over the current pose together with the map:

$$p(x_t, m|z_{1:t}, u_{1:t}) \tag{3.1}$$

Here x_t is the pose at the time t, m is the map which robot is building, $z_{1:t}$ are all the measurements which the robot made up to the time step t, and $u_{1:t}$ are all the controls which the robot performed up to the time step t. It is called online SLAM because it only estimates the variables which persist at the time step t (robots current position and current map). Many of the online SLAM methods are incremental, meaning that they discard past measurements and controls after they are processed. The online SLAM is depicted in Figure 3.2

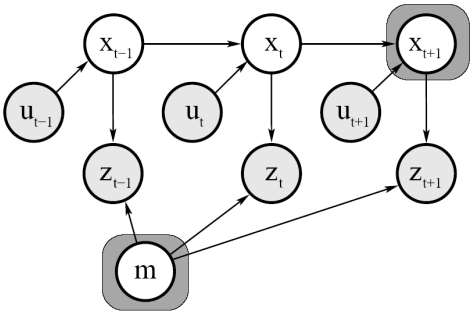

Figure 3.2: Online SLAM: evolves estimating the posterior over the current pose together with the map

2. Full SLAM problem: the posterior over the entire path $x_{1:t}$ together with the map m is calculated, instead of calculating posterior just for the current pose x_t and the map m (as in online SLAM problem): $p(x_{1:t}, m|z_{1:t}, u_{1:t})$ The full SLAM problem is illustrated in Figure 3.3.

The online SLAM problem is actually the result of integrating out past poses from the full SLAM problem:

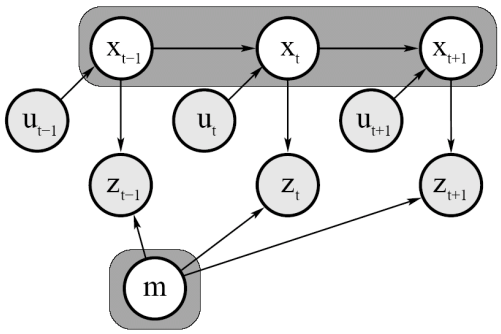

Figure 3.3: Full SLAM: the posterior over the entire path $x_{1:t}$ together with the map m is calculated

$$p(x_t, m|z_{1:t}, u_{1:t}) = \int\int\cdots\int p(x_{1:t}, m|z_{1:t}, u_{1:t})dx_1dx_2\cdots dx_{t-1} \quad (3.2)$$

These integrations in SLAM are usually performed one at a time.

Practically, calculating a full posterior is in most cases infeasible, because of the following two reasons:

1. The continuous parameter space has high dimensionality

2. The number of discrete correspondence variables is large (many SLAM algorithms construct maps with tens thousands of features).

Because of these two reasons, in practice, SLAM algorithms must rely on approximations.

The SLAM problem can be divided into another two categories according to the nature of the estimation problem:

1. Continuous estimation problem: pertains to the location of the objects in the map and the robot's own position. The objects can be landmarks in feature-based representation, or they can be object patches detected by range finders.

2. Discrete estimation problem: has to do with correspondence. Once an object is detected, the SLAM should find the relation of this object to previously detected objects. The reasoning is discrete (either the object is the same as a previously detected object, or it is not the same).

3.1.1 Problems of SLAM

SLAM is usually considered as a chicken and egg problem: Map is needed for localization, but at the same time an accurate pose estimate is needed to build that map. Another problem in SLAM comes from noisy sensors. Because of the noisy sensors, the observation of the landmarks are uncertain, thus the estimated location of the robot and a map are uncertain as well. After each iteration of updating the map, any new feature added to the map will contain more and more error. The error of building a map, and keeping track of the location of the robot, increases over time cumulatively, and it is distorting the map and therefore the robot's ability to accurately determine its location and orientation. Another problem which can occur in SLAM is picking wrong data associations, which can have catastrophic consequences. This problem can occur when the robot has the error in its position, and this causes the robot to make a mistake in data association. The problems of SLAM are visualized in Figure 3.4.

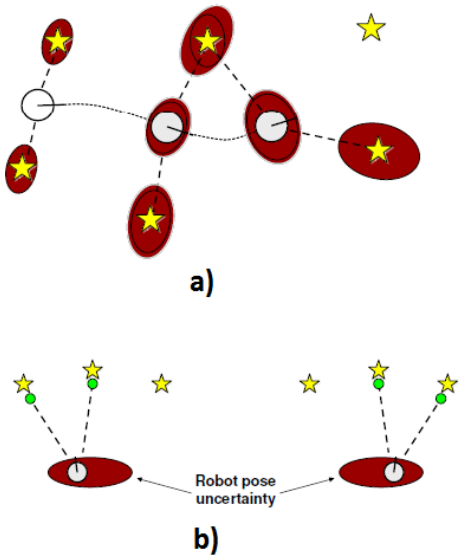

Figure 3.4: Problems of SLAM: a) Robot path error correlates with errors in the map (if there is error in the robot path, there will be also error in the map) b) Pose error correlates data associations

There are techniques to compensate the errors of mapping and localization, such as recognizing features which the robot has previously seen, and accordingly updating the map and robot's trajectory. Some of the SLAM methods use techniques like Kalman filters, particle filters, scan matching of range data to update the current map and robots trajectory using the previously seen landmarks.

3.1.2 Mapping

SLAM in the robotics community is the process of building geometrically consistent maps of the environment. There are different kinds of maps, used to model the environment:

1. Feature-based maps: store landmarks of the environment (e.g. lines,corners, circles, etc.)
2. Occupancy grid maps: store at each cell in the map the probability that the cell is occupied
3. Topological maps: represent the environment, by showing the connectivity of the environment, rather than creating a geometrically accurate map. The algorithms which are building topological maps are not considered as a SLAM methods, because they are not giving geometrically accurate map.

Figure 3.5: Illustration of different types of maps. In feature based maps, features like trees are stored in the map. The example of topological map is a subway connectivity map, where the stations are the nodes and edges are the paths between the stations.

The general task of mapping is given the sensor data to calculate the most likely map:

$$d = \{u_1, z_1, ..., u_n, z_n\} \tag{3.3}$$

$$m^* = \arg\max_m P(m|d) \tag{3.4}$$

Here, d is the sensor data consisting of N actions $(u_1...u_n)$ and N measurements $z_1, ..., z_n$. The most likely map is m^*

Some of the problems in mapping in the SLAM are:

1. The size of the maps (big environments are more difficult to map)
2. Noise in the sensors and actuators: more noise in sensors less accurate maps

3. Self-similar environments are hard to map

4. Cycles: When a robot comes to the same position in the map where it was previously, but from the different path, it must recognize it despite the accumulated error. This is called loop closure.

3.1.3 Sensors Used in SLAM

SLAM method are use different kinds of sensors to acquire data. The sensors typically used in SLAM are: one dimensional (single beam) or 2D- (sweeping) laser rangefinders, 3D Flash LIDAR, 2D or 3D sonar sensors. The SLAM which is using the monocular or stereo cameras is referred as visual SLAM (VSLAM).

3.1.4 Graph-SLAM

Graph-SLAM belongs to the SLAM category of methods, and it solves full SLAM problem (the full robot trajectory and a map). The posterior of the full SLAM problem naturally forms a sparse graph [22]. This sparse graph leads to a sum of nonlinear quadratic constraints. Optimizing nonlinear quadratic constraints gives a maximum likelihood map and corresponding robot trajectory. The Graph-SLAM consists of five poses, which are labelled with $x_0, \cdots x_4$, and two features m_1 and m_2. In the graph, there are two types of arcs: motion arcs and measurement arcs. Motion arcs connect any two consecutive robot positions, and measurement arcs connect poses and features which robot observed. Each edge in the graph is a nonlinear constraint.

In order to compute a map posterior, Graph-SLAM linearises the set of constraints. The result of linearisation of the set of constraints is an information vector. The information matrix inherits the sparseness from the graph which is build by Graph-SLAM. The sparseness allows Graph-SLAM to apply the variable elimination algorithm, by transforming the graph into smaller graph, which is defined over robot poses. The path posterior map is computed using standard inference techniques. The full map posterior is quadratic in the size of the map, and because of that reason it is usually not recovered. The advantage of the Graph-SLAM is that growing of the graph is not computationally expensive, but it requires additional inference when recovering the map and the robots trajectory. The Graph-SLAM is collecting the information about the environment all the time into its graph without resolving that information. Because of that Graph-SLAM can build maps that are very large (e.g. many orders of magnitude larger than EKF SLAM map).

Graph-SLAM calculates posteriors over robot trajectory, and because of that it is not incremental algorithm. It can access to the full data during building the map, and it can apply improved linearisation and data association techniques. It has three important iteration steps in the mapping: the building of the map, the

computation of corresponding variables and the linearisation of the measurements and motion models. Because that iteration steps, Graph-SLAM is constructing the maps with superior accuracy (e.g. more accurate than EKF-SLAM).

3.1.5 Visual SLAM

The visual SLAM belongs to the category of SLAM tasks. It considers using the vision sensors, such as the monocular and stereo cameras. It is doing the simultaneous localization and mapping by extracting and storing in the map the features from the images, and it is matching these features between the consecutive frames in order to find a transformation between the frames, which gives the trajectory of a mobile robot.

A stereo camera is a type of camera with two or more lenses with a separate image sensor. This is simulating the human binocular vision, thus it captures the three-dimensional images.

Figure 3.6: Example of a stereo camera

Monocular camera is a type of camera with only one lens. There is also a special type of camera with 2 lenses, which is providing RGB + depth information for each pixel in the image. Example of such a camera is a Microsoft Kinect camera, which is visualized in the figure 3.7. Kinect camera became extremely popular in the robotics community recently, because it is providing very precise depth measurement. It is successfully used for the visual SLAM. It works with a 30Hz frame rate, has a resolution of 640x480, and it can be used also for smaller resolutions. It reads depth values precisely at the distance more than 40cm and less than 6m.

The visual SLAM problem can be subdivided into 4 categories:

1. Visual SLAM using monocular camera, extracting sparse image features.

2. Visual SLAM using monocular camera, extracting dense image features.

Figure 3.7: Example of a Microsoft kinect camera

3. Visual SLAM using stereo camera, extracting sparse image features.

4. Visual SLAM using stereo camera, extracting dense image features.

Some of the advantages of visual SLAM are:

1. The visual sensors are long-range, high-resolution, passive sensor

2. For the feature detection and feature descriptors it is using low level features (edges, corners), high level features (objects, complex structures)

3. It enables feature tracking and matching techniques between the consecutive camera frames.

Some of the disadvantages of visual SLAM are:

1. Problems in textureless or repetitive environments (e.g. white walls)

2. Difficulty to determine range from single measurement

3. It enables feature tracking and matching techniques between the consecutive camera frames.

4. rgb and depth image noise

3.1.6 RGB-D SLAM

The RGB-D SLAM method is based on fast and accurate RGB-D image registration using multi-resolution surfel maps [26]. The registration approach aligns 640×480 images at a frame-rate of about 10 Hz.

Since small registration errors may accumulate in significant pose drift over time, a graph of probabilistic spatial relations is established and optimized between similar view poses (see Fig. 3.8). A view pose in the graph is denoted as key view and register the current camera frame to the most similar key view in order to keep track of the camera pose. Similarity is measured by distance in translation and rotation between view poses. New key views are added to the graph, if the similarity measure indicates a significant motion of the camera. This also includes a spatial relation between the new key view and the reference key view. In addition, relations between further similar key views are established.

This probabilistic registration method provides a mean and covariance estimate for each spatial relation. The likelihood of the relative pose observation $z = (\hat{x}, \Sigma(\hat{x}))$ of the key view j from view i is obtained by

$$p(\hat{x}|x_i, x_j) = \mathcal{N}\left(\hat{x}; \Delta(x_i, x_j), \Sigma(\hat{x})\right), \tag{3.5}$$

where $\Delta(x_i, x_j)$ denotes the relative pose between the key views under their current estimates x_i and x_j.

From the graph of spatial relations the probability of the trajectory estimate given the relative pose observations is inferred

$$p(x_{1,\ldots,N}|\hat{x}_1, \ldots, \hat{x}_M) \propto \prod_k p(\hat{x}_k|x_{i(k)}, x_{j(k)}). \tag{3.6}$$

This graph optimization problem solve by sparse Cholesky decomposition using the g^{2}o framework [12]. Finally, the mapping framework supports the fusion of the RGB-D images in a single multi-resolution surfel map using the optimized trajectory estimate.

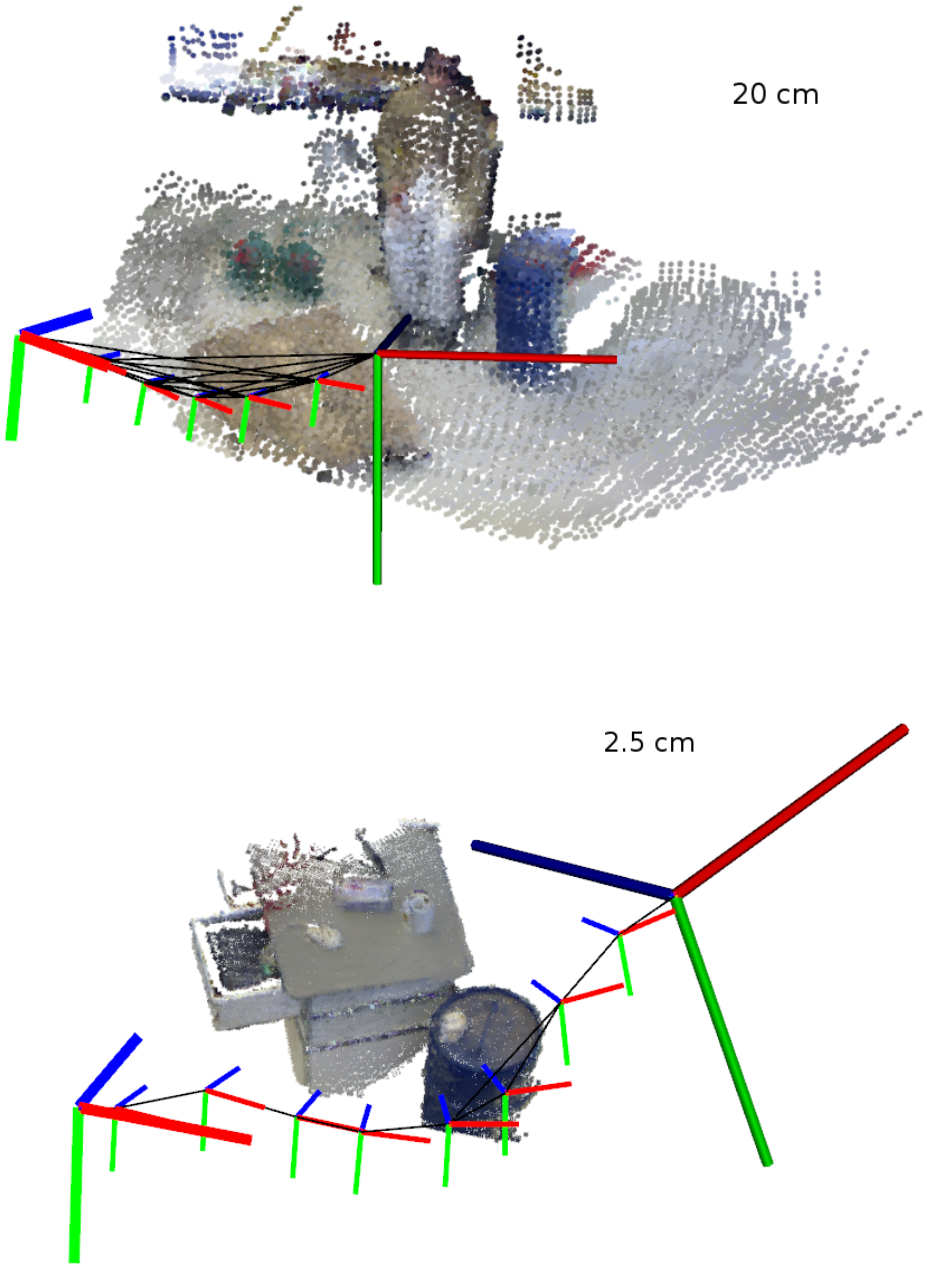

Figure 3.8: Simultaneous localization and mapping is performed by registering multi-resolution surfel maps of RGB-D images and optimizing spatial relations in a key view graph. The example maps are visualized by samples from the surfel distributions at 2.5 cm (bottom) and 20 cm (top) resolution.

3.2 Object Recognition

One of the goals in robotics is to give the robots visual capabilities like humans have, so they can sense their surrounding, understand what they sense and according what they sense take certain actions. Object recognition has the task to find objects of interest in a given image. The robot should be able to recognize several object classes (e.g. tea box, human, chair, computer mouse) and different object instances (plastic chair, metal chair, black chair, white chair). The visual appearance of objects can change due to different shape, color, texture, orientation, size, scale, occlusion and illumination. The objects of interest are not presented alone and isolated in the scenes, but there are other objects classes, which makes the problem more complicated.

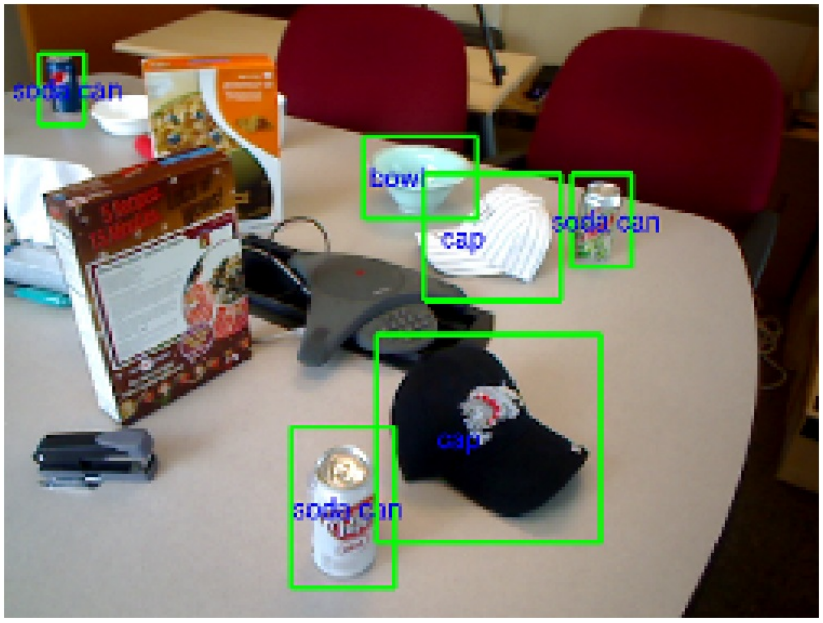

Figure 3.9: Example of object recognition. The task is to recognize several classes and surround them with the rectangle box. http://www.cs.washington.edu

Computational complexity is one of the challenges for the Object recognition task. Some systems need too much time to recognize objects in the single image. An object recognition method should be fast and robust. The task is still considered to be very challenging.

The objects can be recognized using a variety of sensing devices. The most commonly used sensor for robot mapping and navigation, 2D laser range scanners, provides information that is surely not sufficient to recognize objects, and it is restricted to a single plane. 3D lidars give a richer source of information, but its price and working conditions make them difficult to use in robotics. A time-of-flight camera (ToF camera) is able to acquire a 3D image in an indoor scenario in

Figure 3.10: Object-class segmentation. The image on the left is the input image. Right image is the object class segmented image. The objects of interest are human labelled with yellow, canister with green, pallet with blue, and barrel with red.

a convenient way. It calculates the distance based on the known speed of light, by measuring the time of flight of a light signal between the camera and the object for each point of the image. Further, it can only obtain images with a resolution of less than two hundred pixels with a maximum depth of five meters. Recently, the new Microsoft Kinect camera has appeared in the market. It provides depth and color images, has a resolution 640x480 pixels, and has range of 0.7 - 6m.

Classification has the task of assigning the object class label to the previously unseen object instances (is this a chair?). Recognition is the task of identifying a particular object instance (is this my chair?). Detection has the task to (roughly) decide where is the object of interest located in the image (where is the chair?).

3.2.1 Object-Class Segmentation

Object-class segmentation belongs to the object recognition category of tasks. It classifies images pixel-wise into objects and background. It assigns to each recognized pixel in an image a probability that it belongs to the certain class. For example, if the system is able to recognize 3 different object classes (human, barrel, coffee mug) then it would assign to each pixel of the image 3 probabilities. These probabilities are $p_1(pixel = human)$, $p_2(pixel = barrel)$ and $p_3(pixel = coffeemug)$, and each is defined as the probability that a pixel belongs to the class human, barrel and coffee mug. In Figure 3.10 the example of object class image segmentation is illustrated.

3.2.2 Randomized Decision Forests

Randomized decision forests is a precise and fast object recognition method. It consists of a collection of randomized decision trees. Each randomized decision tree consists of nodes and edges which connect the nodes. There are branching

nodes, and leaf nodes. At the leaf nodes, the distribution probabilities for each object class are stored. For example, if the system should train on 3 different classes (coffee mug, tee box and a chair) then at each leaf node, 3 probabilities are stored. They are defined as the probability that a pixels which end up in that leaf node belong to the coffee mug, tea box and a chair. The probabilities distributions are visualized in Figure 3.11. These trees are binary decision trees, meaning that starting from the root node (which is usually at the top of the tree) each branch node has exactly two children.

Each branching node consist of a feature function, and a threshold. Feature function at each node represents one decision made by the classifier. If the feature function applied on an input pixel q returns value less than a threshold τ, then the input pixel q is forwarded to the left child node, otherwise it is forwarded to the right child node.

The reason why randomized decision forests ensemble multiple randomized decision trees is that they achieve lower generalization error. They have been demonstrated to achieve comparable performance to the support vector machines SVMs [2]. Their biggest advantage of the randomized decision forests is the high computational efficiency during the recall, because for each pixel in the image which is being recognized, only several feature functions from the root node down the leaf node need to be calculated. If the algorithm would be implemented on GPU, training can be performed on huge image datasets (e.g. 100000 images).

In order to determine the distribution of the class labels at query pixel, it needs to be evaluated on each decision tree belonging to the collection of decision trees in the randomized decision forests. The query pixel is associated a distribution over class labels from each evaluated decision tree, and the final distribution is the average over all distribution taken from the decision trees. That is, if a randomized decision forests F consist of a collection of 3 decision trees, then a query pixel needs to be evaluated on all 3 decision trees, and it gets the distribution over class labels from each decision tree and outputs the average distribution.

$$p(c|F,q) = \frac{1}{K}\sum_{k=1}^{K} p(c|l_k, T_k, q) \tag{3.7}$$

In the last equation, the F is a randomized decision forests, T_k stands for a k decision tree, K is a total number of decision trees in the forest F, c is a class label assigned to the query pixel q, and l_k is the leaf node which gives the distribution over all class labels for the decision tree T_k.

For training a randomized decision forests, each decision tree is trained independently on a random subset of the training images. At each branching node in a decision tree, many features and thresholds are randomly sampled. The one that separates the training examples in a best way is selected according to the

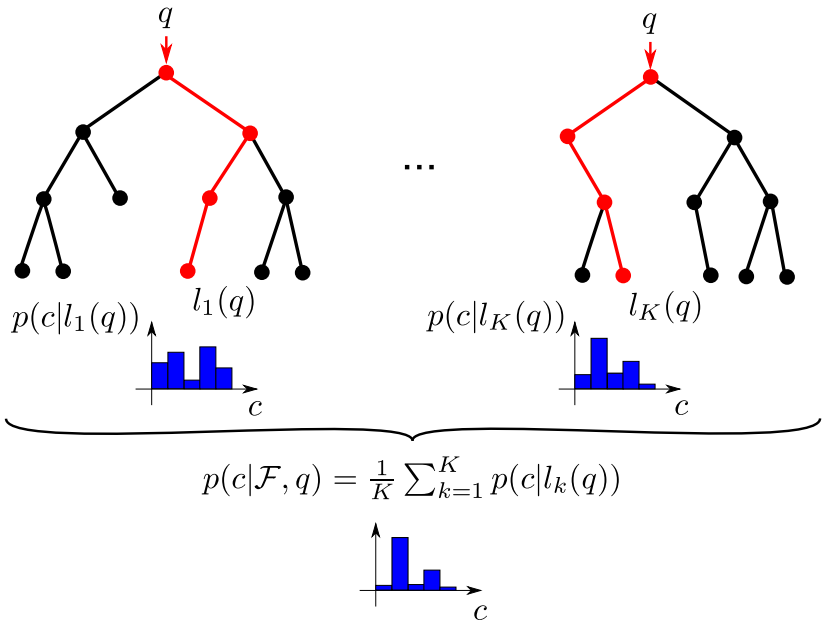

Figure 3.11: Randomized Decision Forests: A forest is an ensemble of decision trees. Each node of the tree makes a binary decision on a pixel. It maintains a classification probability of pixels that reach this node. A query pixel q is assigned the average posterior distribution at the leaves it reaches in the forest.

measure of information gain. This allows to mix different kinds of features such as functions in color and depth cues.

Training the Randomized Decision Forest

The training of the randomized decision forest starts by choosing the K number of decision trees. At the beginning of the training, the whole training dataset is randomly split into K equal size subsets. Each of the k subsets is an input for the training of each k decision trees. Starting at the root node, from each image in the subset, 2000 random pixels are chosen. If the total number of images in the k subset is L, then the total number of pixels at the input of the decision tree will be $2000 * L$.

Each decision tree is build using depth-first approach. One starts at the root and creates left and right child nodes. Then it process the left child node and again makes a left and a right child node. This repeats until the leaf node is reached. Starting from the root node, each node is splitting the input pixels on left and right child. The feature function and a threshold which splits the pixels in the best way according to the measure information gain is chosen, and stored at the node. Further, pixels from the child nodes are again split, until the leaf node is reached. The leaf nodes don't store a best feature function and a threshold, but only the distribution probabilities for each class object.

The information gain is the measure which is used during the training to estimate how good the pair feature function and a threshold $\psi = (\theta, \tau)$ separate the pixels from the input of the node on the left and right subsets:

$$Q_l(\psi) = \{(I, x) | f_\theta(I, x) < \tau\} \tag{3.8}$$

$$Q_r(\psi) = Q \setminus Q_1(\psi) \tag{3.9}$$

In the last two equations, Q is the set of all input pixels at the node, $Q_l(\psi)$ is the set of pixels which are forwarded to the left branching node, and $Q_r(\psi)$ is the set of pixels forwarded to the right branching node. The information gain, is formulated as:

$$Q(\psi) = H(Q) - \sum_{s \in \{l,r\}} \frac{|Q_s(\psi)|}{|Q|} H(Q_s(\psi)) \tag{3.10}$$

The Shannon entropy for the input set of pixels of the node is as following:

$$H(Q(\psi)) = - \sum_{c \in classes} p(c|\psi) log_2(p(c|\psi)) \tag{3.11}$$

Here, $H(Q(\psi))$ is the Shannon entropy for the set of pixels at the input of the node, and $p(c|\psi)$ is the probability that a pixel which reaches the node belongs to the object class c. This probability is as following:

$$p(c|\psi) = \frac{n_c}{N} \tag{3.12}$$

Here, n_c is the number of pixels at the input of the node, belonging to the class c, and N is the total number of the pixels at the input of the node.

Training the Randomized Decision Forest

---

1. Choose the number of decision trees K.
2. Split the whole training dataset on the k partitions, randomly.
3. For each k decision tree
4. Calculate label weights
5. Create a root node
6. Generate M number of candidates of pairs of color feature function and a threhsold $\psi_m^c = (\theta_m^c, \tau_m^c)$

7. Choose the best pair of color feature function and a threhsold $\psi_b^c = (\theta_b^c, \tau_b^c)$ according to the Information gain

8. Generate M number of candidates of pairs of depth feature function and a threhsold $\psi_m^d = (\theta_m^d, \tau_m^d)$

9. Choose the best pair of depth feature function and a threhsold $\psi_b^d = (\theta_b^d, \tau_b^d)$ according to the Information gain

10. Choose between color feature function and depth feature function according to the information gain, $\psi_b = (\theta_b, \tau_b)$

11. Separate the input pixels of the node to the left $Q_l(\psi)$ and right $Q_r(\psi)$ subsets, using the best pair of feature function and threshold $\psi_b = (\theta_b, \tau_b)$

12. Create a left and a right branch node, and forward the left $Q_l(\psi)$ and right $Q_r(\psi)$ pixel subsets to their inputs

13. For left and right branch nodes, If the maximum depth D of the tree is reached, or the number of input pixels is smaller then a minimum node pixel capacity assign the node as the leaf node, otherwise assign the node as the branch node.

14. Go to the step 6

15. END

3.3 Semantic Mapping

Semantic information can help an autonomous robot to act goal-directly, thus part of this information has to be about objects, functionalities, events, or relations in the robot's surrounding. The map is the data structure which consists the space information about the robots environment. Typical robot maps are represented usually in 2D, in 3D, topologically and, additional sensor-relevant information such as specific features, or texture [29]. The nowadays typical purpose of the semantic maps is the robots navigation, planning, prediction, and sensor data interpretation. A semantic map stores an information about entities, i.e., objects, functionalities, or events, that are located in 3D space.

To enable the reasoning about robots navigation, planning, explanation, prediction, and sensor data interpretation, the knowledge about entities is required (e.g the computer mouse is typically standing on the desk left or right from the keyboard, in front of the monitor). In Figure 4.20 is shown the example of semantic map where different objects are stored in the map.

The main problems of semantic maps are how these maps can be automatically acquired, how the semantic knowledge can be integrated with other types of knowledge in the maps (metric, topological, etc), and how it can help robot to plan and execute tasks.

Autonomous robot needs to recognize and localize objects in the maps and store the semantic information in the map. The approach in which the robot uses RGB-D camera to recognize and localize objects, has certain limitations. The RGB-D cameras have valid depth measures only indoors, which constraints the robot to build semantic maps only in the indoor environment. The problem of recognizing objects are: speed and quality of object recognition. RGB-D Cameras work usually at 30Hz, 640x480 resolution images, thus they produce large amounts of data. Although the processing speed and storage capabilities of computers increased significantly in the last decades, the processing high-resolution images is today a very challenging task. Limited computational power constrains object recognition algorithms much more for real-time applications then for offine applications. The hardware is developing more and more over years, and this makes the prediction possible that limited computational power constraints will relax within the next years. Another difficult problem is invariance to object transformations. Many object recognition algorithms require normalization of common variances, such as position, size, and pose of an object. Here, the normalization parameters cannot be estimated without reliable segmentation. Another challenging problem is to recognize ambiguous objects.

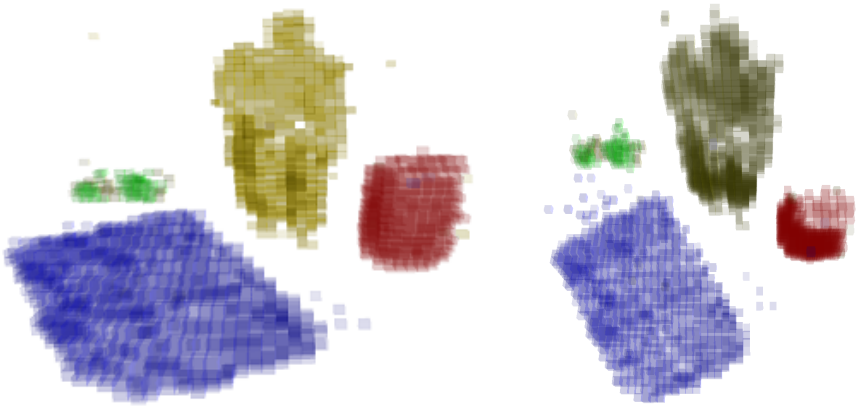

Figure 3.12: Semantic mapping. In the 3D semantic map, the human is labelled with yellow, canister with green, barrel with red and pallet with blue. Image on the left is showing the front view and image on the right top view.

3.4 Pinhole camera model

A 3d point is projected onto the image plane by using a standard pinhole camera model, and perspective transformation [1]:

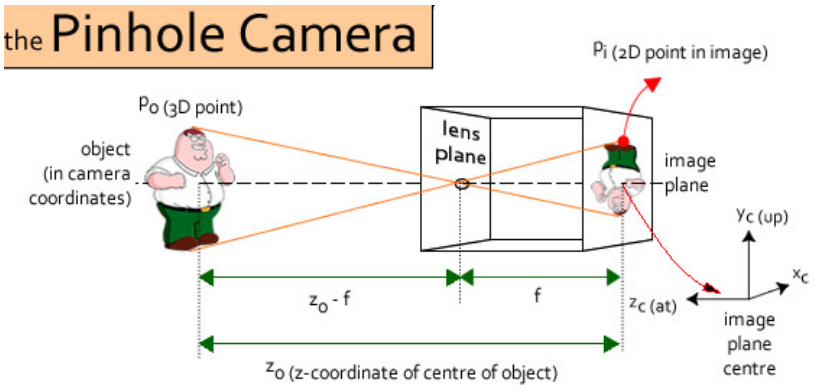

Figure 3.13: Pinhole camera model. The object is inverted in the image plane. Image taken from http://3.bp.blogspot.com

$$s \cdot m' = A[R|t]M' \tag{3.13}$$

$$s \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_1 \\ r_{21} & r_{22} & r_{23} & t_2 \\ r_{31} & r_{32} & r_{33} & t_3 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix} \tag{3.14}$$

In the previous equation, (u, v) are the 2D coordinates of the projected pixel in the image plane, (X, Y, Z) are the coordinates of a 3D point in the world coordinate space which is projected on the image plane. A is a matrix of intrinsic parameters, which incorporates a principal point (c_x, c_y), which is usually at the center of the image, and focal lengths f_x, f_y which are expressed in pixel units. If an image would be scaled by some factor, all of these parameters should be scaled by the same factor. The matrix A is independent of the scene, once when it is estimated, can be always re-used as long as the focal length is fixed. A rotation-translation matrix $[R|t]$ is a matrix of extrinsic parameters. This matrix incorporates the transformation of the current coordinate system of the 3d point to some other coordinate system. This transformation can also be written as:

$$\begin{bmatrix} x \\ y \\ z \end{bmatrix} = R \begin{bmatrix} X \\ Y \\ Z \end{bmatrix} + t \tag{3.15}$$

$$x' = \frac{x}{z} \tag{3.16}$$

$$y' = \frac{y}{z} \tag{3.17}$$

$$u = f_x \cdot x' + c_x \tag{3.18}$$

$$v = f_y \cdot y' + c_y \tag{3.19}$$

Real camera lenses usually have some distortion, in the most cases radial distortion and tangential distortion. So, the above model can be formulated as:

$$\begin{bmatrix} x \\ y \\ z \end{bmatrix} = R \begin{bmatrix} X \\ Y \\ Z \end{bmatrix} + t \tag{3.20}$$

$$x' = \frac{x}{z} A \tag{3.21}$$

$$y' = \frac{y}{z} \tag{3.22}$$

$$x'' = x'(1 + k_1 r^2 + k_2 r^4 + k_3 r^6) + 2p_1 x' y' + p_2(r^2 + 2x'^2) \tag{3.23}$$

$$y'' = y'(1 + k_1 r^2 + k_2 r^4 + k_3 r^6) + p_1(r^2 + 2y'^2) + 2p_2 x' y' \tag{3.24}$$

$$r^2 = x'^2 + y'^2 \tag{3.25}$$

$$u = f_x \cdot x'' + c_x \tag{3.26}$$

$$v = f_y \cdot y'' + c_y \tag{3.27}$$

Here, k_1, k_2, k_3 are radial distortion coefficients, and p_1, p_2 stand for tangential distortion coefficients. The distortion coefficients, k_1, k_2, k_3, p_1, p_2 do not depend on the scene, thus they also belong to the intrinsic camera parameters. They remain always the same no matter the image resolution.

3.5 Octree

An octree is a tree data structure with a property that each internal node has exactly eight children. Octrees are used to partition a three dimensional space by recursively subdividing it into eight octants. Octrees are the three-dimensional analogue of quadtrees. The octree is visualized in Figure 3.14

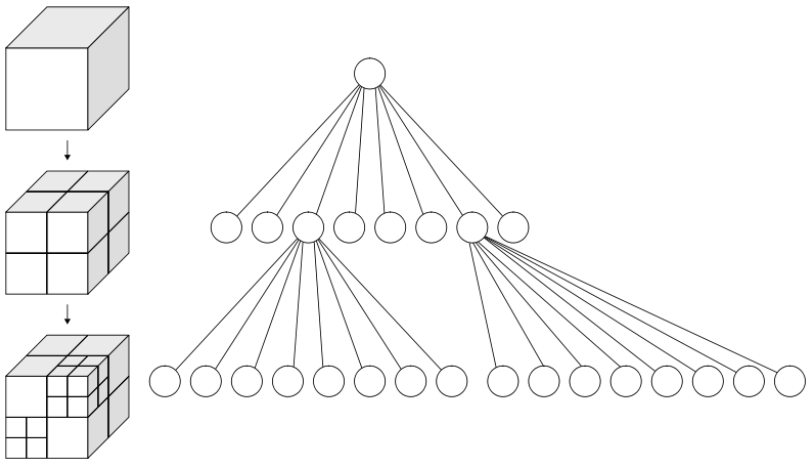

Figure 3.14: Octree visualization. Octree partitions three dimensional space by recursively subdividing it into eight octants. Image taken from http://en.wikipedia.org/

The octree data structure enables to efficiently merge point cloud data from several sources. Point clouds are huge data sets describing three dimensional points, and they have for each point in the point cloud information like distance, color, normals, etc. Additionally, they can be created at high rate and because of that they can occupy a significant amount of memory. These are the reasons for compressing the point clouds and merging in the octree.

Another advantage that ctrees are offering is that at each cell of the octree, the certain value can be stored, which is the same for all the 3D points which belong to the cell.

Chapter 4

Methods

4.1 Image Annotation Tool

Image annotation tool enables cropping of the objects from the background and annotating them in a semi-automatic way. At the input, the tool receives multiple rgb images taken from different camera views for the same scene. For each rgb image, there is associated depth image, transformation between camera views and a point cloud. This data is provided by RGB-D SLAM method [26], using the Kinect camera [11]. After the objects are successfully segmented from the background, the class label is associated to each object in the image. Object classes are represented in the image by using different colors for each class. For example, if one scene consists of a set of 50 images with barrel, canister, human and a euro palette which all need to be annotated in the image, the tool should segment all mentioned objects from the background and assign corresponding class to each object in all 50 images like shown in Figure 4.1.

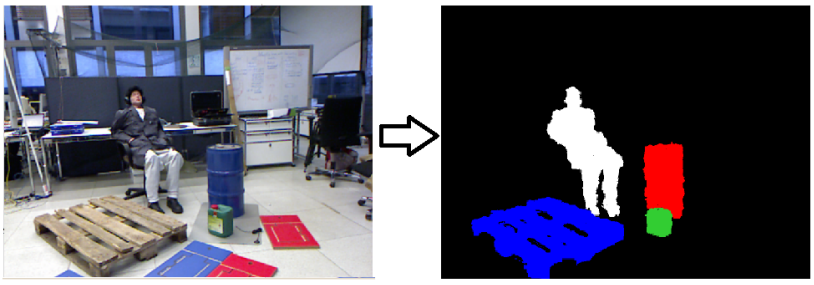

Figure 4.1: Example of annotated objects:barrel, canister, human and a euro palette

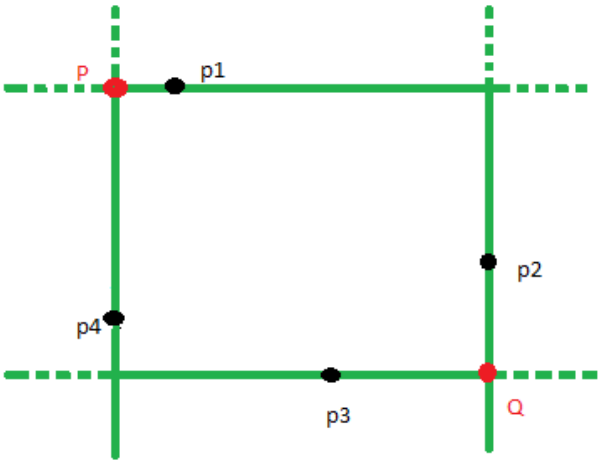

Figure 4.2: Constructing the rectangle box of the 4 projected points $p1, p2, p3, p4$, which are left upper most pixel, right upper most pixel, down left most pixel, and down right most pixel. The points P and Q are the left upper and down right points of the constructed rectangle box.

4.1.1 Tools

Image annotation tool is providing a set of tools which assist the user to segment and annotate multiple images at the same time. Some of these tools are using only rgb image and some are using the depth image to help the user to segment the objects in the image.

1. **Set rectangle**: Sets the rectangle around the object in the rgb image. The user selects first the upper left pixel of the rectangle and then the down right pixel and the rectangle is set using these two points:

$$rect_x = p_x \tag{4.1}$$

$$rect_y = p_y \tag{4.2}$$

$$rect_w = q_x - p_x \tag{4.3}$$

$$rect_h = q_y - p_y \tag{4.4}$$

Here, $rect_x$ and $rect_y$ stand for a upper left pixel of the rectangle box, and $rect_h$ and $rect_w$ are the height and width of the rectangle. p is the upper

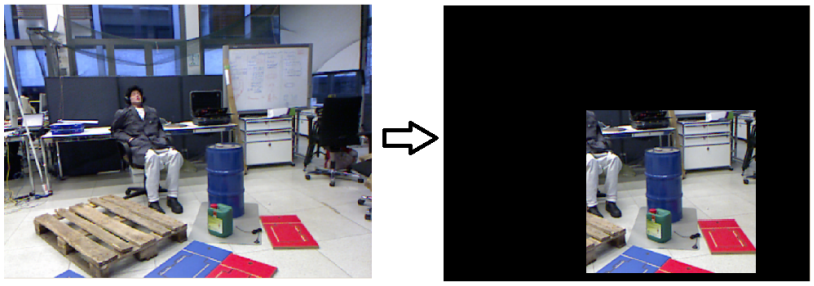

Figure 4.3: Setting the rectangle box around the object. All pixels inside the box are not labelled and out of box assigned as background and cannot be changed by other tools.

left point and q is the lower right point chosen by the user. All the pixels around the rectangle are assigned as background label, and all inside the rectangle are not labelled. The purpose of this tool is to segment the most parts of the image, and assign it as the background, and all other tools which will be used are applied only on the pixels inside the rectangle. This speeds up the time of executing the tools because the tools are applied on less pixels, and also makes sure that the pixels out of the rectangle are staying as the background (other tools cannot label them incorrectly). The example of applying the rectangle box tool can be seen in Figure 4.3.

After the rectangle tool is applied on the seed image, all the pixels inside the box are projected to all other images. Then the rectangle boxes are set on all other images by choosing the left upper most pixel, right upper most pixel, down left most pixel, and down right most pixel. The construction of the projected rectangle box can be seen in Figure 4.2:

The problems of projecting the rectangle to other images is that sometimes the part or most of pixels inside the rectangle box of the seed image are not visible in the images which are taken from totally different views, so the rectangle box would cut off some parts of the object. Another problem is that the shiny or black objects don't have depth values in some parts, thus it is not possible to project pixels from the seed image to the parts which are missing depth values and this can cause that the rectangle box cuts off the part of the object.

2. **Extract plane**: Assigns all the pixels as the background which have the property to belong to the same plane as the seed pixel which is selected by the user. First, the normals of all pixels in the image are calculated using the point cloud of the corresponding image. Normals are smoothed with the block size 25 in order to get more precise normals. Then all the pixels in the rgb image which belong to the same plane as the seed pixel, are on

the similar height and have the similar orientation of the normal as the seed pixel are assigned the background class. In order to see if a point belongs to the same plane as the seed point, the following conditions needs to be met:

$$(p_x - seed_x) \cdot \vec{n_x} \leq \epsilon \tag{4.5}$$

$$(p_y - seed_y) \cdot \vec{n_y} \leq \epsilon \tag{4.6}$$

$$(p_z - seed_z) \cdot \vec{n_z} \leq \epsilon \tag{4.7}$$

Here, *seed* is the point clicked by the user p is the point in the image for which is checked if it belongs to the same plane as the seed point and ϵ is a small constant. This means that the point p is belonging to the same plane as the *seed* point, because the vector drawn from p to *seed* is perpendicular to the normal n, as it is visualized in Figure 4.4. The second condition which needs to be met that the point belongs to the same plane as the seed point, is that their normals should have similar orientation:

$$\alpha = acos(\vec{n_{seed}} \cdot \vec{n_p}) \tag{4.8}$$

Here, α is the angle between the normal of the point p and *seed* normal. If this angle is less then a small constant, then it means that both points belong to the same plane.

This tool is very useful in a scenes where the object is standing on the support plane, then if the user selects the seed point from the support plane, the whole support plane will be assigned the background class, and then the *Depth foreground region* can be used to assign all the object pixels to the foreground class. For this tool, the threshold for the height of the plane and the block size for smoothing the calculation of normals can be adjusted. The problem of this tool is that it assigns always a few millimetres of the bottom of the object as the background. The big advantage of this tool is that it extracts the support plane no matter which color it has, and after applying this tool if the object of interest is not touching other objects in the image, the whole object is surrounded by the background pixels and then it is easy to label the object as the foreground using the tool $usedepthforeground$. The example of the plane segmentation can be seen in Figure 4.5:

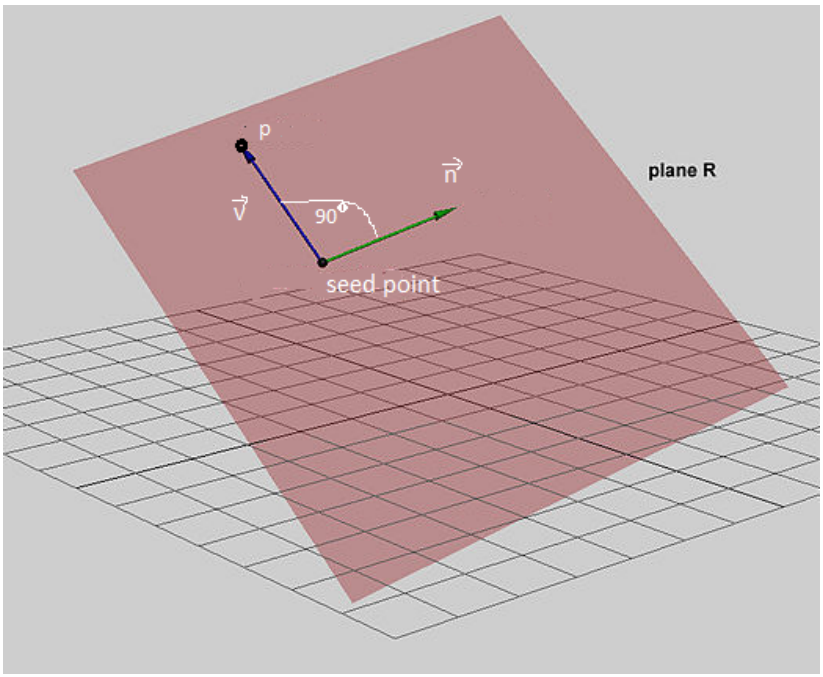

Figure 4.4: Point p is belonging to the same plane as the *seed* point, because the vector drawn from p to *seed* is perpendicular to the normal n

3. **Color foreground region**: Uses the seed pixel taken by the user, and assigns it as the foreground. It assigns further all the neighbouring pixels as the foreground class which have the rgb channels value similar as the seed pixel. This tool is useful in the scenes where the object has the color different from the background as can be seen in the example in Figure 4.6.

$$seed_c h - lowThres_c h \leq neighbor_c h \leq seed_c h + upThres_c h \tag{4.9}$$

Here, $seed_c h$ represents the one of the rgb color channels intensity value of the seed pixel from the rgb image, $neighbor_c h$ stands for one of the eight neighbouring pixels channel value. $lowThres_c h$ and $upThres_c h$ are the lower and upper thresholds for the specific color channel, which can be modified by the user.

4. **Color background region**: similar like the *Color foreground region* but assigned the background class to the pixels. The tool is useful for the situations when the background has specific color and it can be easily annotated as the background. The example of using this tool is shown in Figure 4.7.

5. **Depth foreground region**: Uses the depth image to assign pixels of the object as the foreground. Usually, the objects have the depth jump between

Figure 4.5: Example of extracting the whole support plane of the object barrel. After applying this tool, the tool $usedepthforeground$ can easily annotate the object as the foreground.

the object itself and the background. The tool assigns first the seed point as the foreground and then all the neighbours as the foreground if they satisfy the following condition:

$$seed_d - lowThres_d \leq neighbor_d \leq seed_d + upThres_d \tag{4.10}$$

Here, $seed_d$ represents the depth value of the seed pixel in the depth image, $neighbor_d$ stands for one of the neighbouring pixels depth value. $lowThres_d$ and $upThres_d$ are the lower and upper thresholds for the depth value.

6. **Depth background region**: works similar like the *Depth foreground region* but it assigns pixels the background class. For all tools which extract the regions using color or depth values, it is possible to change the lower and upper thresholds. This enables the user to take advantage of these 4 tools even when the color or depth jump between the object and the background is very low.

7. **Assign class**: Assigns all foreground pixels in the image the chosen class label. The class label is represented with the specific color, so all the foreground pixel are given that color. All the background pixels are given the black color which always stands for the background class. The example can be seen in Figure 4.1

Figure 4.6: Assigning the whole region as foreground, which has the similar rgb channel values as the seed pixel.

8. **Add background pixel**: assigns the pixel chosen by the user as the background class. The tool is useful when small parts of the background are wrongly assigned the foreground label, and this tool can fix the this incorrectly labelled parts. This usually happens after applying the tool $usedepthasforeground$, which labels the most of the object as the foreground, but also some parts of the background which have the similar depth value as the object. Example of this tool is shown in Figure 4.9.

9. **Add foreground pixel**: assigns the pixel chosen by the user as the foreground class, meaning that the pixel belongs to one of the objects of interest. Example is shown in Figure 4.10. This tool is useful for example after applying the *extractplane* tool, the whole support plane is assigned the background class label, but this tool also labels few millimetres of the bottom of the object as the background, so the user can manually correct these parts.

Figure 4.7: Assigning the whole region as background, which has the similar rgb channel values as the seed pixel.

4.1.2 Projection of a Seed Point to Other Images

The main advantage which the Annotation tool is giving is that after one tool is used on one image, the seed point chosen by the user is projected to all other images in the scene and the same tool is applied. For example, if the tool *Extract plane* is used on one image, the plane is extracted first on that image, and then the seed point is projected to all other images and the *Extract plane* tool is used on all images using projected points.

The projection of the seed point from one image to another image is done using the corresponding transformations from the RGBD-SLAM method [26]. Each image has associated transformation from the initial position from the camera. It is represented with a quaternion $(q1, q2, q3, q4)$ which stands for a orientation from the initial camera view and a position vector (x, y, z) which stands for a distance from the initial camera view. The projection of a pixel from the image where the user applied a tool onto all other images is done by applying next 3 steps:

1. A 3D point is calculated for the 2D seed pixel from the corresponding point cloud.

2. The 3D point of a seed pixel, is transformed from the seed coordinate frame to the coordinate frame of the image where the seed pixel is projected. All images have the position of the camera described by the transformation

Figure 4.8: Assigning the whole parts of the object as foreground, for all pixels which have the similar depth values as the seed pixel.

from the initial camera view, which is represented by the orientation and translation from the initial camera view. In order to transform a 3D point of a seed pixel to the other image coordinate frame, we need first to apply the transformation to the initial coordinate frame, and then from the initial coordinate frame the inverse transformation is applied. For example, let an image where the tool is applied be a F_1 frame, and image where we want to project the seed point F_2 frame, and initial frame is F_0. The transformation matrix from F_1 frame to the initial frame is $T_{1,0}$ and the transformation from the frame F_2 to the initial frame is $T_{2,0}$. So on the seed 3d point from the frame F_1 is first applied $T_{1,0}$ transformation, and then inverse $T_{2,0}^{-1}$ transformation because once the seed 3d point is transferred to the initial coordinate frame, it needs to be transferred to the coordinate frame F_2. The transformation between different coordinate frames is visualized in Figure 4.11.

$$\begin{bmatrix} X_2 \\ Y_2 \\ Z_2 \\ 1 \end{bmatrix} = T_{2,0}^{-1} \cdot T_{1,0} \begin{bmatrix} X_{seed} \\ Y_{seed} \\ Z_{seed} \\ 1 \end{bmatrix} \tag{4.11}$$

Here the 3D points (X_2, Y_2, Z_2) and $(X_{seed}, Y_{seed}, Z_{seed})$ are represented in a homogeneous coordinates by adding 1 as the 4th coordinate, because they are multiplied with the transformation matrix which is 4x4 matrix. The

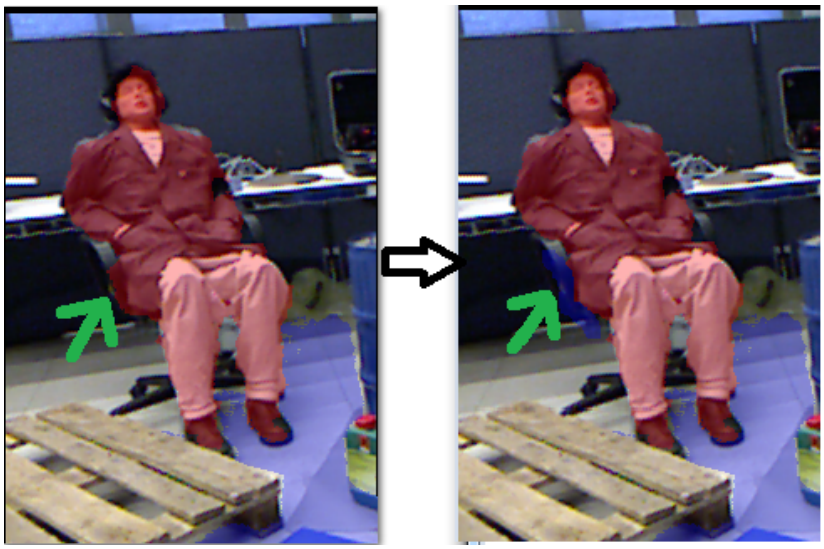

Figure 4.9: Adding a background pixels at the places where the foreground pixels were added wrongly. This happens for example after applying the *usedepthtool*, which is assigning the foreground pixels to the most pixels belonging to the object, but also some additional pixels wrongly assigns as the background.

transformation matrices are constructed using quaternion $(q1, q2, q3, q4)$ which stands for a orientation from the initial camera view and a position vector (x, y, z) which stands for a distance from the initial camera view. The quaternion is transformed to the rotational 3x3 matrix R and the transformation matrix is constructed in the following way:

$$T = \begin{bmatrix} R & t \\ 0 & 1 \end{bmatrix} \tag{4.12}$$

The transformation matrix T is actually the rotation-translation matrix $[R|t]$ from the equation 3.13.

3. The transformed 3D point is then projected onto the image plane using the pinhole camera model, which is explained in the section 3.4.

The rotation-translation matrix $[R|t]$ from the equation 3.13 is constructed using the transformation between the current image viewed and the image onto the seed pixel is projected. The quaternion $(q1, q2, q3, q4)$ is transformed to the rotation matrix R and the position vector (x, y, z) is a translation t.

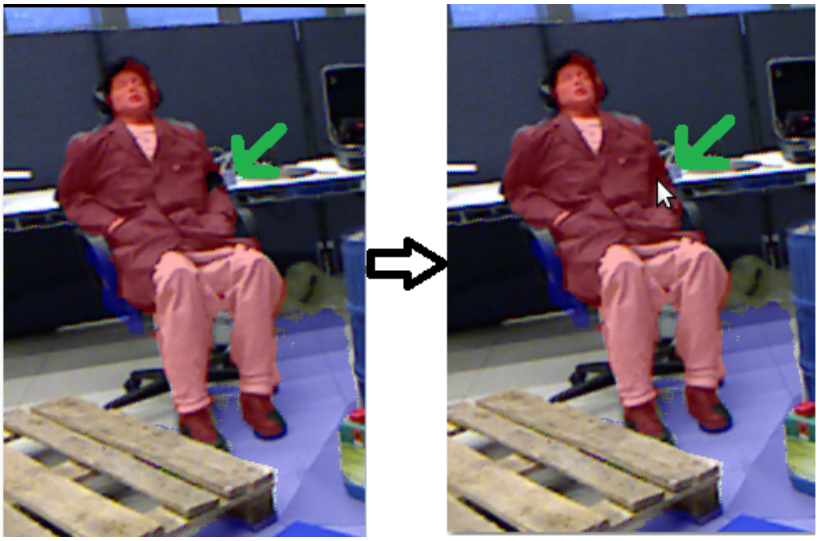

Figure 4.10: Adding a foreground pixels at the places where the foreground pixels were added incorrectly. This tool is helpful for example after applying the *usedepthtool*, some pixels of the object are not assigned as foreground nor background, so the user can easily add the missing pixels as foreground.

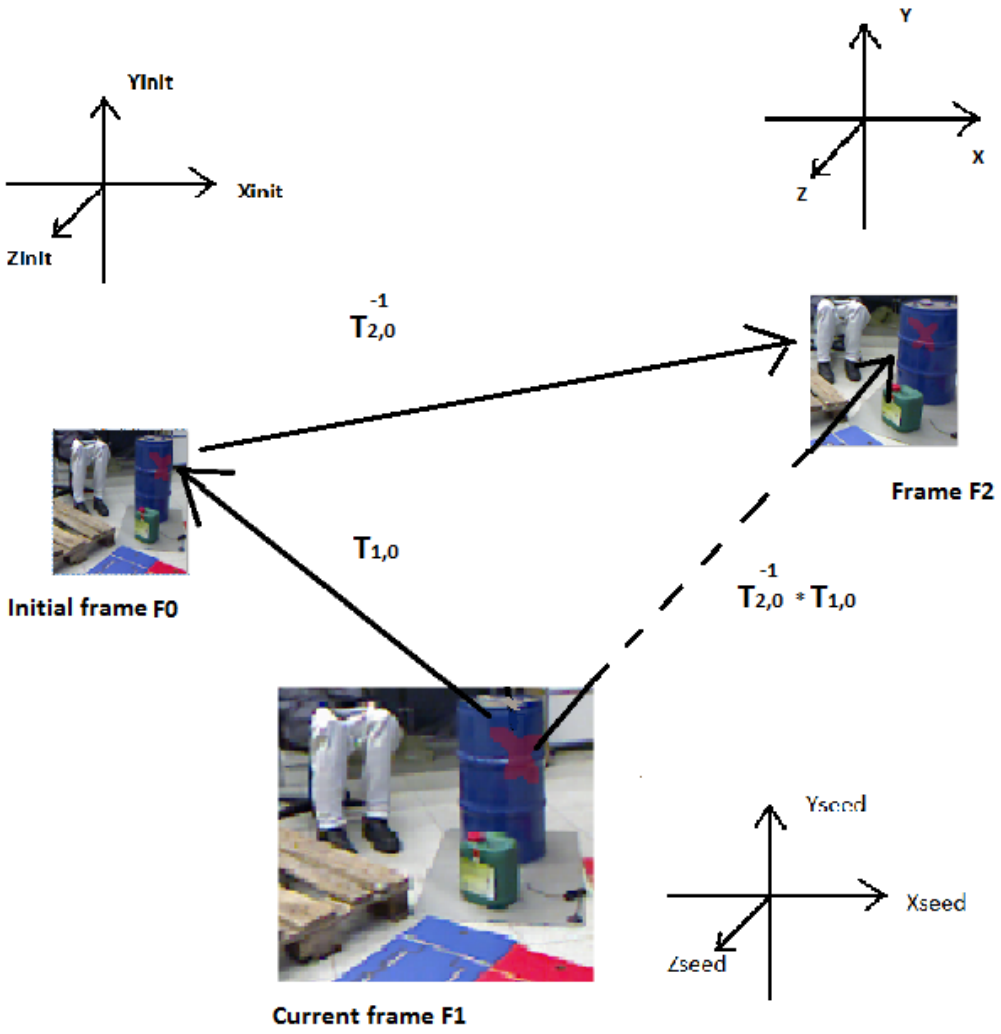

Figure 4.11: The 3D point of the seed coordinate frame is transformed from the seed coordinate frame to the initial frame by applying the transformation $T_{1,0}$ and then from the initial coordinate frame to the frame F_2 by applying the transformation$T_{2,0}^{-1}$. This is analogue as applying directly the transformation $T_{2,0}^{-1}T_{1,0}$ on the seed 3D point

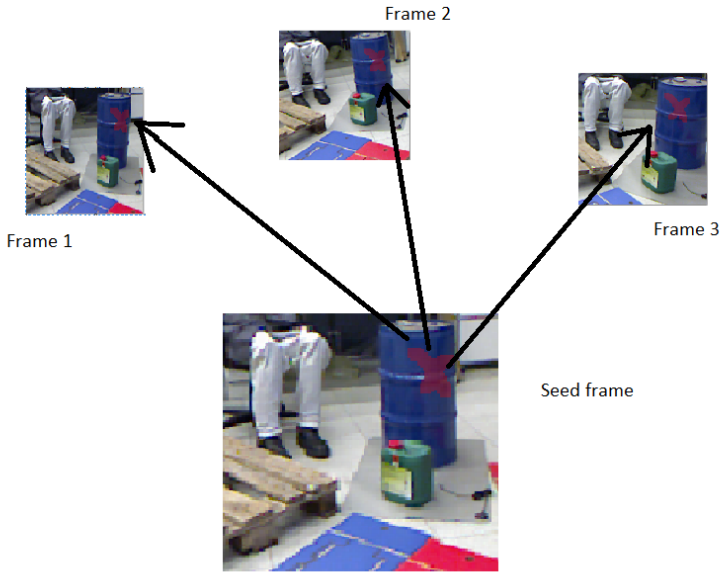

Figure 4.12: The foreground pixels are added to the seed image, and the tool is automatically projecting these foreground pixels to the frames 1, 2 and 3.

4.2 Randomized Decision Forests

For the object recognition method, I used randomized decision forests. This method is chosen because, it is a precise, fast, and robust object recognition method. A query pixel of the image, can be classified by traversing the decision trees from root to leaf node, visiting only several nodes (depending on the depth of the tree) and computing several feature functions. It is scale and translation invariant, but it is not rotational invariant. Randomized Decision Forests can be implemented on GPU, the features can be computed parallel on multiple GPU kernels, which would speed up the training of Randomized Decision Forests and recall of the image.

The Randomized Decision Forests is used for soft labelling of the image pixels, by assigning to each pixel of the image the class distribution (the probabilities that a pixel belongs to the certain class).

In my approach, Randomized Decision Forests are using the rgb and depth images from the Kinect sensor. At each split node, the color and depth features are used. During the computation of the features, the individual pixels are compared, and also regions are compared.

We train the decision trees in a depth-first manner by choosing feature parameters θ and a threshold τ at each node and splitting the pixel set Q accordingly into left and right subsets Q_l and Q_r:

$$\begin{aligned} Q_l(\theta,\tau) &:= \{q \in Q | f_\theta(q) < \tau\}\,, \text{ and} \\ Q_r(\theta,\tau) &:= \{q \in Q | f_\theta(q) \geq \tau\}\,. \end{aligned} \tag{4.13}$$

Since the parameter space cannot be evaluated analytically, we sample P random parameter sets and thresholds (e. g., $P = 2000$) and select feature and threshold that yield maximal information gain

$$I(\theta,\tau) := H(Q) - \sum_{s\in\{l,r\}} \frac{|Q_s(\theta,\tau)|}{|Q|} H\left(Q_s(\theta,\tau)\right), \tag{4.14}$$

where $H(Q) := -\sum_{c\in\mathcal{C}} p(c|Q)\log_2\left(p(c|Q)\right)$ is the shannon entropy of the distribution of training class labels in pixel set Q. This splitting criterion finds feature parameters and threshold that most distinctively seperate the pixel set at a node. Each node is split until a maximum depth is reached in the tree, or the number of pixels lies below a minimum support threshold.

At each leaf node l, we want to maintain the distribution $p(c|l,D)$ of pixels of class c that arrive at the node from the original training set. Since we train the decision tree from pixels with equally distributed class labels, we actually measure the class distribution $p(c|l,Q)$ of training pixels Q at the leaf, i. e.,

$$p(c|l,Q) := p(c(q)|l, q \in Q) = p(c(q)|l, q \in Q, q \in \mathcal{D}). \tag{4.15}$$

The distribution of interest can be obtained by applying Bayes rule:

$$\begin{aligned} p(c|l, Q, \mathcal{D}) &= \frac{p(q \in Q|c(q), l, q \in \mathcal{D})\ p(c(q)|l, q \in \mathcal{D})}{p(q \in Q|l, q \in \mathcal{D})} \\ &= \frac{p(q \in Q|c(q), q \in \mathcal{D})\ p(c(q)|l, q \in \mathcal{D})}{p(q \in Q|q \in \mathcal{D})}. \end{aligned} \tag{4.16}$$

For the desired distribution we obtain

$$p(c(q)|l, q \in \mathcal{D}) = \frac{p(c(q)|l, q \in Q)\ p(q \in Q|q \in \mathcal{D})}{p(q \in Q|c(q), q \in \mathcal{D})} \tag{4.17}$$

We can further reformulate the probability of a pixel of class c to be included in the class-equalized training data Q to

$$p(q \in Q|c(q), q \in \mathcal{D}) = \frac{p(c(q)|q \in Q)\ p(q \in Q|q \in \mathcal{D})}{p(c(q)|q \in \mathcal{D})}, \tag{4.18}$$

and obtain

$$p(c(q)|l, q \in \mathcal{D}) = \frac{p(c(q)|l, q \in Q)\ p(c(q)|q \in \mathcal{D})}{p(c(q)|q \in Q)}. \tag{4.19}$$

By design, $p(c(q)|q \in Q)$ is uniform among class labels and, hence, we incorporate the distribution of classes in the complete training set into the leaf distributions through

$$p(c|l, D) = \eta\ p(c|l, Q)\ p(c|\mathcal{D}), \tag{4.20}$$

where $\eta := p(c|Q) = 1/|\mathcal{C}|$.

4.2.1 Depth Feature

For a pixel q, the depth image feature compares the depth at normalized query positions in the image:

$$f_\theta(q) = d\left(q + \frac{u}{d(q)}\right) - d\left(q + \frac{v}{d(q)}\right) \tag{4.21}$$

Here, $d(q)$ stands for a depth at the pixel q in an image. Parameters $\theta = (u, v)$ stand for pixel offsets. The normalization of the offsets by $\frac{1}{d(q)}$ makes the features scale invariant (modulo perspective effects). If an offset pixel has no valid depth reading or lies beyond the image, the depth $d(q)$ is set to a large positive value.

Individually each feature is weak and it gives only a small information about which class the pixel belongs to, but combining all features at all nodes, in a decision forest they are sufficient to accurately classify the pixel. The feature is computationally efficient, because it only reads at most 3 image pixels and performs at most 5 arithmetic operations.

4.2.2 Color Feature

For a query pixel q, the color image feature compares random two of three channels of the color image represented in the CIE Lab color space at scale-normalized query positions:

$$f_\theta(q) = c_1\left(q + \frac{u}{d(q)}\right) - c_2\left(q + \frac{v}{d(q)}\right) \tag{4.22}$$

Figure 4.13: In this Figure, the two offsets u and v are both on the object, and the depth difference or color difference between them is not big.

In the last equation c_1 and c_2 stand for a randomly chosen color channels, $d(q)$ stands for a depth at the pixel q in an image. Parameters $\theta = (u, v)$ stand for pixel offsets. A Lab color space (CIE Lab) is a color-opponent space with dimension L for lightness and a and b for the color-opponent dimensions. The reason why the rgb image is converted to the CIE Lab color space is because for the color features, it gives better performance. In Figure 4.15 the example of region feature is illustrated.

4.2.3 Region Depth Feature

For a query pixel q, a region depth feature compares the average depth regions at normalized query positions in the image:

$$f_\theta(q) = avg^d\left(q + \frac{u}{d(q)}\right) - avg^d\left(q + \frac{v}{d(q)}\right) \tag{4.23}$$

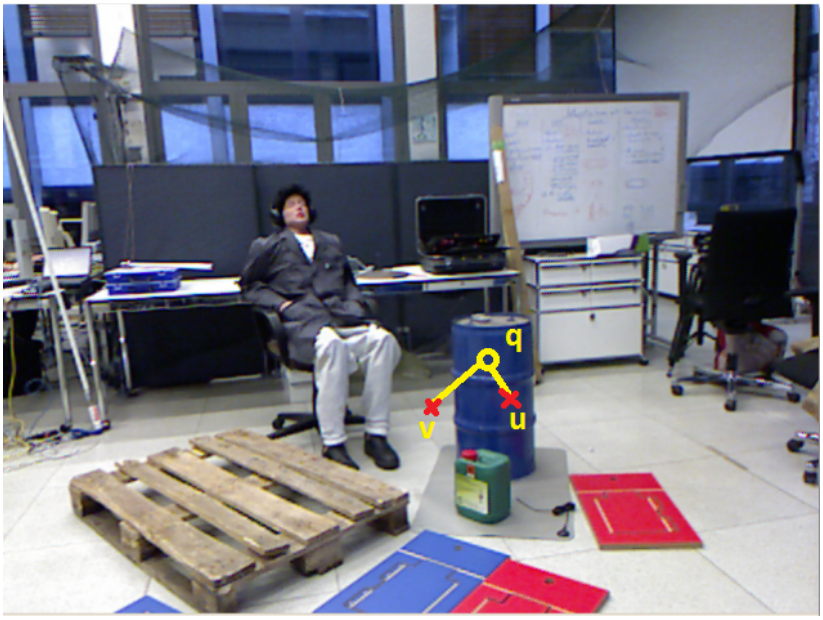

Figure 4.14: In this Figure, the offsets u is on the object and v is on the floor (background), and the depth difference or color difference between them is big.

This feature is similar as the $depthfeature$, with the difference that depth function $d(q)$ is replaced with average depth region avg^d. The region is defined by its width and height, which are randomly chosen at each node. Parameters $\theta = (u, v, w_1, h_1, w_2, h_2)$ stand for pixel offsets u and v, and for a width w_1 and height h_1 of the region of the first offset pixel, and width w_2 and height h_2 of the region of the second offset pixel. The purpose of comparing the average depth regions instead of comparing the individual pixels is to neutralize the Kinect camera sensor noise.

4.2.4 Region Color Feature

For a query pixel q, a region color feature compares the average channels value regions at normalized query positions in the image:

$$f_\theta(q) = avg^{c_1}\left(q + \frac{u}{d(q)}\right) - avg^{c_2}\left(q + \frac{v}{d(q)}\right) \tag{4.24}$$

In the last equation avg^{c_1} and avg^{c_2} stand for a randomly chosen lab channels, $d(q)$is giving the average depth region for a pixel q in an image. Parameters $\theta = (u, v, w_1, h_1, w_2, h_2)$ stand for pixel offsets u and v, and for a width w_1 and height h_1 of the region of the first offset pixel, and width w_2 and height h_2 of the region of the second offset pixel. The purpose of comparing the CIE Lab channel value regions instead of comparing the individual pixels is to neutralize the Kinect camera sensor noise of the color images.

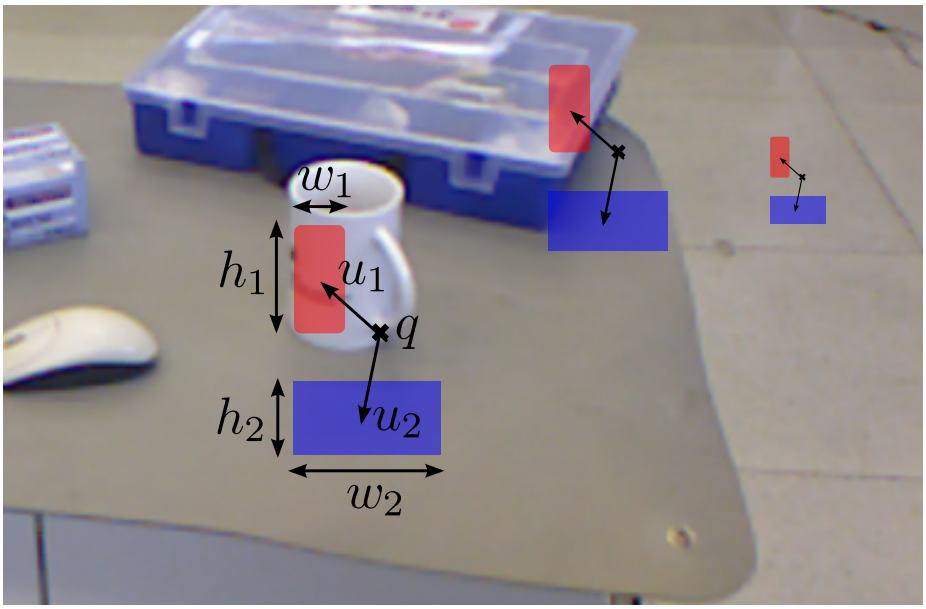

Figure 4.15: Region color and depth features in the random decision classifier compare the average values in two regions relative to the query pixel q. I normalize for perspective scale changes in the image by exploiting the dense depth available in RGB-D images. I scale relative offset locations u_i and region extents w_i, h_i by the inverse of the depth $d(q)$ measured at the query pixel.

4.2.5 Average Region Calculation Using Integral Images

Average depth and color regions are calculated using the integral images to speed up the calculation, since the calculation of the features during the training consumes the most of the time. For each pair of color and depth image, one integral image is constructed consisting of five channels. Three channels are the rgb channels of the color image, one channel is for the depth image, and one is used for counting how many elements in the region have non nan depth value. The value of each channel for the pixel (x,y) of the integral image is the sum of the elements of sub-matrix for the corresponding channel from the element (0,0) to the element (x,y). In Figure 4.16 the integral image is visualized.

The average color region from the integral image for the pixel (x,y) is calculated in the following way:

$$sum_{ch}(q) = P_4^{ch} - P_2^{ch} - P_3^{ch} + P_1^{ch} \tag{4.25}$$

$$avg_{ch}(q) = \frac{sum_{ch}(q)}{N} \tag{4.26}$$

Here, $sum_{ch}(q)$ is the sum of the elements in the region for the pixel q of the channel ch of the integral image, and N is the number of the elements in the

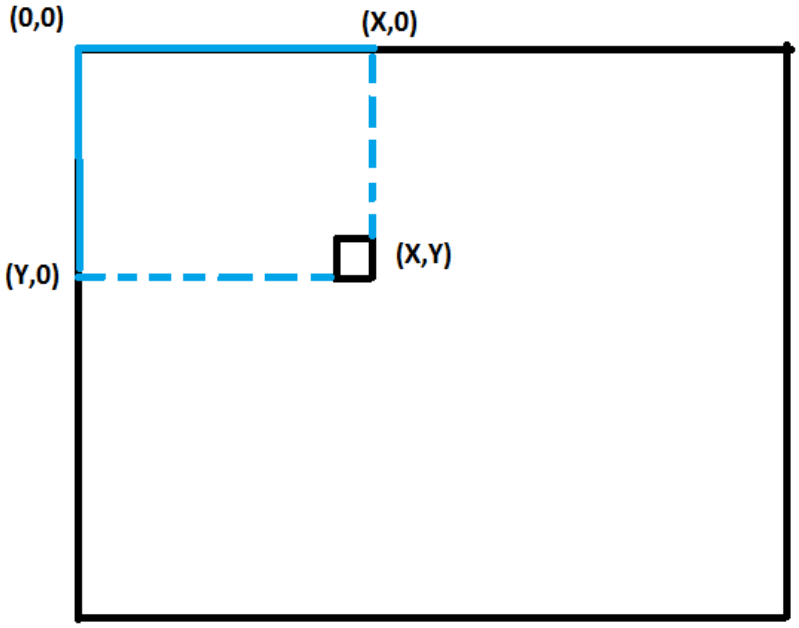

Figure 4.16: Randomized Decision Forests: The value of each channel for the pixel (x,y) of the integral image is the sum of the sub-matrix for the corresponding channel from the element (0,0) to the element (x,y).

region. P_1^{ch} is the upper left element of the region, P_2^{ch} is the down left element of the region, P_3^{ch} is the upper right element of the region and P_4^{ch} is the down right. The elementP_1^{ch} is added in the end, because by subtracting the elements P_2^{ch} and P_3^{ch} from the element P_4^{ch} the region defined by the element P_1^{ch} is twice subtracted and it has to be added once in the end. $avg_{ch}(q)$ is the average color value of the region for the channel ch of the pixel q. The region calculation is illustrated in Figure 4.18.

Depth channel of the integral image has the problem because some pixels in the depth image have nan values (the values which are not valid). To overcome this problem for the calculation of depth channel elements of the integral image, each pixel in the depth image which has the nan value is considered to be zero. Because of this, additional channel needs to be added to the integral image, which counts how many pixels of the depth image in the region have non-nan values. The average depth region from the integral image for the pixel (x,y) is calculated in the following way:

$$sum_d(q) = P_4^d - P_2^d - P_3^d + P_1^d \tag{4.27}$$

$$nonnan(q) = P_4 - P_2 - P_3 + P_1 \tag{4.28}$$

$$avg_d(q) = \frac{avg_d^*(q)}{nonnan(q)} \tag{4.29}$$

Here, $sum_d(q)$ is the sum of the elements of the depth region for the pixel q, $nonnan(q)$ is the number of pixels in the region which have non nan values, and $avg_d(q)$ is the average depth region for the pixel q.

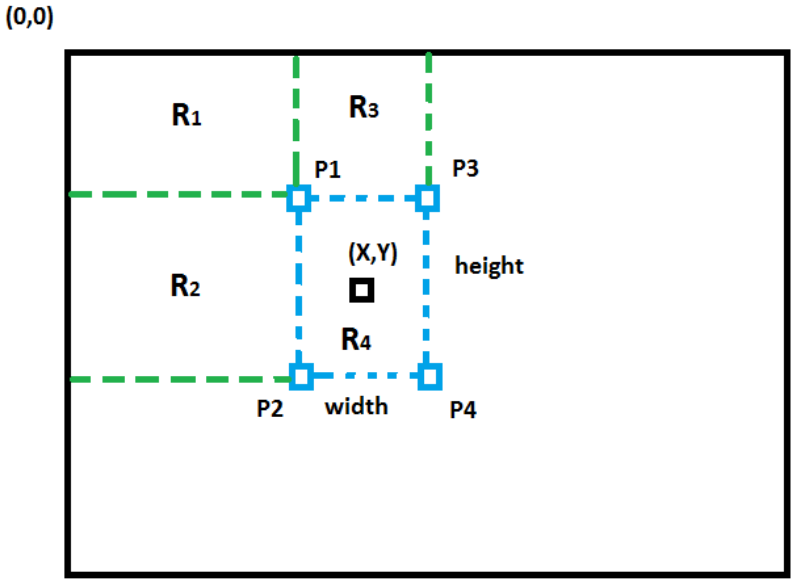

Figure 4.17: Calculation of the average region of the pixel (X,Y) of the integral image. R_1, R_2, R_3, R_4 are the regions defined from the points P_1, P_2, P_3, P_4 to the element (0,0).

4.3 Object-Class Segmentation

After the training of the decision forest is done, all the decision trees are saved to the binary file, so they can be reused in the recall phase. This means that the training is done only once, and can be reused any time in the future for the recall.

The image is softly labelled using the randomized decision forest which means that all the pixels in the image are assigned a class label. For each query pixel of the image, this is done by traversing the query pixel from the top of the tree (root) to the leaf node. At each node in the tree, the pixel is forwarded to the left branch node if the feature function of the node applied on the query pixel is less than a threshold, otherwise the pixel is forwarded to the right branch node. At each branch node, the feature function and a threshold is the one chosen during the training according to the information gain (the feature function and a threshold which separates the pixels at the best way). The traversing of the tree ends when the qeury pixel reaches the leaf node. The distribution associated to the query pixel is the average distribution over all distributions taken from each decision trees:

$$p(c|q) = \frac{1}{K} \sum_{k=1}^{K} p(c|l_k, q) \tag{4.30}$$

Here, $p(c|q)$ is the averaged probability over all decision tree that a query pixel q has a class c, K is the total number of decision trees in the randomized decision forest, and $p(c|l_k, q)$ is the distribution for the class c taken from the decision tree k, at the node l_k.

The query pixel is assigned a class label, by choosing the maximum likelihood class label from the class distribution of the leaf node, where the pixel ended by traversing the tree. For example, if the system is trained on two object classes (e.g. coffee mug, tee box), and the query pixel ends at the leaf node with class distribution p(background | q) = 0.1, p(coffee mug | q) = 0.7, p(tee box | q) = 0.2, then the maximum likelihood class is coffee mug, because it has the highest probability among all in the distribution, and label coffee mug is assigned to the query pixel. If the query pixel has no valid depth measure, than it is not evaluated because all the features are normalized with the depth value of the query pixel.

$$l = \arg\max_{c} p(c|q) \tag{4.31}$$

In the previous equation l is the maximum likelihood class label which is assigned to the pixel q, and $p(c|q)$ is the averaged probability over all decision trees that a query pixel has the class c.

Precision, recall, and accuracy are the measures used for evaluating the performance of the softly image labelling. Precision of class c is defined as the number

of correctly classified pixels in the image as the class c over the total number of pixels classified in the image for the class c:

$$precision_c = \frac{t_p}{t_p + f_p} \tag{4.32}$$

Here, t_p is referred to the true positive, which is the number of correctly classified pixels in the image, and f_p is the false positive, which is the number of pixels in the image, which are wrongly assigned the class c but the teacher class is different than c. Precision has the value 1 if all the pixels which are assigned class c are correctly classified. It can happen that all the pixels which are assigned class c are correctly classified. but not all the pixels from the object class c are labelled as c.

Recall is defined for the class c as the number of correctly classified pixels (as the class c)) over the total number of pixels in the teacher image which are assigned class c:

$$recall_c = \frac{t_p}{t_p + f_n} \tag{4.33}$$

Here, t_p is referred to the true positive, which is the number of correctly classified pixels in the image, and f_n is referred to false negative, which is the number of pixels in the image having the teacher class c, but they are wrongly assigned other class than c. $t_p + f_n$ is the number of teacher pixels in the image for the class c. Recall is showing how good is the object of class c labelled. It will be 1 (the best performance) if all the pixels of the object class c are labelled as class c, no matter if other pixels not belonging to the object class c are assigned wrongly the class c. Recall and precision are only describing the performance if they are shown together.

Accuracy is a measure defined as:

$$accuracy_c = \frac{t_p}{t_p + f_p + f_n} \tag{4.34}$$

where, t_p is referred to the true positive, which is the number of correctly classified pixels in the image, f_n is referred to false negative, which is the number of pixels in the image having the teacher class c, but they are wrongly assigned other class than c, and f_p is the false positive, which is the number of pixels in the image, which are wrongly assigned the class c but the teacher class is different than c. Accuracy is 1 (the best performance) if the f_p and f_n are zero. This happens if all the pixels which have the teacher class c are correctly assigned class c, and no other pixels in the image are assigned class c. Accuracy is a good measure, because it can be compared easily with performance of other algorithms, since it is a single value, and it incorporates both errors from precision and recall.

The precision, recall and accuracy are computed for each class over the whole test dataset, meaning that the number of correctly classified pixels t_p, number of

teacher pixels $t_p + f_n$, and number of incorrectly classified pixels f_p are summed over all images in the test set, and finally the precision, recall and accuracy are computed using these values.

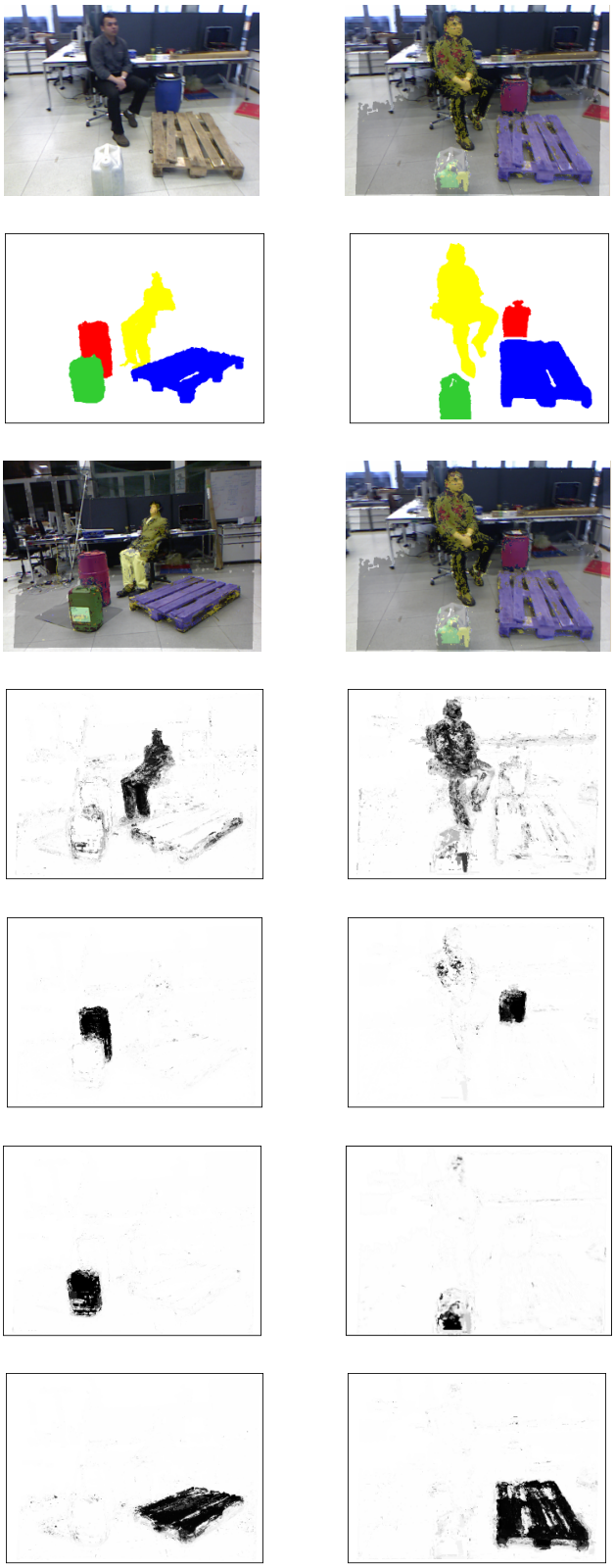

Figure 4.18: Object class segmentation of the images consisting of big objects using randomized decision forests. Left column shows a good classification example. Right column shows bad classification example. Images in the top row are original images, under them are teacher images, followed by classified image, then the image showing the distribution probabilities for objects human, barrel, canister and palette. In the left column all objects are succesfully labelled, and in the right column the canister is poorly labelled.

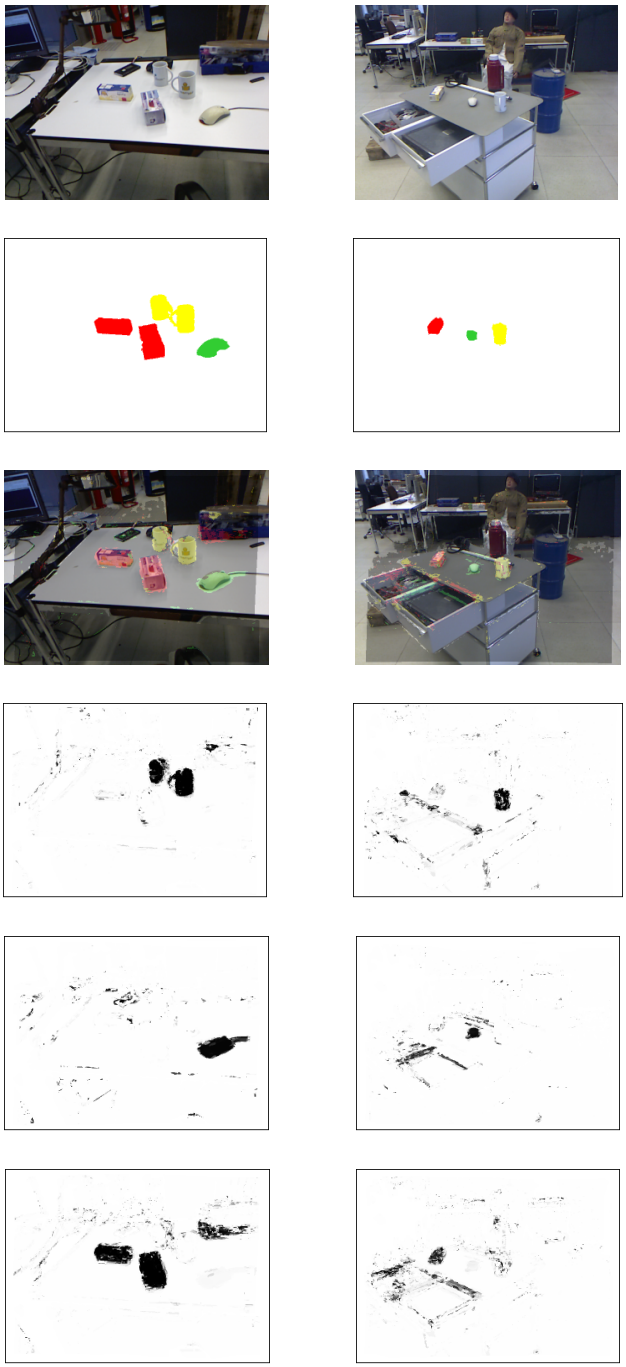

Figure 4.19: Object class segmentation of the images consisting of small objects using randomized decision forests. Left column shows a good classification example. Right column shows bad classification example. Images in the top row are original images, under them are teacher images, followed by classified image, then the image showing the distribution probabilities for objects coffee cup, mouse, and teebox. In the left column all objects are succesfully labelled, and in the right column the objects of interests are generally found in the image, but also the parts of the desk are wrongly labelled

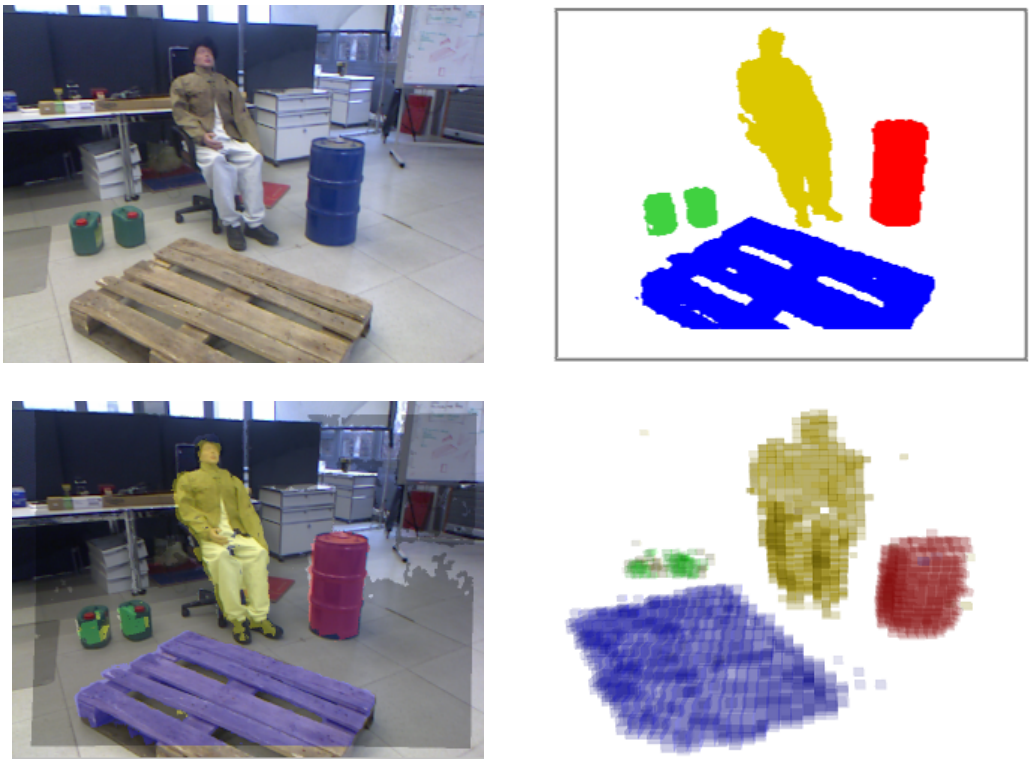

Figure 4.20: Semantic mapping. Upper left: RGB image of a scene. Upper right: Ground truth object-class segmentation. Down left: Back-projected 3D object-class segmentation overlayed on RGB image. Down right: 3D object-class map obtained by fusing multiple views from a SLAM trajectory.

4.4 Semantic Mapping

I integrate my object-class segmentation method with SLAM to fuse the segmentations of individual images in a dense 3D map.

4.4.1 Probabilistic 3D Mapping of Object-Class Image Segmentations

Given the trajectory estimate from the SLAM approach and the depth information in the images, the probabilistic object-class segmentations of the individual views are projected into 3D space and filter this information in a probabilistic octree map. Each voxel v of the octree stores a belief $Bel(c(v))$ that the object class $c(v)$ is present in its volume

$$Bel(c(v)) = p(c(v)|\mathcal{Z}, \mathcal{S}), \tag{4.35}$$

where $\mathcal{Z}$ is the set of RGB-D images with probabilistic labelling and $\mathcal{S}$ is the trajectory estimate. My goal is to integrate segmentation evidence from multiple views in a 3D map and to improve segmentation quality.

I successively project the image pixels into 3D and determine corresponding octree voxels. The belief in the voxel is then updated in a Bayesian framework with the pixel observations $q_{1:N} := \{q_1, q_2, ..., q_N\}$ that fall into the voxel:

$$p(c(v)|q_{1:N}, \mathcal{S}) = \sum_{c(q_1),\cdots,c(q_N)} p(c(v), c(q_1), \cdots, c(q_N)|q_{1:N}, \mathcal{S}) \tag{4.36}$$

Neglecting the known trajectory and applying Bayes rule yields:

$$p(c(v)|q_{1:N}) = \sum_{\cdots} p(c(v), c(q_1), \cdots, c(q_N)|q_{1:N}) \cdot p(c(q_1), \cdots, c(q_N)|q_{1:N}) \tag{4.37}$$

The left term can be further factored using Bayes rule, while for the right term we impose independence between pixel observation. We arrive at

$$p(c(v)|q_{1:N}) = p(c(v)) \sum_{\cdots} \prod_i \eta_i p(c(q_i)|c(v)) p(c(q_i)|q_i), \tag{4.38}$$

where

$$\eta_i = \frac{1}{p(c(q_i)|c(q_{i+1}, \cdots, c(q_N))} \tag{4.39}$$

I approximate $p(c(q_i)|q_i)$ with the output of the RF classifier $p(c(q_i)|q_i, \mathcal{F})$ The probability:

$$p(c(v)) = Bel_0(c(v)) \tag{4.40}$$

incocrporates prior knowledge on the belief. For the distribution:

$$p(c(q_i)|c(v)) = 1_{\{c(v)\}}(c(q_i)) \tag{4.41}$$

I assume a deterministic one-to-one mapping. It follows that

$$p(c(v)|q_{1:N}, \mathcal{S}) = Bel_0(c(v)) \prod_i \eta_i p(c(q_i) = c(v)|q_i, \mathcal{F}) \tag{4.42}$$

which can also be applied recursively.

To back-project the pixel of the image to the octree cell, the corresponding 3D point is taken from the point cloud, and the 3D point is then transformed to the initial coordinate frame (the first image in the scene) and the 3D point is added to the corresponding octree cell. All 3D points are transformed to the initial coordinate frame in order to have all 3D points defined in the same coordinate frames (different images have different coordinate frames). The transformation of the 3D point to the initial coordinate frame is done by multiplying the 3D point in the homogeneous coordinates by the inverse transformation matrix:

$$\begin{bmatrix} X_{init} \\ Y_{init} \\ Z_{init} \\ 1 \end{bmatrix} = T_{n,0} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix} \tag{4.43}$$

Here, $X_{init}, Y_{init}, Z_{init}, 1)$ is the 3D point in the initial coordinate frame, $(X, Y, Z, 1)$ is the 3D point which is transformed to the initial coordinate frame, and $T_{n,0}$ is the transformation matrix from the coordinate frame n to the initial frame. The transformation matrices are constructed using quaternion $(q1, q2, q3, q4)$ which stands for a orientation from the initial camera view and a position vector (x, y, z) which stands for a distance from the initial camera view. The quaternion and a position vector are taken from the RGB-D SLAM method. The quaternion is transformed to the rotational 3x3 matrix R and the transformation matrix is constructed in the following way:

$$T = \begin{bmatrix} R & t \\ 0 & 1 \end{bmatrix} \tag{4.44}$$

The projection of the octree cell to all images belonging to the same scene is done by using the standard pinhole camera model. The cells of the octree are represented in the initial camera position coordinate frame. The 3D position of the cell is first transformed to the coordinate frame of the image where it is projecting, and then the pinhole camera model is used to project a 3D point onto an image plane. Transformation of the 3D point in the homogeneous coordinates $X_{init}, Y_{init}, Z_{init}, 1)$ to the image coordinate frame is done by:

$$\begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix} = T_{n,0}^{-1} \begin{bmatrix} X_{init} \\ Y_{init} \\ Z_{init} \\ 1 \end{bmatrix} \tag{4.45}$$

Here the inverse matrix is used because $T_{n,0}$ is the transformation matrix, which is transforming the point from the nth frame to the initial frame and we here is opposite direction needed.

Chapter 5

Experiments

I evaluate my approach on a datasets containing RGB-D videos of four larger object classes and 3 smaller object classes. The datasets contain 500 training images and 500 test images from 47 and 40 scenes, respectively, with several instances of the object classes in varying configuration. I use precision, recall, and accuracy [8] measures to quantify segmentation quality. I assess the overall accuracy on a testset by summing over the pixel decisions of all classes. Since the background class is semantically different to the object classes, I also measure the segmentation quality of the object classes without background class. In order to assess the quality of the fused semantic maps, I back-project the octree belief over object-classes into the test images.

Table 5.1 shows the average precision, recall and accuracy for the small objects classes in varying configuration. Using only color features gives poor classification performance (accuracy 0.07), since the most small object classes are colorfull (e.g. teabox, cups) and the color feature in this case is not able to recognize the objects. On the other hand, using only depth feature the classification performance is significantly higher than using only color feature. The depth feature is stronger feature than a color feature for the small objects, because the different instances of object classes (e.g. cups, mice) have similar shape and dimensions, but quite different colors. The depth feature is able to recognize all the object classes. The combination of depth and color feature gives slightly worse classification performance comparing to performance using only depth feature. This happens because the color feature for the small object classes is not helping at all in recognizing the objects, in fact it is decreasing the classification performance in most cases by making false decisions in the random decision trees. The combination of color and depth feature has higher performance on the object class mice comparing to using only depth feature, because the mouse is very small object, and there are no big depth jumps between the mouse and the background

Table 5.1: Average precision, recall, and accuracy of max-likelihood object-class segmentation for small objects without background class.

method	precision	recall	accuracy
unnorm. color	0.13	0.14	0.07
norm. color	0.14	0.15	0.07
norm. depth	0.40	0.61	0.32
norm. color + depth	0.35	0.65	0.29
norm. color + depth + 3D	0.64	0.97	0.42

Table 5.2: Average precision, recall, and accuracy of max-likelihood object-class segmentation for small objects with background class.

method	precision	recall	accuracy
unnorm. color	0.95	0.95	0.90
norm. color	0.95	0.95	0.90
norm. depth	0.96	0.96	0.92
norm. color + depth	0.95	0.95	0.91
norm. color + depth + 3D	0.97	0.98	0.95

(mostly due to the noise in the Kinect sensor) so the depth feature alone can't recognize mouse at all, and the combination of color and depth feature is able to recognize mouse. The 3D fusion is significantly outperforming the combination of color and depth features, for the accuracy around 13%. It is interesting that the average recall is improved from 0.65 to 0.97, which means that almost all the pixels belonging to the objects are classified correctly, while the average precision 0.64 is indicating that a part of a background is misclassified as one of the object classes.

Table 5.3 shows average results for different kinds of RF classifiers for the large objects without background class. It can be seen that for the big object classes using only color feature is gives relativly good classification performance. I also conclude that, on this dataset, depth-normalized color is a prominent feature and yields higher accuracy than normalized depth queries alone. The scale normalization of the features using depth enhances the quality of the color features significantly. The color feature is able to recognize the big object classes (unlike small object classes) because the most instances of the big object classes have similar color (e.g. barrels have mostly blue color, pallet have always brown, humans have specific color of the face). I noticed that the humans are overfitted to specific colors. For example, the white color was dominant in most instances of humans (cloth which persons were wearing), and in some cases, the classifier is assigning the human label to the objects in the image which have white color.

Table 5.3: Average precision, recall, and accuracy of max-likelihood object-class segmentation for large objects without background class.

method	precision	recall	accuracy
unnorm. color	0.61	0.48	0.37
norm. color	0.74	0.61	0.51
norm. depth	0.71	0.38	0.33
norm. color + depth	0.78	0.69	0.58
norm. color + depth + 3D	**0.87**	**0.76**	**0.68**

Table 5.4: Average precision, recall, and accuracy of max-likelihood object-class segmentation for large objects with background class.

method	precision	recall	accuracy
unnorm. color	0.80	0.80	0.67
norm. color	0.85	0.85	0.74
norm. depth	0.80	0.80	0.67
norm. color + depth	0.87	0.87	0.78
norm. color + depth + 3D	**0.91**	**0.91**	**0.83**

The classifier is able to recognize humans, as long as the color of the cloth which is human wearing is similar to the cloth of the human instances which were present during the training of the random decision forest. The depth feature obviously doesnt't have problems with colors, but it is not performing better than normalized color feature because the size and shape of the object instances is varying a lot. The combination of color and depth feature is performing better than using only color or depth feature, which indicates that the random decision forest during the training is able to choose the better feature (if color feature is better indicator for barrel which is in most cases blue, than it will choose color feature). The 3D fusion of color and depth image segmentations clearly outperforms the other approaches. It improves on purely image-based segmentations by about 10% for the object classes without background.

Table 5.5 shows the performance of color + depth feature of the small objects, and the performance of the 3D fusion of color and depth image segmentations. It can be seen that the cups are classified significantly better than teaboxes and mice, because the shape of the cups is mostly the same, and cups are in some way rotationally invariant because of the cylindrical shape. The tea boxes are harder to classify because there are very colorful which unables color feature to recognize it, and it is not rotationaly invariant (like cups). The performance of the mouse is the worst, because it is very small so the depth feature can't recognize them, but because of the specific color (black and gray), color feature is able to poorly

Table 5.5: Per-class precision, recall, and accuracy of max-likelihood object-class segmentation for small objects.

	norm. color + depth			norm. color + depth + 3D		
class	prec.	recall	acc.	prec.	recall	acc.
cup	0.42	0.81	0.38	0.76	0.94	0.73
teabox	0.28	0.63	0.24	0.41	0.72	0.35
mouse	0.43	0.47	0.29	0.98	0.20	0.20
background	0.99	0.96	0.95	0.99	0.98	0.97

Table 5.6: Per-class precision, recall, and accuracy of max-likelihood object-class segmentation for large objects.

	norm. color + depth			norm. color + depth + 3D		
class	prec.	recall	acc.	prec.	recall	acc.
palette	0.93	0.84	0.78	**0**.98	**0**.90	**0**.88
barrel	0.92	0.73	0.68	**0**.95	**0**.85	**0**.81
canister	0.74	0.13	0.12	**0**.95	**0**.22	**0**.22
human	0.56	0.59	0.40	**0**.69	**0**.64	**0**.49
background	0.91	0.94	0.86	**0**.92	**0**.97	**0**.89

recognize them.

Table 5.6 is comparing the performance of combination of color and depth feature of the large object classes, and the performance of the 3D fusion of color and depth image segmentations. Overall, the barrels and palletes are recognized in almost all images with a very high accuracy, because they have very specific colors and the color featue is able to recognize them successfully. The canister is classified the worst, becuase there are instaces of canister which have around 50 cm height, and instances which are double smaller which makes the classification harder. Another reason why the canister object class is poorly classified is that there are instances of canister which have white color, and the classifier is overfitted for the humans object class on white color (because most of the humans instances were wearing white cloth), thus the classifier is assigning in most cases the human label on the pixels which actually belong the the canister label. Overall the The 3D fusion of color and depth image segmentations clearly outperforms the performance of using combination of color and depth.

Table 5.7 is comparing the performance of the classifier using different parameters of the minimum support of the leaf node (the leaf node is not split during training if the number of pixels at the node is lower than a minimum support of the leaf node). The best performance is achieved using the minimum support of the leaf node 1000. I conclude that the minimum support of the node 100 is performing worse than 1000 because it is too small number comparing to the

Table 5.7: Average precision, recall, and accuracy of max-likelihood object-class segmentation for small objects without background class for the parameter minimum support of the leaf node.

min support of leaf node	precision	recall	accuracy
100	0.35	0.65	0.29
1000	0.44	0.62	0.35
5000	0.52	0.2	0.18

Table 5.8: Average precision, recall, and accuracy of max-likelihood object-class segmentation for small objects without background class for the parameter maximum depth of the tree.

max depth of the tree	precision	recall	accuracy
10	0.11	0.80	0.10
15	0.35	0.65	0.29
20	0.51	0.45	0.31

total number of input pixels at the root node, so it is overfitting the training data. The minimum support 5000 has the worst performance, because the number of pixels at the leaf node is too large to make a correct decision, and it is indicator that the leaf nodes should be further split in order to recognize the object

Table 5.8 shows the average precision, recall, and accuracy of max-likelihood object-class segmentation for small objects without background class for the parameter maximum depth of the tree (if the node reaches the maximum depth of the tree, it is not split). The maximum depth of the tree 20, shows the best performance among all, but since it just slightly outperforms the performance of maximum depth 15, I would say that maximum depth 15 is better because it generalize better and it classifies the image faster. The more deeper is the tree, it overfitts more the training data. It is interesting that the recall of the maximum depth 10 of the tree is showing very high recall but low precision. It means that the most of the pixels of the objects are classified correctly, but the background is misclassified. This could be possible improved if the prior for the background would be increased (increasing the probability that the pixel belongs to the background class).

Table 5.9 shows the average precision, recall, and accuracy of max-likelihood object-class segmentation for small objects without background class for the parameter maximum feature offset (see Equation 4.21). It shows similar performance for the maximum offsets 30 and 70, but the maximum offset 120 is too large for the small object classes.

Table 5.10 shows the average precision, recall, and accuracy of max-likelihood

Table 5.9: Average precision, recall, and accuracy of max-likelihood object-class segmentation for small objects without background class for the parameter maximum feature offset.

max depth of the tree	precision	recall	accuracy
30	0.35	0.65	0.29
70	0.38	0.61	0.3
120	0.23	0.53	0.19

Table 5.10: Average precision, recall, and accuracy of max-likelihood object-class segmentation for small objects without background class for the different kind of choosing input pixel from training images at the start of the training.

distribution	precision	recall	accuracy
even	0.35	0.65	0.29
not even	0.37	0.54	0.27
120	0.23	0.53	0.19

object-class segmentation for small objects without background class for the different kind of choosing input pixel from training images at the start of the training. Choosing the input pixels evenly (for each class the same number of pixels is chosen) shows better performance than choosing the pixels not evenly. This happens because only small number of pixels are chosen from the small objects like mouse, if the pixels are not chosen evenly.

Chapter 6

Conclusions

In this master thesis, I proposed a novel approach to semantic mapping. I apply object-class image segmentation to recognize objects pixel-wise in RGB-D images. I incorporate depth and color cues into a random decision forest classifier and normalize the features for scale using depth measurements. Based on trajectory estimates obtained with a SLAM method, I fuse the image segmentations into a probabilistic 3D object-class map in order to improve the classification performance of the object-class segmentation. In experiments on two datasets, one for small objects classes and one for large object classes, I demonstrate that my approach provides a 3D segmentation of the object classes, and also significantly improves 2D object-class segmentation performance.

I conclude that the segmentation quality of my approach highly depends on the properties of the underlying object-class image segmentation method. While many other methods exist that demonstrate good segmentation results, the recall efficiency of the segmentation approach is of equal importance for online processing and application in a robotics setting.

The classification performance of the object-class image segmentation method depends on the choice of the training and test images. In order to avoid overfitting (to have good generalization), the large training dataset is needed. It is important that the training dataset consists of many different scenes with different configuration, and that many different instances of objects are present (different colors, shape, size of objects, different background).

The quality of the 3D fusion depends mainly on the quality of the object-class image segmentation method. If the performance of the object-class image segmentation method is poor, then the 3D fusion would make results even worse. On the other hand, if the object-class image segmentation method give good classification performance, then the 3D fusion greatly improves the classification performance. The quality of the 3D fusion also depends on the quality of the

trajectory obtained by the SLAM method. If the trajectory is bad than the 3D fusion makes the classification performance in most cases worse, since the alignment between the frames are bad.

6.1 Future Work

In future work, it would be nice to integrate further descriptive image features like Histograms of Oriented Gradients (HOG)or Fast Point Feature Histograms. In order to scale my approach to larger sets of objects, the combination of multiple random decision forests could be considered. The hierarchical approach would probably help to improve the classification performance, where the category (e.g. canister) would be subdivided into subcategories (e.g. small size canister, large size, green canister, blue canister ...). For some object class (e.g. humans) it would help to subdivide the category into subcategories of different parts of body because they are all specific for each human. The 3D fusion could be improve if the air class would be included in the 3D octree map. All individual images would project the object class pixels to the octree, and beside that, the air would be projected to the octree. The pixels which belong to the air class, are the pixels with around 5cm smaller depth than the object pixels. Beside updating of the cell of the octree using Bayesian approach, the counting procedure could be used.

Bibliography

[1] http://opencv.willowgarage.com. [cited at p. 32]

[2] A. Bosch, A. Zisserman, and X. Munoz. Image classification using random forests and ferns. In *Proc. of the Int. Conf. on Computer Vision (ICCV)*, 2007. [cited at p. 11, 27]

[3] L. Breiman. Random forests. *Machine Learning*, 45(1):5–32, 2001. [cited at p. 11]

[4] R. O. Castle, G. Klein, and D. W. Murray. Combining monoSLAM with object recognition for scene augmentation using a wearable camera. *Image Vision Computing*, 28(11):1548 – 1556, 2010. [cited at p. 9, 10]

[5] J. Civera, D. Galvez-Lopez, L. Riazuelo, D. Tardos, and J. M. M. Montiel. Towards semantic SLAM using a monocular camera. In *Proc. of the IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)*, 2011. [cited at p. 9, 10]

[6] N. Dalal and B. Triggs. Histograms of oriented gradients for human detection. In *Proc. of the IEEE Computer Society Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2005. [cited at p. 12]

[7] N. Engelhard, F. Endres, J. Hess, J. Sturm, and W. Burgard. Real-time 3D visual SLAM with a hand-held camera. In *Proc. of RGB-D Workshop on 3D Perception in Robotics at European Robotics Forum*, 2011. [cited at p. 9]

[8] M. Everingham, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman. The pascal visual object classes (VOC) challenge. *International Journal of Computer Vision*, 88(2):303–338, 2010. [cited at p. 63]

[9] P. Henry, M. Krainin, E. Herbst, X. Ren, and D. Fox. RGB-D mapping: Using depth cameras for dense 3D modeling of indoor environments. In *Proc. of International Symposium on Experimental Robotics (ISER)*, 2010. [cited at p. 9]

[10] A. Ion, J. Carreira, and C. Sminchisescu. Image segmentation by figure-ground composition into maximal cliques. *Proc. of the Int. Conf. on Computer Vision (ICCV)*, 2011. [cited at p. 11, 13]

[11] Microsoft Kinect. 2010. [cited at p. 35]

[12] R. Kuemmerle, G. Grisetti, H. Strasdat, K. Konolige, and W. Burgard. g2o: A general framework for graph optimization. In *Proc. of the IEEE Int. Conf. on Robotics and Automation (ICRA)*, 2011. [cited at p. 23]

[13] L. Ladicky, C. Russell, P. Kohli, and P. H. S. Torr. Associative hierarchical crfs for object class image segmentation. In *Proc. of the Int. Conf. on Computer Vision (ICCV)*, 2009. [cited at p. 11, 12]

[14] Durrant-Whyte Leonard. Mobile robot localization by tracking geometric beacons. [cited at p. 15]

[15] M. Marszatek and C. Schmid. Accurate object localization with shape masks. [cited at p. 12]

[16] D. Meger, P.-E. Forssén, K. Lai, S. Helmer, S. McCann, T. Southey, M. Baumann, J. J. Little, and D. G. Lowe. Curious george: An attentive semantic robot. *Robotics and Autonomous Systems*, 56(6):503–511, 2008. [cited at p. 9, 10]

[17] A. Nüchter and J. Hertzberg. Towards semantic maps for mobile robots. *Robotics and Autonomous Systems*, 56(11):915–926, 2008. [cited at p. 9, 10]

[18] A. Ranganathan and F. Dellaert. Semantic modeling of places using objects. In *Proc. of the Int. Conf. on Robotics: Science and Systems*, 2007. [cited at p. 11]

[19] R. Riedmiller and H.Braun. A direct adaptive method for faster. [cited at p. 13]

[20] F. Schroff, A. Criminisi, and A. Zisserman. Object class segmentation using random forests. In *Proc. of the British Machine Vision Conference*, 2008. [cited at p. 12]

[21] H. Schulz and S. Behnke. Learning object-class segmentation with convolutional neural networks. In *Proceedings of the European Symposium on Artificial Neural Networks (ESANN)*, 2012, to appear. [cited at p. 11, 12]

[22] Wolfram Burgard Sebastian Thrun and Dieter Fox. Probabilistic robotics. [cited at p. 20]

[23] J. Shotton, A. Fitzgibbon, M. Cook, T. Sharp, M. Finocchio, R. Moore, A. Kipman, and A. Blake. Real-time human pose recognition in parts from single depth images. In *Proc. of the IEEE Computer Society Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2011. [cited at p. 11, 13]

[24] J. Shotton, M. Johnson, and R. Cipolla. Semantic texton forests for image categorization and segmentation. In *Proc. of the IEEE Computer Society Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2008. [cited at p. 11]

[25] J. Stückler and S. Behnke. Combining depth and color cues for scale- and viewpoint-invariant object segmentation and recognition using random forests. In *Proc. of the IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)*, 2010. [cited at p. 13, 14]

[26] J. Stückler and S. Behnke. Robust real-time registration of RGB-D images using multi-resolution surfel representations. In *Proc. of the 7th German Conference on Robotics (ROBOTIK)*, 2012, to appear, for review: http://www.ais.uni-bonn.de/papers/robotik2012_mrsreg.pdf. [cited at p. 9, 10, 22, 35, 42]

[27] M. Tomono and Y. Shin'ichi. Object-based localization and mapping using loop constraints and geometric prior knowledge. In *Proc. of the IEEE Int. Conf. on Robotics and Automation (ICRA)*, 2003. [cited at p. 9]

[28] S. Vasudevan, S. Gächter, V. Nguyen, and R. Siegwart. Cognitive maps for mobile robots-an object based approach. *Robotics and Autonomous Systems*, 55(5):359–371, 2007. [cited at p. 9, 10]

[29] M. Hebert W. Burgard. World modeling, in: Springer handbook of robotics. 2008. [cited at p. 3, 31]

[30] H. Zender, O. Martinez Mozos, P. Jensfelt, G.-J.M. Kruijff, and W. Burgard. Conceptual spatial representations for indoor mobile robots. *Robotics and Autonomous Systems*, 56(6):493 – 502, 2008. [cited at p. 9, 10]

List of Figures

List of Tables