TECHNISCHE UNIVERSITÄT
CHEMNITZ

Aufgabenstellung zur Masterarbeit

im Studiengang Master Information and Communication Systems
Fakultät für Elektrotechnik und Informationstechnik
Professur Hochfrequenztechnik und Theoretische Elektrotechnik

zum Thema

Multi-layer random forests with auto-context for object detection and pose estimation

für

Frau Supritha Prasad
geb. am 30. Oktober 1992 in Mysore

DLR Deutsches Zentrum
für Luft- und Raumfahrt

Betreuer : Dr. Ulrich Hillenbrand (DLR)
Prüfer : Prof. Dr. rer. nat. Madhukar Chandra (TUC)
Ausgabedatum : 01.09.2018
Abgabedatum : 08.02.2019
Tag der Abgabe : 07.02.2019

Supritha Prasad
“Multi-layer random forests with auto-context for object detection and pose estimation”
Master Thesis, Technische Universität Chemnitz, 2019

Declaration of Authorship

I hereby certify that this Master thesis is all my own work and uses no external material other than that acknowledged in the text. This is a true copy of the Master thesis, including any required final revisions, as accepted by my supervisor.

Chemnitz, 07.02.2019

Supritha Prasad

Acknowledgement

I wish to express my sincere gratitude to everyone who helped and guided me in completing my thesis.

The research involved in this work and its completion would not have been possible without the timely and appropriate guidance of Dr. Ulrich Hillenbrand. I am highly indebted to him for his invaluable support, constant supervision and granted freedom at every stage of the thesis at the German Aerospace Center (DLR).

I am extremely grateful to Prof. Dr. Madhukar Chandra for his sincere guidance and encouragement extended to me from the Technische Universität Chemnitz.

Further, I wish to express my sincere thanks to Prof. Dr.-Ing. Alin Albu-Schäffer and Dr. rer. nat. Rudolph Triebel for providing me with such a wonderful learning opportunity and be associated with the Institute of Robotics and Mechatronics at DLR. I take this opportunity to also thank all the other members of the Perception and Cognition department for their kind assistance and encouragement.

To my parents Mr. Prasad Mallaradhya and Dr. Latha Shivamurthy, thank you for encouraging me in all of my pursuits and inspiring me to follow my dreams. This thesis is dedicated to my grandparents who I miss dearly - I hope that I am making you all proud down here.

My baby brother, Prathik Prasad, I thank you for being my confidant and I am extremely grateful that I have you in my life.

Finally, I thank my boyfriend Kiran Kamarthi for being there every time I needed him and for encouraging me to be a better me.

Abstract

Deep neural networks are multi-layer architectures of neural networks used in machine learning. In recent years, training such systems through Deep Learning techniques has shown remarkable improvements over more traditional techniques in the domains of image-based classification and object category recognition. However, a well-known downside of Deep Learning is that generally, for training these very-high-dimensional systems, a very large amount of training samples is required.

It is a common understanding that the multi-level structure of data processing that is learned and applied may be a critical factor for the success story of deep neural networks. Highly task-optimized representations may be gradually built through a sequence of transformations that can abstract and hence generalize from the specific data instances.

In this Master Thesis, the aspect of deep multi-layered data transformation will be transferred from neural networks onto a very different learning scheme, the Random Forests. The latter is a highly competitive and general-purpose method for classification and regression. Even large Random Forests usually do not have the high amount of parameters to optimize during training, and the number of hyper-parameters and architectural varieties is much lower than for the deep neural networks. Training with a much smaller amount of samples is hence possible.

Some variants of a layered architecture of Random Forests is investigated here, and different styles of training each forest is tried. The specific problem domain considered here is object detection and pose estimation from RGB-D images. Hence, the forest output is used by the pose hypothesis scoring function and the poses are optimized using RANSAC-based scheme. Experiments on the pose annotated Hinterstoisser and T-LESS datasets prove the performance of the multi-layer random forests architecture.

Keywords: Random Forests, Auto context, RGB-D images, 6D pose estimation, object detection, 3D computer vision, robotics

Contents

Acknowledgement . . . . **iv**

Abstract . . . . **v**

Contents . . . . **vi**

List of Figures . . . . **viii**

List of Tables . . . . **ix**

List of Abbreviations . . . . **x**

1 Introduction **1**
1.1 Proposed object pose estimation pipeline 2
1.2 Motivation 3
1.3 Structure 4

2 Related Work **5**
2.1 Random forests and its variants 5
2.2 Auto-context 6
2.3 Pose estimation 7

3 Abstract Forest Model **9**
3.1 Decision trees 9
  3.1.1 Data points for training and testing 10
  3.1.2 Features 12
  3.1.3 Split function 13
  3.1.4 Tree training 13
  3.1.5 Tree testing 15
  3.1.6 Energy models 15
  3.1.7 Leaf prediction models 17
3.2 Combining trees into a forest ensemble 18
3.3 Key model parameters 21
3.4 Building deep random forests 22
3.5 Auto-context framework 23

4 Implementation **25**
4.1 3D object coordinate labelling 25
4.2 Split node features 27
4.3 Object label and coordinate predictions 29
  4.3.1 Training 29

4.3.2 Testing . . . 30
4.4 Using auto-context information . . . 32
4.5 Multi-layered architecture of random forests . . . 33
4.6 Object pose estimation . . . 34
4.6.1 Pose estimation overview . . . 34
4.6.2 Scoring function . . . 36

5 Datasets . . . 39
5.1 Hinterstoisser dataset . . . 39
5.2 T-LESS dataset . . . 41

6 Experiments and Discussion . . . 46
6.1 Evaluation metrics . . . 46
6.1.1 Pose estimation . . . 46
6.1.2 Object detection . . . 46
6.2 Experimental set-up . . . 47
6.2.1 Without auto-context . . . 48
6.2.2 With auto-context . . . 49

7 Conclusion and Future Work . . . 53

Bibliography . . . 54

List of Figures

1.1 Computer vision algorithms at DLR 1
1.2 Illustration of the pipeline for RGB-D input images. 2

3.1 Decision tree to check if a photo represents indoor or outdoor scene . 10
3.2 Basic notations in a tree 11
3.3 Diagram illustrating a decision boundary 14
3.4 Using information gain to construct a tree 16
3.5 Diagram illustrating a probabilistic leaf model 18
3.6 A random forest testing with 4 trees 19
3.7 Ensemble model posterior $p(y|v)$ 20
3.8 Illustration of a deep forest ensemble 22
3.9 Illustration of the procedures of the auto-context 24

4.1 SCoRF pipeline for camera localization 25
4.2 Object coordinates as dense correspondences 26
4.3 Computing feature responses using a depth image 27
4.4 Depth adaptive pixel difference features 28
4.5 Depth adaptiveness 28
4.6 Proxy classes 30
4.7 Random forest training 30
4.8 Reducing the uncertainty of object label and coordinates predictions . 32
4.9 Visualization of the search for optimal pose 34
4.10 Components of the scoring function 37

5.1 15 objects in the dataset created by Hinterstoisser et al. 40
5.2 Training images for the object duck 40
5.3 Training images for the object cat 40
5.4 Test scene with the object of interest surrounded by dense clutter . . 41
5.5 Objects included in the T-LESS dataset 42
5.6 Sample training images for object 2 43
5.7 Sample training images for object 25 43
5.8 Test scenes overlaid with colored 3D object models at ground truth poses 44
5.9 Sample test images of scene 1 with objects 2, 25, 29 and 30 45

6.1 Results on the two datasets for different tree counts 48
6.2 Results with the auto-context structure for 3 trees in each forest . . . 49
6.3 Results with the auto-context structure for 10 trees in each forest . . 50
6.4 Results at each layer for 10 trees in each forest 50
6.5 Results for pose estimation with the multi-layered architecture 51

List of Tables

6.1 Default training parameter settings 47
6.2 Default testing parameter settings 47
6.3 Accuracy when testing with different number of trees in a forest . . . 48
6.4 Accuracy with different number of layers of forests 49
6.5 Accuracy with different number of layers for tree count 10 49
6.6 Pose estimation results on the Hinterstoisser data set 51

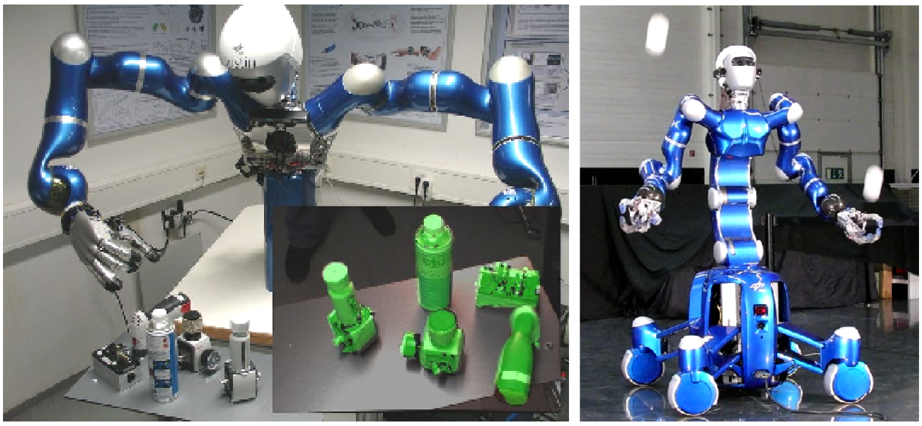
List of Abbreviations

CART	Classification and Regression Trees
DLR	Deutschen Zentrums für Luft- und Raumfahrt (English: German Aerospace Center)
HOG	Histogram of Oriented Gradients
ICP	Iterative Closest Point
ID3	Iterative Dichotomiser 3
IG	Information Gain
IoU	Intersection over Union
LINEMOD	MultiMODal approach for LINEearizing the memory
LRU	Lightweight Rover Unit
MAP	Maximum A Posteriori
MLE	Maximum Likelihood Estimation
RANSAC	RANdom SAmple Consensus
RGB	Image with Red Green Blue colour channels
RGB-D	RGB image with an additional Depth channel
ROBEX	Robotic Exploration Under Extreme Conditions
SCoRe Forests	Scene Coordinate Regression Forests
SIFT	Scale Invariant Feature Transform
T-LESS	Texture-less
ToF	Time of Flight

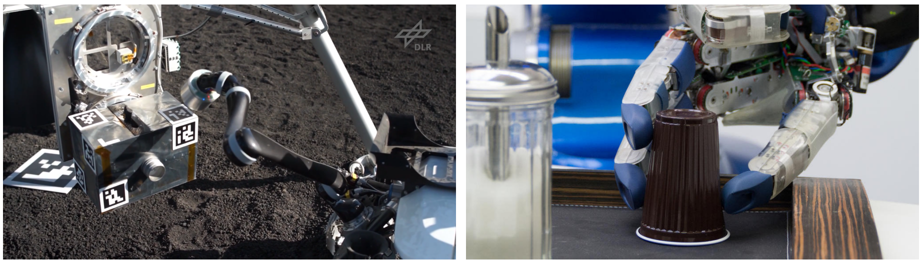
1 Introduction

Recent advances in industrial and service robotics are gradually expanding the application domain of robotics from laboratory settings to a domestic environment. Since a large number of the traditional perception approaches are adapted for fixed environments, the ability of robust interpretation is one of the key components of a robotic system at present.

Many applications such as human-robot cooperation and interaction, autonomous driving and augmented reality require the position and orientation of the neighbouring objects. In current missions at the German Aerospace Center (DLR), this is demonstrated using the robots Rollin' Justin and Lightweight Rover Unit (LRU). The task of object detection and pose estimation is accomplished, for example using geometry-based techniques, circle detectors (for ball catching scenario) or visual markers as shown in Figure 1.1.



(a) Matching objects from a database [1]　(b) Using circle detector [2]



(c) Visual markers in ROBEX [3]　(d) Grasping a plastic cup

Figure 1.1: Computer vision algorithms at DLR

In order to grasp a plastic cup as seen in Figure 1.1(d), the robot has to recognize the cup and estimate it's 6D pose accurately. 6D object pose estimation is the process of determining the 3D translation and 3D rotation from a camera coordinate frame to an object coordinate frame. Translation and rotation are alternative terms for position and orientation. Therefore, estimating the 6D pose is a crucial step in object manipulation tasks. The objects are captured by 2D or 3D sensors and three kinds of information are generally used for recognition - colour (RGB), depth or both (RGB-D).

In this work, a novel vision algorithm is proposed and the specific problem domain considered is the object detection and pose estimation from RGB-D images. In particular, a learning scheme is developed using a multi-layered architecture of random forests and trained to locate and classify the objects in a scene image.

1.1 Proposed object pose estimation pipeline

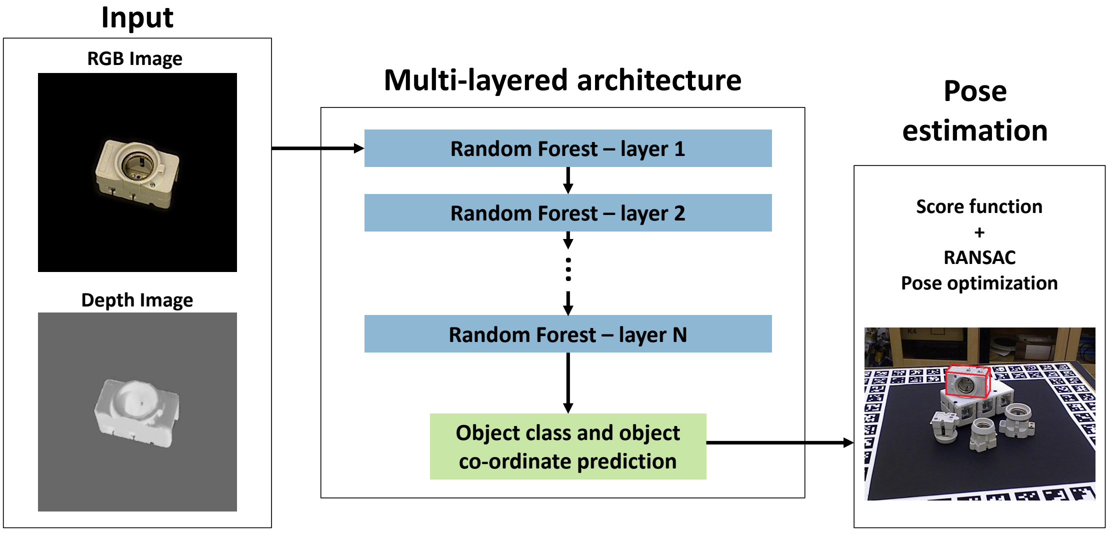


Figure 1.2: Illustration of the pipeline for RGB-D input images.

Inspired by the success of current research, the performance of the new multi-layered architecture is evaluated by estimating the 6D pose of objects. Therefore, the proposed pipeline consists of two main parts: processing through the multi-layered architecture of random forests and estimation the pose.

An RGB-D image is taken as the input and processed by the pipeline as shown in Figure 1.2. In the first step, the multi-layered random forests process the entire image and obtains the model correspondences. This approach is motivated by the recent works where a continuous object-part labelling known as object coordinates is used at different processing stages for pose estimation from a pre-segmented RGB-D image [4, 5].

The basic idea of multiple layers of random forests in [6] has been the starting point of this study. Here, each layer of the random forest receives feature information processed by its preceding layer and outputs its processing result to the next layer. The concept of auto-context is introduced from the second layer, where the image appearances (observed data) are integrated with the context information [7].

The second step consists of pose estimation, where the pose hypotheses are sampled based on the observed depth values and the coordinate predictions from the N^{th} layer of random forest [8]. Subsequently, these hypotheses are refined and a final estimate is chosen.

1.2 Motivation

Deep neural networks are multi-layer architectures of neural networks used in machine learning. In recent years, training such systems through deep learning techniques has shown remarkable improvements over more traditional techniques in the domains of image-based classification and object category recognition.

The multi-level structure of data processing that is learned and applied in the deep neural networks may be a critical factor for their success story. Highly task-optimized representations may be gradually built through a sequence of transformations which are abstract and hence generalising from the specific data instances.

However, a well-known downside of deep learning is that generally, a very large amount of training samples is required for training these very high dimensional systems. Also, another disadvantage of deep learning is their black box nature of not being able to comprehend what is learned. For example, if an image of a book is given as an input to a deep neural network and it predicts it to be a door, it is difficult to interpret the reasons for this wrong prediction. It is easier to understand the cause of its mistake when human interpretable features are accessible. In comparison, other algorithms like decision trees make it much easier to understand what is going on. This is important because, in some domains, interpretability is essential.

In this Master Thesis, the aspect of deep multi-layered data transformation is transferred from neural networks onto a very different learning scheme, the random forests. The latter is a highly competitive and general-purpose method for classification and regression. Even large random forests usually do not have a high amount of parameters to optimize during training. The number of hyper-parameters and architectural varieties is much lower for random forests than for deep neural networks. Training with a much smaller amount of samples is hence possible.

Several variants of a layered architecture of random forests have been investigated here, and different styles of training each forest are tried. The specific problem domain considered is the object detection and pose estimation from RGB-D images.

The Hinterstoisser and T-LESS datasets are used for training and testing [9, 10]. These data are currently used in the public SIXD challenge for object detection and pose estimation, the results of which will serve as a measure of the actual state of the art [11, 12].

1.3 Structure

The next chapter gives an overview of the previous work related to random forests and object pose estimation. In chapter 3, the basics of random forests and the multi-layered architecture are presented. Chapter 4 introduces the approach for estimating the object pose by combining the benefits of machine learning with principles from geometry. Chapter 5 is concerned with the data used for training and evaluation. Chapter 6 contains the experiments themselves and discussion. Finally, conclusion and future work is presented in Chapter 7.

2 Related Work

During the last decade, computer vision has seen a large shift towards machine learning based methods. This has massively improved the accuracy and robustness for tasks like object recognition, object detection and semantic segmentation. In addition, the development of low-cost consumer RGB-D cameras such as Microsoft Kinect has impacted the research in computer vision as well as robotics. These sensors are based on the principle of time of flight (ToF) and provide more stable depth maps than passive approaches. Compared to the RGB image based methods, the extra spatial information provided by utilizing the depth image yields more practical solutions for object pose estimation.

The task of detecting an object and estimating it's 6D pose using random forests is being explored since a long time in various research fields, such as computer graphics, robotics, and pattern matching. The most relevant works are addressed below.

2.1 Random forests and its variants

Random forests are one of the most successful ensemble methods for solving regression and classification problems. They compare favourably with respect to other techniques [13] and have lead to one of the biggest success stories of computer vision in recent years: the Microsoft Kinect. In the years since the introduction of random forests by Breiman [14], they have grown from a single algorithm to an entire framework of models [15, 16]. There have been several approaches to modify and improve their prediction performance and are applied in various fields.

Gall et al. [17] introduced Hough forests, which are random forests adapted to perform a generalized Hough transform in an efficient way. Hough forests improved the performance of the Hough transform for object detection on a categorical level and were extended to new domains such as object tracking and action recognition. Class labels and pose parameters were predicted for local image patches. The decision trees were trained to jointly minimize uncertainty of object class and pose, and nodes alternated between classification and regression objectives. Therefore, Hough forests were regarded as task-adapted codebooks of local appearance allowing fast supervised training and fast matching at test time. Hough voting scheme was used where every pixel casted a vote in some quantized prediction space such as object center and scale across the image and the cell with the most votes was considered the winner. They achieved good detection accuracy for object class and predicted coarse object poses. The concept of training over both Hough votes and object segmentation has been adopted in this thesis.

Zhou and Feng [6] proposed a novel deep forest ensemble method called gcForest (multi-Grained Cascade Forest) with a cascade structure. In contrast to deep neural networks which require great effort in hyper-parameter tuning, gcForest was much easier to train. Excellent performance was achieved by almost same settings of hyper-parameters in gcForest. The training process of gcForest was efficient and users could control training cost according to the available computational resource. Furthermore, in contrast to deep neural networks which require large-scale training data, gcForest worked well even with small-scale training data.

The multi-layer concept presented in this thesis is inspired by gcForest of Zhou and Feng. Finding a layered architecture of random forests that gives gain in prediction accuracy over ordinary random forests and Hough forests is hence desirable both for furthering our methodological understanding of learning systems and for practical purposes.

2.2 Auto-context

Another research direction has focused on an interesting idea of how to give random forests the ability to learn internal descriptions of the input data so as to generalise the final classifier.

The introduction of contextual information as additional features in low-level classifiers was initially proposed by Tu and Bai [7] in their work. Given a set of training images and their corresponding label maps, a classifier was first trained on local image patches to classify each pixel. The classification maps evaluated by the learned classifier were then used as context information, in addition to the original image patches, to train a new classifier. The idea was to train stacked classifiers where early classifiers provide input for subsequent classifiers. This auto-context algorithm was iterated to approach the ground truth. Therefore, local appearance features from image patches were combined with long-range context features for predicting class label of the local image patches and resulted in a substantial gain in performance.

In this line of research, Montillo et al. [18, 19] presented a new classifier called as entangled decision forests where random forests were trained in an entangled setting. The decisions made in each tree node were entangled with decisions made in higher level nodes of other trees. Therefore, intermediate classifier outputs were stacked with original input data. The entangled forest of trees had each tree processing a different image patch. This resulted in higher prediction accuracy and captured long range semantic context with shortened decision time.

Kontschieder et al. [20] introduced Context-Sensitive Decision Forests to interpret Hough forests with contextual information in the decision forest framework. Instead of providing only posterior classification results from an earlier level of classifier during learning and testing, they additionally provided regression (voting) information as it was used in Hough forests. They showed that auto-context can be deployed for intermediate output in a Hough forest for pedestrian detection but they did not smooth the intermediate output.

Auto-context has also been used for vision applications such as object segmentation [21] and pose estimation [22, 4]. Moreover, its scope goes beyond high-level vision and has the potential to be used for a wide variety of problems of multi-variate labelling. In a similar spirit, a general auto-context framework is implemented to learn an integrated low-level context model in this thesis.

2.3 Pose estimation

The vast majority of publications in the field of object detection and pose estimation deal with rigid and articulated objects, coarse and accurate poses, and more recently, in registered color and depth images. Techniques that explicitly refer to the task of estimating full 6D poses of rigid object instances are discussed below.

Early pose estimation methods such as the one presented by Gordon and Lowe [23] were based on matching sparse features for instances with sufficient texture from RGB images. Initially, SIFT features were extracted from the reference images and pair-wise correspondences were established. These correspondences were used to build a metrically accurate 3D model of the object and all its feature locations. Later, features detected in the current video frame were matched to those of the model for computing the pose.

A popular alternative to sparse feature-based approaches are templates. They work well for texture-less objects where sparse feature detectors fail to identify salient points. A template is a global descriptor that captures the appearance of an object from a certain view point. In practice, a set of templates are generated for all possible discrete viewing angles and scales to describe the object as a whole. During test time, each template from the set is compared to the input image at all possible 2D locations. A matching score exceeding a pre-defined threshold will yield a detection. As each template is associated with a specific view point, a template match contain information about object scale and rotation along with its 2D location. Therefore, a template match translates to an estimate of the 6D object pose. However, this estimate is limited in its accuracy as the set of templates represent only a discrete set of object poses. Hence, most template-based methods apply some kind of post-processing, e.g. ICP refinement, to improve the final result. Templates do not require an extensive training phase and have shown to be very efficient during test time.

Ferrari et al. [24] used the template-based approach relying on matching contours for object detection. The success of this methods was limited as contours tend to be unstable when extracted from natural images. Alternatively, object class templates like HOG (histogram of oriented gradients) and HOG-based deformable parts model were described in [25, 26]. Since they were invariant to object appearance, their ability to discriminate exact view points were limited for accurate 6D object pose estimation.

Hinterstoisser et al. [27, 10] proposed the LINEMOD templates, which combined gradient and normal cues from RGB and depth images respectively for robust object detection. Each object was represented by 1100+ templates covering different combinations of scale and viewing angles and the template match gave an initial pose estimate further refined using ICP. Although the matching function of LINEMOD templates was very fast to compute, the number of templates needed increased linearly with the number of objects and with the extent of the pose space to cover.

This thesis is inspired by the Scene Coordinate Regression (SCoRe) forests introduced by Shotton et al. [28] for camera re-localization from RGB-D images. Camera localization is a subtask of pose estimation where the complete input image is covered by the object. The SCoRe forest used only simple depth and RGB pixel comparison features and hence removed the need for a traditional pipeline of feature detection, description and matching. The forest was trained on a particular scene, using RGB-D images with known camera poses. The depth maps and camera poses were sufficient to compute scene coordinate training labels at every pixel. These labels were used in a standard regression objective to learn the forest and predict an estimate of each pixel's correspondence to 3D points in the scene's world coordinate frame. Based on these correspondences, an initial set of hypothesized camera poses was inferred at sparse and randomly sampled pixels. Pre-emptive RANSAC later iterated through more pixels to count inliers and refine the hypothesized poses.

Taylor et al. [29] aimed at another approach using the idea of coordinate representation for human pose estimation. They applied a regression forest to an image window centred around each pixel. Each leaf node in the forest contained a distribution over coordinates on the Vitruvian manifold. The most confident mode across the forest was taken at each pixel as the correspondence to the model. This allowed the use of standard continuous optimization over energy function and the result was a one-shot pose estimation.

Motivated by the Vitruvian manifold model, Brachmann et al. [8] extended the concept of scene coordinate regression to that of object coordinate regression by jointly predicting object labels and object coordinates. Recently, they [4] considered the task of developing an auto-context regression framework which iteratively reduces uncertainty in object label and object coordinate predictions.

In this thesis, the concept of a multi-layered architecture of random forests is combined with object coordinate prediction in an auto-context framework. Some variants of the layered architecture have been investigated here, and different styles of training each forest are tried. Their performance is evaluated by detecting and estimating the 6D pose of objects, using a single RGB-D image.

3 Abstract Forest Model

Recent years have seen an explosion of forest-based techniques in the machine learning, vision and medical imaging literature. They are fast, robust to noise and offer possibilities for explanation and visualization of their output.

Random forest algorithm produces a great result most of the time even without hyper-parameter tuning. It is also one of the most used algorithms, because of its simplicity and the fact that it can be used for both classification and regression tasks. Random forest, as the name implies, is an ensemble method which combines several decision trees to produce better predictive performance than utilizing a single decision tree. Therefore, the basic principles of decision trees are discussed in this section first and extended to the random forest algorithm later. The general mathematical notation is also introduced here, which will be used throughout the thesis.

3.1 Decision trees

First introduced in 1960's, decision trees are a non-parametric supervised learning method used for classification and regression [16]. They reappeared after discovering that higher accuracy can be achieved by ensembles of slightly different trees on previously unseen data.

A decision tree typically starts with a single node called the root. Each internal node and the root node stores a split (or test) function to be applied to the incoming data, leading to branches. Each of these outcomes leads to additional nodes, which branch off into other possibilities. Each terminal node also called as leaf stores the final answer (predictor). This gives it a treelike shape. Therefore, each leaf in the decision tree is responsible for making a specific prediction. For regression trees, the prediction is a value, such as price. For classifier trees, the prediction is a target category, such as outdoor or indoor.

Consider a decision tree structure as illustrated in Figure 3.1 to check if a photo represents an indoor or outdoor scene. An algorithm is constructed to automate this task by hierarchically representing a series of tests. Each internal node in a decision tree is associated with one such question.

The image pixels from the photo is given as an input to the root node and a test is applied to it. The upper half of the image is examined if it is blue or not, considering that blue might represent the sky. Based on the result of this first test, the whole image data is then sent to the left or right child. A new question is asked here, such as, if the lower half is also blue. If it is not, then the probability that

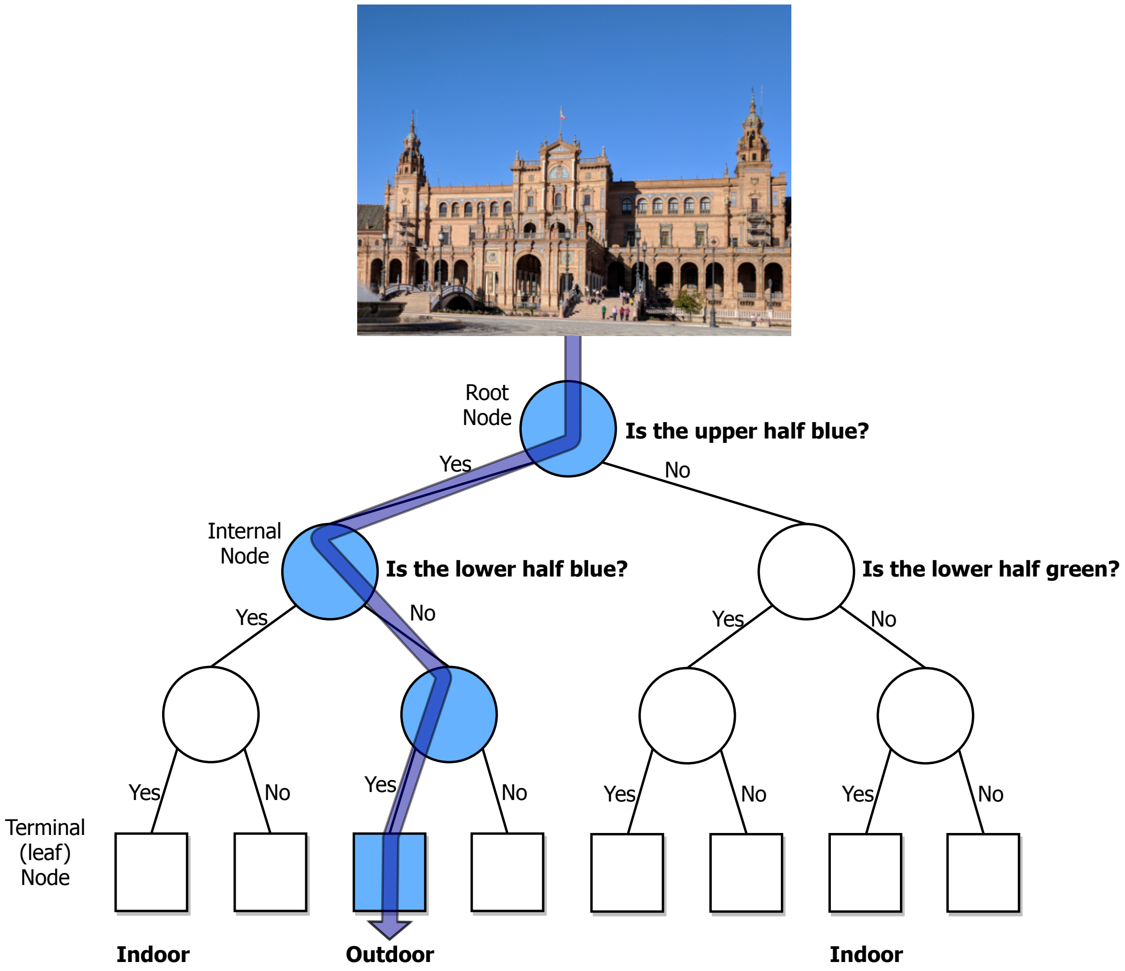


Figure 3.1: Decision tree to check if a photo represents indoor or outdoor scene

this photo is an outdoor scene increases. However, if the lower half of the photo is also blue then the photo might be an indoor scene of a blue wall. Similarly, another test is applied and so on until the image reaches a leaf. The leaf contains the most probable answer, either outdoor or indoor, based on the answers to all the questions asked during the tree descent.

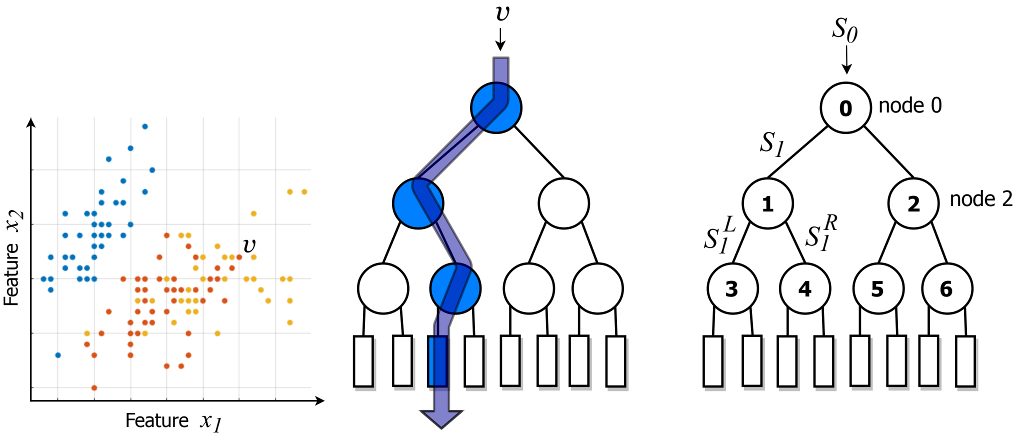
Every question that is asked in the nodes leads to the correct region of decision space and thus, increases the confidence of the response at each step gradually.

3.1.1 Data points for training and testing

Input data are represented as a collection of points defined by their feature responses. Figure 3.2(a) illustrates a data point, denoted by a vector,

$$v = (x_1, x_2, ..., x_n) \in \mathbb{R}^d \tag{3.1}$$

The components x_i represent some attributes of the data point called features. In a computer vision application, v may correspond to a pixel in an image and the x_is



(a) Input data (b) Testing (c) Training

Figure 3.2: Basic notations in a tree

represent the responses of a chosen filter bank at that particular pixel. The features used in this thesis are explained in Section 3.1.2 in detail.

During testing, a split node applies a test to the input data v and sends it to the appropriate child. The process is repeated until a leaf node is reached, as shown in Figure 3.2(b).

A training point is a data point for which the attributes may be known and used to compute tree parameters. For the example shown in Figure 3.1, a training set would be a set of photos with labels "indoor" or "outdoor". Therefore, a training set denoted by S_0 is a collection of different training data points. Training a decision tree involves sending the entire training set S_0 into the tree and optimizing the parameters of the split nodes so as to optimize a chosen energy function, as shown in Figure 3.2(c).

In a supervised learning task, a training point is a pair (v, y) where v is the input data point (feature vector) and y represents a generic known label. In an unsupervised training task, the training points are represented only by their feature response and there is no associated label.

In decision trees, subsets of training points are associated with different tree branches. Consider the nodes to be numbered in breadth-first order starting from 0 for the root, as shown in Figure 3.2(c). If S_1 denotes the subset of training points reaching node 1, then S_1^L and S_1^R will denote the subsets going to the left and to the right children of node 1 respectively. For each split node j in a binary tree, the following properties apply:

$$S_j = S_j^L \cup S_j^R, \qquad S_j^L \cap S_j^R = \emptyset, \qquad S_j^L = S_{2j+1}, \qquad S_j^R = S_{2j+2} \tag{3.2}$$

3.1.2 Features

A specific image feature may represent a pixel or a whole object in an image I. A vector space associated with the feature vectors is often called a feature space $\mathcal{F}$ and the image pixels are mapped to elements of the feature space. The feature space can often be represented in different ways, such as color components, length, area, circularity, gradient magnitude, gradient direction, or simply the gray-level intensity value. However, more complicated feature representations are also possible.

The choice of features and the number of features depends on the type of data point as well as the application. Since the dimensionality n (cf. equation (3.1)) of the feature space can be extremely large in theory, it is neither possible nor necessary to extract all n dimensions of v. Instead, only a small portion of n is extracted as necessary. The features computed will be a subset selected from the set of all possible features,

$$\phi(v) = (x_{\phi_1}, x_{\phi_2}, ..., x_{\phi_{n'}}) \in \mathbb{R}^{n'} \tag{3.3}$$

where n' denotes the dimensionality of the subspace and $\phi_i \in [1, n]$ denote the selected dimensions. In most applications, $n' \ll n$; n' may be as small as 1 or 2.

In this thesis, the feature design for the tree nodes is inspired by [28, 30] where the features are computed by pixel comparisons. The features consider depth or RGB differences from pixels in the vicinity of pixel p and capture local patterns of context. Thus, they are simple and extremely fast to evaluate. They are depth-adapted to make them largely depth invariant.

The feature responses can be computed as follows:

$$f^{da-d}(\theta, p) = D\left(p + \frac{\delta_1}{D(p)}\right) - D\left(p + \frac{\delta_2}{D(p)}\right) \tag{3.4}$$

$$f^{da-rgb}(\theta, p) = I\left(p + \frac{\delta_1}{D(p)}, \gamma_1\right) - I\left(p + \frac{\delta_2}{D(p)}, \gamma_2\right) \tag{3.5}$$

Here, δ indicates a 2D offset, $D(p)$ returns the depth at pixel position p, and $I(p, \gamma)$ indicates an R, G or B channel pixel lookup according to γ. Abbreviations 'da-d' and 'da-rgb' stand for depth adaptive depth differences and depth adaptive RGB differences.

The division by $D(p)$ makes the features depth invariant and is in spirit to [30]: at a given point, a fixed world space offset indicates whether the pixel is close or far from the camera. If an offset pixel lies on the background or outside the bounds of the image, the depth probe $D(p)$ is given a large positive constant value.

3.1.3 Split function

A decision tree is considered optimal when it represents the most data with the fewest number of levels or questions. Each node is associated with a test function, also known as split function or weak learner. The split functions play a crucial role both in training and testing. A test function at a split node j can be formulated as a function with binary outputs,

$$h(v, \theta_j) : \mathbb{R}^d \times \mathcal{T} \rightarrow \{0, 1\} \tag{3.6}$$

where $\theta_j \in \mathcal{T}$ denote the split parameters associated with the j^{th} node and $\mathcal{T}$ represents the space of all split parameters. The outputs 0 and 1 decide if the data point v arriving at the split node is sent to its left or right child node.

The parameters of a general weak learner model can be denoted as

$$\theta = (\phi, \psi, \tau) \tag{3.7}$$

where $\phi = \phi(v)$ is a filter function (cf. equation (3.3)) which selects a subset of features from the entire vector v, ψ defines the geometric primitive used to separate the data and τ denotes the thresholds used in the binary test. The optimization given in equation (3.12) is then defined over all these three sets of parameters.

In this work, each split node in the tree stores a unique set of parameters,

$$\theta_j \subseteq \{\delta_1, \delta_2, \gamma_1, \gamma_2, z, \tau_f\} \tag{3.8}$$

with $z \in \{da - d, da - rgb\}$ indicating the type of feature to use. τ_f denotes a threshold on the feature response that decides whether a pixel goes to the left or the right child of a node.

3.1.4 Tree training

Building a mathematical model of decision tree consists of two basic phases - training phase and testing phase (cf. Section 3.1.5). Training and learning are the same thing. Given some data, called the training set, a tree model is built. This model generally will try to predict one variable based on all the others.

In particular, the split functions stored at the internal nodes are ‘‘learned” automatically as they are the key factor for the functioning of a tree. Thus, the training phase takes care of selecting the type and parameters of the split function $h(v, \theta)$ associated with each split node j. This is done by optimizing a chosen objective function defined on an available training set.

The optimization is performed by searching over a discrete set of samples of possible parameter settings θ. This happens in a greedy manner.

At each node j , depending on the incoming training set S_j we learn the function that "best" splits S_j into S_j^L and S_j^R. This problem is formulated as the maximization of an objective function I at the j^{th} split node,

$$\theta_j = \arg\max_{\theta \in \mathcal{T}} I(S_j, \theta) \tag{3.9}$$

Given the set S_j and the split parameters θ, the corresponding left and right sets are uniquely determined, based on the output 0 or 1, according to equation (3.6). The objective function $I(S_j, \theta)$ is then computed using the parent set S_j, children sets S_j^L and S_j^R and splitting parameters θ as inputs. I can be based on information gain i.e. splitting the training set S_j such that the resulting child nodes are as pure as possible containing mostly training points of a single class.

The structure of the tree has to be decided during training as it depends on how and when to stop growing various branches of the tree. All decision trees need stopping criteria or it would be possible, and undesirable, to grow a tree in which each class occupies its own node. The resulting tree would be computationally expensive, difficult to interpret and would probably not work very well with new data.

Therefore, avoiding growth of full trees will have positive effects in terms of generalization. For example, consider the Figure 3.3. The dotted curve is a decision boundary that accurately separates out the two classes in the training data but a diagonal red line is probably a better decision boundary for new cases.

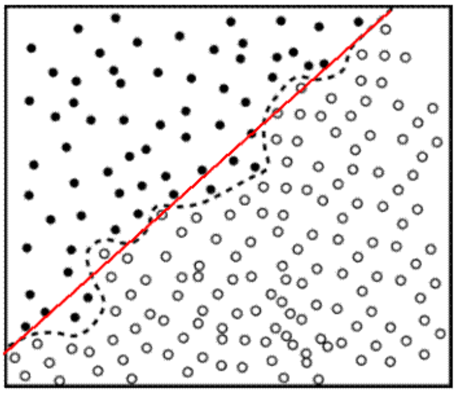


Figure 3.3: Diagram illustrating a decision boundary [31]

The following conditions are some common stopping criteria that can be applied:

- All instances in the training set belong to a single class
- The maximum tree depth, td_{max} has been reached
- The information gain at the node is less than the minimum value i.e. when the attributes of the training points within the leaf node are similar to one another
- The number of training points in the terminal node is less than the minimum number of training points required for a parent node

Hence, a learned tree structure is obtained at the end of the training phase where optimum weak learners (split functions) are associated with each node resulting in different set of training points at each leaf.

3.1.5 Tree testing

During the testing phase, the tree model is tested on a sample called as testing data whose class labels are known but not used for training the model. Therefore, the performance of that tree is evaluated using the test set.

Given a previously unseen data point v, a decision tree hierarchically applies a number of previously selected tests. Note that the node tests have been selected during training (cf. 3.1.4) and remain fixed during testing. Starting at the root, each split node applies its associated test function $h(v, \theta_j)$ to v. Depending on the result of this binary test, the data point v is sent to the left or right child. This process is repeated until the data point reaches a leaf node. The leaf nodes contain a predictor/estimator (e.g. a classifier or a regressor) which associates an output (e.g. a class label or a continuous value) with the input v.

3.1.6 Energy models

Algorithms for constructing optimized decision trees usually work top-down, by choosing a variable at each level that best splits the data. The objective function, I used during training is essential in constructing decision trees that will perform the desired task.

The result of the optimization problem determines the parameters of the weak learners, which in turn determine the path followed by a data point and thus its associated prediction. Therefore, the energy model determines the prediction and estimation behaviour of a decision tree [16].

Different algorithms use different metrics for measuring the homogeneity of the target variable within the subsets. Some examples of metrics used in building a decision tree are

- Gini Index for CART (Classification and Regression Trees) algorithm.
- Information Gain for ID3 (Iterative Dichotomiser 3), C4.5 and C5.0 algorithms.

Information gain is based on the concept of entropy and information content from information theory. Entropy evaluates the homogeneity of a sample. If the sample is completely homogeneous the entropy is 0 and if the sample is equally divided it has entropy of 1.

For the dataset S, the entropy is calculated as

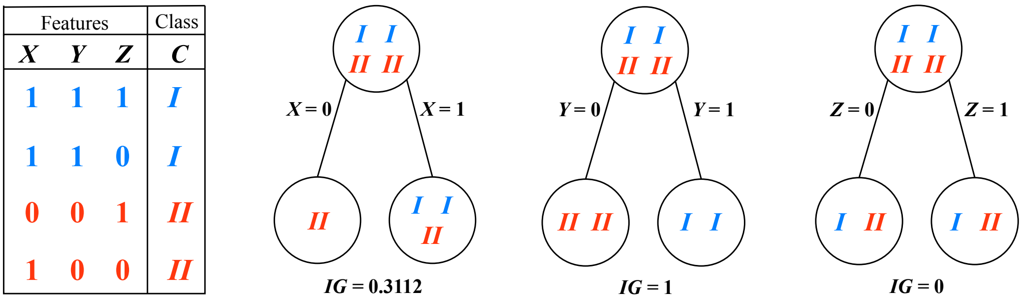
$$\mathcal{H}(S) = -\sum p(S) \log p(S) \tag{3.10}$$

The information gain is based on the decrease in entropy after a dataset is split on an attribute. Constructing a decision tree is all about finding attribute that returns the highest information gain (i.e. the most homogeneous branches). It is calculated as

$$IG = \mathcal{H}(S) - \sum_{i \in \{L,R\}} \frac{|S^i|}{|S|} \mathcal{H}(S) \tag{3.11}$$

Here the dataset S is split into two subsets S^L and S^R. Calculating the weighted average entropy of the child sets avoids splitting off children containing very few points.

Consider a simple binary classification problem for a training set with 3 features (X, Y and Z) and 2 classes (I and II indicated by different colors) as shown in Figure 3.4. The aim is to separate the 2 classes as much as possible and learn the parameters that best split the training data. This example describes how information gain can be used as a training objective function to build a tree using the ID3 algorithm.



Features			Class
X	***Y***	***Z***	***C***
1	1	1	*I*
1	1	0	*I*
0	0	1	*II*
1	0	0	*II*

(a) Training set (b) Split on X (c) Split on Y (d) Split on Z

Figure 3.4: Using information gain to construct a tree

To create a tree, a root node is considered initially. The attribute with highest information gain for the full training set is determined since it best classifies the data. This attribute is used at the root of the tree. The process is repeated at each branch until the 2 classes are separated. Therefore, a top-down greedy search through the space of possible decision trees is performed.

Steps to split the dataset at a node using information gain and thus, construct a decision tree:

Step 1: The entropy for the whole dataset is computed.
Out of 4 instances, 2 are classified as class I and 2 as class II. Therefore, using equation (3.10), entropy

$$\mathcal{H}(S) = -(2/4)\log_2(2/4) - (2/4)\log_2(2/4) = 1$$

Step 2: For every feature, the information gain is calculated using equation (3.11) and child entropies.

- **Split on feature X** results in child entropies $\mathcal{H}(X = 0) = 0$, $\mathcal{H}(X = 1) = 0.9184$ and information gain $IG = 1–(1/4)(0)–(3/4)(0.9184) = 0.3112$.
  Therefore, if X is the best attribute, this node would be further split.
- **Split on feature Y** results in child entropies $\mathcal{H}(X = 0) = 0$, $\mathcal{H}(X = 1) = 0$ and information gain $IG = 1–(1/2)(0)–(1/2)(0) = 1$.
  This corresponds to the best condition with low child entropies of 0 and a high information gain of 1.
- **Split on feature Z** results in child entropies $\mathcal{H}(X = 0) = 1$, $\mathcal{H}(X = 1) = 1$ and information gain $IG = 1–(1/2)(1)–(1/2)(1) = 0$.
  Therefore, this corresponds to the worst condition with high child entropies of 1 and no information gain.

Step 3: Feature with the largest information gain is chosen at the decision node. The training data is split based on the feature Y at the root node. Thus, maximizing the information gain helps select the split parameters which produce the highest confidence (lowest uncertainty) in the final distributions.

Step 4: A branch with entropy 0 is a leaf node and a branch with entropy more than 0 needs further splitting.

Step 5: The ID3 algorithm is run recursively on the non-leaf branches, until all data is classified.

3.1.7 Leaf prediction models

Besides estimating optimal weak learners and tree structures, the training phase also has to learn good prediction models to be stored at the terminal nodes.

After training, each leaf node remains associated with a subset of labelled training data. During testing, a previously unseen point traverses the tree until it reaches a leaf. Since the split nodes act on features, the input test point is likely to end up in a leaf associated with training points which are similar to itself. Thus, the label associated with the test point will be similar to that of the training points in that leaf.

In general, conditional distributions $p(c|v)$ or $p(y|v)$ are used to predict the label associated with the input test point. They are label statistics that are gathered in the leaf, where c and y represent the categorical and continuous labels, respectively. v is the data point that is being tested in the tree and the conditioning denotes that the distributions depend on the specific leaf node reached by the test point.

For example, in classification, the conditional distribution $p(c|v)$ is estimated where c indicates a discrete class index, as shown in Figure 3.5.

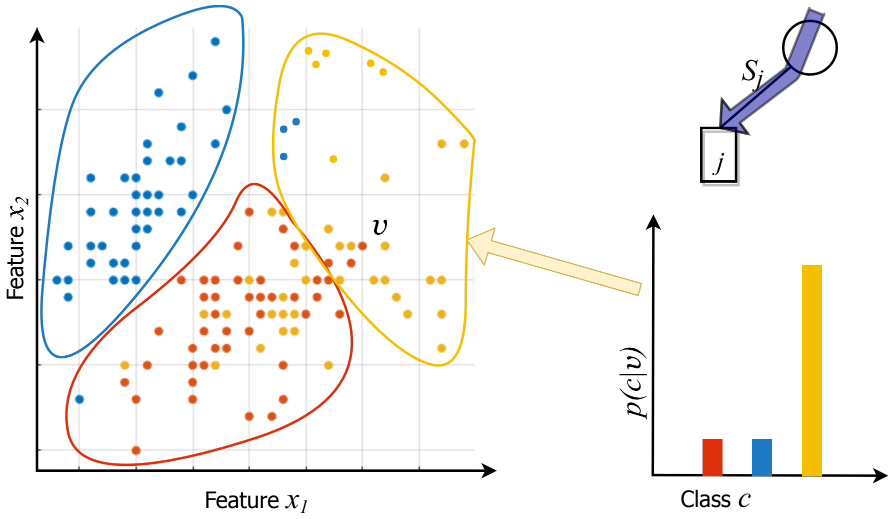


Figure 3.5: Diagram illustrating a probabilistic leaf model

Therefore, every leaf node is associated with a predictor model such as Maximum Likelihood Estimation (MLE) and Maximum A Posteriori (MAP). A MAP estimate may be obtained as

$$c = \arg\max_c p(c|v) \qquad or \qquad y = \arg\max_y p(y|v) \tag{3.12}$$

for the categorical and continuous cases, respectively.

However, in general it is preferable to keep the entire distribution around until the final moment where a decision must be taken, rather than taking an early point estimate. This allows for reasoning about the prediction uncertainty.

3.2 Combining trees into a forest ensemble

Decision tree learners can create over complex trees that do not generalize the data well. Hence, they are prone to over fitting, especially when a tree is particularly deep. One way to combat this issue is by setting a maximum depth but this will be at the expense of error due to bias. Another way is to build a simpler model with less variance sample to sample but ultimately it will not be a strong predictive model.

Ensemble methods combines several decision trees to produce better predictive performance than utilizing a single decision tree. The main principle behind the ensemble model is that a group of weak learners come together to form a strong learner.

Bagging or bootstrap aggregation is one of the techniques to perform ensemble decision trees. The idea is to create several subsets of data from training sample chosen randomly with replacement. Now, each collection of subset data is used to train their decision trees. As a result, an ensemble of different models is obtained. Average of all the predictions from different trees are used which is more robust than a single decision tree. By introducing randomness in the way each base estimator is built, the variance of the final model is reduced. In other words, over fitting is avoided.

Random Forest is an extension of bagging that is built on decision trees. It takes one extra step - in addition to taking the random subset of data, it also takes a random selection of features at each split node rather than using all features to grow trees. These two elements of randomness ensure that each tree will be different. Therefore, a random decision forest is an ensemble of randomly trained decision trees and is characterized by the same components as the decision trees.

In a forest with T trees, the variable $t \in \{1, ..., T\}$ is used to index each component tree. All trees are trained independently. During testing, each test point v is simultaneously pushed through all trees (starting at the root) until it reaches the corresponding leaves. Combining all tree predictions into a single forest prediction may be done by a simple averaging operation, such as in classification, using

$$p(c|v) = \frac{1}{T}\sum_{t=1}^{T} p_t(c|v) \tag{3.13}$$

where $p_t(c|v)$ denotes the posterior distribution obtained by the t^{th} tree.

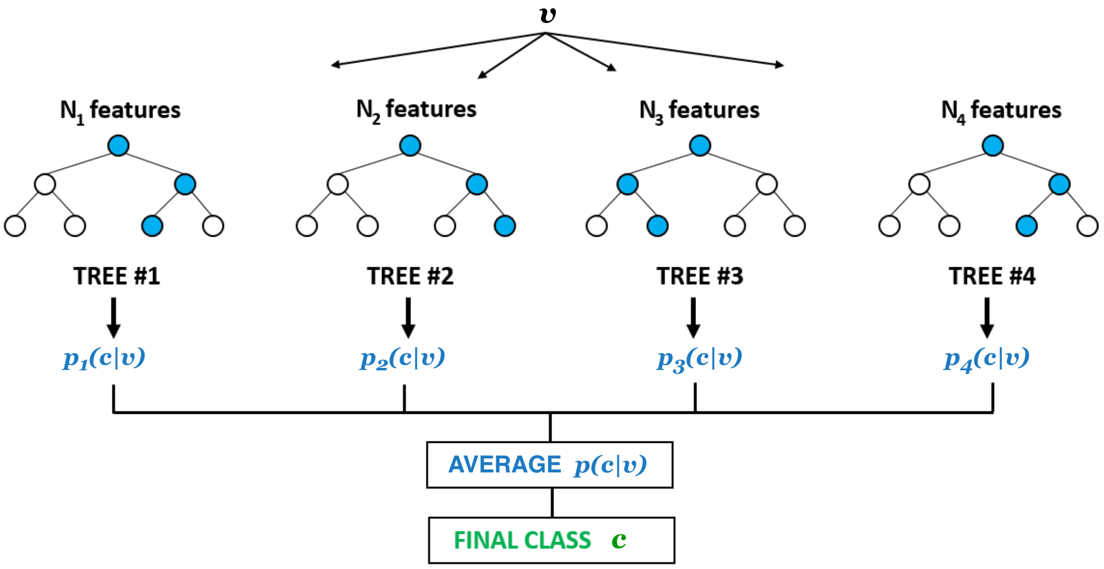


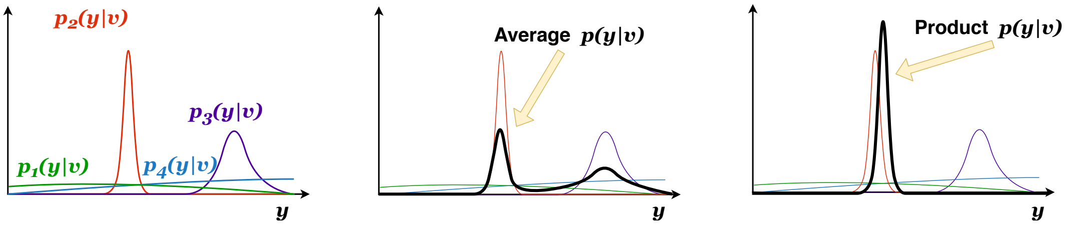
Figure 3.6: A random forest testing with 4 trees

A random forest testing is illustrated in Figure 3.6 with $T = 4$ trees in the forest. During testing the same unlabelled test input data v is pushed through each component tree. At each internal node a test is applied with a random selection of features and the data point is sent to the appropriate child. The process is repeated until a leaf is reached. At the leaf, the stored posterior $p_t(c|v)$ is read off. The forest class posterior $p(c|v)$ is obtained as the average of all tree posteriors. This is used to predict the final class c.

Alternatively, with the partition function Z ensuring probabilistic normalization, the tree outputs could also be multiplied together using

$$p(c|v) = \frac{1}{Z}\prod_{t=1}^{T} p_t(c|v) \tag{3.14}$$

Consider another example of a trained forest with $T = 4$ regression trees [16]. Multiple tree outputs are combined to predict the continuous variable y, as shown in Figure 3.7. For a test data point v, the corresponding tree posteriors $p_t(y|v)$ is modeled here as Gaussians, with $t \in \{1, ..., 4\}$. As illustrated, some trees produce peakier (more confident) predictions than others.



(a) Posteriors of 4 trees (b) Averaging all tree posteriors (c) Product of all tree posteriors

Figure 3.7: Ensemble model posterior $p(y|v)$

Both the averaging and the product operations produce combined distributions (shown in black) which are heavily influenced by the most confident, most informative trees. Therefore, such simple operations have the effect of selecting the more confident trees out of the forest. This selection is carried out at a leaf-by-leaf level and the more confident trees may be different for different leaves. Averaging many tree posteriors also has the advantage of reducing the effect of possibly noisy tree contributions. The product based ensemble model produces sharper distributions and may be less robust to noise. Alternative ensemble models are also possible, such as use of a Naive-Bayes strategy, for combining tree outputs.

3.3 Key model parameters

The (hyper)parameters in a random forest are either used to increase the predictive power of the model or to make the model faster.

The model parameters that influence the behaviour of a decision forest most are:

- **Forest size T :** This is the number of trees the algorithm builds before taking the maximum voting or taking averages of predictions. In general, a higher number of trees increases the performance and makes the predictions more stable, but it also slows down the computation.
- **Maximum tree depth td_{max} :** Learning very deep trees can lead to overfitting. Thus, a single decision tree needs pruning in order to overcome the overfitting issue. However, in random forest, this issue is eliminated by randomly selecting the variable td_{max}. A standard random forest algorithm grow the full decision tree without pruning.
- **Maximum features :** This is the maximum number of features a random Forest is allowed to try in an individual tree. Increasing this parameter generally improves the performance of the model as at each node a higher number of options can be considered. However, this is not necessarily true as this decreases the diversity of individual tree which is the USP of random forest. Also, the speed of the algorithm decreases when maximum features is increased. Hence, an optimal number has to be chosen to strike the right balance.
- **Minimum samples :** This is the minimum sample leaf size that is required to split an internal node. A smaller leaf makes the model more prone to capturing noise in train data. Generally, a minimum leaf size of more than 50 is preferred. However, multiple leaf sizes should be tried to find the most optimum for the use case.
- **Randomness parameter ρ :** ρ controls the amount of randomness within each tree along with the amount of correlation between different trees in the forest. Large values of ρ correspond to little randomness and thus large tree correlation. When $\rho = |T|$, the forest behaves very much as if it was made of a single tree. Small values of ρ correspond to large randomness in the training process. Thus the forest component trees are all very different from one another.

Other parameters that influence the forest's accuracy and its estimated uncertainty are the choice of features, weak learner model and training objective function.

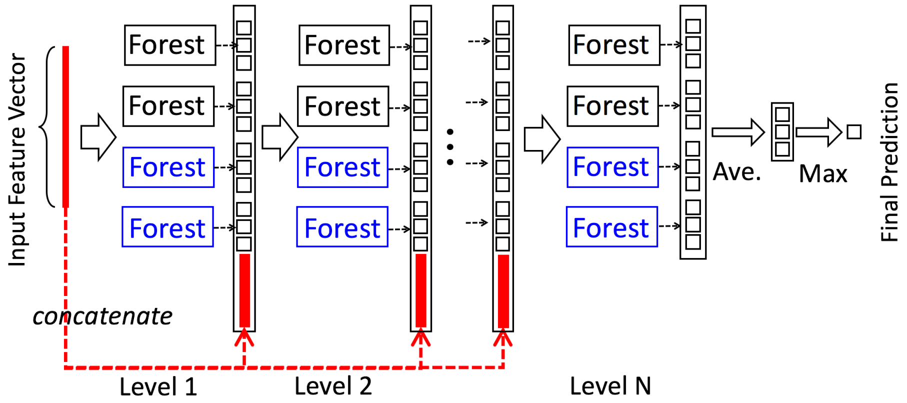
When training a forest it is important to visualize its trees as well as other intermediate variables (e.g. the features and parameters chosen at each node), to make sure the forest has the expected behavior.

3.4 Building deep random forests

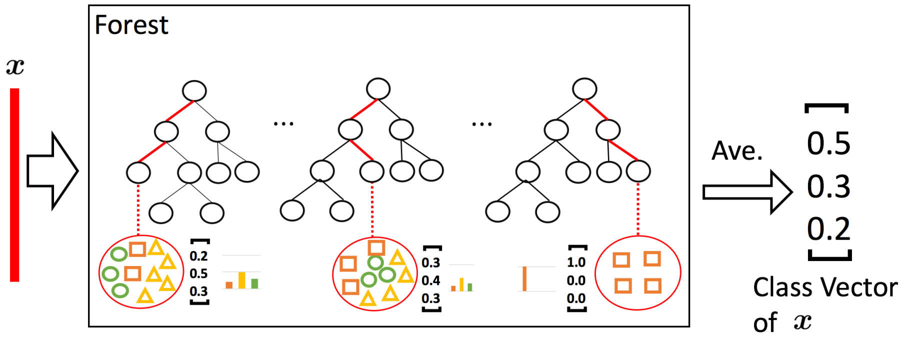
The success of deep neural networks has inspired many to wonder whether other learners could benefit from deep layered architectures. Few frameworks generalizing the deep neural network architecture have been proposed where neurons are replaced by other classifiers. In this work, a network is considered where each layer is a type of random forest, with neurons composed of the individual decision trees. Therefore, such networks can be quickly trained layer-by-layer instead of paying the high computational cost of training a neural network all at once.

In this section, a general description of Zhou and Feng's deep forest ensemble [6] is given since it forms the basis for the multi-layer concept presented in this thesis. Details relating to the application of this general idea along with auto context are in the Section 4.

A cascade forest structure called as gcForest (multi-Grained Cascade Forest) was proposed by Zhou and Feng as illustrated in Figure 3.8. Every layer in the ensemble consists of two random forests (black) and two completely-random tree forests (blue), with each forest containing 500 trees. Each level of cascade receives feature information processed by its preceding level, and outputs its processing result to the next level.



(a) Cascade structure of a gcForest



(b) Generation of a class vector

Figure 3.8: Illustration of a deep forest ensemble [6]

There are three classes to predict, as shown in Figure 3.8(b). Given an instance, each forest will produce an estimate of class distribution, by counting the percentage of different classes of training examples at the leaf node where the concerned instance falls, and then averaging across all trees in the same forest. The estimated class distribution forms a class vector, which is then concatenated with the original feature vector to be input to the next level of cascade. Thus, each forest will output a three-dimensional class vector, which is then concatenated for re-representation of the original input.

After expanding a new level, the performance of the whole cascade will be estimated on validation set and the training procedure will terminate if there is no significant performance gain. Thus, the number of cascade levels is automatically determined.

In contrast to most deep neural networks whose model complexity is fixed and require large-scale training data, gcForest [6] adaptively decides its model complexity by terminating training when adequate and can work well even when there are only small-scale training data. More importantly, gcForest has much fewer hyper-parameters, and has shown excellent performance across various domains by using the same parameter setting.

In this thesis, a modified concept of gcForest has been adapted where the deep architecture of decision forests only require the previous layer's output. In [6], each layer passes both the class probabilities predicted by the random forests and the original data to each subsequent layer. However, in this work, only the output of the previous layer of forests is passed to the next layer. This reduces the spatial complexity of network training and testing. Moreover, fewer trees in each layer seem to be sufficient here. For example, as described in Section 6, there are only 10 decision trees in each layer whereas [6] uses 4 random forests of 1,000 trees each (or 4,000 trees per layer). Another distinction is that the final routine here uses information gain to calculate node splits, whereas gcForest implements gini impurity.

In both [6] and this thesis, the decision tree networks can be trained efficiently without the use of backpropagation. Each layer remains static once trained, and so the training data can be pushed through to train the next layer. Hence, the training time for a multi-layer forest should be much faster than training time for a traditional deep neural network architecture.

3.5 Auto-context framework

As explained in previous sections, every tree in the forest predicts the final class per pixel. In the multi-layered architecture, this concept of a simple random forest is extended to an auto-context framework. This is because a region around a local image patch may contain substantial amount of "structural" information, i.e. neighboring object class predictions are statistically dependent. Therefore, the prediction step is adapted into an auto-context structure.

Auto-context was first introduced in [7]. Figure 3.9 illustrates the classification map updated at each round for the horse segmentation problem. The blue rectangles indicate the whole set of context pixels from which red rectangles are selected contexts in training.

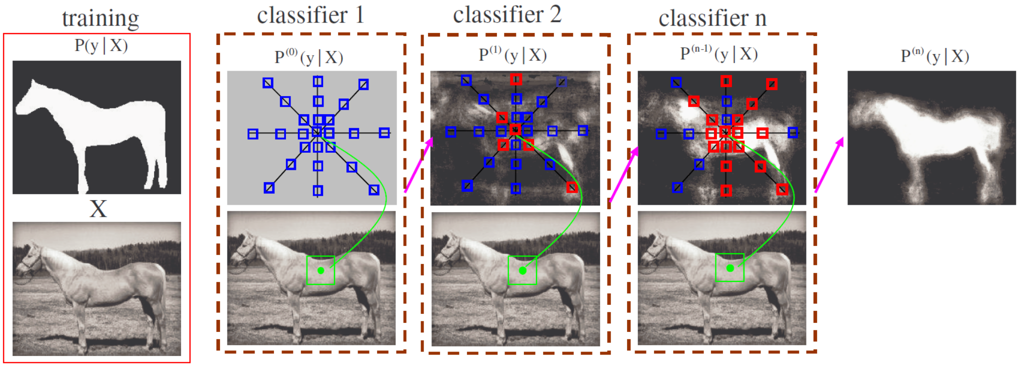


Figure 3.9: Illustration of the procedures of the auto-context [7]

Given a set of m training images and their corresponding label maps $S = \{(X_j, y_j), j = 1...m\}$, a classifier is first learned on local image patches. For each image X_j , probability maps $P_j^{(0)}(y|X)$ are constructed with uniform distribution on all the labels. These discriminative probability maps by the learned classifier are then used as context information, in addition to the original image patches, to train a new classifier. The algorithm then iterates to approach the ground truth $P(y|X)$.

Therefore, the auto-context algorithm integrates the image appearances (observed data) together with the context information by learning a series of estimators. In this work, an efficient estimator is used to iteratively reduce the uncertainty of object class label predictions along with object coordinate predictions as described in Section 4.4.

4 Implementation

In this section, an approach for estimating the object pose from an RGB-D image is introduced which combines the benefits of machine learning with principles from geometry. The concept of object coordinate pixel labelling is described followed by how this labelling is used to train a model for each object. Later, a scoring function is formulated which assesses the pose hypothesis by evaluating the forest at a sparse set of pixels.

4.1 3D object coordinate labelling

Visual variabilities such as lighting, occlusion and reflections make it extremely difficult to predict correspondences between an image and an object. In the earlier days of computer vision, researchers tried to hand-engineer image features with the goal to identify objects under varying viewing angles, lighting, etc. Since 2010, learned features have proven increasingly superior. Thus, inspired by [28, 29, 8], the aim is to learn to predict correspondences in this work.

In [28], the correspondences were directly predicted from an image pixel to a point in the scene's 3D world coordinate frame, called as scene coordinates. An overview of the pipeline is shown in Figure 4.1.

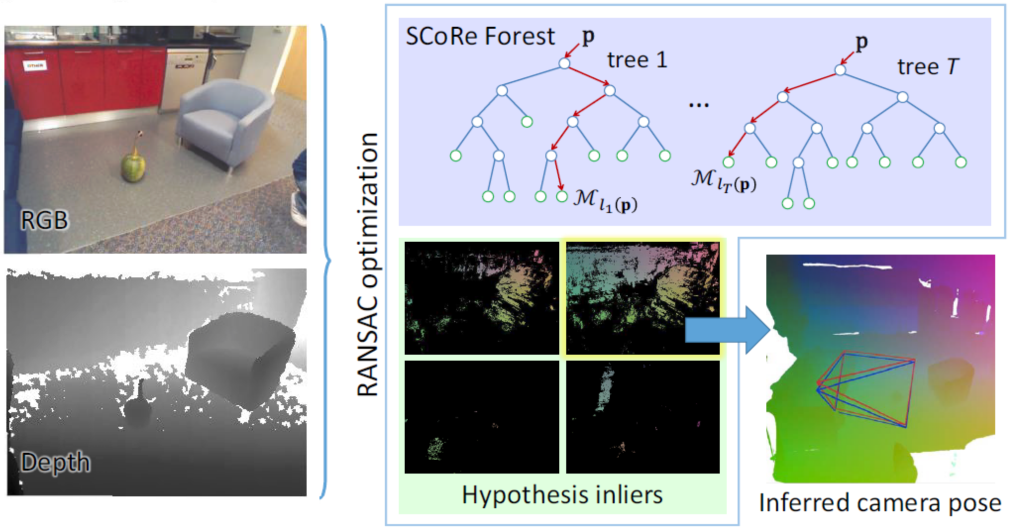
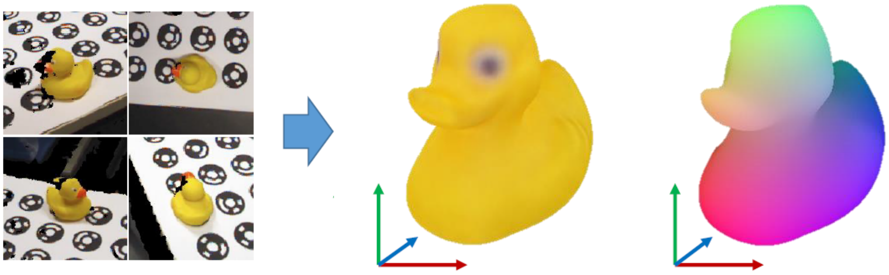


Figure 4.1: SCoRF pipeline for camera localization [28]

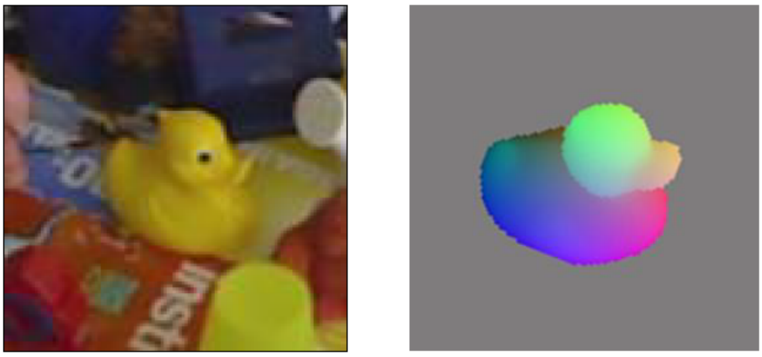
The depth maps and camera poses were sufficient to compute scene coordinate training labels at every pixel. These labels were used in a standard regression objective to learn the forest. This forest, known as scene coordinate regression forest (SCoRF), was used for camera localization.

Camera localization is a special case of object pose estimation where the object of interest occupies the complete input image. If the object of interest occupies only a small fraction of the input image, as considered in [8] and in this thesis, most scene coordinate predictions will be wrong. Therefore, the concept of scene coordinate regression was extended to that of object coordinate regression in [8] since the SCoRF pipeline cannot be directly applied to object pose estimation. Thus, using object coordinates, accurate 6D poses of small objects can be estimated in this thesis.

Object coordinates are a dense correspondence representation used to learn to regress for each pixel of an input image the location on the object's surface. This is named after the coordinate reference frame of the object, as shown in Figure 4.2.



(a) Acquisition (b) 3D Model (c) Object Coordinates



(d) Input Image (e) Object Coordinate Prediction

Figure 4.2: Object coordinates as dense correspondences

A 3D model of an object, e.g. created by 3D scanning, defines a coordinate frame illustrated by three coloured axes specific to the object. Each point on the object surface has a unique coordinate in this frame, i.e. object coordinate. The object coordinates are visualized by mapping their X/Y/Z components to the RGB cube, as shown in Figure 4.2(c).

A discriminative model is trained which is able to regress object coordinates and segmentation of object simultaneously from its object coordinate ground truth. Therefore, a dense 3D object coordinate labelling is predicted, as shown in Figure 4.2(e), along with a dense object ID labelling.

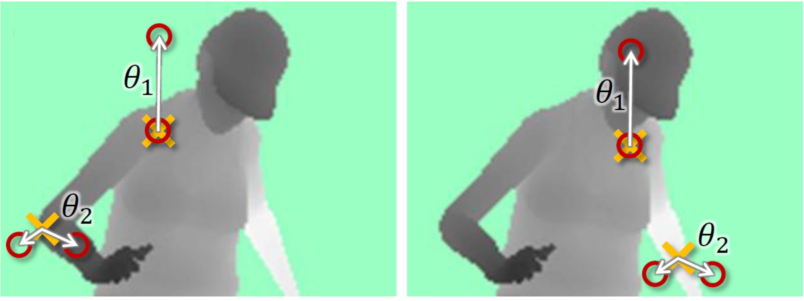
Object coordinates are predicted for each pixel based on image patches, i.e. based on its local image neighbourhood. While this restricts the available information depending on the patch size, it has large advantages in terms of generalization capabilities. For example, partial object occlusion will affect some patches but not others, for which object coordinates can still be predicted reliably.

Hence, by learning to predict object coordinates, the model can be robust to many sources of visual variability like lighting changes or occlusion just by adding them to the training set. The learned model can either develop invariant features, or remember the object appearance under different image conditions.

4.2 Split node features

Each random forest in the model uses pixel differences on RGB and depth as features evaluated in the tree splits. These feature responses are calculated using equations (3.4) and (3.5), inspired by [30].

Figure 4.3 illustrates two features at different pixel locations. The yellow crosses indicates the pixel p. The red circles indicate the offset pixels as defined in equation (3.4).

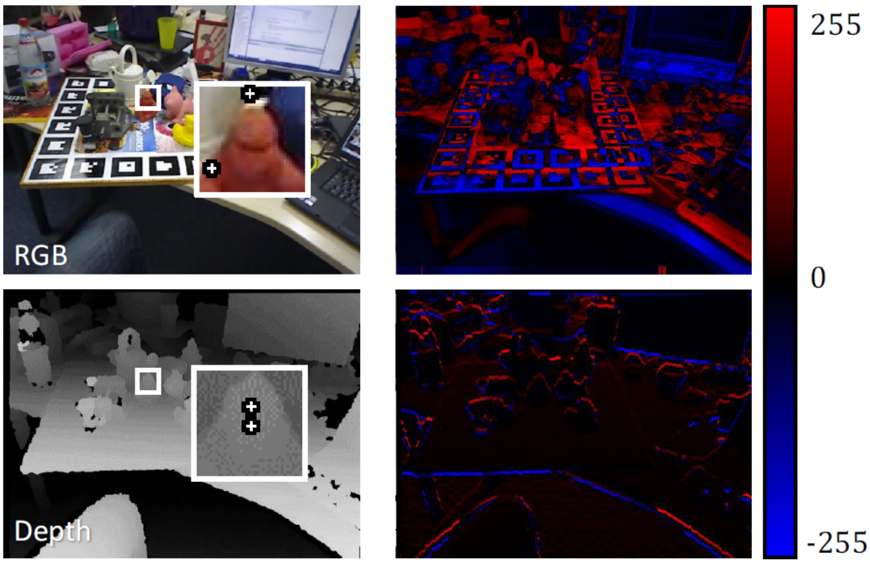


(a) large depth difference (b) smaller response

Figure 4.3: Computing feature responses using a depth image [30]

In Figure 4.3(a), the two example features give a large depth difference response. In Figure 4.3(b), the same two features at new image locations give a much smaller response. Feature $f(\theta_1)$ looks upwards: large positive response is obtained for pixels p near the top of the object, but a value close to zero for pixels p lower down the object. Feature $f(\theta_2)$ may instead help find thin vertical structures. Individually these features provide only a weak signal about which object the pixel p belongs to, but in combination in a decision forest they are sufficient to accurately disambiguate all trained objects.

Figure 4.4 show the feature responses for one RGB feature f^{da-rgb}, and for one depth feature f^{da-d}. The magnified inlay show an input patch with feature offsets δ_1 and δ_2 marked by crosses.



(a) Pixel differences (b) Feature responses

Figure 4.4: Depth adaptive pixel difference features

As mentioned before, feature offsets are depth adaptive, i.e. input patches are scale-normalized according to the depth at the center pixel. Consider the patch size for two input images taken with different distance between object and camera. Figure 4.5 illustrates how the scale adapted patch covers similar portions of the object.

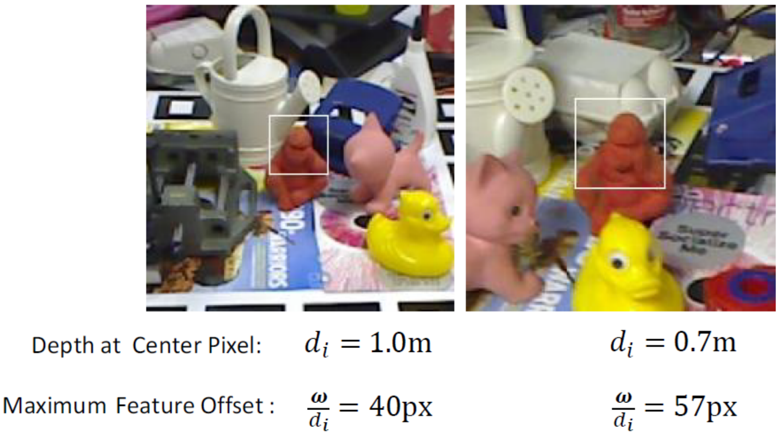

Depth at Center Pixel: $d_i = 1.0\text{m}$ $d_i = 0.7\text{m}$
Maximum Feature Offset: $\frac{\omega}{d_i} = 40\text{px}$ $\frac{\omega}{d_i} = 57\text{px}$

Figure 4.5: Depth adaptiveness

The design of these features was strongly motivated by their computational efficiency: no preprocessing is needed; each feature need only read at most 3 image pixels and perform at most 5 arithmetic operations. Also, they are robust to occlusions.

4.3 Object label and coordinate predictions

Given an object c and pixel p, a random forest is modelled to predict the distribution over object labels $P_p(c)$ for each pixel in the image. The set $\mathcal{C} \in \mathbb{N}$ contains all objects known to the system. Furthermore, a distribution of coordinates $P_p(y|c)$ in object space $y \in \mathcal{Y} \subseteq \mathbb{R}^3$ is predicted jointly.

4.3.1 Training

To train the forest, samples are drawn uniformly from within the object segmentation and up to a certain distance along the object boundary if outside the segmentation. This distance is set to 20% of the maximum feature size. Samples outside the segmentation with sufficient overlap with the object contain valuable information about the object boundary. This is necessary for the application of auto-context discussed in next section. The decision forest is trained for all the objects and a background class. The background class is generated for the first layer using random responses at each pixel, followed by median filtering. This is reused in the subsequent layers.

The training procedure is the same for all trees of the forest and the training images are a combined set of training pixels. Each training pixel p is characterized by its depth $D(p)$, its color $I(p, rgb)$, its object instance label c_p and its object coordinate y_p. The continuous object coordinate labels y are quantized to obtain discrete object coordinate labels $\hat{y}$. This quantization enables the use of standard information gain classification objective during training, which performs better than the regression objective.

In [8], a regular $5 \times 5 \times 5$ grid over the object bounding box is used for quantization. For some objects, this results in a very unbalanced distribution over proxy classes. Inspired by [4], randomly selected object coordinates of the training data are taken as the cluster centers and quantized using nearest neighbor assignment. Therefore, the quantized object coordinates $\hat{y}$ serves as proxy classes, as visualized in Figure 4.6. They are used for coordinate regression and classification trees are built.

At each tree depth level, a pool of feature parameters θ_j given by equation (3.8) is sampled uniformly. Feature thresholds τ_f are also chosen randomly by calculating the feature response at a random training pixel. If a feature access a pixel outside the object segmentation during training, a uniform color noise is returned.

The selection of features at each node is based on a split score, as mentioned in Section 3.1.6. The information gain IG (c.f. equation (3.11)) is defined over the joint distribution $P(\hat{y}, c)$. θ_j is selected such that the information gain in objects

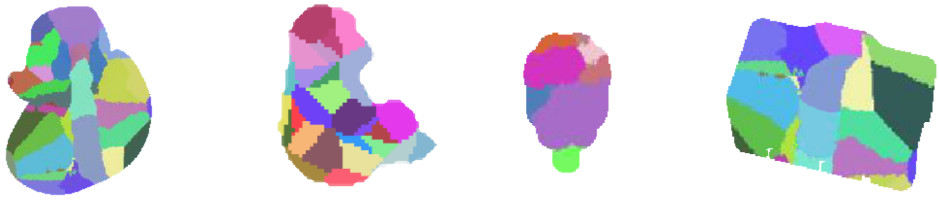


Figure 4.6: Proxy classes

and proxy classes is maximized. This divides the data to go to left and right child nodes, where the feature selection process repeats. This process iterates until a stopping criterion is met. The splitting is further stopped if a maximum depth has been reached, or not enough training pixels arrived at a node.

Therefore, the tree structure is constructed based on quantized object coordinates. Training pixels from all objects are pushed through the tree t and all continuous locations y are recorded for each object c at each leaf l^t . This gives a conditional distribution $P(y|c, l^t)$ per leaf which is approximated by storing only the mode $y_c(l^t)$ with most supporting samples. For each object and leaf, mean-shift is evaluated with a Gaussian kernel and a bandwidth of 2.5cm. Furthermore, the percentage of pixels coming from each object c is stored at each leaf to approximate the distribution of object affiliations $P(c|l^t)$ at the leaf. Figure 4.7 summarizes the forest training procedure.

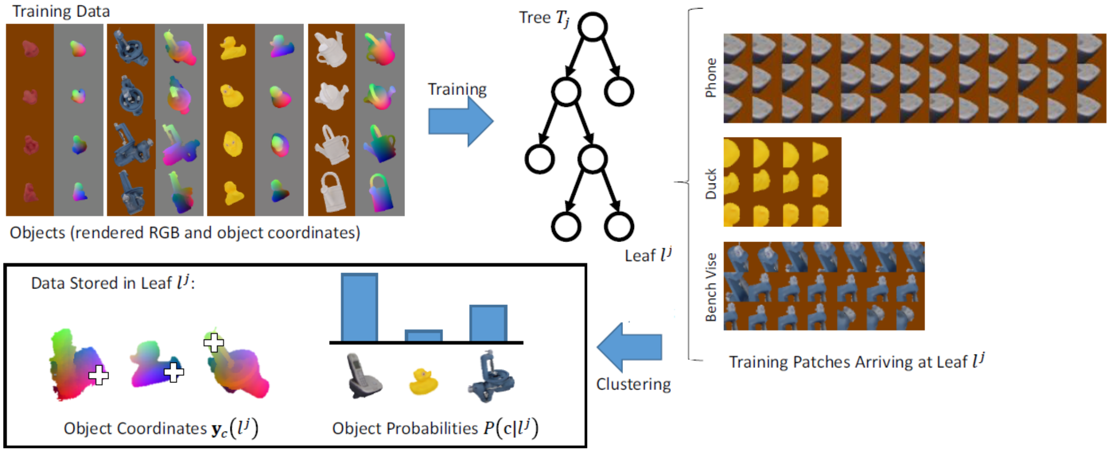

Training Data
Objects (rendered RGB and object coordinates)
Training
Tree T_j
Leaf l^j
Phone
Duck
Bench Vise
Training Patches Arriving at Leaf l^j
Clustering
Data Stored in Leaf l^j:
Object Coordinates $\mathbf{y}_c(l^j)$
Object Probabilities $P(c|l^j)$

Figure 4.7: Random forest training

4.3.2 Testing

At test time, all the pixels of an RGB-D image are pushed through every tree of the forest, thus associating each pixel p with a distribution $P(c|l_p^t)$ and one prediction $y_c(l_p^t)$ for each tree t and each object c. Here, l_p^t is the leaf outcome of pixel p

in tree t. The leaf outcome of all trees for a pixel p is summarized in the vector $l_p = (l_p^1, ..., l_p^t, ..., l_p^{|T|})$. For each pixel p in the image and for each object c, $P_p(c)$ is calculated by combining $P(c|l_p^t)$ stored at leafs l_p^t. Similarly, combining all the object coordinate predictions $y_c(l_p^t)$ yields $y_c(l_p)$.

$P_p(c)$ is referred as the object probability since

$$P_p(c) \approx P(c|l_p) \tag{4.1}$$

This is because $P_p(c)$ is the approximate probability $P(c|l_p)$ that a pixel p belongs to object c given it ended up in the combined set of leaf nodes $l_p = (l_p^1, ..., l_p^t, ..., l_p^{|T|})$. Therefore, the object probability is calculated using Bayes' theorem as,

$$P(c|l_p) = \frac{P(c)P(l_p|c)}{P(l_p)} \tag{4.2}$$

Since the trees were trained to separate pixels according to both object affiliation and their position in object space, $P(l_p|c)$ can be approximated as

$$P(l_p|c) \approx \prod_{t=1}^{|T|} P(l_p^t|c)$$

where $P(l_p^t|c)$ is the conditional probability for a leaf outcome l_p^t given the pixel is part of object c. Simplifying, we get

$$P(l_p^t|c) = \frac{P(l_p^t)P(c|l_p^t)}{P(c)}$$

where $P(l_p^t)$ is the a priori probability of the leaf outcome l_p^t.

$P(l_p)$ is the probability that a pixel ends up in the leafs l_p regardless the object it belongs to. Hence,

$$P(l_p) = \sum_{c' \in \mathcal{C}} P(c') \prod_{t=1}^{|T|} P(l_p^t|c')$$

$$P(l_p) = \sum_{c' \in \mathcal{C}} P(c') \prod_{t=1}^{|T|} \frac{P(l_p^t)P(c'|l_p^t)}{P(c')} \tag{4.3}$$

Therefore, using equations (4.2) and (4.3) in equation (4.1),

$$P(c|l_p) = \frac{P(c) \prod_{t=1}^{|T|} \frac{P(l_p^t)P(c|l_p^t)}{P(c)}}{\sum_{c' \in \mathcal{C}} P(c') \prod_{t=1}^{|T|} \frac{P(l_p^t)P(c'|l_p^t)}{P(c')}} \tag{4.4}$$

where $P(c)$ is a priori probability that a pixel is part of an object c. Assuming it is the same for each object,

$$P(c|l_p) = \frac{\prod_{t=1}^{|T|} P(l_p^t)P(c|l_p^t)}{\sum_{c' \in \mathcal{C}} \prod_{t=1}^{|T|} P(l_p^t)P(c'|l_p^t)}$$

Cancelling out the factor $\prod_{t=1}^{|T|} P(l_p^t)$ in both numerator and denominator, the object probability reduces to

$$P(c|l_p) \approx P_p(c) = \frac{\prod_{t=1}^{|T|} P(c|l_p^t)}{\sum_{c' \in \mathcal{C}} \prod_{t=1}^{|T|} P(c'|l_p^t)} \tag{4.5}$$

4.4 Using auto-context information

As explained in the previous section, every tree in the forest predicts an object coordinate point estimate per pixel. A standard random forest is used for this task. In the multi-layered architecture, this concept of a simple random forest is extended to an auto-context framework. This is because the dense object coordinates may contain a substantial amount of "structural" information, i.e. neighboring object coordinate predictions are statistically dependent. Therefore, the prediction step is adapted into an auto-context structure.

The auto-context algorithm, introduced in [7], integrates the image appearances (observed data) together with the context information by learning a series of estimators. In this work, object coordinates prediction from previous layer are concatenated instead of probability maps from layer 2. Given a set of training images, local image patches of size 11 × 11 are considered. The context region is taken as 3 times the patch size.

An efficient estimator is used to iteratively reduce the uncertainty of object coordinates and class label predictions. Inspired by the Geodesic Forests approach in [32], the outputs of neighbouring pixels are coupled by smoothing the predictions before passing them to the next forest. Figure 4.8 indicates the probabilities of object labels and object coordinates that are jointly predicted for every pixel of the input image in each layer.

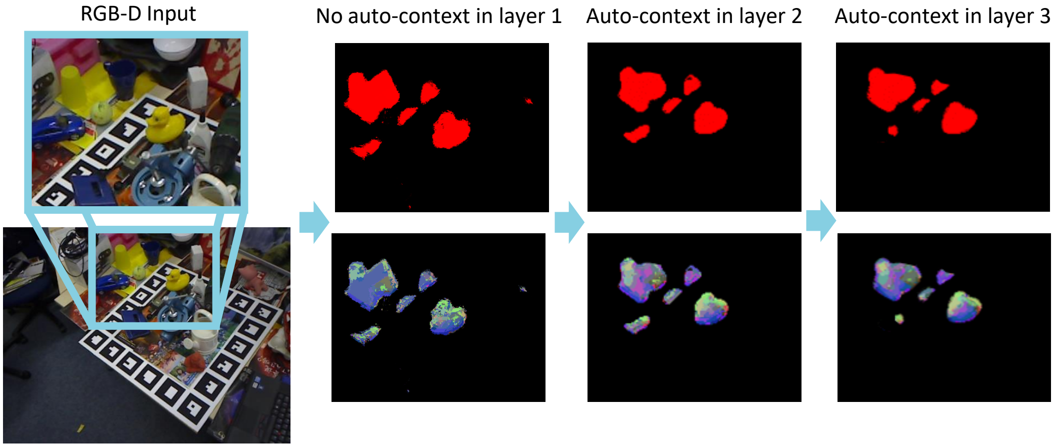


Figure 4.8: Reducing the uncertainty of object label and coordinates predictions

In general, there will be two types of features that can be chosen by each forest in the multi-layered framework: (1) image features computed on the local image patches centered at the current pixel, and (2) context information on the probability maps. Initially, the probability maps are uniform, and thus, the context features are not selected by the first estimator. The trained estimator then produces new probability maps which are used to train another estimator.

4.5 Multi-layered architecture of random forests

The deep forest ensemble (gcForest) described in Section 3.4 has inspired this concept of multi-layered framework. But instead of an ensemble of four forests, one auto-context random forest is used in each layer. This will reduce the uncertainty of the predictions as much as possible.

Multiple layers of forests T^i are modelled, where $i \in \{0, ..., d\}$ denotes the layer number. The first layer T^0 is trained exactly as described in Section 4.3. All subsequent forests T^{i+1} have access to the output of the previous forest T^i, namely object probabilities $P_p^i(c)$ and object coordinate predictions $y_c(l_p)$ of pixel p.

The outputs of neighbouring pixels are coupled by smoothing the predictions before passing them to the next forest. Instead of a geodesic filter as used in [32], a median filter is used in a local neighbourhood of each pixel, similar to [4]. Thus, median-smoothed object probabilities $P_p^i(c)$ can be defined using the filter as

$$m_{\mathcal{C}}^i(c,p) = \underset{P'_p \in \mathbb{R}}{\arg\min} \sum_{j=0}^{N-1} |P_j^i(c) - P'_p(c)| \tag{4.6}$$

where m is the median (filter's output) and N is the number of pixels in the neighbourhood of pixel p. This will be forwarded to the next layer of forest T^{i+1} as a feature,

$$f_{\mathcal{C}}(p,\theta) = m_{\mathcal{C}}^i(c, p+\delta) \tag{4.7}$$

where the parameter vector θ consists of pixel offset δ and the object index c. The smoothing of the object coordinate prediction $y_c(l_p)$,done in a similar way, results in the feature

$$f_{\mathcal{Y}}(p,\theta) = m_{\mathcal{Y}}^i(c, p+\delta) \tag{4.8}$$

Therefore, the forests T^i where $i \in \{1, ..., d\}$ are trained in the same way as described before in Section 4.3, but the set of feature types is increased by $f_{\mathcal{C}}$ and $f_{\mathcal{Y}}$. Starting with the second layer $i = 1$, the prediction of the previous forest is calculated for each training image. The resulting auto-context feature channels are stored sub-sampled.

4.6 Object pose estimation

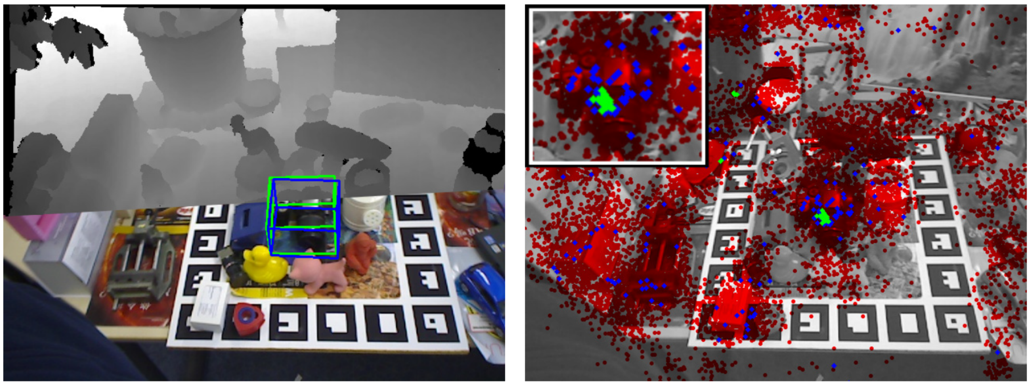
Given an input image, the aim is to estimate the 6D pose of an object c (3D rotation and 3D translation). The pose h_c is defined as the rigid body transformation that maps a point from object space y_c into camera space e, i.e. $e = h_c y_c$.

As described in previous section, the multi-layered forests predict both 3D object coordinates and object instance probability for each pixel of the input image. This is the first stage of the thesis. In the second stage, the pose of an object is estimated as explained briefly in Section 4.6.1. Inspired by [8], the forest output is used to generate pose hypotheses and the poses are optimized using the scoring function as described in Section 4.6.2.

4.6.1 Pose estimation overview

Initially, pose hypotheses are sampled based on observed depth values and coordinate predictions from the forest. Subsequently, these hypotheses are evaluated using a RANSAC-based algorithm in order to find the pose which maximizes the score of equation (4.10).

Figure 4.9 visualizes the process. RGB-D test image is shown in Figure 4.9(a) with upper-half depth image and lower-half RGB image. The estimated 6D pose of the object camera is illustrated with a blue bounding box and the respective ground truth with a green bounding box.



(a) RGB-D test image

(b) Pose optimization

Figure 4.9: Visualization of the search for optimal pose [8]

The inlet in Figure 4.9(b) is a zoom of the center area. The algorithm optimizes the scoring function in a RANSAC-like fashion over a large, continuous 6D pose space. The 6D poses, projected to the image plane, which are visited by the algorithm are color coded: red poses are disregarded in a very fast geometry check; blue poses are evaluated using the scoring function during intermediate fast sampling; green poses are subject to the most expensive refinement step.

Generating pose hypotheses

A single pixel p_1 is drawn from the image using a weight proportional to the object probability $P_{c,p}$ for each pixel p. Two more pixels p_2 and p_3 are drawn from a square window around p_1 using the same method. The size of the window is calculated from the diameter of the object δ_c and the observed depth value $D(p_1)$ of the first pixel, $w = f\delta_c/D(p)$ where f is the focal length. Sampling is done efficiently using an integral image of $P_{c,p}$. For each of the three pixels drawn, coordinates in camera space e_{p_1}, e_{p_2} and e_{p_3} is calculated using the observed depth.

Then, tree indices t_1, t_2 and t_3 are randomly chosen for the three pixels to read out the three associated object coordinate predictions. Together with the camera coordinates, this yields a set of three 3D-3D correspondences $(e_{p_1}, y_c(l_{p_1}^{t_1}))$, $(e_{p_2}, y_c(l_{p_2}^{t_2}))$ and $(e_{p_3}, y_c(l_{p_3}^{t_3}))$. Kabsch algorithm [33] is used to calculate the pose hypothesis h_c that minimizes the squared distance between the three camera coordinate and object coordinate pairs.

The object coordinate predictions of the random forest are very noisy and contain many outliers. Therefore, a geometric check is performed to quickly identify and discard erroneous hypotheses. The predicted object coordinate is mapped into camera space using h_c and a transformation error is calculated as

$$e_{p_1,t_1}(h_c) = \|e_{p_1} - h_c y_c(l_{p_1}^{t_1})\| \tag{4.9}$$

which is the Euclidean distance to the corresponding camera coordinate. Similarly, errors $e_{p_2,t_2}(h_c)$ and $e_{p_3,t_3}(h_c)$ are computed for the remaining pixels.

A pose hypothesis h_c is accepted only if none of the three distances is larger than 5% of the object's diameter δ_c. The process is repeated until a fixed number of 210 hypotheses are accepted. All accepted hypotheses are scored according to equation (4.10).

Refinement

The top 25 hypotheses are selected with respect to the scoring function and refined. To refine a pose h_c, the set of pixels $M_c(h_c)$ supposedly belonging to the object c are iterated over as done for score calculation. For every pixel $p \in M_c(h_c)$, the error $e_{p,t}(h_c)$ is calculated for all trees t.

Let $\hat{t}$ be the tree with the smallest error $e_{p,\hat{t}}(h_c) \leq e_{p,t}(h_c)$ for pixel p. Every pixel p where $e_{p,\hat{t}}(h_c) < 20mm$ is considered an inlier. The correspondence $(e_p, y_c(l_p^t))$ for all inlier pixels are stored and used to re-estimate the pose with the Kabsch algorithm.

This process is repeated until the score of the pose according to Equation (4.10) no longer increases, the number of inlier pixels drops below 3 or a total of 100 iterations is reached.

The final estimate

The pose hypothesis with the highest score after refinement is chosen as the final estimate. The formulation of the task as an optimization problem, however, allows for the use of any general optimization algorithm to further increase the precision of the estimate.

4.6.2 Scoring function

Estimating the pose is formulated as an optimization problem with respect to some scoring function s. The score of a pose hypothesis is calculated by comparing synthetic images rendered using h_c with the observed depth values and the predictions of the multi-layered forests. The scoring function consists of three main components and is formulated as

$$s_c(h_c) = \lambda^{depth} s_c^{depth}(h_c) + \lambda^{coord} s_c^{coord}(h_c) + \lambda^{seg} s_c^{seg}(h_c) \tag{4.10}$$

where the factors λ^{depth}, λ^{coord} and λ^{seg} reflect the reliability of the different observations. The three components in the scoring function are described as:

The depth component

This component measures deviations between the observed and ideal rendered depth images by comparing them. It is defined as

$$s_c^{depth}(h_c) = -\frac{\sum_{p \in M_c(h_c)} f(D_p, \hat{D}_p(h_c))}{|M_c(h_c)|} \tag{4.11}$$

where $M_c(h_c)$ is the object mask, i.e. the set of pixels belonging to object c. It is derived from the pose h_c by rendering the object into the image and the pixels with missing depth values are excluded. The term $\hat{D}_p(h_c)$ is the depth at pixel p produced by rendering the 3D model of object c with pose h_c. $|M_c(h_c)|$ in the denominator normalizes the depth component to make it independent of the object's distance to the camera. A robust error function

$$f(D_p - \hat{D}_p(h_c)) = min(|D_p - \hat{D}_p(h_c)|, \tau_d)/\tau_d$$

is used to handle inaccuracies in the 3D model, where τ_d is the cutoff threshold.

The coordinate component

This component measures the consistency of pose hypothesis and object coordinates prediction by calculating the deviations between the object coordinates $y_c(l_p^t)$ predicted by the forest and rendered object coordinates $\hat{y}_{p,c}(h_c)$.

$$s_c^{coord}(h_c) = -\frac{\sum_{p \in \hat{M}_c(h_c)} \sum_{t=1}^{|T|} g_c(y_c(l_p^t)\hat{y}_{p,c}(h_c))}{|\hat{M}_c(h_c)|} \tag{4.12}$$

where $\hat{M}_c(h_c)$ is the set of pixels belonging to object c excluding pixels with no depth observation and pixels where the object probability is smaller than the threshold i.e.

$P_(c, p) < \tau_c$ (since pixels with small $P_(c, p)$ do not provide reliable object coordinate predictions $y_c(l_p^t)$). The term $\hat{y}_{p,c}(h_c)$ denotes object coordinates rendered using the 3D model of object c with pose h_c. Again, a robust error function

$$g_c(y_c(l_p^t)\hat{y}_{p,c}(h_c)) = min(\|y_c(l_p^t) - \hat{y}_{p,c}(h_c)\|^2, \tau_y)/\tau_y$$

is used to handle inaccuracies in the 3D model, where τ_y is the cutoff threshold.

The segmentation component

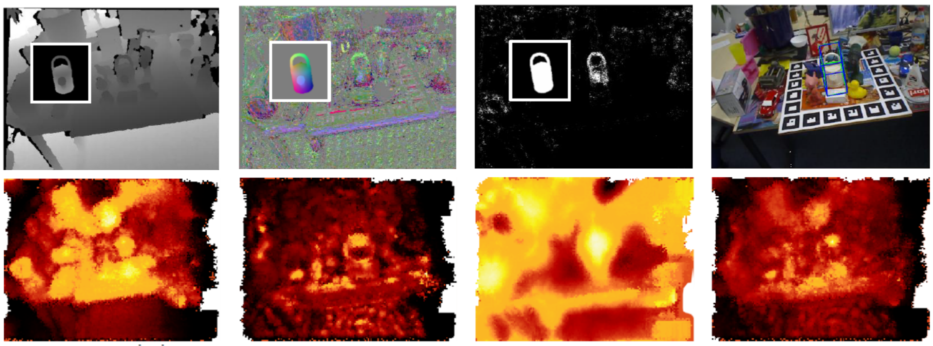
This component compares a rendered segmentation with the soft segmentation, i.e. the object probabilities $P(c|l_p^t)$, predicted by the forest. It punishes pixels inside the ideal segmentation $M_c(h_c)$ which are according to the forest unlikely to belong to the object. It is defined as

$$s_c^{seg}(h_c) = -\frac{\sum_{p \in M_c(h_c)} \sum_{t=1}^{|T|} -logP(c|l_p^t)}{|M_c(h_c)|} \tag{4.13}$$

Final scoring function

Since the scoring components are all normalized, $s_c(h_c) = 0$ is returned whenever the number of pixels considered becomes very small i.e. when $|\hat{M}_c(h_c)| < 100$.

Figure 4.10 visualizes the composition of scoring function and benefits of each component for the Can object.



(a) Depth (b) Coordinate (c) Segmentation (d) Final energy

Figure 4.10: Components of the scoring function [8]

Different input channels from which individual components of the scoring function are calculated is illustrated at the top. The depth channel, object coordinate predictions of the last layer, object probability predictions, and the RGB image with the final pose estimate (blue) compared to ground truth (green) as shown from left to right. As an inlay for each channel, the corresponding object renderings are shown which are checked for consistency by the scoring function. At the bottom, dense score maps for each score component and the full scoring function are calculated for visualization.

Note that for pose optimization, scores are evaluated only very sparsely for a few pose hypotheses.

A dense sampling of the 6D pose space for the input image is created to calculate the score for each pose sample. Later, the highest score is projected into the image for each pixel ray. Bright color means high score. While different scoring components display strong local maxima, their combination usually shows the strongest maxima at the correct pose.

5 Datasets

The SIXD Challenge 2017 [11, 12], organized at the 3rd International Workshop on Recovering 6D Object Pose at ICCV 2017, proposed a set of datasets for evaluating the task of 6D localization of a single instance of a single object. This task reflects the industry-relevant bin-picking scenario where a robot needs to grasp a single arbitrary instance of the required object, e.g. a component such as a bolt or nut, and perform some operation with it. The six datasets collected are namely Hinterstoisser, T-LESS, TUD Light, Rutgers, Tejani and Doumanoglou, and are publicly available.

Each dataset contains a set of texture-mapped 3D object models and training and test RGB-D images annotated with ground-truth 6D object poses. The 3D object models were created using KinectFusion-like systems for 3D surface reconstruction [34, 35]. The proposed scenes cover a wide range of cases with a variety of objects in different poses and environments including multiple instances, clutter and occlusion.

The approach described in this thesis was initially evaluated on the widely used dataset of Hinterstoisser et al. [10] and later on the T-LESS dataset [9]. Synthetic and real images of isolated, i.e. non-occluded, objects were used for training with the Hinterstoisser and T-LESS datasets respectively. Both the datasets consist of real test images that were captured in scenes with varying complexity, often with clutter and occlusion.

5.1 Hinterstoisser dataset

The Hinterstoisser dataset, also known as the LINEMOD dataset, has been the most commonly used dataset for 6D object pose estimation with accurate 6D pose annotations. It consists of 15 RGB-D image sequences corresponding to 15 different objects of interest, as illustrated in Figure 5.1. Here, 13 out of the total 15 textureless household objects with discriminative color, shape and size is considered. The other 2 objects were omitted since proper 3D models were missing.

Each sequence consists of approximately 1,200 frames sampling the upper view hemisphere of the corresponding object including ±45 in-plane rotation. The objects feature little amount of texture and are approximately 15cm to 30cm in diameter.

For each frame, the rotation and translation of the object is given relative to the camera. The distance between the object of interest and the camera ranges between 65cm and 115cm. The camera calibration parameters are also known with a focal length of 573px and a principal point at (325, 242).

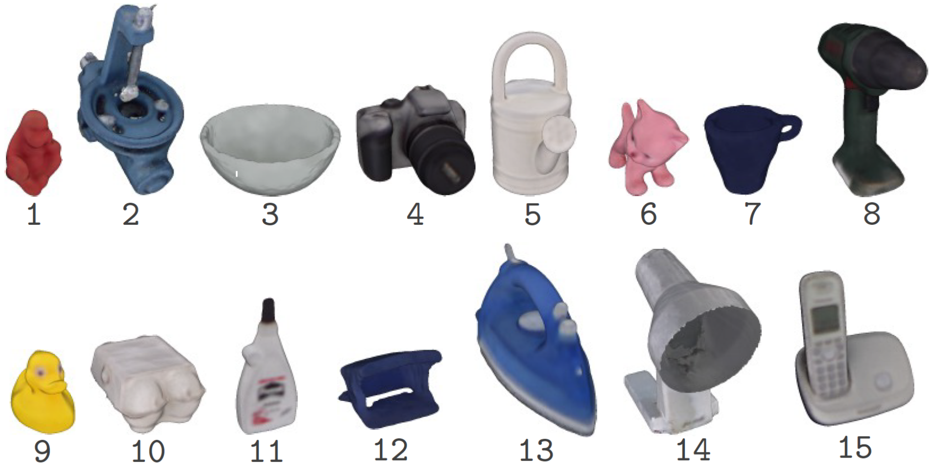


Figure 5.1: 15 objects in the dataset created by Hinterstoisser et al.

The training images show individual objects from different viewpoints and were obtained by rendering of the 3D object models. These synthetic images for the objects duck and cat are illustrated in Figures 5.2 and 5.3.

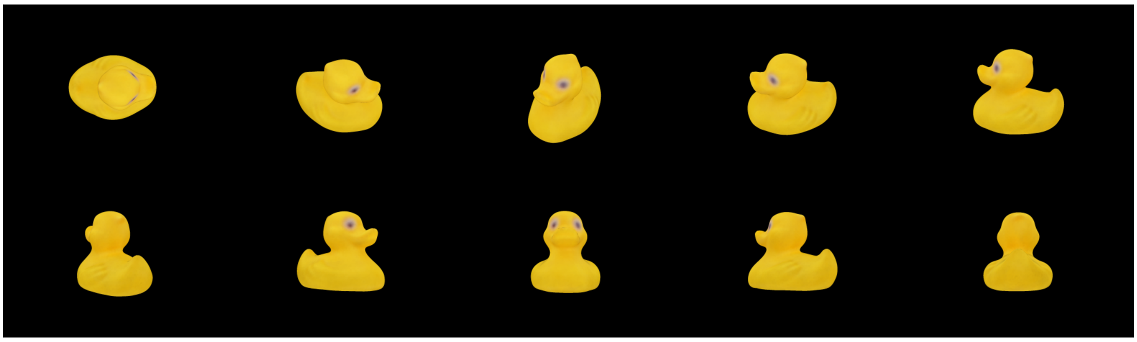


Figure 5.2: Training images for the object duck

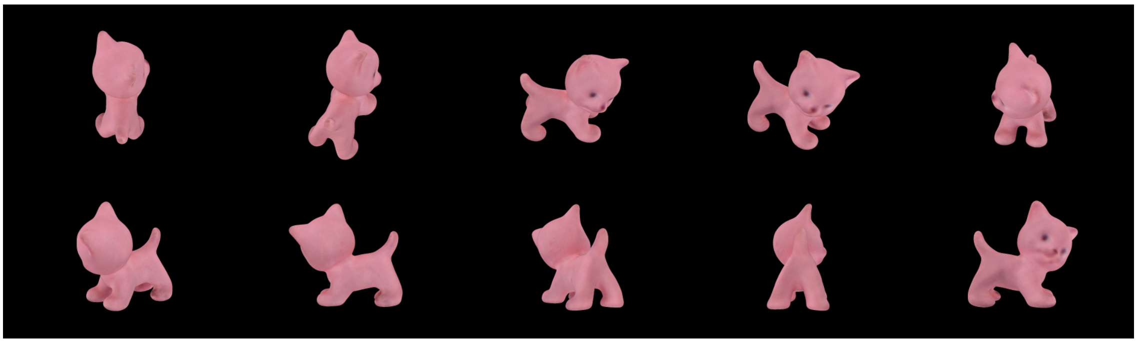


Figure 5.3: Training images for the object cat

Each object is associated with a real test image set showing one annotated object instance with significant clutter but only mild occlusion, as shown in Figure 5.4. It also provides ground-truth annotation for all other instances of the modeled objects in one of the test sets. For each test frame, the ID of the object of interest is given, and only the pose has to be estimated. Performance is measured as the percentage of test frames per sequence where the pose has been estimated correctly, subject to some threshold. This threshold is defined via the average distance of transformed vertices of the object's 3D model.

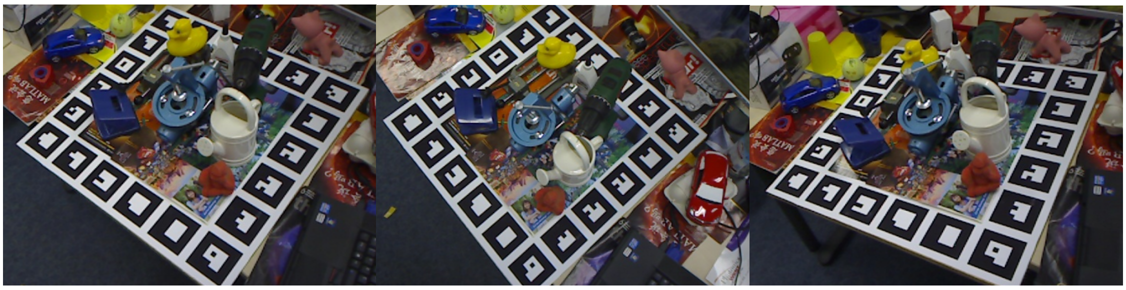


Figure 5.4: Test scene with the object of interest surrounded by dense clutter

The object of interest is usually displayed in the center of the frame surrounded by dense clutter on a work desk. The clutter consists predominately of objects from the respective other image sequences but their pose is not annotated. The specific clutter configuration changes throughout each sequence, probably to avoid severe occlusion. Therefore, the object of interest is never occluded substantially. Apart from that, the imaging conditions among all sequences are static, i.e. the camera type, lighting conditions and the global desk setup are static.

5.2 T-LESS dataset

The T-LESS dataset consists of 30 industry-relevant objects with no significant texture and no discriminative color or reflectance properties, as shown in Figure 5.5. The objects exhibit symmetries and mutual similarities in shape and/or size. Compared to other datasets, a unique property is that some of the objects are parts of others. For example, objects 7 and 8 are built up from object 6, object 9 is made of three copies of object 10 stacked next to each other, whilst the center part of objects 17 and 18 is nearly identical to object 13. Objects exhibiting similar properties are common in industrial environments.

The dataset includes training and test images captured with a triplet of sensors, i.e. a structured light RGB-D sensor Primesense Carmine 1.09, a time-of-flight RGB-D sensor Microsoft Kinect v2, and an RGB camera Canon IXUS 950 IS. The sensors were time-synchronized and had similar perspectives. All images were obtained with an automatic procedure that systematically sampled images from a view sphere,

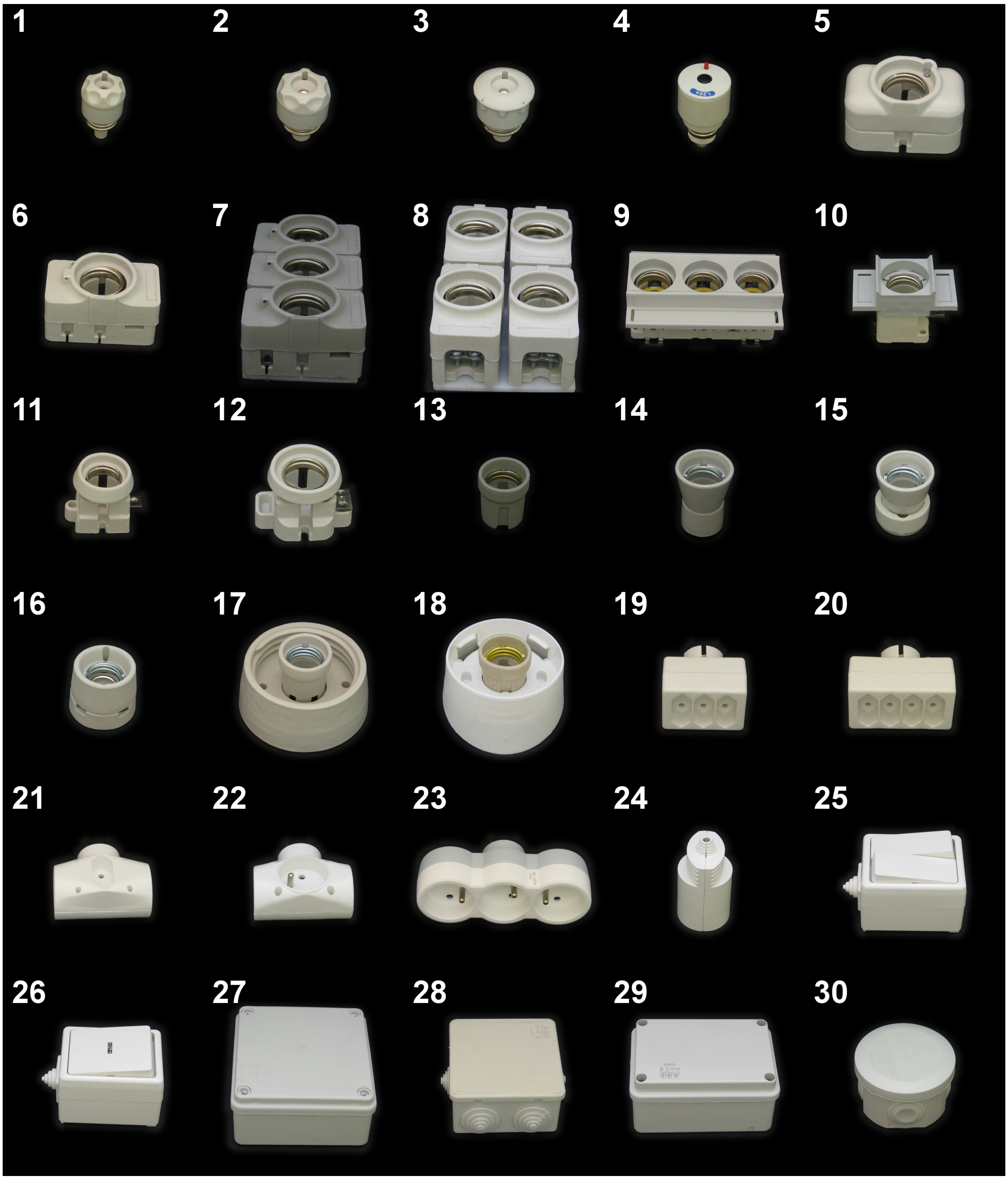


Figure 5.5: Objects included in the T-LESS dataset

resulting in ∼39K training and ∼10K test images from each sensor.

The depth of object surfaces in the training and test images is in the range 0.53m - 0.92m, which is within the sensing range of the used Primesense Carmine sensor that is 0.35m - 1.4m. To remove irrelevant parts of the scene in the images periphery, the provided images are cropped versions of the captured ones. Resolution of the provided images are 400×400 px for training images and 720×540 px for test images.

The training images depict objects in isolation with a black background from a full view sphere, as shown in Figures 5.6 and 5.7 for objects 2 and 25 respectively.

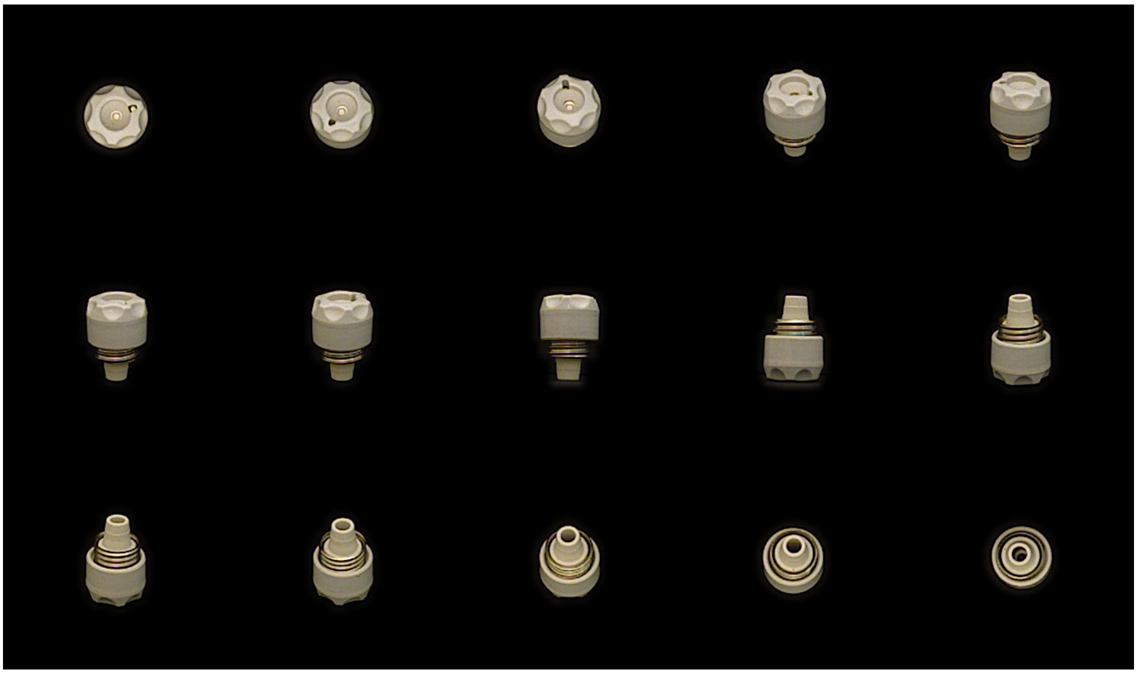


Figure 5.6: Sample training images for object 2

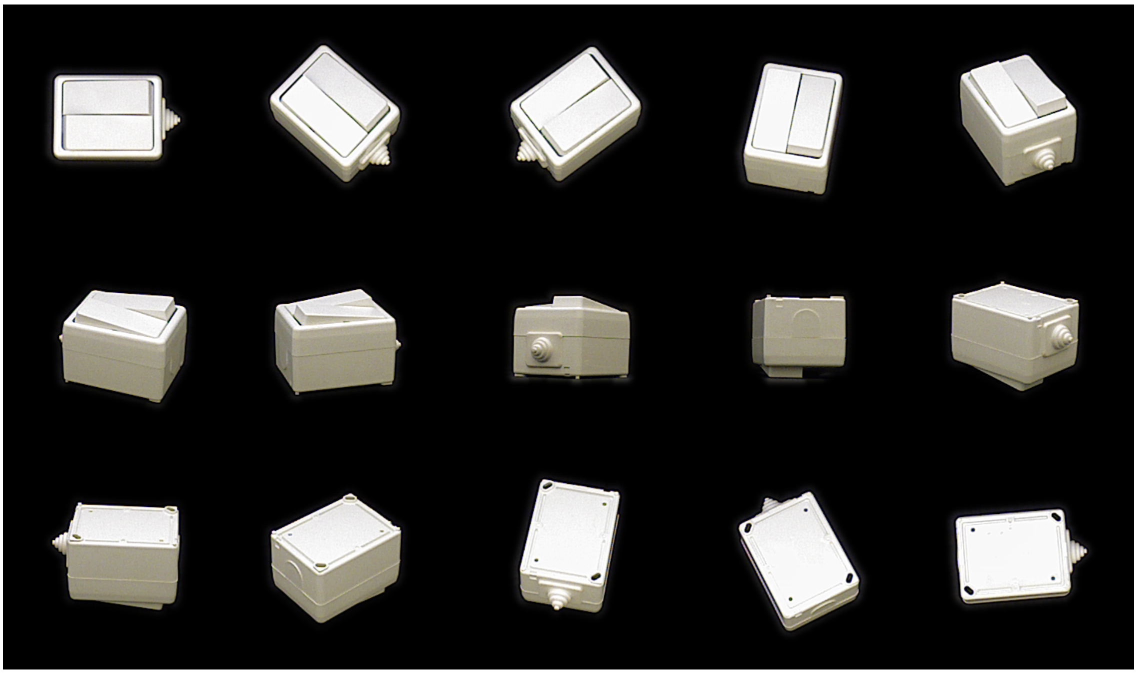


Figure 5.7: Sample training images for object 25

These images were obtained with a systematic acquisition procedure which uniformly sampled elevation from 85° to −85° with a 10° step and the complete azimuth range with a 5° step. Views from the upper and lower hemispheres were captured separately, turning the object upside down in between. In total, there are $18 \times 72 = 1296$ training images per object. Exceptions are objects 19 and 20, for which only views from the upper hemisphere were captured, specifically 648 images from elevation 85° to 5°.

The test images originate from 20 table-top scenes with arbitrarily arranged objects, as shown in Figure 5.8 overlaid with colored 3D object models at the ground truth poses.

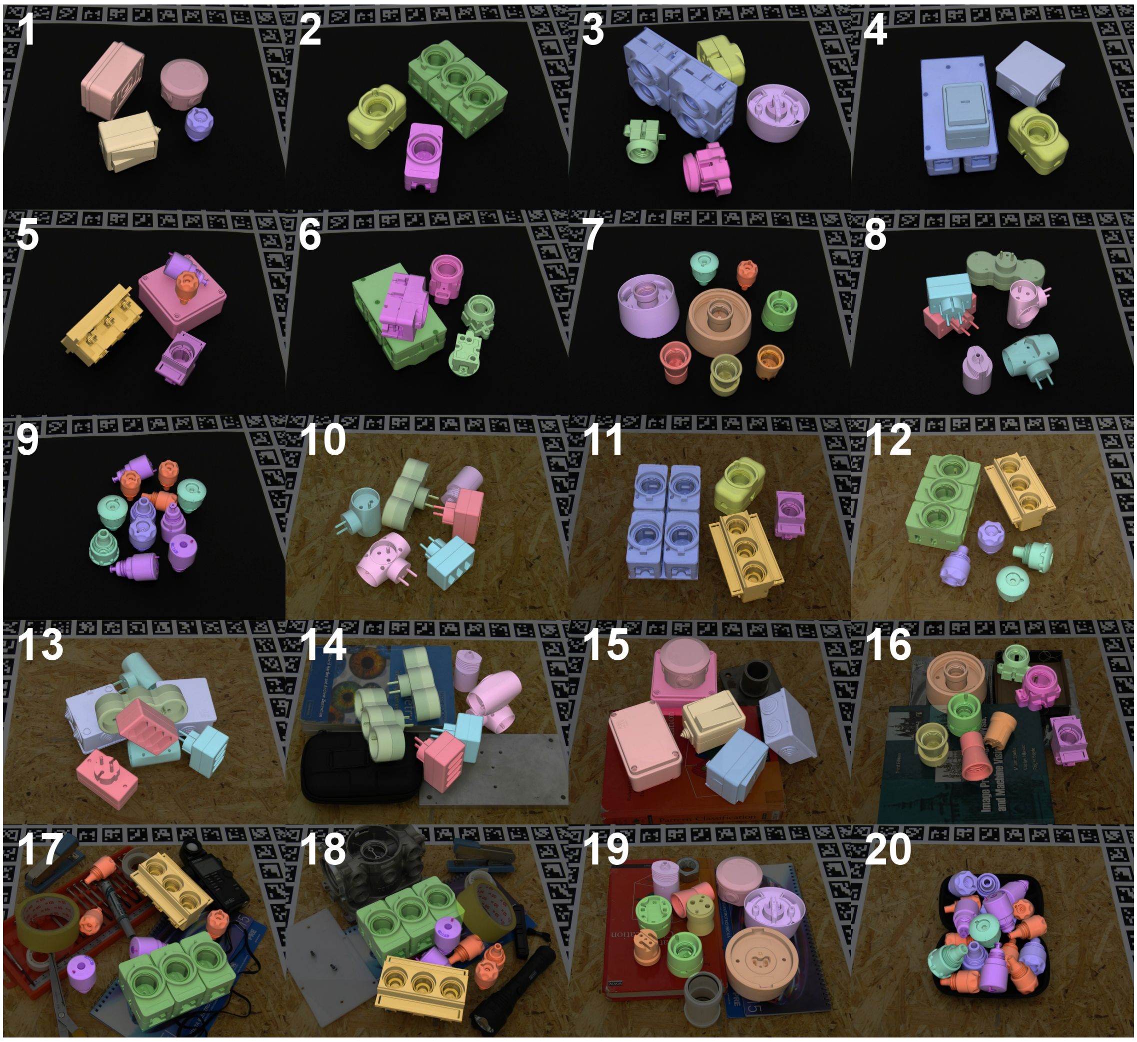


Figure 5.8: Test scenes overlaid with colored 3D object models at ground truth poses

Complexity of the test scenes varies from those with several isolated objects and a clean background to very challenging ones with multiple instances of several objects and with a high amount of occlusion and clutter.

The test images were captured from a view hemisphere with a 10° step in elevation (ranging from 75° to 15°) and a 5° step in azimuth. A total of $7 \times 72 = 504$ test images were captured per scene. Sample images are shown in Figure 5.9

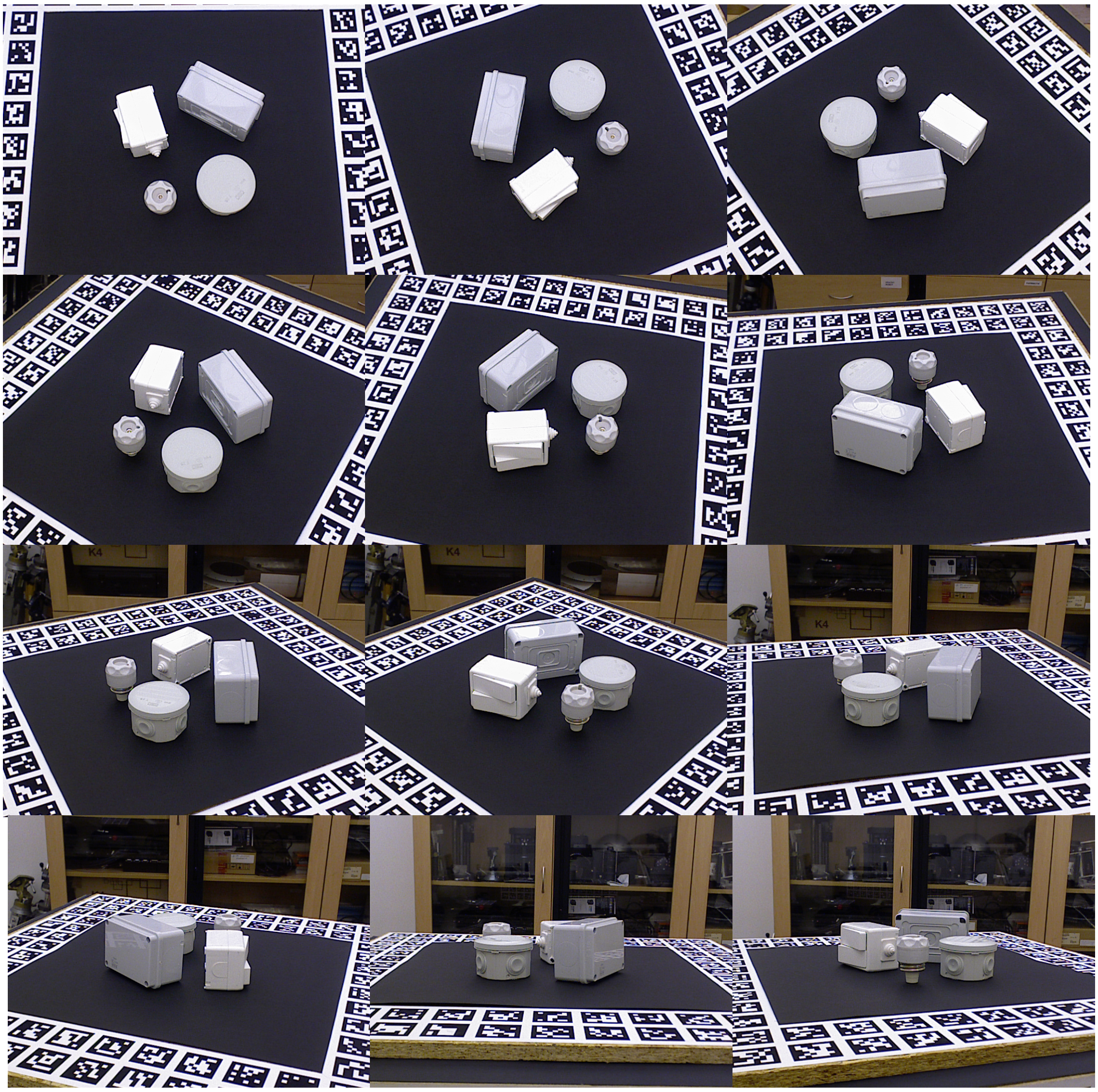


Figure 5.9: Sample test images of scene 1 with objects 2, 25, 29 and 30

Additionally, the dataset contains two types of 3D mesh models for each object; one manually created in CAD software and one semi-automatically reconstructed from the training RGB-D images. All occurrences of the modeled objects in the training and test images are annotated with accurate ground truth 6D poses.

Therefore, Hinterstoisser and T-LESS datasets are used to evaluate the approach presented in this thesis and the results from the SIXD Challenge for object pose estimation will serve as a measure of the actual state of the art.

6 Experiments and Discussion

As mentioned in Section 5, the multi-layer random forests architecture is evaluated on the widely used dataset of Hinterstoisser et al. and the T-LESS dataset. The dataset of Hinterstoisser et al. provides synthetic training and real test data. The T-LESS dataset provides real training and real test data with realistic noise patterns.

6.1 Evaluation metrics

6.1.1 Pose estimation

The goal is to evaluate the accuracy in pose estimation for each object per image. It is known which object is present. One possible metric for evaluation would be to compute the percentage of test images where the pose of the object in question was estimated correctly. Hintertoisser et al. [10] define a threshold on the average distance of transformed 3D points. The exact tolerance to translational and rotational error depends on object size and shape. In this thesis, the test protocol of Brachmann et al. [4] is followed where they additionally propose the 2D projection measure.

This measure is calculated using a 3D model or a point cloud of the object. The estimated pose is accepted, if the average re-projection error of all model vertices is below τ_p (called as 2D Projection). To calculate the re-projection error, the model vertices are projected into the image using the ground truth pose h and the estimated pose $\tilde{h}$. The average distance of the projections of corresponding vertices is calculated as

$$\frac{1}{|\mathcal{V}|} \sum_{v \in \mathcal{V}} \| Chv - C\tilde{h}v \| < \tau_p \tag{6.1}$$

where $\mathcal{V}$ is the set of all object model vertices and C is the camera matrix.

6.1.2 Object detection

In many applications, the presence of an object is unknown and has to be established first. This can be done by defining a threshold on the score of the final pose estimate. The system would report a detection only if this score is below the threshold.

The full pipeline is run to extract one hypothesis per image and a 2D bounding box is extracted based on this hypothesis. The bounding boxes of all images of a sequence are ranked according to their hypothesis' score. The ground truth bounding box is extracted using the ground truth pose. A detection is considered correct if the intersection over union (IoU) of detected bounding box and ground truth bounding box is at least 50%.

6.2 Experimental set-up

For the experiments, the architecture depicted in Figure 1.2 is used. The framework is trained on images of objects from the Hinterstoisser and T-LESS datasets as described in Section 5 and tested on scenes such as shown in Figures 5.4 and 5.9.

The decision forests are trained such that at each node, 500 color features and depth features are sampled. In each iteration, 1000 random pixels per training image are chosen and collected in the current leafs. The splitting stops if either less than 50 pixels arrive at each leaf or if the tree depth is more than 64.

While parameters such as the number of layers d and tree counts $|T|$ are varied, the following parameters are set constant as stated in Tables 6.2 and 6.1 for all the pose estimation experiments.

Training Parameters	
Number of features generated at each node	1000
Ratio of $da - d$ to $da - rgb$ features	0.5
Maximum feature offset	10 pixel meters
Random pixels per image to learn tree structure	1000
Random pixels per image to learn leaf distributions	5000
Stopping criterion: minimum number of pixels per node	50
Stopping criterion: maximum tree depth	64
Object coordinate proxy classes	125

Table 6.1: Default training parameter settings

Testing Parameters	
Score depth component weight λ^{depth}	1.5
Score coordinate component weight λ^{coord}	1
Score segmentation component weight λ^{seg}	1
Threshold τ_d used in s_c^{depth}	50 mm
Threshold τ_y used in s_c^{coord}	$(0.2 \cdot \delta_c)^2$ with object diameter δ_c
Threshold τ_c used in s_c^{coord}	10^{-8}
Number of hypothesis to be sampled	210
Threshold used during sampling of poses	$0.05 \cdot \delta_c$ with object diameter δ_c
Inlier threshold used in refinement	20 mm
Number of hypothesis to be refined	25

Table 6.2: Default testing parameter settings

6.2.1 Without auto-context

Multiple layers of forests T^i are modelled in the architecture, where $i \in \{0, ..., d\}$ denotes the layer number. The first layer T^0 is trained exactly as described in Section 4.3. In this approach, only the first layer is considered and the tree count is varied.

Every tree in the forest predicts an object class and an object coordinate point estimate per pixel. Therefore, the pose is estimated using a standard random forest without any auto-context information. The rate of correctly estimated poses per object for different tree counts is as shown in Figure 6.1.

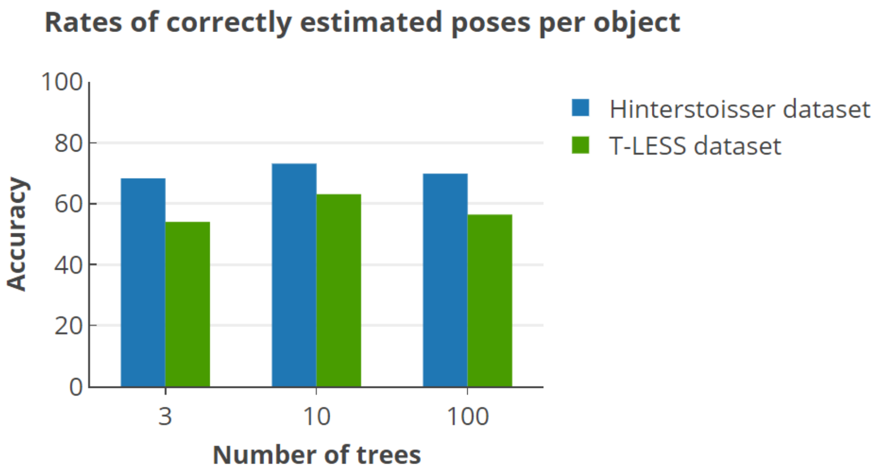


Figure 6.1: Results on the Hinterstoisser and T-LESS dataset for different tree counts

	Number of trees		
	3	10	100
Hinterstoisser	68.83	73.13	69.83
T-LESS	54	63.1	56.4

Table 6.3: Accuracy when testing with different number of trees in a forest

Table 6.3 summarizes the results for tree counts 3, 10 and 100. For the synthetic training images of the Hinterstoisser dataset, an average of 68.83% is scored with 3 trees in the forest. For 10 trees, the average rate is improved to 73.13% while using 100 trees leads to over-fitting.

To verify that the approach is not restricted to synthetic training data, experiments were performed by training with real images. Using real training images from the T-LESS dataset results in 63.1% correctly estimated poses on average for a tree count of 10.

6.2.2 With auto-context

From the second layer of the multi-layered architecture, all subsequent forests T^{i+1} have access to the output of the previous forest T^i, namely object probabilities $P_p^i(c)$ and object coordinate predictions $y_c(l_p)$ of pixel p. This concept of a simple random forest is extended to an auto-context framework. This is because the dense object coordinates may contain a substantial amount of “structural” information, i.e. neighboring object coordinate predictions are statistically dependent.

Therefore, there is an improvement in the recall rate as observed in Table 6.4 with the auto-context structure for tree count 3, when compared to the results of only the first layer (without auto-context).

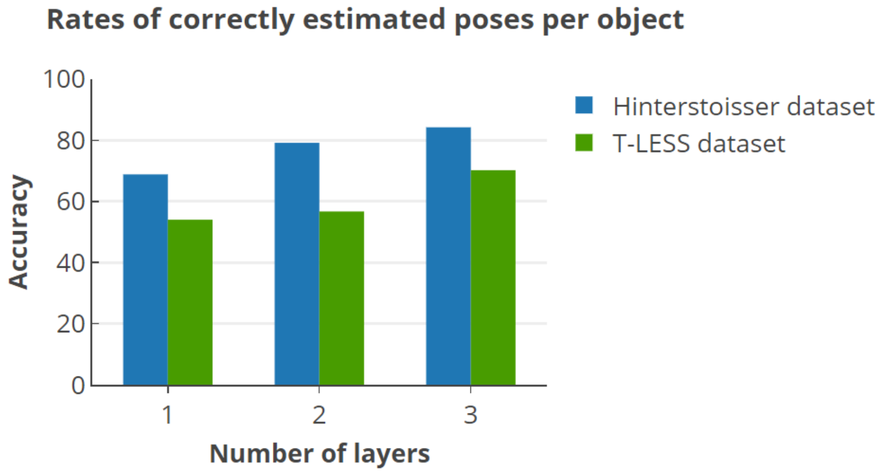


Figure 6.2: Results with the auto-context structure for 3 trees in each forest

	Number of layers		
	1	2	3
Hinterstoisser	68.83	79.16	84.23
T-LESS	54	56.7	70.2

Table 6.4: Accuracy with different number of layers of forests

The performance of the multi-layered architecture is evaluated for a tree count of 10 in each forest. The results improve with every layer, as shown in Table 6.5. At the end of 3 layers, the accuracy increases to 84.73% and 73.6% for the two datasets.

	Number of layers		
	1	2	3
Hinterstoisser	73.16	82.16	84.73
T-LESS	63.1	70.7	73.6

Table 6.5: Accuracy with different number of layers for tree count 10

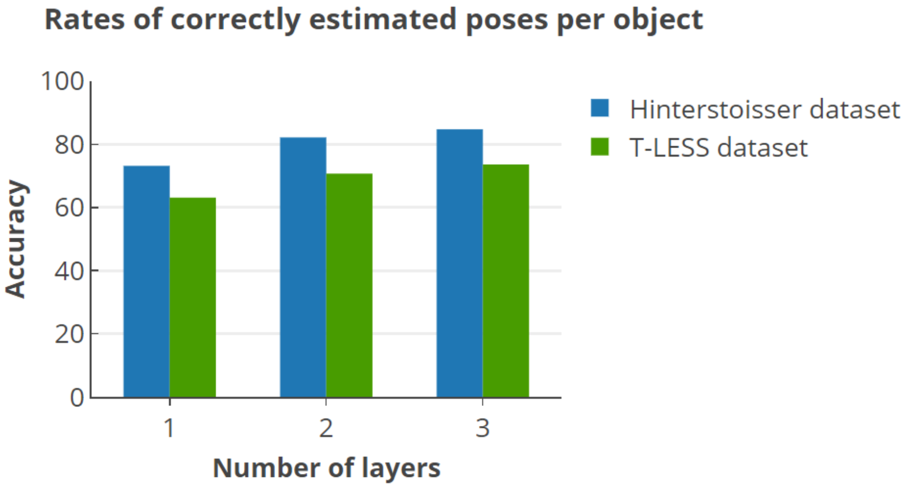


Figure 6.3: Results with the auto-context structure for 10 trees in each forest

Therefore, the auto-context algorithm integrates the image appearances (observed data) together with the context information by learning a series of estimators. In this work, it is used to iteratively reduce the uncertainty of object coordinates and class label predictions at each layer. Figure 6.4 indicates the probabilities of object labels and object coordinates that are jointly predicted for every pixel of the input image in each layer.

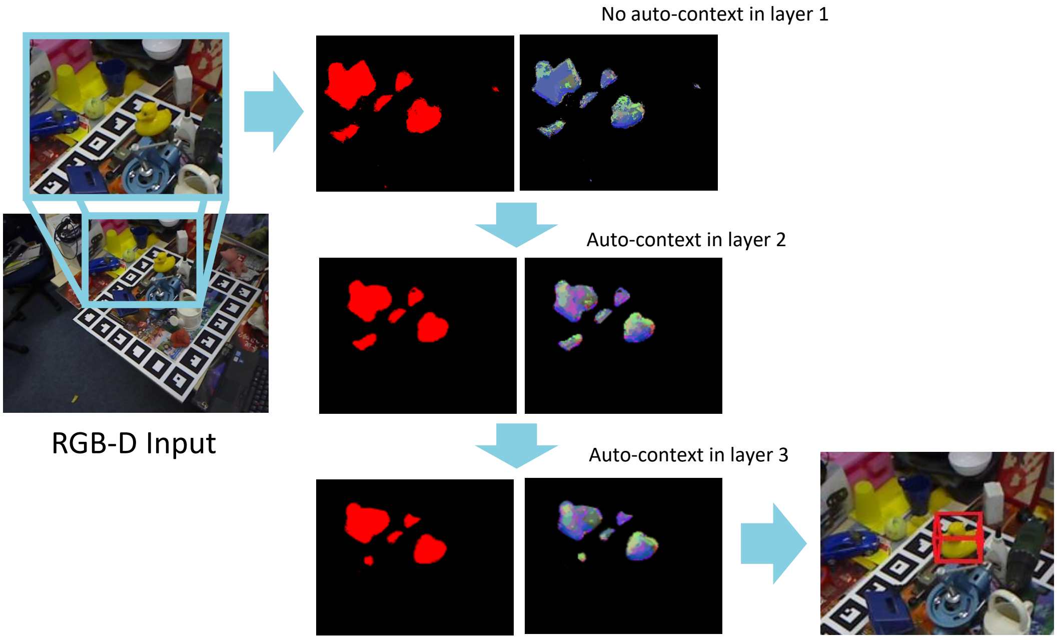
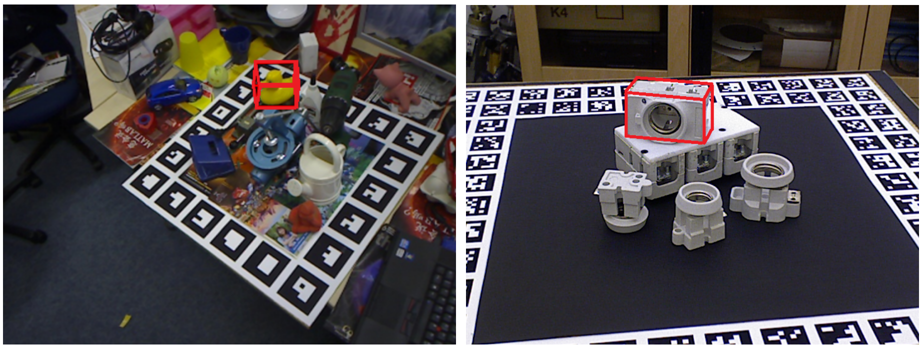


Figure 6.4: Results at each layer for 10 trees in each forest

So far, the pipeline has one learned component: An auto-context random forest. While it achieves good results, it is unclear how to adjust the training of the random forest for increasing the accuracy of pose estimates. The quality of object coordinate predictions cannot be improved directly. However, during training, the information gain of proxy classes is optimized per split node of the random forest in a greedy fashion. The relation of this training objective and the pose accuracy is not straight forward, especially because the RANSAC-based pose optimization is robust to errors in the object coordinate prediction. For example, the auto-context improves the quality of object coordinate predictions. However, this did not translate to more accurate pose estimates.

Examples for pose estimation with the multi-layered architecture is as shown in Figure 6.5.



(a) Hinterstoisser dataset (b) T-LESS dataset

Figure 6.5: Results for pose estimation with the multi-layered architecture

The results are compared to the RGB-D pose estimation pipelines of Brachmann et al. [8, 4], as shown in Table 6.6. The publicly available binaries are used to measure the accuracy. The results of a recent method of Krull et al. is also compared which is an extension of the pipeline of Brachmann et al. [8] by utilizing a CNN in the final pose optimization stage.

	Hinterstoisser
Brachmann et al. [8]	52.1
Krull et al.	73.1
Brachmann et al. [4]	82.1
Multi-layer RF	84.73

Table 6.6: Pose estimation results on the Hinterstoisser data set

The implementation of the proposed framework takes approximately 50s to estimate the pose of one object on a single frame. In Section 4.3, the procedure to train a forest that predicts segmentations and object coordinates simultaneously is discussed. The runtime of the forest scales logarithmically in the number of objects.

In Section 4.6, a scoring function is formulated which assesses the pose hypothesis by evaluating the forest at a sparse set of pixels. Later, a hypothesis sampling scheme for RANSAC is discussed which decides for which object a hypothesis should be created based on predicted object label distributions. Therefore, the pose optimization stage is performed sublinearily to the number of objects. The whole system is able to process objects in approximately 50s with a CPU only implementation. A large boost is expected for a GPU port as this implementation runs on CPU and is not optimized.

7 Conclusion and Future Work

In this thesis, the potential of the multi-layer random forests architecture has been shown. Given an RGB-D image, a framework which can estimate the 6D pose of a rigid object instance is presented. It is based on random forests that learn to decide, for each pixel of the input image, whether it belongs to the object and where on the object surface it is located. A subsequent scoring function and RANSAC-based geometric optimization yields a stable and accurate estimate of the object pose with low runtime. The performance of the pipeline is measured in terms of accuracy of estimated poses.

The multi-layer random forests framework is computationally efficient for object detection and pose estimation as the features are computed by pixel comparisons. The features consider depth or RGB differences from pixels in the vicinity of a pixel. Each feature need only read at most 3 image pixels and perform at most 5 arithmetic operations. Thus, they are simple, extremely fast to evaluate and robust to occlusion. Also, as seen in Section 6, Random Forests usually do not have high amount of parameters to optimize during training. The architectural varieties is much lower than for deep neural networks. Hence, training with smaller datasets is possible.

The main limitation of this work is that pose optimization needs to be performed for each object the forest has been trained with. Objects not present in the image can only be discarded after optimization by thresholding the final pose score. Additionally, the experiments in Section 6 were limited to cases with no or little amount of occlusion. Experiments on highly occluded data resulted in fairly well detection of objects but pose estimates were inaccurate.

Certainly, a future goal is to evaluate on the complete T-LESS dataset to compare with the BOP [12]. Also, the pipeline may be adapted to estimate poses from RGB inputs only. Preliminary experiments showed that the network can be trained and tested using plain RGB data, relying only on the discriminative predictions of the random forest to estimate poses.

While this framework supports a wide variety of object types, there are still many objects which are particularly challenging. Specifically, the pipeline should be extended to handle articulated objects such as toys or a cupboard with a moving door, instead of rigid objects. For such freely deformable objects, the prediction of object coordinates based on image patches is still sensible. The deformation is likely to change the global shape of an object but less so the local appearance. However, the subsequent RANSAC-based pose optimization procedure needs to be investigated further.

Bibliography

[1] S. Kriegel, M. Brucker, Z.-C. Marton, T. Bodenmuller, and M. Suppa, ‘‘Combining object modeling and recognition for active scene exploration,’’ in *Intelligent Robots and Systems (IROS), 2013 IEEE/RSJ International Conference on*. IEEE, 2013, pp. 2384--2391.

[2] B. Bäuml, O. Birbach, T. Wimböck, U. Frese, A. Dietrich, and G. Hirzinger, ‘‘Catching flying balls with a mobile humanoid: System overview and design considerations,’’ in *Humanoid Robots (Humanoids), 2011 11th IEEE-RAS International Conference on*. IEEE, 2011, pp. 513--520.

[3] ‘‘Robex mond-analog-mission sizilien 2017,’’ accessed: 09-10-2018, Video at 1min 43sec. [Online]. Available: https://youtu.be/-wXQf0b1bqQ

[4] E. Brachmann, F. Michel, A. Krull, M. Ying Yang, S. Gumhold *et al.*, ‘‘Uncertainty-driven 6d pose estimation of objects and scenes from a single rgb image,’’ in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2016, pp. 3364--3372.

[5] J. Valentin, M. Nießner, J. Shotton, A. Fitzgibbon, S. Izadi, and P. H. Torr, ‘‘Exploiting uncertainty in regression forests for accurate camera relocalization,’’ in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2015, pp. 4400--4408.

[6] Z.-H. Zhou and J. Feng, ‘‘Deep forest: Towards an alternative to deep neural networks,’’ *arXiv preprint arXiv:1702.08835*, 2017.

[7] Z. Tu and X. Bai, ‘‘Auto-context and its application to high-level vision tasks and 3d brain image segmentation,’’ *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 32, no. 10, pp. 1744--1757, 2010.

[8] E. Brachmann, A. Krull, F. Michel, S. Gumhold, J. Shotton, and C. Rother, ‘‘Learning 6d object pose estimation using 3d object coordinates,’’ in *European conference on computer vision*. Springer, 2014, pp. 536--551.

[9] T. Hodan, P. Haluza, Š. Obdržálek, J. Matas, M. Lourakis, and X. Zabulis, ‘‘T-less: An rgb-d dataset for 6d pose estimation of texture-less objects,’’ in *Applications of Computer Vision (WACV), 2017 IEEE Winter Conference on*. IEEE, 2017, pp. 880--888.

[10] S. Hinterstoisser, V. Lepetit, S. Ilic, S. Holzer, G. Bradski, K. Konolige, and N. Navab, ‘‘Model based training, detection and pose estimation of texture-less 3d objects in heavily cluttered scenes,’’ in *Asian conference on computer vision*. Springer, 2012, pp. 548--562.

[11] T. Hodan, “Sixd challenge 2017.” [Online]. Available: http://cmp.felk.cvut.cz/sixd/challenge_2017/

[12] T. Hodan, F. Michel, E. Brachmann, W. Kehl, A. GlentBuch, D. Kraft, B. Drost, J. Vidal, S. Ihrke, X. Zabulis *et al.*, “Bop: Benchmark for 6d object pose estimation,” in *Proceedings of the European Conference on Computer Vision (ECCV)*, 2018, pp. 19--34.

[13] R. Caruana, N. Karampatziakis, and A. Yessenalina, “An empirical evaluation of supervised learning in high dimensions,” in *Proceedings of the 25th international conference on Machine learning.* ACM, 2008, pp. 96--103.

[14] L. Breiman, “Random forests,” *Machine Learning*, vol. 45, no. 1, pp. 5--32, Oct 2001. [Online]. Available: https://doi.org/10.1023/A:1010933404324

[15] A. Criminisi, J. Shotton, E. Konukoglu *et al.*, “Decision forests: A unified framework for classification, regression, density estimation, manifold learning and semi-supervised learning,” *Foundations and Trends® in Computer Graphics and Vision*, vol. 7, no. 2--3, pp. 81--227, 2012.

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

[17] J. Gall, A. Yao, N. Razavi, L. Van Gool, and V. Lempitsky, “Hough forests for object detection, tracking, and action recognition,” *IEEE transactions on pattern analysis and machine intelligence*, vol. 33, no. 11, pp. 2188--2202, 2011.

[18] A. Montillo, J. Shotton, J. Winn, J. E. Iglesias, D. Metaxas, and A. Criminisi, “Entangled decision forests and their application for semantic segmentation of ct images,” in *Biennial International Conference on Information Processing in Medical Imaging.* Springer, 2011, pp. 184--196.

[19] A. Montillo, J. Tu, J. Shotton, J. Winn, J. E. Iglesias, D. N. Metaxas, and A. Criminisi, “Entanglement and differentiable information gain maximization,” in *Decision Forests for Computer Vision and Medical Image Analysis.* Springer, 2013, pp. 273--293.

[20] P. Kontschieder, S. R. Bulò, A. Criminisi, P. Kohli, M. Pelillo, and H. Bischof, “Context-sensitive decision forests for object detection,” in *Advances in neural information processing systems*, 2012, pp. 431--439.

[21] J. Shotton, M. Johnson, and R. Cipolla, “Semantic texton forests for image categorization and segmentation,” in *Computer vision and pattern recognition, 2008. CVPR 2008. IEEE Conference on.* IEEE, 2008, pp. 1--8.

[22] V. Ramakrishna, D. Munoz, M. Hebert, J. A. Bagnell, and Y. Sheikh, “Pose machines: Articulated pose estimation via inference machines,” in *European Conference on Computer Vision.* Springer, 2014, pp. 33--47.

[23] I. Gordon and D. G. Lowe, “What and where: 3d object recognition with accurate pose,” in *Toward category-level object recognition.* Springer, 2006, pp. 67--82.

[24] V. Ferrari, T. Tuytelaars, and L. Van Gool, “Object detection by contour segment networks,” in *European conference on computer vision.* Springer, 2006, pp. 14--28.

[25] N. Dalal and B. Triggs, “Histograms of oriented gradients for human detection,” in *Computer Vision and Pattern Recognition, 2005. CVPR 2005. IEEE Computer Society Conference on*, vol. 1. IEEE, 2005, pp. 886--893.

[26] P. F. Felzenszwalb, R. B. Girshick, D. McAllester, and D. Ramanan, “Object detection with discriminatively trained part-based models,” *IEEE transactions on pattern analysis and machine intelligence*, vol. 32, no. 9, pp. 1627--1645, 2010.

[27] S. Hinterstoisser, S. Holzer, C. Cagniart, S. Ilic, K. Konolige, N. Navab, and V. Lepetit, “Multimodal templates for real-time detection of texture-less objects in heavily cluttered scenes,” in *Computer Vision (ICCV), 2011 IEEE International Conference on.* IEEE, 2011, pp. 858--865.

[28] J. Shotton, B. Glocker, C. Zach, S. Izadi, A. Criminisi, and A. Fitzgibbon, “Scene coordinate regression forests for camera relocalization in rgb-d images,” in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2013, pp. 2930--2937.

[29] J. Taylor, J. Shotton, T. Sharp, and A. Fitzgibbon, “The vitruvian manifold: Inferring dense correspondences for one-shot human pose estimation,” in *Computer Vision and Pattern Recognition (CVPR), 2012 IEEE Conference on.* IEEE, 2012, pp. 103--110.

[30] 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 *Computer Vision and Pattern Recognition (CVPR), 2011 IEEE Conference on.* Ieee, 2011, pp. 1297--1304.

[31] A. Fielding, “Decision trees in more detail: How trees are built and pruned,” accessed: 20-11-2018. [Online]. Available: http://www.alanfielding.co.uk/multivar/crt/dt_example_04.htm

[32] P. Kontschieder, P. Kohli, J. Shotton, and A. Criminisi, “Geof: Geodesic forests for learning coupled predictors,” in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2013, pp. 65--72.

[33] W. Kabsch, “A solution for the best rotation to relate two sets of vectors,” *Acta Crystallographica Section A: Crystal Physics, Diffraction, Theoretical and General Crystallography*, vol. 32, no. 5, pp. 922--923, 1976.

[34] R. A. Newcombe, S. Izadi, O. Hilliges, D. Molyneaux, D. Kim, A. J. Davison, P. Kohi, J. Shotton, S. Hodges, and A. Fitzgibbon, “Kinectfusion: Real-time dense surface mapping and tracking,” in *Mixed and augmented reality (ISMAR), 2011 10th IEEE international symposium on.* IEEE, 2011, pp. 127--136.

[35] F. Steinbrücker, J. Sturm, and D. Cremers, “Volumetric 3d mapping in real-time on a cpu.” in *ICRA*, 2014, pp. 2021--2028.