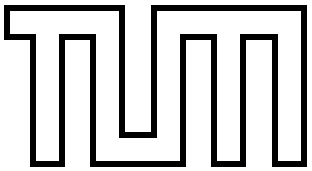


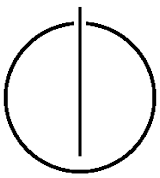
FAKULTÄT FÜR INFORMATIK

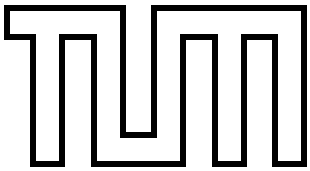
DER TECHNISCHEN UNIVERSITÄT MÜNCHEN

Bachelorarbeit in Informatik

Object detection and segmentation in cluttered scenes through perception and manipulation

Julius Adorf





FAKULTÄT FÜR INFORMATIK

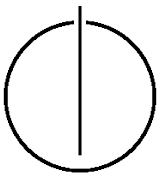
DER TECHNISCHEN UNIVERSITÄT MÜNCHEN

Bachelorarbeit in Informatik

Object detection and segmentation in cluttered scenes through perception and manipulation

Objekterkennung und Segmentierung durch Perzeption und Manipulation

Author:	Julius Adorf
Supervisor:	Prof. Michael Beetz, Ph.D.
Advisor:	M.Sc. Dejan Pangercic
Date:	July 14, 2011



I assure the single-handed composition of this bachelor's thesis, only supported by declared resources.

München, 14 July 2011 Julius Adorf

Abstract

Recognition of textured objects is fundamental in many robotic applications in household environments. This work addresses the perception task of finding a rigid textured object and its 3D pose in a cluttered scene such that the cluttered scene can subsequently be resolved by successive removal of objects by a robot.

A system is presented that learns local 2D features and 3D models from objects. It is able to detect an object and estimate its pose in a cluttered scene by matching observed local 2D features against the learned models. Confidence values provide a measure of goodness for estimated object poses. For the object with the highest confidence value, the pose is refined in order to reduce error, and that estimate provides a basis for the grasping pipeline of a robot. The system is based on the existing, yet immature and unstable textured object recognition stack in the Robot Operating System. Recent developments in object recognition are reviewed in this context, especially the Oriented BRIEF feature detector and descriptor.

Experiments have shown that in 82% of cluttered scenes in a validation set, an object is correctly recognized in the scene within a margin for rotational error of 20 degrees and for translational error of 3 cm. A live test on the robot revealed limitations as well as promising aspects of the approach.

Zusammenfassung

Die Erkennung von Objekten spielt in vielen Anwendungen der Robotik in Haushalten eine wichtige Rolle. Diese Arbeit beschreibt die Aufgabe, die Lage und Orientierung eines starren, texturierten Körpers in einer Szene mit vielen weiteren Objekten zu bestimmen. Auf diese Art und Weise sollen die Objekte nacheinander von einem Roboter entfernt werden können.

Ein System wird vorgestellt, welches lokale 2D-Merkmale und 3D-Modelle von einem Objekt erstellt. Indem das System beobachtete lokale 2D-Merkmale mit den Modellen assoziiert, kann es ein Objekt erkennen, und dessen Lage und Orientierung im Raum bestimmen. Konfidenzwerte geben Aufschluss über die Zuverlässigkeit für die Schätzungen der Lage und der Orientierung der Objekte. Die erste Schätzung mit dem höchsten Konfidenzwert wird verfeinert. Die verfeinerte Schätzung reduziert den Fehler und ermöglicht dem Roboter, das Objekt zu greifen. Das System beruht auf der bestehenden, noch nicht ausgereiften Textured-Object-Recognition-Bibliothek aus dem Robot Operating System. Jüngste Entwicklungen aus der Objekterkennung werden vorgestellt, insbesondere darunter der Oriented-BRIEF-Algorithmus zur Bestimmung und Beschreibung von lokalen 2D-Merkmalen.

Experimente zeigen, dass die Lage und die Orientierung eines Objekts in 82% der zur Validierung ausgewählten Szenen erfolgreich bestimmt werden, mit einer Fehlertoleranz von 3 cm Versatz und 20 Grad Winkelunterschied. Ein Testlauf auf dem Roboter zeigt sowohl Grenzen als auch vielversprechende Aspekte des Ansatzes auf.

Acknowledgments

To my family

My family has patiently supported me during this four-month quest in science and technology. My appreciation to my equally dear, and loyal friends. Thanks to Dejan Pangercic, my advisor, who was the first to introduce me into the challenges to be found in the world of robotics and computer vision. Lucian Goron was never short of pragmatic advice. Thumbs up for Manuel Baierl for finding a money- and time-saving shortcut to the laboratory. The discussions with Martin Kreuzeder helped to shape my understanding about geometry, proving useful in several parts of this work. Many thanks to Hans-Martin Adorf for his generous comments, and his advice on style. Thanks to Gabriel Adorf and Christian Schöne, who both taught me about the importance of graphics, and the basics in how to create them. Yet, my drawings are no match for their's. Thanks to Chanrith Siv for helping me to remove the roughest edges in my English writing. Thanks to Alexander Shishkov, Ethan Rublee, and Vincent Rabaud for their comments on software that was not supposed to be used by the public. I am grateful for the help of Zoltan-Csaba Marton and Thomas Rühr, who helped me operating the robot at Technische Universität München. Thank you.

Contents

List of Figures

List of Tables

1. Introduction

1.1. Intended Purpose

Object recognition is one of the fundamental tasks in computer vision. One of its applications is in robotics. Especially in household environments, it is often necessary for a robot to detect the presence of an object, as well as to find the pose of the object such that the robot can grasp it and make further decisions on a semantic level.

A particular application addressed by this work involves a robot in a household environment. The robot is given the task to go shopping, to empty the shopping bag onto a table, and to move the items to the appropriate storage facilities. When the items are laid out on the table, they form a cluttered scene. Most of these objects are rigid bodies that can be distinguished by their characteristic texture, which is composed of pictorial descriptions of the product, labels, and company logos. High levels of occlusion and the presence of several such textured objects in a single scene renders recognition of these shopping items difficult. Since the objects may be arbitrarily oriented, prior assumptions about their principal axes are likely invalid.

The system presented in this work enables a Personal Robot 2 (PR2[1]) to detect an object and estimate its pose in a cluttered scene to provide sufficient information to a grasping pipeline, such that the clutter can be resolved by removing items one-by-one from the scene. Although, in the shopping scenario, the robot may have prior knowledge about what objects are present in the scene, this information is not used in this work, in order to develop a system that is general enough to be applied to other application scenarios.

1.2. Problem Statement

The challenge is therefore to correctly identify a rigid textured object, to determine its pose in a scene with clutter and high occlusion, and to subsequently provide enough information to allow the robot to grasp the object and to remove it from the cluttered scene.

1.3. Related Work

In literature many different approaches to object recognition are described. The techniques using local features have been especially relevant to this work.

Tuytelaars and Mikolajczyk qualitatively review local feature detectors and descriptors [1], and provide guidance for making an appropriate choice between a multitude of available detectors and descriptors. Lowe's work on the scale-invariant feature transform explains the principles of the scale-space pyramid and rotation invariance of features [2]. Rosten and Drummond introduce the FAST feature detector [3]. Calonder et al. describe BRIEF, a

[1] http://www.willowgarage.com/pages/pr2/overview

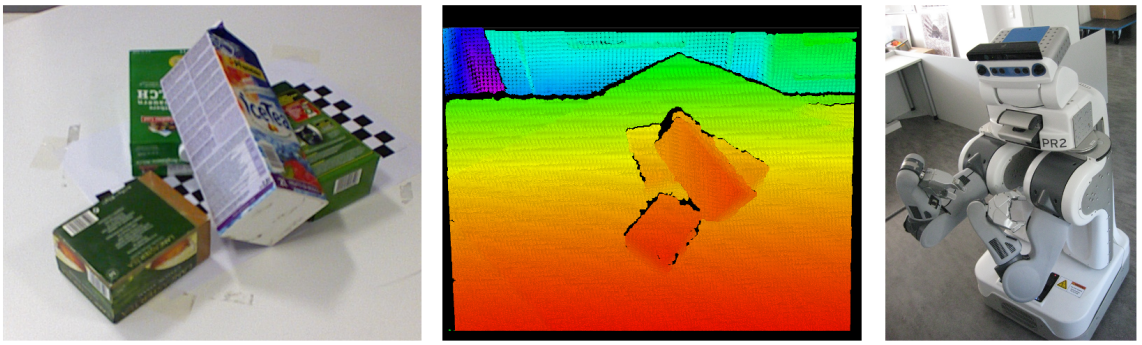


Figure 1.1.: From left to right: (a) Four rigid textured objects occluding each other in a cluttered scene; (b) a 3D-visualization of the same scene, obtained with a Kinect RGB-D camera; (c) the PR2 robot at Technische Universität München with a Kinect camera mounted on its top.

feature descriptor whose generated descriptor vectors are efficient to match [4]. Oriented BRIEF, derived by Rublee et al. from FAST and BRIEF, is the feature detector and descriptor chosen for this work [5]. A common sub-task in object recognition is matching observed features with model features. Gionis et al. introduce, and Slaney and Casey summarize Locality Sensitive Hashing [6, 7], which finds approximate nearest neighbours. Fischler and Bolles excellently explain RANSAC and deal with the perspective-n-point problem [8]. Dogar and Srinivasa recently researched the problem of robustly grasping objects in a cluttered scene [9]. Forsyth and Ponce's work on computer vision served as a reference and is what influenced the terminology in this work [10]. The same applies to Gonzalez and Woods in the field of image processing [11]. Alpaydin's excellent introduction to machine learning [12] has influenced our experiment design and parameter optimization. Fawcett published a detailed guide on how to evaluate the performance of classifiers through analysis of Receiver Operating Characteristics [13]. Melsa and Cohn cover decision and estimation theory [14].

1.4. Selected Approach

General Approach

Objects generally differ in shape and appearance. Recognizing the shopping items in a scene as shown in Figure 1.1 (a) is difficult. In general, shopping items are often shaped similarly. Worse, different products are sold in packagings of the same shape. The distinguishing feature of these objects is texture.

Thus, it seems natural to rely on texture to recognize objects. We originally considered to combine the information from textured surfaces with the information contained in the shapes of the rigid bodies, however, we solely used texture information, and based our system on the existing Textured Object Detection (`TOD`) stack in the Robot Operating System.

We required our system to run on a PR2 robot at Technische Universität München, as depicted in Figure 1.1 (c). The PR2 is a robotic platform produced by Willow Garage. A Kinect RGB-depth camera is mounted on top of the robot. It simultaneously provides colour images and depth measurements. The Kinect camera measures the time-of-flight of near-infrared rays to compute the distance between camera and 3D points. The typical depth range of a Kinect camera is between 0.8 and 3.5 meters [15]. A depth image obtained by this camera is visualized in Figure 1.1 (b).

In the presented approach, we first learn a model from each rigid textured object. An object is rotated in front of the camera, which takes images and 3D scans from different views of the object. For each view, local 2D features are extracted from the greyscale image. Figure 1.2 shows both an image of a textured object (a) and the keypoints extracted from a greyscale version of the image (b). The local 2D features, together with the associated 3D points on the object's surface, form the *model features*. The model features make up the model of this object. The 3D part of the model, a sparse point cloud, is described in a standard coordinate system.

Figure 1.2.: From left to right: (a) Textured object *haltbare_milch*; (b) the keypoints of local 2D features detected on textured surface of *haltbare_milch*

Given an image of a scene, we can use the models to recognize instances of the template objects. For clarity, we call a scene a *query scene* if we want to recognize objects in it. First, local 2D features are extracted from a greyscale *query image* depicting the query scene. These features are called *query features* in order to distinguish them from model features. Nearest-neighbour search permits to find *correspondences* between a query feature and one or more model features. Some of these correspondences for a query scene are shown in Figure 1.3. Based on these correspondences, a model-fitting algorithm that is robust with respect to outliers, aligns models to the 3D scene such that their projections onto the image plane explain the observed data. Finally, if an *aligned model* sufficiently explains the correspondences, the system *guesses* the object to appear in the query scene at the pose of the aligned model. A *confidence value* measures the degree to which a guess explains the observed data. Although we do not explicitly make use of a statistical model, the general idea is based on the likelihood principle.

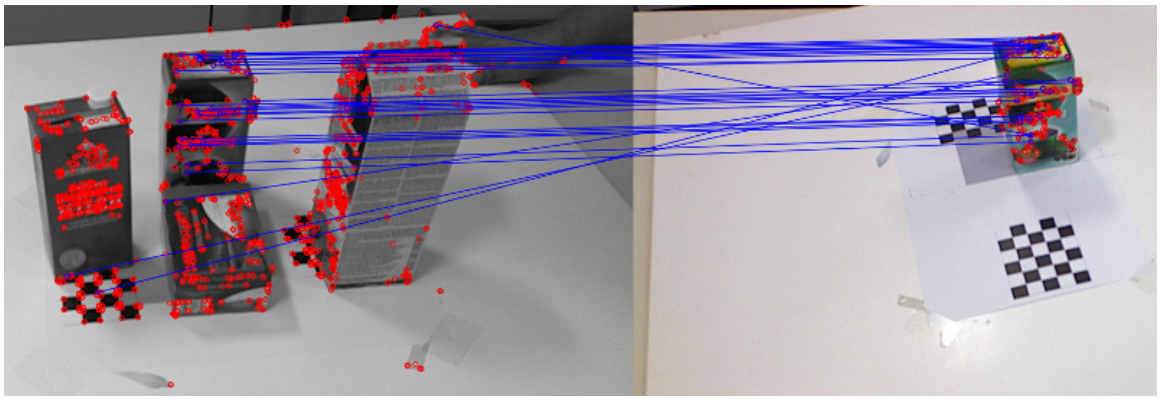


Figure 1.3.: The (hypothetical) correspondences between the query image (left), and a selected view of the template object (right)

Implementation

The basic steps in learning models and recognizing objects are already implemented in the Textured Object Detection (`TOD`) stack. `TOD` supports the use of different local 2D feature detectors and descriptors. It supports a range of nearest-neighbour search algorithms that find correspondences between feature sets. Finally, `TOD` uses RANSAC as the aforementioned model-fitting algorithm for estimating the objects' poses. The confidence value is measured by the number of correspondences consistent with this pose.

We encounter cluttered scenes (Figure 1.1 a) that exhibit high occlusion rates, as well as show objects in arbitrary orientations. For our application, it is sufficient to resolve the clutter object by object. Hence, we chose to concentrate on finding the pose of only one object in the scene. In the following, when we are talking about *recognizing* an object, we mean both detecting its presence and estimating its pose.

We present a new system called `CLUTSEG`, built on top of `TOD`. It has three purposes: Primarily, it adapts `TOD` to our purpose of resolving a cluttered scene by locating one object in the scene. Second, it helps choosing between the multitude of available feature detectors and descriptors, and selecting one of the nearest-neighbour search algorithms. Third, it helps to find good parameter values since `TOD` and `CLUTSEG` have many parameters.

`CLUTSEG` locates an object by ranking the initial guesses from `TOD` by confidence value. Then, the best-ranked guesses are refined until a guess with sufficiently high confidence value is found.

We chose Oriented BRIEF (ORB) as a feature detector and descriptor. It yields binary descriptor vectors, which can be compared efficiently. We traded exactness for speed and chose Locality Sensitive Hashing for finding approximate nearest neighbours between these binary vectors. ORB is a recent development, and we were curious about how it performs in an application.

2. Theory

This chapter provides the theoretical foundations for building the 3D object recognition system as presented in Chapter 3. We review the required mathematical tools together with the basics of 3D object recognition where models are learnt from multiple views. We discuss Oriented BRIEF, a new feature detector and descriptor. We resort to approximative nearest neighbour search, which permits to quickly establish correspondences between the query image and the models. Then, we show how pose estimation is related to other problems, and how to solve it in the presence of noise and mismatches in the correspondences. Finally, this chapter shows how the implemented system relates to the framework of classifiers and estimators, which is helpful for the evaluation given in Chapter 4.

2.1. Prerequisites in Geometry

A model of an object is described in its own coordinate system. Another coordinate system is defined by the depth camera. In this section, we explain how a coordinate system attached to the camera and a coordinate system attached to the object are related by proper rigid transformations. We precisely define the pose of an object with respect to the camera in terms of a proper rigid transformation. We also describe how pose estimation is related to camera calibration and the perspective-n-point problem.

2.1.1. Rigid Transformations

A *rigid transformation* of coordinates is defined as an orthogonal transformation followed by a translation in Euclidean space. An orthogonal transformation is either a reflection or a rotation. A *proper rigid transformation* additionally requires the orthogonal transformation to be a rotation (the definitions of rigid transformations found in literature are inconsistent, though: Reflections are excluded from rigid transformations in [10], but included in [16], we stick to the latter). In three dimensions, a proper rigid transformation of coordinates can be represented by a rotation matrix $R \in \mathbb{R}^{3\times3}$ and a translation vector $\mathbf{t} \in \mathbb{R}^3$.

$$T : \mathbb{R}^3 \to \mathbb{R}^3, \mathbf{p} \mapsto R\mathbf{p} + \mathbf{t} \tag{2.1}$$

The inverse transformation is given by

$$T^{-1} : \mathbb{R}^3 \to \mathbb{R}^3, \mathbf{p} \mapsto R^{-1}(\mathbf{p} - \mathbf{t}) = R^T(\mathbf{p} - \mathbf{t}) \tag{2.2}$$

A proper rigid transformation (PRT) can be used to describe a rigid motion of an object with respect to a reference coordinate system. In physics, this is called an active transformation, and explains the relationship between two different points in the same coordinate system.

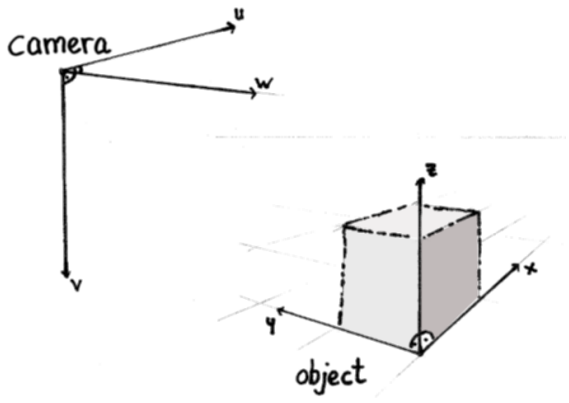


Figure 2.1.: The camera coordinate system and the object coordinate system

Another interpretation is to see a PRT as a passive transformation. The PRT transforms vector coordinates between two different coordinate systems. The passive interpretation is used throughout this work.

2.1.2. Object-Camera Transformations

3D points can be specified in coordinates with respect to any coordinate system. The question arises which coordinate systems are convenient and how many are necessary. In our case, the two different coordinate systems discussed in the following prove sufficient.

The *camera coordinate system* (c) is attached to the camera. Two of its basis vectors define the image plane and the remaining basis vector is directed towards the scene. The camera coordinate system is convenient for the robot. For example, if the robot is programmed to grasp an object at two points, it needs to know the camera coordinates of the two grasping points. The *object coordinate system* (o), on the other hand, is attached to the object. The model features are located in the object coordinate system, which is also called model coordinate system in literature [17]. Both coordinate systems are shown in Figure 2.1. The PRT between object coordinates and camera coordinates is called the *object-camera transformation* and is denoted by ${}^{c}_{o}T$. Subsequently, its inverse PRT is called *camera-object transformation* and denoted by ${}^{o}_{c}T$:

$$ {}^{o}_{c}T = {}^{c}_{o}T^{-1} \tag{2.3} $$

We stick to the subscript and superscript notation introduced in [10]. Here is an example that indicates the usefulness of these definitions. To compute the distance d between the camera and the object origin $\mathbf{0} = (0,0,0)^T$, we may use

$$ d = \| \, {}^{c}_{o}T \, (\mathbf{0}) \, \|_2 \tag{2.4} $$

The object-camera transformation defines the relationship between camera and object. Since we are only interested in the *relative* location and orientation of the object with respect to the camera, we simplify matters by assuming the camera to be fixed and by always looking

at scenes from the eyes of the robot. Note that if there are multiple objects in a scene, then there exists one object-camera transformation for each object. For example, if there are two objects 1 and 2 in the scene, the distance d_{12} between them may be computed by

$$d_{12} = || \, {}^{c}_{o}T_1 \, (\mathbf{0}) - \, {}^{c}_{o}T_2 \, (\mathbf{0}) \, ||_2 \tag{2.5}$$

where ${}^{c}_{o}T_1$ and ${}^{c}_{o}T_2$ are the respective object-camera transformations, which uniquely determine the object poses with respect to the camera.

We can reformulate our problem of recognizing objects as

> "Find the object-camera transformation for each of the objects in the scene."

In the special case of a single object

> "Find the object-camera transformation for one of the objects in the scene."

2.1.3. Camera Model

In our selected approach, we construct models by associating local 2D features extracted from the image plane with corresponding 3D points in the scene, as shown in Figure 1.2. In the recognition process, models are aligned to the scene and projected onto the image plane to verify whether they explain the observed data. In both cases, it is required to know the nature of the relationship between the scene and its 2D image, which is explained by the camera model.

Since both `TOD` and `CLUTSEG` use OpenCV, an open source library for computer vision, we adopt their camera model. It is a pinhole camera model, where a perspective projection is used to determine the projection of a 3D point onto the image plane [18]. This plane is conveniently defined in front of the centre of perspective [10, 18].

In the previous section, it was shown that estimating the pose of an object with respect to the camera is equivalent to estimating the object-camera transformation. There is a third way to look at this problem, and this is the estimation of the extrinsic parameters of the camera. The extrinsic parameters relate the camera pose to some given coordinate system in its environment. Thus, equivalently to the problem definitions given previously in Section 2.1.2, we can see our object recognition problem as

> "Find the extrinsic camera parameters with respect to the coordinate system of one of the objects in the scene."

Estimation of camera parameters belongs to the topic of *camera calibration*. In our case, the intrinsic parameters of the Kinect camera are factory-specified. `TOD` uses camera calibration techniques from OpenCV in order to find the extrinsic parameters of the camera; given that the correspondences between n 3D points and their n projections onto the image are known, the extrinsic parameters can be estimated. It is an instance of the *perspective-n-point* (PnP) problem [8].

2.2. Object Recognition

In the previous chapter, we discussed how to formulate our problem of object recognition. Here we introduce the basic mechanisms for learning models from multiple views, based on the characteristic features in the object's appearance. We show how feature detectors and descriptors find these features, especially using Oriented BRIEF. We cover how nearest-neighbour search finds correspondences between query features and model features. Finally, based on a set of correspondences, we introduce a general RANSAC-based approach of estimating the pose of objects in the scene.

2.2.1. Local Features

In object recognition, the question arises on how to generate a description of an object, and how to describe a query scene. We can distinguish local approaches from global approaches. An example for a global approach is template matching. Yet, in scenes with occlusion, local methods are clearly preferable [17]. Object recognition using local features is based on the detection of keypoints in an image. A multitude of different methods is available, but they all aim at choosing keypoints that are repeatable for the same object over different images. A 2D local feature consists of a keypoint and a feature descriptor that captures information about its local image neighbourhood. Feature descriptors are often vectors and their distance in terms of a vector norm provides a means for measuring the similarity between two features.

Learning Models from a Single View

Lowe describes an approach to object recognition based on local features where each model is learnt from a single view of the template object [2]. Given a query image, local features are extracted and matched against the model (Figure 2.2). This approach works well, if the object in the query scene has approximately the same orientation as in the template view. Lowe reports that using SIFT features, discussed in Section 2.2.4, it is possible to recognize objects up to a rotation of 20 degrees [2]. Unfortunately, considering only a single view is insufficient if query scenes show objects in arbitrary orientations.

Learning Models from Multiple Views

The single-view scheme can be extended to allow for recognizing objects in arbitrary orientations. In order to accomplish this, a description of a template object is based on local features extracted from multiple views of the object. Local features from a query scene are matched against the model that incorporates features recorded from different viewpoints (Figure 2.3).

Generally, there is no need to incorporate all collected features into the model. A strategy on how to select features from multiple views is presented in [17]. Learning models from multiple views is the basis of this work.

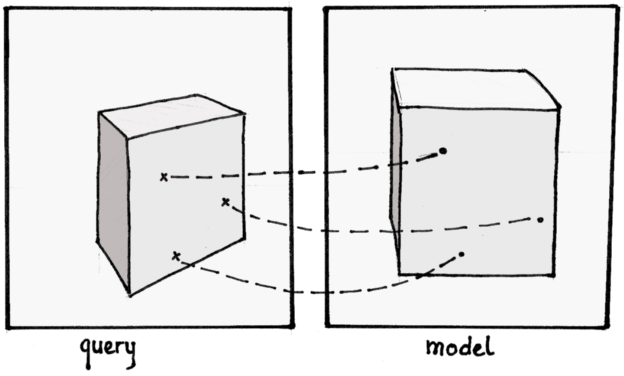


Figure 2.2.: Object recognition with models learned from a single view. The example shows three correspondences between query features (×) and model features (•).

2.2.2. Feature Detectors

A feature detector designates keypoints in an image. There exists a multitude of feature detectors in literature, and more efforts have been spent into inventing new ones than comparing existing ones [19].

There is generally no such thing as the best feature detector. As often in engineering, all we can do is making the best choice for the task at hand. An ideal feature detector though, should be *efficient*, and it should find features that are *repeatable*, *informative*, *local*, and *accurate* [1].

Repeatability describes the chances that the same point on an object shown from two different viewpoints and viewing conditions is designated keypoint on both images [19]. Thus, repeatability is important when trying to find correspondences between two images. A feature detector designates keypoints after examining the local image neighbourhood. These neighbourhoods can for example be deformed by noise, image discretisation, a change of viewpoint or a change in lighting conditions [1]. These deformations have to be addressed by a feature detector to achieve repeatability. In order to achieve invariance as regards scale and rotation, features can be extracted at multiple scales, and orientations can be assigned to keypoints (Table 2.1).

An informative feature identifies an image neighbourhood that shows high variation in intensity. If not, the features would be difficult to distinguish. For example, we would not like to have features detected on non-textured background, or on areas that carry little information.

A local feature only carries information about a small neighbourhood of a point. This ensures that it is still repeatable despite of occlusion. A small neighbourhood also leads to less informative features, enforcing a trade-off between locality and informativeness [1].

2.2.3. Feature Descriptors

A feature descriptor describes the image neighbourhood of a point. It is often a vector. Ideally, two corresponding keypoints should have similar descriptor vectors, such that cor-

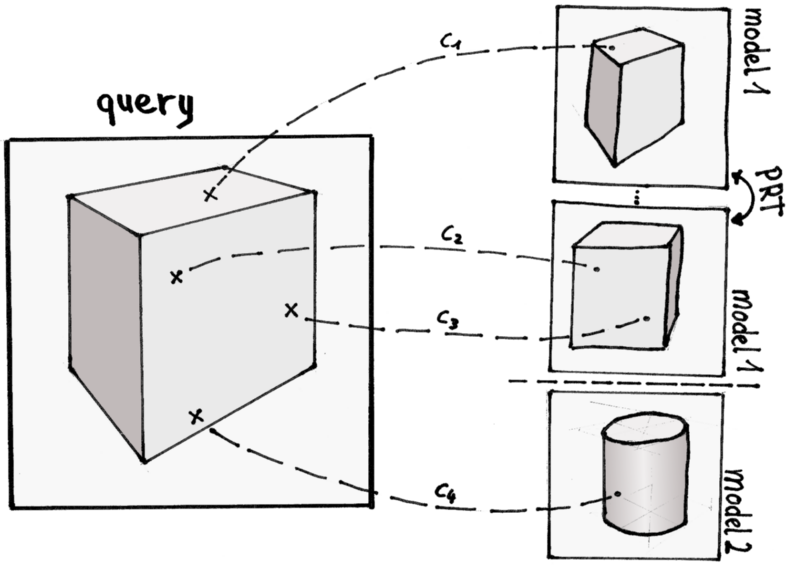


Figure 2.3.: Object recognition with models learned from multiple views. The example shows correspondences between query features (×) and model features (•). Correspondence c_1 is matched to the right model, but the location is incorrect. Correspondences c_2 and c_3 are correct. Finally, correspondence c_4 is mismatched.

respondences can be found by nearest neighbour search. A feature descriptor that permits for efficient matching is BRIEF [4].

2.2.4. Selected Detectors and Descriptors

Some of the existing detectors and descriptors are relevant for this work. We briefly introduce the Scale-Invariant Feature Transform (SIFT) [2, 20]. We use SIFT features and the Speeded-Up Robust Features (SURF) [21] in order to compare them in terms of speed with the Oriented BRIEF (ORB) [5], the latter being used in our implementation. All SIFT, SURF, and ORB are both feature detectors and descriptors.

Detector	Multi-scale	Oriented Keypoints
SIFT	yes	yes
SURF	yes	yes
FAST	no	no
ORB	yes	yes

Table 2.1.: Properties of selected feature detectors. SIFT, SURF, and ORB are also feature descriptors

SIFT

The scale-invariant feature transform became popular in object recognition. It is desirable that images of objects at different scales produce more or less the same features. SIFT achieves this by selecting keypoints from a scale-space pyramid in order to generate scale-invariant features. SIFT is both a feature detector and a descriptor. Lowe further describes a system that lets correspondences vote in Hough space, whose dimensions are location, orientation and scale [2]. When bins contain more than a predefined number of votes, the object is said to be recognized at this pose. SIFT is applied to panorama stitching [22].

FAST

FAST is a feature detector that designates keypoints by looking at a Bresenham circle of 16 pixels in diameter around a candidate keypoint. If nine contiguous pixels of the diameter pixels are all brighter (a), or all darker (b) than the centre pixel plus (a), or minus (b) a certain threshold, then the circle centre is designated keypoint [3]. FAST is efficient because it can rule out certain candidate keypoints by reasoning on the diameter pixels values north, east, south and west of the centre. FAST-ER builds on this idea, but reasons about the diameter pixels in an order optimized by ID3; the information gain is computed on a set of training images [19]. `OPENCV` 2.2 just contains an implementation of FAST, presumably because of its smaller code size.

BRIEF

BRIEF generates binary feature descriptors for a given set of keypoints. The BRIEF descriptor vectors are sensitive to rotation [4]. They are formed by a bit string computed from the image neighbourhood of a given keypoint. Each bit stores the result of comparing pixel intensities between two 2D points in the neighbourhood, which are called *test locations*. The test locations are initially sampled from a 2D Gaussian distribution once, and together form a *test pattern* [4]. Finding nearest neighbours of binary descriptors according to their Hamming distance is faster than for real-valued features using the L2 norm.

Oriented BRIEF

Oriented BRIEF (*ORB*) is a combination of the FAST feature detector and the BRIEF feature descriptor, plus modifications that render the extracted features orientation-invariant [5]. It was designed to outperform SIFT and SURF at least in terms of computation time. We here refer to the implementation available in OpenCV 2.3 [1].

ORB computes image moments for assigning orientations to keypoints detected by FAST. Image moments of order $(p+q)$ are defined (see [11]) for a 2D digital image f of dimension MxN as

$$m_{pq} = \sum_{x=0}^{M-1} \sum_{y=0}^{N-1} x^p y^q f(x,y) \tag{2.6}$$

[1] http://code.ros.org/svn/opencv/branches/2.3/opencv/modules/features2d

ORB computes the moments m_{01} and m_{10} for a circular neighbourhood of a keypoint and derives an orientation α for the keypoint:

$$\alpha = \text{arctan2}\,(m_{01}, m_{10}) \tag{2.7}$$

Consider the vector $(m_{01}, m_{10})^T \in \mathbb{R}^2$. The angle of this vector becomes the orientation of the keypoint. If the keypoint neighbourhood were a probability distribution, then this vector would contain the means of the marginal distributions of x and y, respectively.

A problem arises, when rotating the BRIEF test pattern according to the FAST keypoint orientation. There is a correlation between the image moments and the BRIEF vectors obtained from a rotated test pattern. Such a correlation reduces the entropy of computed BRIEF descriptors. The solution is to compute a test pattern that has only little correlation. ORB uses test locations as depicted in Figure 2.4.

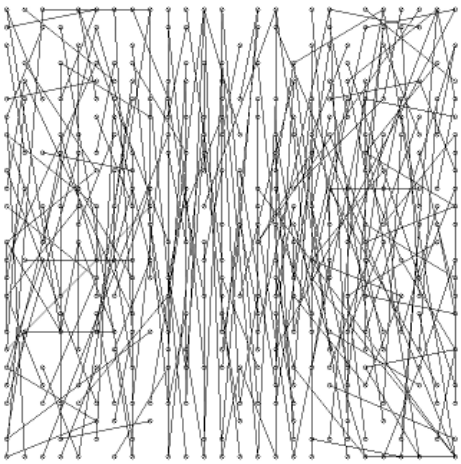


Figure 2.4.: The 256 test locations of ORB

2.2.5. Feature Matching

In object recognition that is based on local features, correspondences between known views of the object and the query image can be established by matching the query features with model features. The computation time depends on the size of the descriptor vector, the nearest neighbour algorithm and the used distance metric. If exactness is not required, i.e. approximate nearest neighbours are sufficient, then speed-ups can be achieved. Computing the Hamming distance between binary features is faster than computing the L2-norm for real-valued feature descriptors [4]. This holds especially true when using the `POPCNT` instruction, which is available at least in AMD and Intel architectures [23, 24].

Locality Sensitive Hashing

SIFT, ORB and other feature detectors and descriptors are used to describe an image in terms of local features. Nearest neighbour matching can then find correspondences between two feature sets. Locality Sensitive Hashing (LSH) quickly finds nearest neighbours, and

trades exactness in favour of a reduction in computation time. Not always does it return the absolute nearest neighbour. LSH is a randomized algorithm that performs multiple scalar projections of a high-dimensional vector, and reasons that the scalar projections are similar for similar input vectors. It can be extended to find nearest neighbours for binary features. LSH has applications in object recognition, image retrieval, music retrieval, and identification of duplicates [7].

2.2.6. Pose Estimation

The previous sections showed how the problem of recognizing objects is related to the estimation of extrinsic parameters, and in particular, to the perspective-n-point (PnP) problem. The PnP problem can be solved by RANSAC.

RANSAC stands for *RANdom SAmple Consensus*. It is a randomized algorithm that fits a parametrized model to the data. It is robust with respect to outliers. The pose estimation problem in this context can be reduced to the location determination problem or the equivalent PnP problem [8]. Generally, this algorithm randomly chooses a small subset of the data that fully determines the model parameters. Then it forms a consensus set of *inliers*, that is, all data points that are consistent with the model up to a certain margin of error. RANSAC constructs a fixed number of such consensus sets and then chooses the model with the largest consensus to compute the final estimate.

Pose estimation using RANSAC can be categorized as a method of *pose consistency* or *alignment*, where first a pose is hypothesized, and then the hypothesized pose is verified for support by the remaining data [10].

2.3. Machine Learning

This section shows how object recognition relates to machine learning, especially to classification and estimation theory. It also covers the basics of a machine learning experiment, which are required for optimizing parameters and for understanding the strengths and weaknesses of an object recognition system. When evaluating the system, a measure of *goodness* needs to be defined.

2.3.1. Classification

It is difficult to perceive our object recognition problem as a pure classification problem. Assume, there were no interest in the pose of an object. Assume further that objects are unique in an image. Then object recognition can be seen as a *multi-label classification* problem, as described in [25]. An image is described by a set of labels, and conversely, the classification result is a set of labels that predict which objects are in the image. The Receiver Operating Characteristics (ROC) can be used to evaluate standard two-class classifiers [13, 14]. Standard measures of goodness are the true and false positive rates that can be computed from confusion matrices. In case of multi-label classification, it is possible to define similar metrics [25]. Unfortunately, even if object pose were not an issue, assuming objects to appear uniquely on an image might be unrealistic. For example, a shopping bag, as in the application scenario of this work, is best represented by a multiset, and not by a multi-label set. Thus, also a scene with shopping items is best represented by a multiset.

Given a finite set of labels, the number of multi-label sets is finite, the number of multisets is not. A problem with an infinite number of decisions is an estimation problem rather than a classification problem [14]. For example, trying to define a specificity metric on it is difficult, since the number of negatives in a multi-set is generally infinite.

2.3.2. Estimation

Estimation can be regarded as the continuous generalization of classification [14]. Clearly, if it is known which objects are in a scene, object recognition can be perceived as a pure estimation problem. For each object on the scene, the six degrees of freedom of an object's pose have to be estimated. The squared and absolute errors of an estimate are possible measures of goodness that are generally available for estimators.

2.3.3. Recognition

Our recognition problem neither fits a pure classification nor a pure estimation problem. In this work, a measure of "goodness" sees the object recognition system as a combination of classifier and estimator. The estimation of "goodness" is conditioned to the classifier making the right choice, and scenes that contain multiple instances of the same object are not considered.

2.3.4. Parameter Optimization

In the presented system, the algorithms for detecting, describing and matching features are parametrized, as well those for pose estimation and those for making the final decision. This leads to the question of how to find a parameter set such that the system performs well in the application scenario.

Validation Set

Choosing good parameters for parametrized algorithms is a general problem in computer science. Solutions for these problems include adaptive methods that estimate the parameters directly from the data. For example, in adaptive numerical quadrature, grid points are selected according to an error estimate gained directly from function samples. Optimization techniques find a parameter set that optimizes performance on a validation set, where ground truth is avaible. This optimization technique is well-known in machine learning.

Test Set

It is interesting to estimate the error which a classifier or estimator is expected to make when working on new instances. This is the generalization error which cannot be estimated on the validation set. Instead a separate test set is required to estimate the generalization error [12].

3. Implementation

This chapter describes our CLUTSEG system for recognizing an object in a cluttered scene. It takes a query image as input. As a result it shall detect the presence of an object, find its pose in the query scene, and mark points on the object.

CLUTSEG is based on the existing Textured Object Detection (TOD) Library, which we discuss in detail.

3.1. Software Dependencies

CLUTSEG depends on many other open-source libraries. Of these, we briefly explain (Figure 3.1): OPENCV for operating on images, PCL for dealing with 3D data, ROS for interfacing with the robot, SQLITE for storing and retrieving experimental data, and R for analysing data.

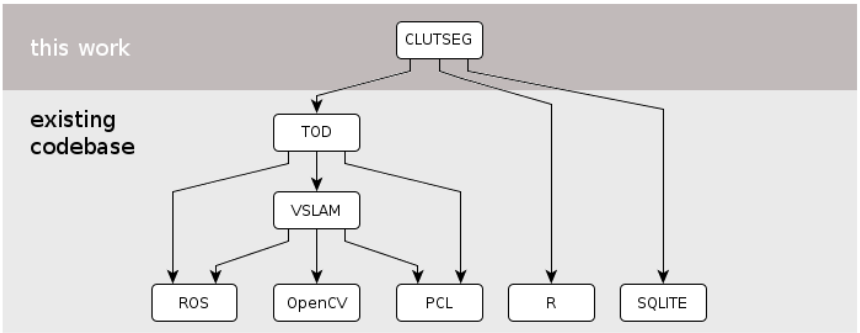


Figure 3.1.: CLUTSEG depends (amongst others) on TOD, ROS, PCL, OPENCV, VSLAM, R, and SQLITE.

OpenCV is an open-source library for computer vision [18]. It supports the representation, the input and the output of images as well as many image processing operations. It offers methods for camera calibration and feature detection, which are used by CLUTSEG and TOD.

The *Robot Operating System* (ROS) is an open-source framework for the development of robot applications, having its roots in the STAIR project at Stanford University [26]. In ROS, distributed nodes communicate by publishing and subscribing to topics. In our case, the interesting topics were the image and the 3D point cloud, together forming the sought-after depth images. These images and other data can be recorded in so-called bags. ROS contains the VSLAM package, which is only used by TOD because it contains a RANSAC implementation for solving the PnP problem.

The *Point Cloud Library* (PCL) is concerned with processing 3D data [27]. A point cloud is a set of points in n-dimensional space. For example, the output of the Kinect camera can

be represented by a point cloud where each point is described by its Cartesian coordinates with respect to the camera and by its observed colour.

SQLite is a reliable file-based relational database that is well-suited for local applications. It is thoroughly tested with over a million tests, and hassle-free to use. It provides convenient command-line tools that allow for exporting data quickly into human-readable formats.

R is an open-source environment for statistical computing and graphics. It excels in good documentation. R supports many different sources of data, and in particular also a `SQLITE` database.

3.2. Textured Object Detection Library

In this section we analyse the software underlying the `CLUTSEG` system. We explain how we modified it and improved it to better fulfil our requirements. We describe how to extract models of template objects from multiple views, and cover the elementary pipeline that later recognizes instances in query scenes.

The Robot Operating System contains the packages *tod*, *tod_training* and *tod_detecting*. The system formed by these three `ROS` packages is called `TOD` for easier reference [1]. Throughout this work, `TOD` has been used in Subversion revision 50479, and modifications are explicitly mentioned in the following. At this revision, `TOD` was experimental, unstable, and not covered by automatic tests. The system participated in the *Solutions in Perception Challenge* contest at ICRA 2011 in Shanghai for comparison.

`TOD` offers two modes. One targets the Kinect RGB-D camera. The second mode is optimized for working better with a Prosilica MP 5 camera [28]. While the latter clusters correspondences prior to pose estimation, the former does not. `TOD` recognizes objects solely by intensity and depth information; colour information is ignored. `CLUTSEG` uses `TOD` in the Kinect mode.

3.2.1. Learning

In general, an object recognition system needs models of the objects which it shall be able to recognize. In `TOD`, a model is a set of model features learnt from multiple views of the template object. Each of the views is processed in a pipeline (Figure 3.2). Afterwards, the resulting model features from selected views are unified, and together form the model of the object.

Model Features

Given a single view, let $\mathbf{p}$ be the 3D object coordinates of a point on the template object. Let $\mathbf{p}'$ be the projection of $\mathbf{p}$ on the image plane, and $\mathbf{d}$ the descriptor vector that describes the local image neighbourhood of point $\mathbf{p}'$. We call $(\mathbf{p}, \mathbf{d})$ a *model feature*.

`TOD` conceptually generates a model feature in three steps. First, a feature detector selects a keypoint on the image plane. Second, a feature descriptor vector is computed, which describes the neighbourhood of the keypoint. Third, the descriptor vector is associated with the object coordinates of the 3D point that corresponds to the keypoint. The keypoint itself is of no further interest.

[1] German readers will note the unfortunate choice in abbreviating "textured object detection".

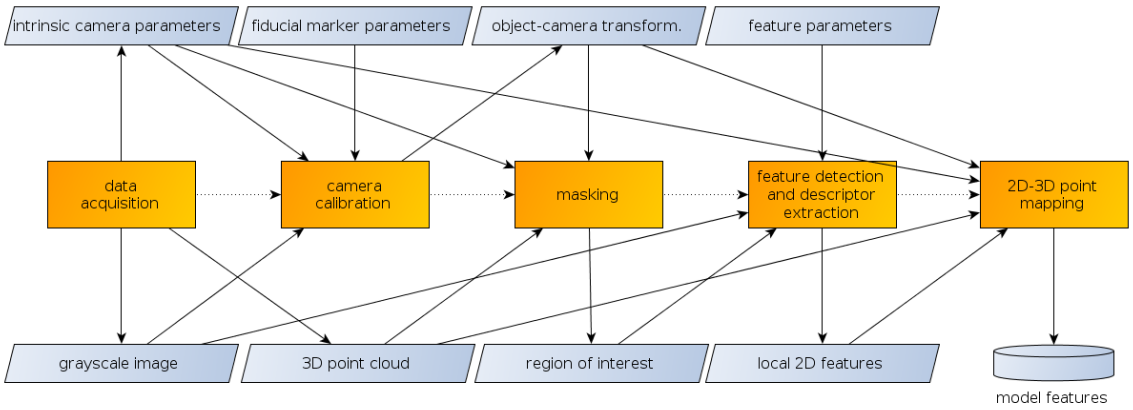


Figure 3.2.: The pipeline of TOD used for learning models. The rectangular nodes represent processes, the slanted nodes represent data.

Model Extraction

TOD offers a toolchain that extracts model features from a template object on a rotating table. The template object is either manually or automatically rotated, and a depth camera takes images from multiple views. The latter consist of a depth image and a greyscale image each. The views are the basic units processed in a pipeline that comprises three stages for each view: First, the computation of the object-camera transformation from the image. Second, the segmentation of the region of interest that shows the object on the image. Third, the extraction of local 2D features from the greyscale image, followed by the construction of model features from the region of interest.

Camera Calibration

Given a view, the first step is to obtain the object-camera transformation by estimating the extrinsic parameters of the camera. This is required for every view because the object pose changes relative to the camera, and conversely, the camera pose changes relative to the object. The camera calibration is based on chessboard-like fiducial markers that are glued onto the rotating table on two opposite sides (Figure 3.3). OPENCV offers camera calibration methods that help to find the chessboard corners and then, based on these corners, estimate the extrinsic parameters by solving the perspective-n-point problem.

There is a subtlety in the relationship between the fiducial markers and the object coordinate system. The camera calibration computes the PRT ${}^{c}_{f}T$ between the *fiducial coordinate system* (f) attached to the fiducial markers and the camera coordinate system. The origin in fiducial coordinates is defined halfway between the two chessboards. Yet, we need the object-camera transformation ${}^{c}_{o}T$. The solution here consists in just *defining* the object coordinate system to be the same as the coordinate system attached to the fiducial markers, that is

$$ {}^{c}_{o}T : \mathbb{R}^3 \to \mathbb{R}^3, \mathbf{p} \mapsto {}^{c}_{f}T(\mathbf{p}) \tag{3.1} $$

As long as one of the two chessboards is detected, the transformation can be estimated,

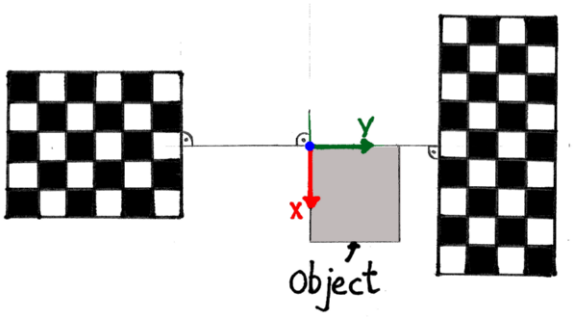


Figure 3.3.: The fiducial coordinate system and the object coordinate system are defined to be the same.

which proves useful when the template object occludes the chessboard further away from the camera. The chessboards were sometimes not detected in the image at all. In such cases, we used dithered binary images as a fallback. This improved reliability. We singled out 23 cases of the original images where camera calibration failed. Some tested image processing operations work better than others on these 23 images (Table 3.1). Three preprocessed images are shown in Figure 3.4.

Pre-processing method	Imagemagick command	Success
Dithering to Binary	`convert -monochrome $1 $2`	23/23
Colour Reduction	`convert -colors 2 $1 $2`	23/23
Local Adaptive Thresh.	`convert $1 -lat 25x25 -threshold 50% $2`	23/23
Thresholding	`convert $1 -threshold 50% $2`	19/23
Sharpening	`convert -sharpen 0x12 $1 $2`	2/23
Closing	`convert -morphology Close Disk $1 $2`	1/23
Opening	`convert -morphology Open Disk $1 $2`	0/23
Median Filtering	`convert -median 2 $1 $2`	0/23

Table 3.1.: Various image processing operations have been applied to the set of 23 images where detecting the chessboards initially failed. When converting these images to dithered binary images, the chessboard was detected on all 23 of them.

Segmentation of the Region of Interest

The model must only includes features that belong to the template object. Hence, the extraction of model features must be confined to the region in the image that shows the object but no background.

`TOD` segments the 3D point cloud of the template object using a box-shaped pass-through filter of a predefined size at the object origin ${}^{c}_{o}T\,(\mathbf{0})$. Afterwards, a radius outlier search removes all points with less than 20 neighbours within 2 cm. `TOD` projects the segmented point cloud onto the image plane. The set of projected points form a mask that is supposed to be aligned as close as possible to the true region of interest.

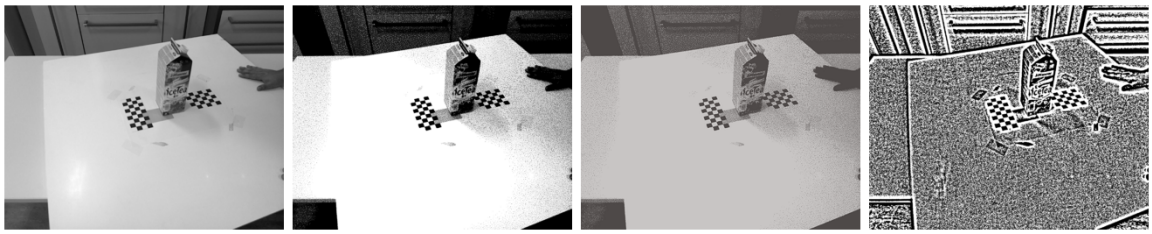


Figure 3.4.: From left to right: (1) Original image where pose estimation failed; after (2) Dithering to Binary; after (3) Colour Reduction; after (4) Local Adaptive Thresholding. On images 2, 3, and 4, the extrinsic parameters were successfully estimated.

Extraction of Model Features

The final step in learning requires the construction of a set of model features from the depth image. Local 2D features are extracted using any one of the feature detectors and descriptors provided in `OPENCV`. Each of these features is described by a keypoint $\mathbf{p}'$ on the image plane and a descriptor vector $\mathbf{d}$. A camera model, whose intrinsic parameters are known, uniquely associates the keypoint $\mathbf{p}'$ with a 3D point, described in camera coordinates $\mathbf{p_v}$ in the scene. Applying the camera-object transformation, the object coordinates $\mathbf{p}$ are obtained by $\mathbf{p} = {}^{o}_{c}T\,(\mathbf{p_v})$. This yields a model feature $(\mathbf{p}, \mathbf{d})$. `TOD` stores all learned models in a *modelbase*.

In the actual implementation, `TOD` applies the inverse object-camera transformation just before calling RANSAC for pose estimation, but it is equivalent, conceptually simpler, and computationally faster to do this only once while extracting features.

3.2.2. Recognition

This section shows how `TOD` can be used to recognize instances of template objects in query scenes, making use of the models learned in the previous section (Figure 3.5). The recognition process in `TOD` can be split into three parts: First, the extraction of query features. Second, the matching of query features and model features to obtain a set of correspondences between query image and models. Third, the estimation of object poses based on this set of correspondences.

Query Features

Let $\mathbf{p}$ be a 2D point on a query image, and $\mathbf{d}$ its descriptor vector. Then we call $(\mathbf{p}, \mathbf{d})$ a *query feature*. Hence, such a query feature is nothing but a keypoint-descriptor pair produced by SIFT, SURF or ORB, or any other feature extractor, and we introduce this term to clearly tell features from the query and the model apart. Note that query features do not include 3D information.

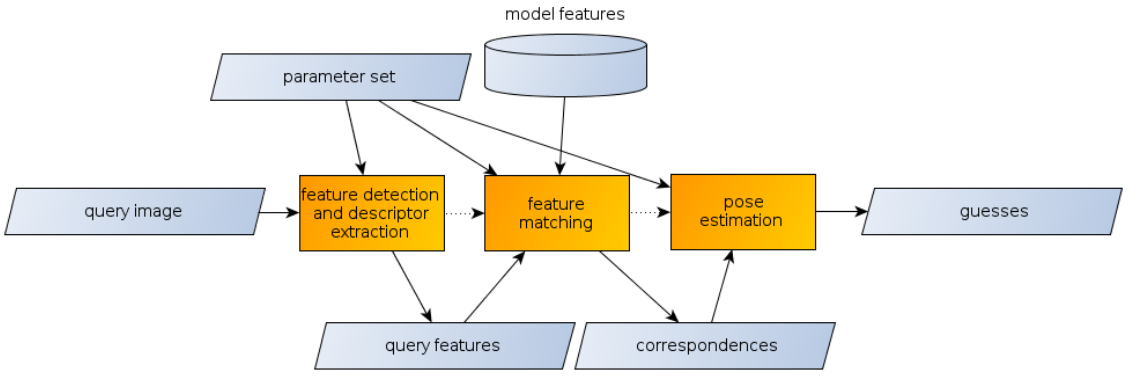


Figure 3.5.: TOD recognition pipeline (Kinect mode).

Extraction of Query Features

The extraction of query features is similar to the extraction of model features. Both query feature and model feature extraction can be configured, so TOD allows to try different feature detectors with configurable parameters. TOD also provides rBRIEF, an experimental modified version of BRIEF, which we used in some experiments but we then replaced in favour of ORB from OPENCV.

Feature Matching

Since the image is not segmented, these query features emanate from different textured objects, or from some textured background. Matching these features shall find correspondences between these query features and model features. Let C_i be the set of correspondences where a query feature has been matched with a model feature from the i-th template object. Finally, let $C = (C_1, \ldots, C_n)$ denote the partition of all correspondences for the query scene, where n is the number of template objects described in the modelbase. TOD integrates different nearest-neighbour algorithms. TOD offers a brute-force matching algorithm which we use in Section 4.3.1 for comparison. We use Locally Sensitive Hashing in TOD for matching.

TOD finds up to k nearest neighbours for each query feature. The maximum number k of correspondences per query feature can be controlled by parameter `knn`. TOD implements the ratio test, as described in [20]. If a model feature is matched with multiple query features, only the correspondence with the minimum distance is retained. This way, TOD ensures that the mapping from query features to model features is injective.

Guess

The hypotheses TOD generates for a given query scene are called *guesses*. A guess is a triple $(i,\ \hat{T},\ S)$, where i denotes the template object which TOD believes to see on the scene, $\hat{T}$ the estimated pose of the object in terms of an object-camera transformation and $S \subseteq C_i$ the consensus set of inliers.

Pose Estimation

Given the correspondences C, `TOD` generates a set of guesses. The correspondences are treated separately for each template object, that is `TOD` looks at each correspondence set C_i. For the i-th template object, RANSAC is called iteratively. If RANSAC returns with a pose estimate $\hat{T}_{ij}$ in the j-th iteration and the size of the consensus set S_{ij} is greater or equal to a parameter `min_inliers_count`, then `TOD` infers that there is an instance of the template object at the estimated pose and generates a guess $(i, \hat{T}_{ij}, S_{ij})$. Note that the inlier sets S_{ij} are disjoint because the RANSAC call in the j-th iteration is told to operate only on the set $C_i - \bigcup_{k=1}^{j-1} S_{ik}$, and C forms a partition. If not, then iteration is stopped, and `TOD` continues with the correspondence set for the next template object, if any. This means that the inlier sets of two different guesses never share a correspondence.

The `ROS` package `posest` in stack `vslam` contains an implementation of RANSAC for the PNP problem, which is used by `TOD`. It is roughly similar to the RANSAC/LD algorithm presented in [8]. It solves the P4P problem. The largest consensus set found is fed to the PNP-solver in `OPENCV` (`cv::solvePnP`), which then computes the final pose estimate.

Figure 3.6 shows a model aligned to the scene. Altogether, for a given query scene, `TOD` takes the query image and generates an unordered set of guesses, which is the output of the `TOD` recognition process.

3.3. Clutter Segmentation Tool

This section discusses our implementation of the `CLUTSEG` object recognition system on top of `TOD`. We introduce three core concepts that enable `CLUTSEG` to recognize the next object that is to be resolved from the clutter: First, a guess ranking helps to discard guesses with low confidence. Second, guess refinement aims at spending more computational resources on specifically improving pose estimates for guesses with high confidence value, and returning only the one with the highest confidence value. Third, guess rejection reduces the probability of settling on the wrong guess. Finally, this section covers the tools that help to find good parameters for the clutseg system, and help to collect statistics to evaluate its performance.

3.3.1. Guess Ranking

A core idea of `CLUTSEG` is that we can have more confidence in a guess, when it is supported by a large consensus set of inliers. It remains an assumption throughout this chapter that more inliers are positively correlated with the probability of the guess being correct. Measures for correctness are defined in Section 3.4.2.

A *guess ranking* is a function r that assigns a real-valued score to a guess $(i, \hat{T}, S)$. Different rankings can be defined and plugged into `CLUTSEG`, such as a ranking based on proximity to the camera, or a combination of multiple guess rankings. Yet, the ranking of interest is a guess ranking r_S that assigns scores to guesses according to the number of inliers

$$r_S : (i, \hat{T}, S) \mapsto |S| \tag{3.2}$$

The number of inliers is not only interesting because of the assumed correlation with probability of being correct. It also simplifies the robotic task of grasping an object by supplying more 3D points.

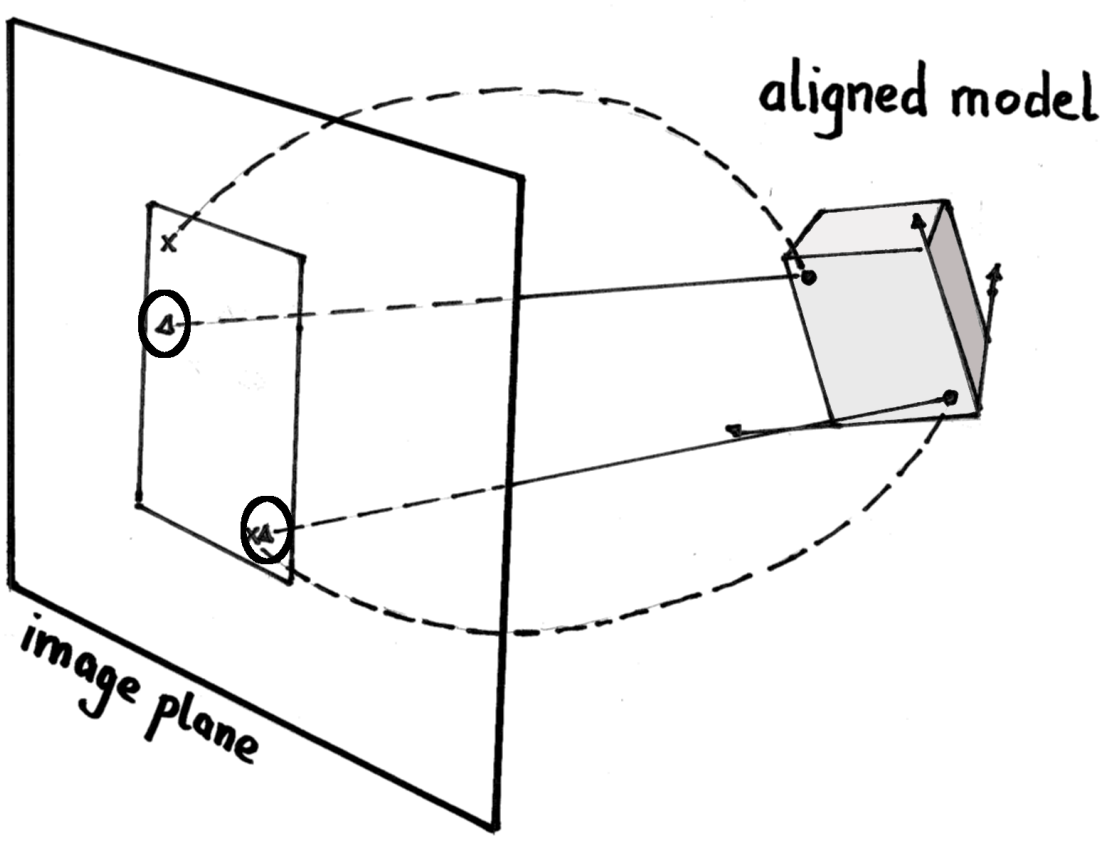


Figure 3.6.: A model aligned to the query scene. The query features ($\times$) correspond to model features ($\bullet$). The correspondence belonging to the lower right query feature is consistent with the pose estimate because the aligned model point projected onto the image plane ($\triangle$) is within a certain distance of the query feature; this correspondence is said to be an *inlier*, or equivalently, it is said to belong to the consensus set for the aligned model. The other correspondence is designated *outlier*.

3.3.2. Guess Refinement

Given a guess, the idea behind *guess refinement* is to spend additional computation resources on selected guesses to reduce the expected error in pose estimates. Guess refinement in `CLUTSEG` takes one of the guesses $(i, \hat{T}, S)$. `CLUTSEG` matches the query features only against the model features of the i-th template object, as if the modelbase contained only one model. The pose estimation using RANSAC leads to a new set of guesses, for instances of the i-th template object only. The best guess according to the chosen guess ranking is designated the *refined guess*.

Guess refinement aims at finding more matches between query features and the single object whose pose is to be refined. By ignoring other models only correspondences between the query scene and one model are found. Such an approach is optimistic, and possibly dangerous, especially in cases where the guess to refine is a false positive. The validity of such an approach has at least to be empirically verified.

3.3.3. Guess Rejection

A technique denoted as *guess rejection* reduces the chance of returning an incorrect guess. `CLUTSEG` rejects guesses whose guess ranking is smaller than a configurable parameter value `accept_threshold`. How guess rejection exactly works and in which scenarios it is useful is explained in the following section.

3.3.4. Recognition

This subsection describes the full recognition process. It puts the ideas of guess ranking, guess refinement and guess rejection into context.

Given a query scene, the `CLUTSEG` recognition process first calls `TOD` once to produce a set of initial guesses D for all objects in the modelbase. A ranking defines a total order on D. The inlier ranking r_S is used for this by default. `CLUTSEG` chooses the first ranked guess in D and computes the refined guess. If the ranking of the refined guess is greater or equal than `accept_threshold`, it is accepted and returned as the result of the recognition process. In case the refined guess is rejected or no refined guess was made at all, the algorithm proceeds to the next guess in D, and repeats the refinement step. This iteration is performed until either a refined guess is accepted, or `CLUTSEG` declares the scene to be empty because none of the initial guesses in D lead to a refined guess.

This scheme was built with the intention to avoid the problem of finding a good operating point for `TOD` in ROC space. It is sufficient that `TOD` returns a set D where the high-ranked guesses are true positives. For one of the high-ranked guesses, there are two cases to consider. If the high-ranked guess is a true positive, then the refinement step should either confirm this guess, or improve it, respectively. If the high-ranked guess is a false positive against expectations, then the refinement step might still lead to a guess. We expect this refined guess to have low ranking, and therefore to be rejected; in such a case, other high-ranked guesses are examined.

This two-staged recognition process requires the query features to be extracted once. Matching must be done at least twice. In the refinement step, there are fewer features to match than in the initial step, the *detection step*.

3.4. Parameter Optimization

.

This section reviews the parameters of the CLUTSEG system, discusses their nature and explains the approach that has been used to find a reasonable parameter set. It introduces statistics that measure the performance of the CLUTSEG system on sets of query scenes where ground truth is available.

3.4.1. Parameter Space

The CLUTSEG system has a fair amount of parameters. Many of these parameters have already been introduced in the previous chapters, others still need to be described. Table 3.2 shows an overview of all configurable system parameters, except the parameters for feature extraction, which depend on the choice of feature extractor, and which are treated separately.

Parameter	Range
`detect_matcher_type`	categoric
`detect_knn`	integer
`detect_do_ratio_test`	boolean
`detect_ratio_threshold`	real
`detect_ransac_iterations_count`	integer
`detect_max_projection_error`	real
`detect_min_inliers_count`	integer
`refine_matcher_type`	categoric
`refine_knn`	integer
`refine_ransac_iterations_count`	integer
`refine_max_projection_error`	real
`refine_min_inliers_count`	integer
`accept_threshold`	real

Table 3.2.: CLUTSEG parameters for matching and pose estimation

Even if we only consider five different values for each of the ten of these parameters, we obtain $5^{10} \approx 10^6$ different parameter sets. Assume we would like to test two different matchers and three different feature detectors. This would result in around $6 \cdot 10^6$ parameter sets. A full factorial design for finding optimal parameters, as recommended in [12], becomes difficult. The pragmatic approach chosen in this work is based on three ideas: First, we make an a-priori choice in algorithms. Second, tools were developed together with CLUTSEG that support visual inspection of experiment results in order to support an intuitive analysis. Third, the exploration of parameter space is done selectively, and parameter sets in promising regions are more thoroughly investigated. The danger of such an approach is the chance to only find a local optimum instead of the targeted global optimum, and the drawback is that results have to be formulated with even more care since they are not supported by a full grid of samples in parameter space.

Feature Extraction Parameters

Feature extraction parameters control the extraction of local 2D features. A choice needs to be made for the feature detector and the feature descriptor. Feature detectors again need to be configured. The requirements described in Chapter 1 call for a feature detector that produces features in good quantities, and it shall be reasonably fast. Hence, we chose ORB, which takes three main parameters. The number of levels in the scale-space pyramid is denoted by `octaves`, the magnification factor between two levels is called `scale_factor`. Parameter `n_features` is a hint for ORB on how many features are desired.

Feature Matching Parameters

The matching process in the detection step and the refinement step have some parameters in common. It is possible to plug different matching algorithms into `TOD`. The choice is denoted by `detect_matcher_type` (`refine_matcher_type`), but partly dictated by the prior choice of feature descriptor. A binary feature descriptor calls for a matcher that is well-designed to work with binary vectors. Also, brute-force is hardly a choice except for use as comparison to approximate nearest neighbour search algorithms. We chose the Locally Sensitive Hashing algorithm implemented in `TOD`. Parameter `detect_knn` (`refine_knn`) is a hint to the matching algorithm about how many neighbours per query feature shall be retrieved. The ratio test ([20]) can be enabled or disabled by `detect_do_ratio_test` and configured by `detect_ratio_threshold` for the detection step. In the refinement step it does not make sense and is disabled by default.

Pose Estimation Parameters

The pose estimation step in detection and refinement depends on three parameters each. RANSAC is controlled by `detect_ransac_iterations_count` (`refine_ransac_iterations_count`) and `detect_max_projection_error` (`refine_max_projection_error`). No further pose estimates for an object are generated once RANSAC returns an estimate with a consensus set of size less than `detect_min_inliers_count` (`refine_min_inliers_count`).

3.4.2. Measured Statistics

This subsection describes statistics that are recorded by `CLUTSEG` for a set of scenes where ground truth is available. We explain how errors and scores are computed for a given query scene or whole sets of query scenes. We cover the statistics that are recorded in the learning and the recognition process, which are aimed at simplifying the reasoning about the system's performance and about the influence of parameters.

Learning Statistics

When constructing the models from multiple views, some simple statistics are recorded. The model construction is part of `TOD`'s codebase. It had to be modified in order to keep track of the minimum, maximum, and average number of features extracted on the views. Also, the number of times, where the computation of the object-camera transformation failed due to undetected chessboards, is tracked. The time required for building the modelbase is recorded as well. Results are presented in Chapter 4.

Ground Truth

The error CLUTSEG makes on a query scene can only be computed when ground truth is available. The ground truth data for a scene consist of a set of labels that tell which objects can be seen at which location and which orientation in the scene. Thus, the ground truth G can be described as a set of labels

$$G = \{(i,\ {}^{c}_{o}T_i) : \text{object } i \text{ is on scene at pose } {}^{c}_{o}T\} \tag{3.3}$$

Classification Error

CLUTSEG only attempts to partially estimate ground truth. It does not try to label all objects in the scene, but only one. Mapping to ROC terminology is not straight-forward. Ignoring the estimated pose, it is possible to put a guess made by CLUTSEG into one of four categories: (a) True positive, if the guessed object is actually on the scene; (b) false positive if, the guessed object is not on the scene; (c) true negative, if there is no guess and the scene is empty; (d) false negative, if the scene shows objects but CLUTSEG did not make a guess.

Unfortunately, this scheme does not cover the case in which two objects have been confused. In the application scenario, several objects are expected to be on the scene.

Pose Estimation Error

The error in the estimated pose can only be computed in case the guessed object is actually visible in the scene. In case it is visible, the translational error e_t given ground truth pose ${}^{c}_{o}T$ and pose estimate ${}^{c}_{o}\hat{T}$ is given by the distance between the estimated location and the true location of the object origin $\mathbf{0} = (0, 0, 0)^T$:

$$e_t = \left|\left|\ {}^{c}_{o}\hat{T}\,(\mathbf{0}) - \ {}^{c}_{o}T\,(\mathbf{0})\ \right|\right|_2 \tag{3.4}$$

The orientation error e_α is computed from the ground truth orientation r and estimated orientation $\hat{r}$, specified in axis-angle representation, and using the dot-product, we have

$$e_\alpha = \left|\left|\arccos\left(\frac{\mathbf{r}^t\ \hat{\mathbf{r}}}{||\mathbf{r}||_2\ ||\hat{\mathbf{r}}||_2}\right)\right|\right| \tag{3.5}$$

Rodrigues' rotation formula can be used for efficiently converting back and forth between rotation matrix and axis-angle representation, and is readily available in OPENCV.

Guess Scores

Here, we present a score that measures CLUTSEG's performance on a single query scene. The goal is to find a score function that closely models the utility of a guess made for a query scene.

If the guessed object is not on the scene, clearly the utility of such a guess is zero. Also, we consider the utility to be zero, if the orientation and translation error is beyond a certain margin of error. The idea is therefore to introduce a combined classification and estimation score.

Let $\alpha_{max} = \frac{\pi}{9}$ and $t_{max} = 3$cm be the maximum tolerable errors in orientation and translation, respectively. The choices are consistent with the *Solutions in Perceptions Challenge*

2011. Then, for a query scene with ground truth G and a guess g, we define the *guess score* function u as

$$u : G, g \mapsto \begin{cases} 1 & \text{if scene is empty and no guess made} \\ 0 & \text{if guessed object is not on scene} \\ 1 - \min\{1, \frac{e_t^2}{t_{max}^2} + \frac{e_\alpha^2}{\alpha_{max}^2}\} & \text{if guessed object is on scene} \end{cases} \tag{3.6}$$

The presented guess score function is piecewise continuous. Thus, it retains more information than a simple statistic that records the binary observation whether the guess is within or beyond an error margin. The influence of error in pose is bounded, and outliers do not influence average the score function too much. On the other hand, it does not provide useful gradient information in regions of bad parameter sets, given the function's flat surface beyond a certain region defined by t_{max} and α_{max}.

Success Rate

We define an object to be successfully recognized if and only if its estimated pose does not exceed the error margins for translation (t_{max}) and rotation (α_{max}). `CLUTSEG` produces only one guess per scene. The *success rate* is the ratio between the number of scenes where `CLUTSEG` successfully recognized an object and the number of scenes in total.

Recognition Statistics

`CLUTSEG` also collects statistics about the recognition process that do not involve ground truth. These statistics cover the number of guesses, matches, inliers in the detection and in the refinement stage. The runtime is recorded as well.

3.4.3. Experiment Strategy

Experience showed that it takes around 2-7 minutes to evaluate `CLUTSEG`'s performance on a validation set with 21 images. Hence it was vital to make a decision which regions of parameter space to explore first and which regions to explore next. In this work, the strategy was to select a few regions, and then to pursue search in the neighbourhood of those parameter sets that performed well in the experiments conducted so far. To a certain extent, the strategy resembles genetic programming.

3.4.4. Experiment Runner

In this subsection we describe `CLUTSEG`'s experiment runner that permits to test different parameter sets against a set of scenes. We describe how a database was employed for keeping track of parameter sets and experiment results, the visualization tools that make the results accessible for analysis, and the major implementation decisions that saved computing resources.

Experiment Database

CLUTSEG uses a SQLITE database for keeping track of parameter sets and the recorded statistics, as introduced in Sections 3.4.1 and 3.4.2, respectively. The experiment runner can be executed in the background. When being idle, it polls the database for new parameter sets. New parameter sets are tested for validity with computationally cheap tests. The failing ones are marked and skipped, thus one failing experiment does not kill the process. A no-frills object-relational mapper provides a convenient interface to work with the database. The primary purpose of the database consists in selecting experiment inputs and outputs in a central, accessible and convenient location. Few efforts have been spent on normalization, performance tuning or any other aspects that were not related to its primary purpose. For convenience, data can be selected from various database views.

Modelbase Cache

Extracting models is computationally expensive and may take several minutes. As long as the feature parameters do not change in several experiments, there is no need to regenerate the models, and the modelbase is retrieved by the SHA1 hashcode of the feature parameters. A modelbase for four template objects takes up around five Megabytes on the filesystem, depending on the choice of feature detector and descriptor. TOD stores the modelbase as compressed YAML (Yet Another Markup Language) text files. If disk space were scarce and reading the modelbase quickly into memory were an issue, then binary files would be a better choice.

Visualization Tools

CLUTSEG employs OPENCV and R for visualizing the recognition results in the query scene and the statistical data acquired in experiments. It provides functions for visualizing guesses and ground truth in parallel, showing inliers, translational and rotational errors in the manner of a heads-up display. It provides command-line tools to inspect views of the modelbase, that is the region of interest, the extracted keypoints, and the estimated pose. Imagemagick is used to generate montage images that show the results of one experiment at once. Finally, TOD allows to visualize matches between the query scene and the modelbase. R is used for pulling data from the database and for plotting the data.

4. Evaluation

This chapter presents the experimental results of the CLUTSEG system on the validation set, and results from a live test on the PR2 robot at Technische Universität München. It covers how the modelbase was built, how validation data, ground truth, and the test data have been obtained. We discuss the parameter set found by experimentation. Scenes are presented in which the system performs well. We also cover the limitations and issues that were revealed during the evaluation.

4.1. Data Acquisition

This section shows how all the data required for constructing the modelbases and the validation set was obtained. The modelbases and the validation set required raw data, which have been collected in a similar fashion and the same hardware and software setup.

4.1.1. Common Setup

We used the Willow Garage PR2 robot at TUM to collect raw data. On top of the PR2, a Kinect RGB-D camera was mounted. A wooden board with two chessboards attached was setup in about one meter distance from the camera. This board served as a rotating table. The chessboards were printed on white DIN-A4 paper using a laser printer, and around them was enough white margin, as recommended in the documentation of method `findChessboardCorners` in OPENCV. The recording took place in an indoor laboratory environment whose lighting conditions were roughly similar to the ones to be expected in an application scenario. The board was manually rotated, carefully and not too fast. The objects must not move with respect to the fiducial markers, which are attached to the object coordinate system.

The resolution of the image was 1280x1024 pixels, and the dense point cloud had a resolution of 640x480. The sampling rate was set to about one depth image per second. The camera matrix needs to be recorded, as it is required at least in solving the point-to-point correspondence problem in pose estimation.

4.1.2. Modelbases

TOD requires multiple views from the template object. For the modelbases, we saved about 60–70 views per object to a bag file, which has been observed to work better than a bag with only 40 views. For each object, around 1 Gigabyte of data was recorded. The next preparatory steps consisted in uncompressing the bag files with a script included in both CLUTSEG and TOD, in estimating the object-camera transformation for each view, and in masking for each view as described in Section 3.2.1. These post-processed raw data provide the masks and the object-camera transformation which can be used to create several

modelbases using different feature detectors and descriptors. The resulting models of each template object were much smaller in size than the original raw data. The models took up around 1 Megabyte of disk space each, three magnitudes less than the original raw data. The feature detectors are responsible for this data reduction. In the experiments, ORB selected 394 keypoints on average per image of the modelbase raw data.

Figure 4.1.: The four objects *assam_tea*, *haltbare_milch*, *jacobs_coffee*, and *icedtea*, which were used in evaluation.

We collected raw data for four different rigid textured objects (Figure 4.1). These are items commonly found in German supermarkets. These are *assam_tea*, *haltbare_milch*, *icedtea*, and *jacobs_coffee*. The objects have different properties. For example, *assam_tea* is smaller than *icedtea*. The front of *icedtea* exhibits characteristic texture. Its back, though, only shows smallprint. *jacobs_coffee* shows texture that repeats on different sides of the object. Finally, *haltbare_milch* contains characteristic texture which is confined only to certain regions on its surface.

A practical consequence of Equation 3.1 is that a template object must not move with respect to the fiducial markers when collecting raw data. This would be equivalent to redefining the object coordinate system.

4.1.3. Validation Set

The validation set, which was used for optimizing parameter values, was collected similarly to the data for the modelbases. Again, the fiducial markers were required for computing ground truth for the objects on the validation scenes. We recorded around 50–60 views of various scenes, each containing three objects from the modelbase. Out of those, we chose 21 scenes to form the validation set. We chose them in roughly equal angle increments such that the variance in viewpoint between two scenes is maximized. The scenes show the objects with varying levels of occlusion. Each object appears in upright orientation.

Given a scene, ground truth requires the computation of the object-camera transformations ${}^c_oT_i$, ${}^c_oT_j$, and ${}^c_oT_k$ for the three objects i, j and k in the scene. Figure 4.2 shows three coordinate systems, defined with respect to the fiducial markers. The object coordinate system was defined for an object when the model was being learnt. Therefore, an object must be placed in the validation scene such that its object coordinate system aligns with

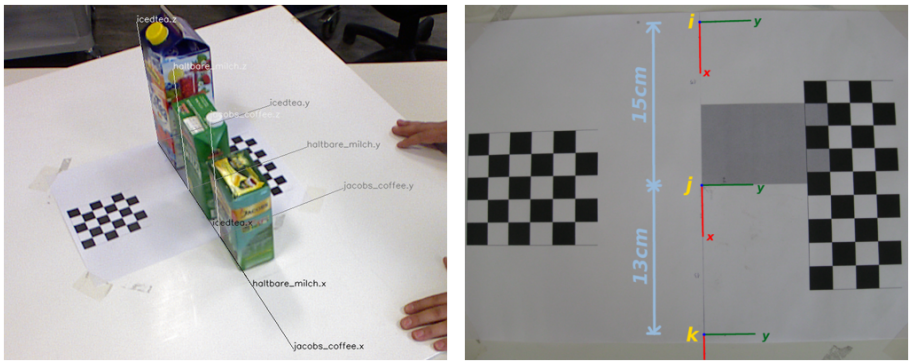


Figure 4.2.: From left to right: (a) A validation scene showing *jacobs_coffee*, *haltbare_milch*, *icedtea*, and the ground truth; (b) three coordinate systems that must align with the object coordinate systems of the three instances.

one of the three coordinate systems as depicted in Figure 4.2. We placed object j in the centre between the fiducial markers, such that

$$ {}^{c}_{o}T_{j} : \mathbf{p} \mapsto {}^{c}_{f}T\,(\mathbf{p}) \tag{4.1} $$

In other words, object j was placed on the rotating table in the same way as when raw data for a modelbase was being collected. Pencil markers on the template objects helped us to remember the correct arrangement. Imprecision was difficult to avoid when we placed the objects manually on the table. This introduces some noise in the ground truth data.

The other two objects i and k were placed upright next to object i, with offset of -0.15 and 0.13 (in metres) in x-direction of fiducial coordinates

$$ {}^{c}_{o}T_{i} : \mathbf{p} \mapsto {}^{c}_{f}T \left(\mathbf{p} - \begin{pmatrix} 0.15 \\ 0 \\ 0 \end{pmatrix} \right) \tag{4.2} $$

$$ {}^{c}_{o}T_{k} : \mathbf{p} \mapsto {}^{c}_{f}T \left(\mathbf{p} + \begin{pmatrix} 0.13 \\ 0 \\ 0 \end{pmatrix} \right) \tag{4.3} $$

4.2. Experiments

Here we describe the regions in parameter space that have been covered by our experiments. We discuss the observations made during experimentation and present the parameter set that has been found to work well on the validation set.

4.2.1. Searched Parameter Space

In total, 4326 experiments with different parameter sets have been conducted. In 2251 experiments, we used FAST as a feature detector and rBRIEF as a feature descriptor. In 2075 experiments, we chose ORB instead.

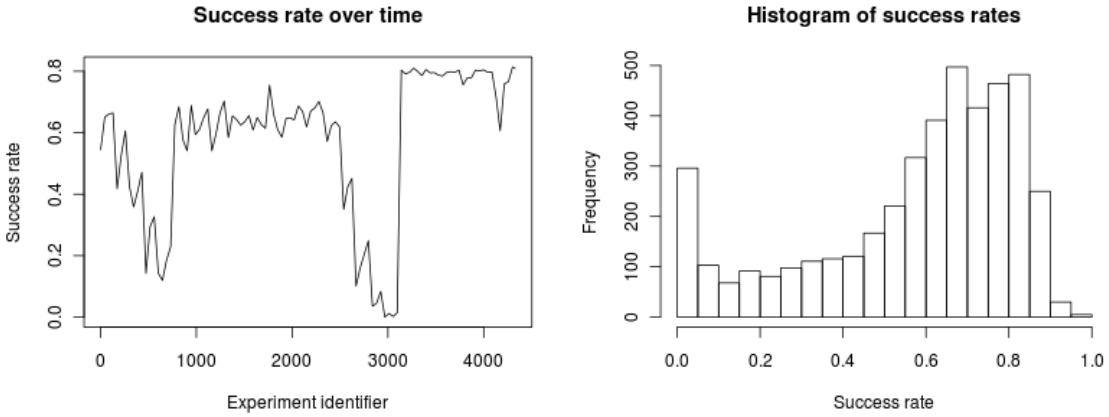


Figure 4.3.: From left to right: (a) the smoothed scatter plot of success rates of 4326 experiments in total. We assigned identifiers to the experiments in increasing order with time. The patterns reflect our tactics in choosing which part of parameter space to explore next. (b) The observed distribution of success rates in all experiments. Bad parameter values result in a zero success rate. Only a few parameter sets led to success rates greater than 90%.

Altogether, the experiments took more than 341 hours of wall-clock time to compute, an average of 4.7 minutes per experiment. Table C.1 lists the hardware and software configuration.

4.2.2. Observations

Finding good parameters has been an interactive process. New parameter sets have been evaluated in batches. A smoothed scatter plot shows the achieved success rates on the validation set over time (Figure 4.3 a). Experiments 3105–4129 have explored the neighbourhood of a promising experiment 2336. This explains the plateau starting with experiment 3100. The few experiments where `CLUTSEG` achieved a success rate of greater than 90% on the validation set (Figure 4.3 b) are outliers. This is indicated by averaging runs with the same parameter values as a prior experiment with a high success rate.

4.2.3. Selected Parameter Set

Here we present a parameter set, dubbed SPS, that has been found to work well on the validation set. It contains values for all configurable system parameters. Table 4.1 provides values for feature extraction. We used the same feature extraction parameters for the modelbase and for the query scenes. We used the default values from `OPENCV` for parameters `octaves` and `scale_factor`. We chose the value for accept_threshold empirically such that it exceeded the number of inliers in cases where a mislabelling was observed.

Table 4.2 shows parameter values for the detection and refinement stages. We used Locality Sensitive Hashing in both stages to match the binary features produced by ORB.

Parameter	Value
detector_type	ORB
descriptor_type	ORB
extractor_type	ORB
n_features	5000
octaves	3
scale_factor	1.2

Table 4.1.: The selected CLUTSEG parameters for feature extraction.

Parameter	Value
detect_matcher_type	LSH-BINARY
detect_knn	3
detect_do_ratio_test	0
detect_ratio_threshold	0.8
detect_ransac_iterations_count	1000
detect_max_projection_error	7
detect_min_inliers_count	8
refine_matcher_type	LSH-BINARY
refine_knn	3
refine_ransac_iterations_count	1000
refine_max_projection_error	5
refine_min_inliers_count	17
accept_threshold	10

Table 4.2.: The selected CLUTSEG parameters for feature matching and pose estimation.

4.3. Performance

The performance of the CLUTSEG system has been tested on the validation set, and in a live test on the PR2 robot. This section reviews CLUTSEG's performance for a parameter set that worked well on the validation set.

4.3.1. Performance on Validation Set

Since CLUTSEG uses randomized algorithms, we have run it with the SPS values for 20 times on the validation set to measure the average performance. The system correctly recognized an object within error bounds of 3 cm and 20 degrees in rotation in 82% of the validation scenes. One of the scenes where CLUTSEG successfully recognized an object is visualized in Figure 4.4, which shows the ground truth with labelled axes, the estimated pose, and the keypoints of query features that found both a match with a model feature and support the pose estimate.

The guesses that remained within the afore-mentioned error bounds showed an average error of 1.2 cm in translation, and 4 degrees in rotation. When only considering the guesses

which exceeded the error bounds, the average translational error was 6.9 cm, and the average rotational error was 2 degrees. `CLUTSEG` achieved an average guess score of 0.62. A guess was made in every scene. Objects were not confused in any of the scenes.

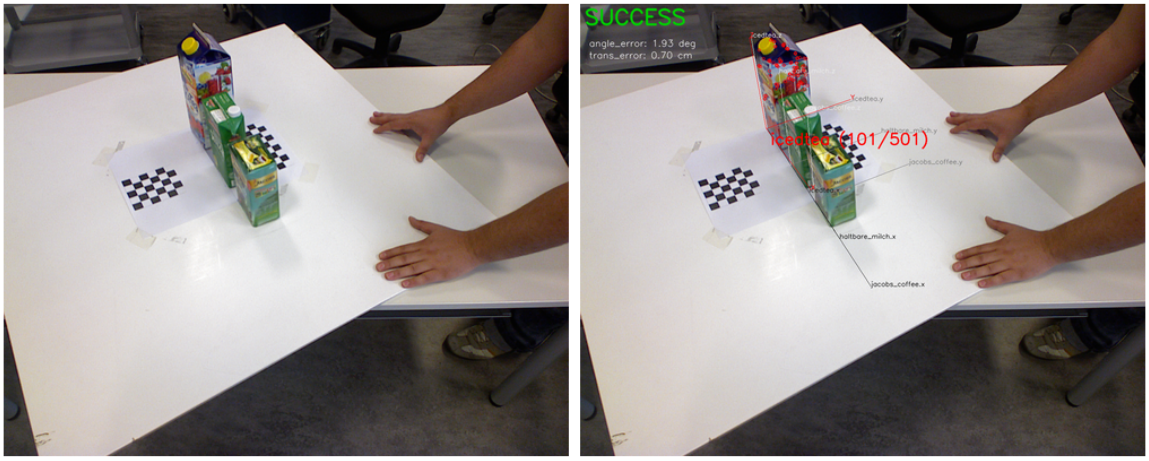


Figure 4.4.: From left to right: (a) A validation scene; (b) a visualization of the estimated pose for *icedtea*.

On average, 2344 keypoints have been extracted from a validation scene. In the detection stage, where `TOD` generates a set of initial guesses (Section 3.2.2), LSH found 1082 correspondences on average. At least 1262 query features remained unmatched. On average, an initial guess was supported by 46 correspondences.

In refinement, LSH found 377 correspondences on average, all of them matching a single object. The average guess in refinement was supported by 71 correspondences. The average pose estimate returned by `CLUTSEG` had a confidence value of 109.

The initial guess that was selected for refinement had a support of 144 correspondences. The confidence values in the detection stage and the refinement stage cannot be compared because which of the correspondences are designated inlier depends on the `refine_max_projection_error` parameter.

Recognizing an object took 17 seconds on average using the setup described in Table C.1. Using brute-force nearest neighbour search instead of Locality Sensitive Hashing, the system took 54 seconds per image. Brute-force search yielded 1068 correspondences per image, very similar to LSH. Since `TOD` ensures injectivity in the mapping of query features to model features, and since LSH does not guarantee to find k nearest neighbours, the number of correspondences found by either LSH or brute-force matching does not need to match. So, LSH ran at least a factor 3 faster than brute-force search. When measuring only the runtime for feature matching alone, we observed LSH to be more than a magnitude faster than brute-force search. The success rate of LSH on the validation set was 81%.

For the selected parameter values, the success rate of 82% is over-optimistic with regard to the test scenes, because the validation set was used for optimizing the system. For ground

Model	Number of model features
assam_tea	4623
haltbare_milch	6981
icedtea	15184
jacobs_coffee	8912

Table 4.3.: The number of model features of *assam_tea*, *haltbare_milch*, *icedtea*, and *jacobs_coffee*.

truth was available for the validation set, more precise data can be presented on it than for the live test.

4.3.2. Performance in Live Test

This section shows the results of a live test of `CLUTSEG` on the PR2. We tested whether the system is able to recognize an object in cluttered scenes with high occlusion rates and objects in arbitrary orientation (Figure 4.5); how duplicates in the scene affect the recognition process (Figure 4.6); in which setups `CLUTSEG` confuses objects (Figure 4.7); and whether the system is able to recognize an object in the presence of many textured objects in the scene (Figure 4.8). Finally, we tested whether the PR2 is able to grasp a recognized object (Figure 4.9).

The guess refinement only matches query features against the model features of one single object (Section 3.3.2). The figures in this section show the number of correspondences (inliers) consistent with the estimated pose, and the total number of correspondences between the query image and the single model. For example, consider the guess with label *haltbare_milch (10, 201)* from Figure 4.5 (a). Its consensus set contains 10 inliers. Between the query image and the model of *haltbare_milch*, 201 correspondences were found in total.

Recognition worked better when the objects were in upright position. `CLUTSEG` had more difficulties with objects in arbitrary orientations. Figure 4.5 (a) shows a scene where *haltbare_milch* was recognized. Only 10 inliers were consistent with its estimated pose, which is also off the ground truth. Figure 4.5 (b) presents a similar scene where *icedtea* was accurately recognized.

In scenes containing *icedtea* and at least one other object from the modelbase, we observed that `CLUTSEG` has a strong tendency to recognize *icedtea*. We tested each of the six possible pairings of the four objects in the modelbase. The relation, where $A > B$ denotes "A is recognized if shown together with B in a scene" was observed to be *icedtea* > *haltbare_milch* = *jacobs_coffee* > *assam_tea*. The four objects are described by a different number of model features (Table 4.3). From the initial guesses, `CLUTSEG` selects the one with the highest guess ranking, which explains this observation.

`CLUTSEG` and `TOD` do not perform any clustering of correspondences prior to pose estimation. With duplicates on the scene, the number of inliers consistent with the estimated pose of an object decreases. This can be explained by `TOD` making the mapping from query features to model features injective. Yet, even in the presence of many duplicate objects, `CLUTSEG` still recognizes objects (Figure 4.6).

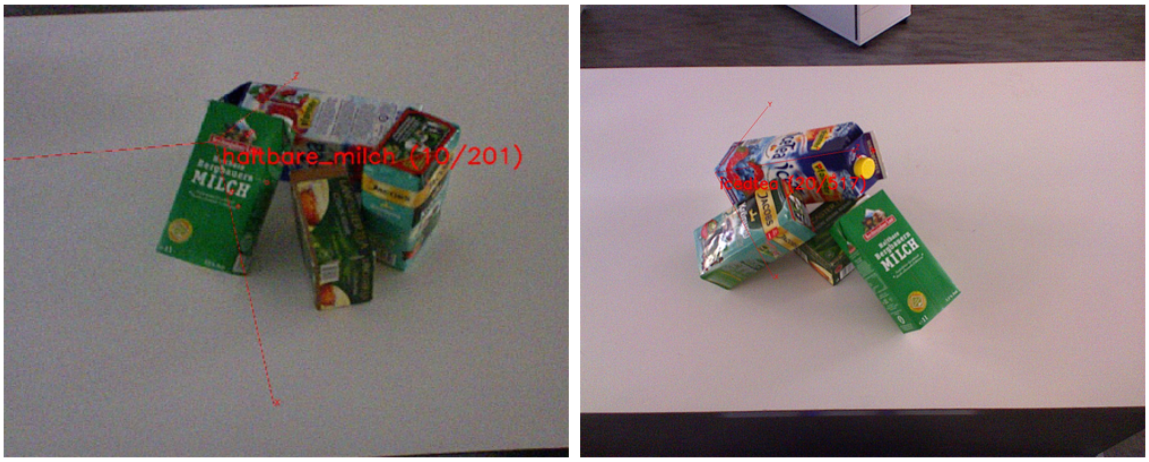


Figure 4.5.: Two cluttered scenes with all four objects from the model base as seen by the PR2. From left to right: (a) *haltbare_milch* was recognized with 10 inliers out of 201 correspondences between the image and the model of *haltbare_milch*. The pose is fairly off the ground truth pose, yet the inliers belong to the right object. (b) `CLUTSEG` precisely estimated the pose of *icedtea* with a confidence value of 155 inliers out of 516 correspondences in a scene with high occlusion rate.

Some shopping items do not only resemble each other in shape, but also in appearance. The (wild-berries) *icedtea* and the peach-flavored ice tea were confused by `CLUTSEG` (Figure 4.7 a). `CLUTSEG` guessed correctly after adding (wild-berries) *icedtea* to the scene (Figure 4.7 b).

The recognition of objects is generally rendered more difficult when many textured object are on the scene; it is more likely that the nearest neighbour of a query feature is not a true correspondence between object and model. How difficult it becomes is an open question. We tested scenes with many textured objects. Most of them did not belong to the modelbase though. `CLUTSEG` encountered no difficulties in the scenes depicted by Figure 4.8 nor in similar scenes tested.

The grasping algorithm is based on computing the centroid of the inlier cloud. The gripper of the PR2 is unfortunately not large enough to grasp *icedtea*. The maximum aperture of the gripper permits the PR2 to grasp *assam_tea*, *haltbare_milch* and *jacobs_coffee*, but there is almost no margin for error. This made the grasping in the live test hard. However, in all test runs, we observed the PR2 to correctly steer its manipulator correctly towards the object. A successful attempt where the PR2 grasps the recognized *assam_tea* is shown in Figure 4.9.

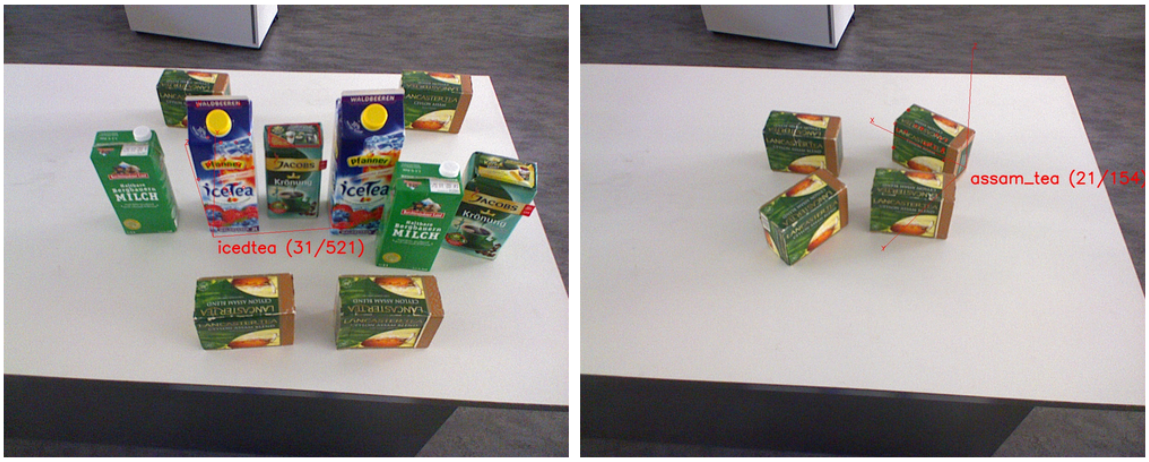


Figure 4.6.: From left to right: (a) *icedtea* was recognized despite of the presence of many duplicates. (b) `CLUTSEG` recognized one of the four items of *assam_tea*.

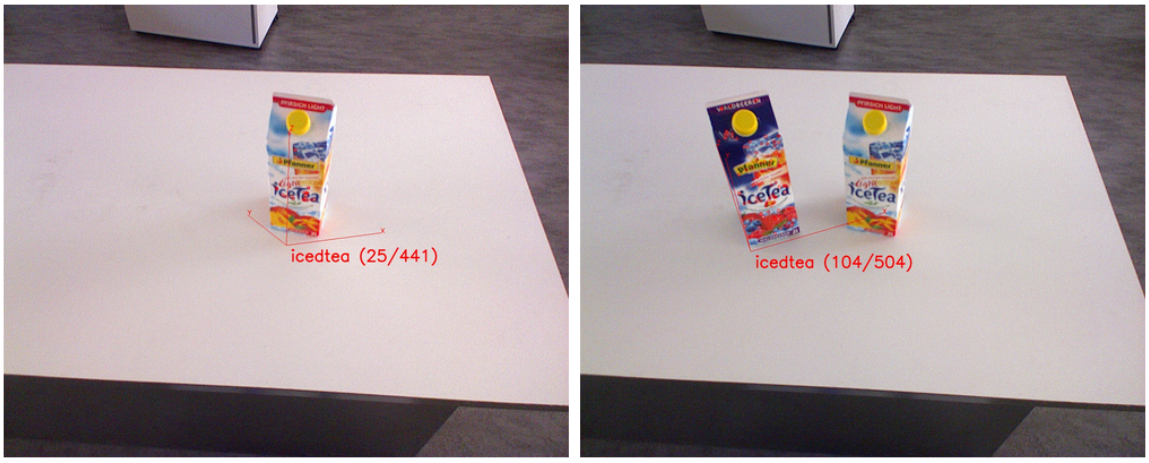


Figure 4.7.: From left to right: (a) The (wild berries) *icedtea* in the modelbase was confused with peach ice tea. (b) The additional presence of (wild berries) *icedtea* resolves the confusion.

Figure 4.8.: Many textured objects in the scene render recognition more difficult. From left to right: (a) *icedtea* was recognized with 40 inliers in a scene with 11 textured objects. (b) After removing *icedtea* and its easy-to-confuse peach-flavored variant, *jacobs_coffee* was recognized with 21 inliers out of 284 correspondences.

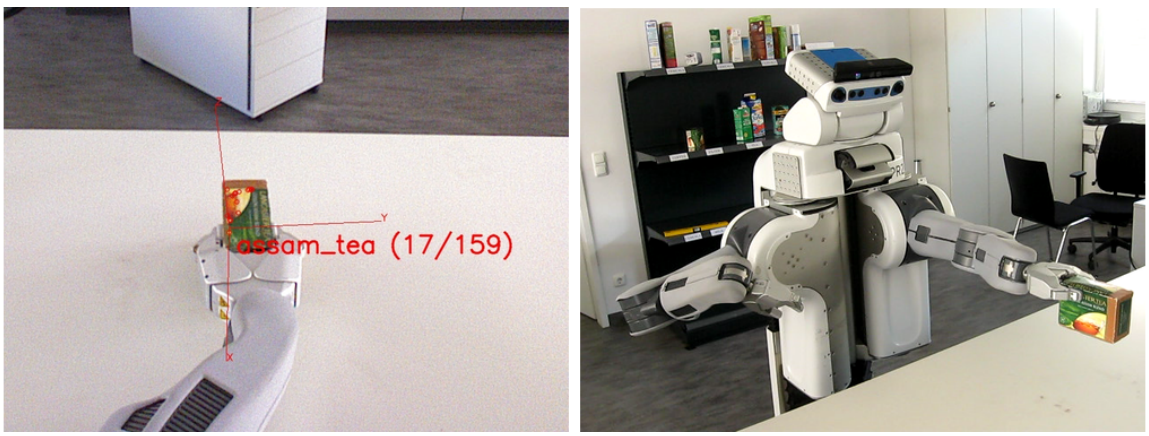


Figure 4.9.: The PR2 grasps *assam_tea*.

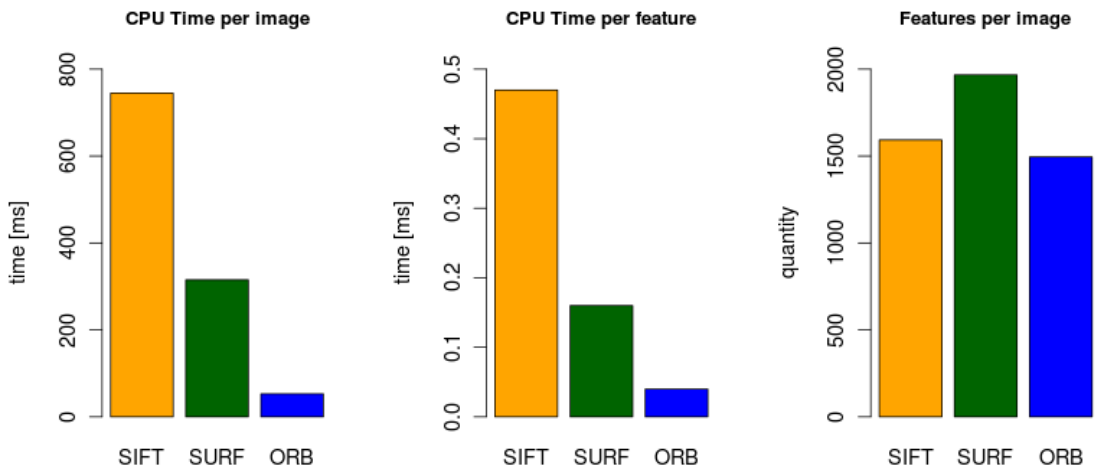


Figure 4.10.: Comparison of SIFT, SURF and ORB in terms of speed and number of features.

4.3.3. ORB

This subsection discusses results of experiments with ORB. It covers a benchmark that compares computation speed of ORB, SIFT, and SURF. It shows how ORB responds to the number of desired features, a parameter that cannot be found in many other feature detectors.

Computational Speed

We first compared SIFT, SURF and ORB in terms of computational speed and the quantity of features produced. We used the default parameters from `OPENCV` (SVN revision 5465), except for ORB, where the number of desired features was set to 1500.

Figure 4.10 shows the CPU time per image, the CPU time per keypoint, and the number of keypoints, averaged over 21 images from the validation set. None of the three detectors run in parallel, and the measured CPU time was roughly equivalent to wall-clock time. The hardware and software configuration of the notebook used for the benchmark is shown in Table C.2.

ORB ran about 14 times faster than SIFT, and about 5 times faster than SURF. Because ORB required only 52.86 milliseconds for computing keypoints and descriptors for one image, the amount of time `CLUTSEG` spends in computing query features becomes negligible compared to the time spent in matching and pose estimation.

Number of Features

In this work, it has been necessary to control the number of features selected by a feature detector. In general, the number of features selected by a feature detector should be con-

Figure 4.11.: From left to right: (a) *assam_tea*, (b) *clutter*, (c) *house*, and (d) *mandrill*.

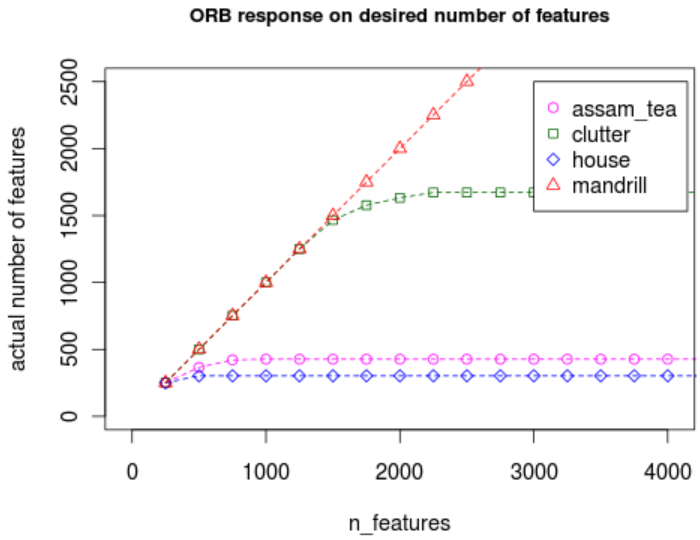


Figure 4.12.: The response of ORB on the number of desired features.

figurable, using a simple and intuitive threshold [1]. ORB has such a threshold, and our evaluation indicates that this threshold is effective.

The parameter `n_features` specifies the number of desired features produced by ORB. Other feature detectors and descriptors in `OPENCV`, such as SIFT and SURF do not provide such a parameter. Neither does the FAST feature detector. The parameter *n_features* is simple and intuitive. Yet, ORB does not make any guarantee about the actual number of features extracted on the image. The parameter is just a hint. The question arises how well ORB responds to different numbers of desired features. ORB selects features based on the Harris corner measure [5].

Figure 4.12 plots the number of desired features against the number of features that were actually obtained by ORB. The four test images *assam_tea*, *clutter*, *house*, and *mandrill* (Figure 4.11) carry different amount of informations, though all were sampled at the same resolution of 512x512 pixels. Increasing values for `n_features` have been tried until the

number of actual keypoints reaches its maximum for all four images.

Mandrill has fine-grained texture, and ORB extracted maximally 13865 keypoints on this image. This number is greater than for image *house*, where only 303 features were chosen for all values of `n_features` greater or equal than 500. The actual number was never observed to exceed the desired number of keypoints by more than 2 keypoints.

4.4. Issues

4.4.1. Repetitions in Texture

The system had difficulties to recognize *icedtea* when shown from a certain angle. It turned out that *icedtea* has smallprint almost all over its backside (Figure 4.13). The same set of letters repeatedly appear over a large area, which probably makes it hard to obtain informative features. Furthermore, focus becomes an issue.

Figure 4.13.: Repetitive texture on template object *icedtea*.

4.4.2. Consensus for Distant Pose Estimates

Generally, the system relies on the size of the consensus set to be a strong indicator for the goodness of an estimate. We observed that in one experiment and scene, `CLUTSEG` estimated a distance of 88 metres between *icedtea* and the camera (Figure 4.15). This bad estimate was almost two orders of magnitude off the ground truth. Yet it still had support by 14 inliers — enough in this scene to make it the best-ranked guess.

An explanation for the large consensus set is that the projection error is a measure on the image plane. Given a scene, consider any correspondence $c_i = (q_i, p_i) \in \mathbb{R}^2 \times \mathbb{R}^3$ between a query feature with keypoint q_i and a model feature with 3D point p_i. Let ${}^c_o\hat{T}$ be a pose estimate. Let $U : \mathbb{R}^3 \to \mathbb{R}^2$ denote perspective projection for the calibrated camera. Define the projection $\hat{q}_i$ of the model point p_i, aligned to the scene by ${}^c_o\hat{T}$, as

$$\hat{q}_i := U\left({}^c_o\hat{T}\,(p_i) \right) \tag{4.4}$$

The correspondence c_i is designated inlier if and only if the L2-distance between query keypoint and the projection of the model point aligned to the scene is less than a threshold r, precisely

$$c_i \text{ is an inlier w.r.t. } {}^c_o\hat{T} \quad \Leftrightarrow \quad || \, q_i - \hat{q}_i \, ||_2 \leq r \quad \wedge \quad p_i \text{ is visible} \tag{4.5}$$

Now, let us model the scenario, where a distant pose estimate was generated for an object that is close to the camera in truth. Let ${}^c_oT$ denote ground truth for ${}^c_o\hat{T}$.

$$e_t \left({}^c_o\hat{T}, \, {}^c_oT \right) \geq l \tag{4.6}$$

Choose l large enough that the projection of the aligned model spans only one pixel $\hat{q}$ on the digital image plane (Figure 4.14). For the n correspondences of this object, this means

$$\forall i \in [n] : \hat{q}_i \approx \hat{q} \tag{4.7}$$

Plugging Equation 4.7 into Equation 4.5, we obtain an inlier criteria for the bad estimate:

$$\forall i \in [n] : \quad c_i \text{ is an inlier w.r.t. } {}^c_o\hat{T} \quad \Leftrightarrow \quad || \, q_i - \hat{q} \, ||_2 \leq r \quad \wedge \quad p_i \text{ is visible} \tag{4.8}$$

All correspondences within a circle of radius r around $\hat{q}$ are designated inliers. If this circle contains many keypoints on the query image, the resulting consensus set of the pose estimate might grow larger than expected. The effect can be neglected if radius r is small enough. In the experiment that exhibited the phenomenon, the radius was set to $r = 12$ (pixels). Note that this radius is equivalent to the parameter `max_projection_error`.

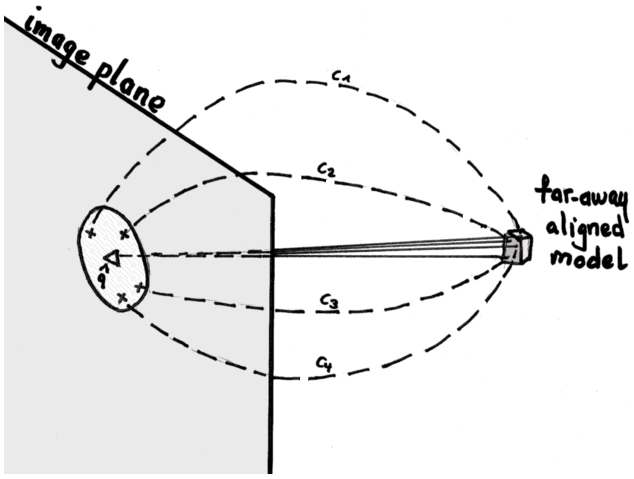


Figure 4.14.: An example, where all points of a far-away aligned model are projected to the same point ($\triangle = \hat{q}$) on the image plane. All four correspondences c_1, c_2, c_3, and c_4 are designated inliers because all the associated query keypoints ($\times$) fall into the circle around point $\hat{p}$.

There are at least three hypothetical solutions to this issue. One might set the parameter `max_projection_error` to a lower value, which is what we have done. The bad pose

estimates could be pruned away by the a-priori assumption that objects are within a certain range. Finally, the radius r could be replaced by a function that depends on the estimated distance $\left|\left|{}^{c}_{o}\hat{T}\,(\mathbf{0})\right|\right|_2$.

Figure 4.15.: On a test scene, the badly aligned pose still results in many inliers.

4.4.3. Incomplete Models

When recording data for a modelbase, the bottom of each template object is facing the rotating table, and is thus invisible to the camera from all viewpoints. This does not matter as long as instances of these template objects are standing upright. This does matter however, when instances occur in arbitrary orientation.

4.4.4. Systematic Error in Models

Unfortunately, analysis of the 3D models used in the evaluation revealed that the models have been corrupted by a systematic error. The 3D point clouds that make up the 3D models do not meet our expectations. We compared our models with the `TOD` models of *campbells_chicken_noodle*, *downy*, *fat_free_milk*, and *good_earth_tea* from the `TOD` tutorials [1]. Whilst *downy*'s model seems valid when projected onto a plane (Figure 4.16 a), *icedtea*'s model does not (Figure 4.16 b-c).

Figure 4.17 shows the *xy-coordinate histograms* of the x, y-coordinates of the model points. These histograms were computed by projecting the model points orthogonally onto the xy-plane, counting the number of points that fall into the same bin of 320x320 bins (pixels) in total. Outliers are not included in the histogram. Both object coordinate axes are drawn onto the histograms. The histogram clearly shows the erroneous models, especially in the case of *icedtea*'s; its xy-coordinate histogram has roughly the form of a hash ('#').

In comparison, the `TOD` model histograms look as expected (Figure 4.18). Even the texture-less lid on the top of *fat_free_milk* can be made out on the histogram (Figure 4.18 c). Just like *icedtea*, *fat_free_milk* is a cuboid. Hence, their xy-coordinate histograms should be similar, which is not the case.

[1] http://vault.willowgarage.com/wgdata1/vol1/tod_kinect_bags/training . Accessed 5 July 2011.

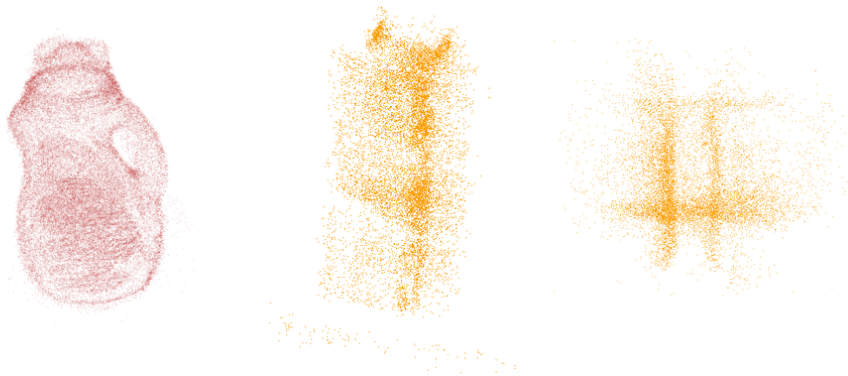


Figure 4.16.: From left to right: (a) Correct model of *downy*; (b) side view of *icedtea* model; (c) top view of *icedtea* model.

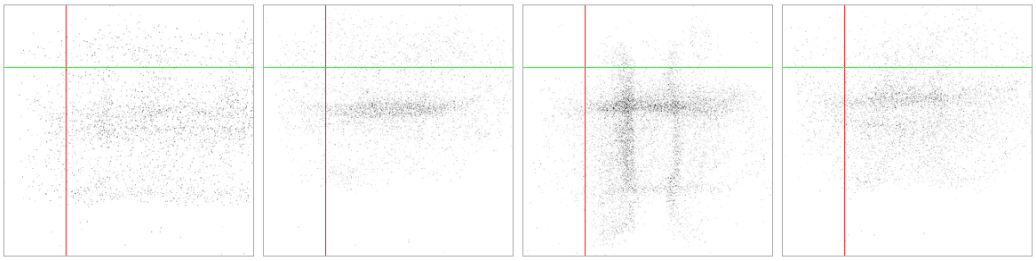


Figure 4.17.: From left to right: *xy*-coordinate histogram of (a) *assam_tea*, (b) *halt-bare_milch*, (c) *jacobs_coffee*, (d) *icedtea*.

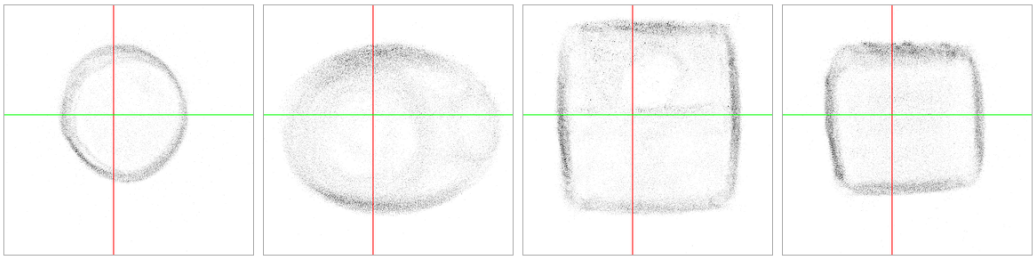


Figure 4.18.: From left to right: *xy*-coordinate histogram of (a) *campbells_chicken_noodle*, (b) *downy*, (c) *fat_free_milk*, (d) *good_earth_tea*.

Unfortunately, the source for the systematic error is yet unknown, although the odds are that the issue is related to camera calibration and/or the accuracy of the depth image. Despite the issues presented in this section, the performance on the validation set and the live test is promising. It remains to see whether better models can further improve results.

5. Conclusion

5.1. Key Results

While our evaluation reveals that many issues remain to be solved, we have shown that an object recognition system can be built on three basic concepts: First, extracting local 2D features using off-the-shelve feature detectors and descriptors. Second, nearest-neighbour search to find correspondences between the scene under investigation and the models. Third, a model-fitting algorithm that aligns models to the scene such that they are consistent with these correspondences.

Our experiments showed that the Oriented BRIEF feature detector and descriptor are 14 times faster than SIFT, and 5 times faster than SURF. ORB is interesting for real-time applications. Oriented BRIEF is one of the first feature detectors and descriptors that permit us to conveniently specify the number of features to be extracted.

Locally-Sensitive Hashing proved to work well with binary features; it was a magnitude faster than brute-force matching in the validation scenes, and both the success rate and the average guess score were hardly impacted by trading exactness for speed.

Although RANSAC turned out to be surprisingly robust with respect to outliers and noise, in our experiments, it achieved so only with 1000 iterations. Hence, most of the time was spent in RANSAC when recognizing objects in a scene, compared to the nearest-neighbour search with Locally-Sensitive Hashing, or to the negligible amount of time spent in generating local 2D features with Oriented BRIEF.

`CLUTSEG` and `TOD` use randomized and approximate algorithms, and can cope with noisy data. They are exposed to high levels of uncertainty. This calls for a statistical analysis, which can only be accomplished if the systems are designed from the beginning to incorporate methods for collecting statistical data.

5.2. Contribution

We have helped to fix some issues with `TOD`, `ROS` and `OPENCV`. We provided a test case that helped to make the SIFT implementation in `OPENCV` respect the image mask provided when extracting features using SIFT. A problem with the DynamicFeatureAdaptor in `OPENCV` which adaptively tries to extract a pre-specified number of features from an image was fixed soon after our bug report. We have helped improve the YAML implementation for `OPENCV` 2.3, and reported some usability issues for the point cloud viewer in `ROS`. A number of issues have been fixed with our help in `TOD`.

We have given an early introduction to the new Oriented BRIEF feature detector and descriptor in `OPENCV`, and have shown that it outperforms SIFT and SURF in terms of speed in our experiments. We have demonstrated that Oriented BRIEF can be used together with Locality-Sensitive Hashing.

We formulated the problem of estimating the pose of an object in terms of camera calibration and showed how it can be reduced to solving the perspective-n-point problem. We showed how RANSAC, when used for solving the perspective-n-point problem, can find bad solutions in case the projection error threshold on the image plane is chosen too large.

We presented the use of dithered binary images in camera calibration with chessboard-based fiducial markers in order to achieve more reliability in the chessboard detection.

We implemented `CLUTSEG` to transform the experimental `TOD` library into a towards a system that reveals its inner workings. We discussed the issues that showed up in its evaluation, without hiding that many problems still remain to be solved.

We provided an experiment runner that served as a robust tool for optimizing the system parameters. Our experiments verified the assumption that it is a good idea to base confidence values on the number of correspondences that are consistent with pose estimates, and that starting from a fairly inexact set of initial guesses, we can obtain a guess with high confidence by refining the initial guess with the highest ranking.

5.3. Future Work

Our experiments suggest a number of improvements. The collection of raw data for models can be improved. We could do away with the fiducial markers by rotating the objects with a robotic manipulator. The ground truth can be computed by the transformations that correspond to the joints of the robotic manipulator. This should get rid of the systematic error we observed in our collected models. Certainly, it simplifies the collection of raw data, as there is no longer the need to prepare a rotating table with fiducial markers, and no need to provide a description to the robot on how to recover the ground truth from fiducial markers. Furthermore, a robotic manipulator (grasping the object) permits to move the template object in all six degrees of freedom. Hence, the models could include model features extracted from a template object in arbitrary orientations; thus accounting for the objects to appear in arbitrary orientations in the query scenes.

The performance of `TOD` and `CLUTSEG` is not yet sufficient to robustly recognize objects in a cluttered scene in which objects appear in arbitrary orientations and high occlusion rates (see Figure 1.1 a, Figure 4.5).

One idea would be to take the uncertainty in the correspondences into account. We can weigh correspondences according to the distance between query feature descriptor and model feature descriptor. We let RANSAC draw correspondences randomly with probabilities proportional to the weights, instead of drawing from a uniform distribution. This way, RANSAC will more often consider close correspondences between the query image and the modelbase; we expect that this permits to run RANSAC with fewer iterations. Even more, the confidence value for a pose can be computed from the sum of weights of all those correspondences that are consistent with this pose, a voting scheme that accounts for uncertainty. These ideas about taking uncertainty into account are not new; a method based on model feature uncertainty, and which also involves weighting correspondences is presented in [17].

Whichever method is used, experiments will tell which method works best. These experiments require the availability of the ground truth for test scenes. The collection of ground truth in this work has been limited to multiple objects standing upright on a table.

Future work should involve computing the ground truth for scenes with objects in arbitrary orientations.

Finally, much of the data observed from a query scene is not being used. The information gained by extracting local features from the 2D appearance could be supplemented by also considering features from the 3D shape of objects.

Appendix

A. Classes

The following table lists selected classes in `CLUTSEG` and its dependencies that are related to concepts mentioned in this work. This mapping is meant to provide a rough guidance that helps close the gap to the implementation.

Package	**Class**	**Corresponds to**
`tod`	`Features2d`	set of local 2D features extracted from a single image
`tod`	`Features3d`	set of model features extracted from a single view
`opencv_candidate`	`Camera`	camera model, including extrinsic parameters
`opencv_candidate`	`PoseRT`	proper rigid transformation
`opencv_candidate`	`Pose`	proper rigid transformation
`tod`	`TrainingBase`	modelbase
`tod_detecting`	`Guess`	guess
`tod_detecting`	`GuessGenerator`	pose estimation
`clutseg`	`Experiment`	experiment
`clutseg`	`Paramset`	set of parameter values
`clutseg`	`Response`	result of an experiment
`clutseg`	`GuessRanking`	guess ranking
`clutseg`	`ResponseFunction`	performance measure

Table A.1.: A mapping between the terminology in the theory and in the implementation.

B. Bug Reports

This is a list of related bug reports filed in the course of this work. For most of them, patches have been supplied.

Library	Issue	Title
OPENCV	1044	SiftFeatureDetector::detect ignores mask parameter
OPENCV	1127	cv::Exception when reading empty YAML file
OPENCV	1169	ORB n_features=0
ROS	5072	pcd_viewer in pcl_visualization fails to load files in directories ending on .pcd
ROS	5073	pcd_viewer in pcl_visualization does not restore terminal state when sent to background
TOD	5083	Getting OpenCV errors when using recognizer in tod_detecting
TOD	5084	train_all.py in tod_training is broken and probably redundant
TOD	5085	frecognizer in tod_detecting fails with "Assertion 'pos < m_num_bits' failed."
TOD	5086	description of command-line options of frecognizer in tod_detecting is out-dated
TOD	5093	Matcher::add uses object indices rather than object identifiers
TOD	5097	drawProjections in GuessGenerator is broken
TOD	5103	tod_training detects features outside of region of interest
TOD	5104	frecognizer detects way too many keypoints
TOD	5111	Workaround for OpenCV 1044: Features outside of mask included in training
VSLAM	5056	Seed is not initialized for RANSAC random number generator in vslam/posest

Table B.1.: Related bug reports.

C. Hardware and Software Setup

Component	Type
Mainboard	ASUS K8V SE Deluxe
CPU	AMD Athlon64 3000+ S754
RAM	1 Gigabyte DDR-RAM
OS	Ubuntu 10.10 Maverick

Table C.1.: The hard- and software configuration for parameter optimization.

Component	Type
CPU	Intel Core i5-560M
RAM	2 Gigabyte DDR3
OS	Ubuntu 10.10 Maverick

Table C.2.: The hard- and software configuration for computation speed benchmark.

Glossary

aligned model a model inserted into the scene with a certain pose. 3

camera coordinate system describes 3D points relative to the camera. 6

camera-object transformation transforms coordinates from the camera coordinate system into the object coordinate system. 6

confidence value size of the consensus set found by RANSAC. 3

correspondence a supposed match between a query feature and a model feature; found by nearest-neighbour search. 3

guess a pose estimate for a specific object; has a confidence value. 3

guess ranking orders guesses by a measure of "goodness", for example by confidence value. 21

guess refinement tries to estimate the pose of a specific object in the scene as accurate as possible. 23

guess rejection rejects a guess if it is not "good" enough, for example if it has a too low confidence value. 23

guess score assigns a score that describes the distance of a guess to the ground truth in a scene. 27

inlier a correspondence that is consistent with a pose estimate up to a certain margin of error; member of the consensus set returned by RANSAC. 13

model feature a local feature; the basic building block for a model of a template object. 3

object coordinate system describes the 3D points of an object or a model. 6

object-camera transformation transforms coordinates from the object coordinate system into the camera coordinate system. 6

query feature a local 2D feature extracted from a query image. 3

query image the image of a query scene taken by the camera. 3

query scene a scene under investigation where objects are to be recognized. 3

template object used for learning a model, as opposed to the instances in a query scene. 3

Bibliography

[1] T. Tuytelaars and K. Mikolajczyk, "Local Invariant Feature Detectors: A Survey," *Foundations and Trends in Computer Graphics and Vision*, vol. 3, no. 3, pp. 177–280, 2007.

[2] D. G. Lowe, "Object Recognition from Local Scale-Invariant Features," in *Proceedings of the Seventh IEEE International Conference on Computer Vision*, vol. 2, pp. 1150–1157, IEEE, 1999.

[3] E. Rosten and T. Drummond, "Machine learning for high-speed corner detection," *Machine Learning*, vol. 1, no. 1, pp. 1–14, 2006.

[4] M. Calonder, V. Lepetit, C. Strecha, and P. Fua, "BRIEF : Binary Robust Independent Elementary Features," in *Computer Vision - ECCV 2010* (K. Daniilidis, P. Maragos, and N. Paragios, eds.), vol. 6314 of *Lecture Notes in Computer Science*, pp. 778–792, Springer, 2010.

[5] E. Rublee, V. Rabaud, K. Konolige, and G. Bradski, "ORB: an efficient alternative to SIFT or SURF," in *Proceedings of the 13th IEEE International Conference on Computer Vision*, 2011. To appear.

[6] A. Gionis, P. Indyk, and R. Motwani, "Similarity Search in High Dimensions via Hashing," in *Proceedings of the 25th International Conference on Very Large Data Bases*, VLDB '99, (San Francisco, CA, USA), pp. 518–529, Morgan Kaufmann Publishers Inc., 1999.

[7] M. Slaney and M. Casey, "Locality-Sensitive Hashing for Finding Nearest Neighbors," *Signal Processing Magazine, IEEE*, vol. 25, pp. 128 –131, March 2008.

[8] M. A. Fischler and R. C. Bolles, "Random Sample Consensus: A Paradigm for Model Fitting with Applications to Image Analysis and Automated Cartography," *Communications of the ACM*, vol. 24, pp. 381–395, June 1981.

[9] M. Dogar and S. Srinivasa, "Push-Grasping with Dexterous Hands: Mechanics and a Method," in *Proceedings of 2010 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2010)*, pp. 2123–2130, October 2010.

[10] D. Forsyth and J. Ponce, *Computer Vision: A Modern Approach.* Pearson Education, 2003.

[11] R. C. Gonzalez and R. E. Woods, *Digital Image Processing.* New Jersey: Pearson Prentice Hall, Third ed., 2010.

[12] E. Alpaydin, *Introduction to Machine Learning.* MIT Press, 2 ed., 2010.

[13] T. Fawcett, "An introduction to ROC analysis," *Pattern Recognition Letters*, vol. 27, no. 8, pp. 861 – 874, 2006.

[14] J. Melsa and D. Cohn, *Decision and estimation theory*. McGraw-Hill, 1978.

[15] PrimeSense, "PrimeSensor Reference Design 1.08 Datasheet," 2010.

[16] A. I. R. Galarza and J. Seade, *Introduction to Classical Geometries*. Birkhäuser, 2007.

[17] A. R. Pope and D. G. Lowe, "Probabilistic Models of Appearance for 3-D Object Recognition," *International Journal of Computer Vision*, vol. 40, pp. 149–167, November 2000.

[18] G. Bradski and A. Kaehler, *Learning OpenCV: Computer Vision with the OpenCV Library*. O'Reilly, 2008.

[19] E. Rosten, R. Porter, and T. Drummond, "FASTER and better: A machine learning approach to corner detection," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 32, pp. 105–119, 2010.

[20] D. G. Lowe, "Distinctive Image Features from Scale-Invariant Keypoints," *International Journal of Computer Vision*, vol. 60, no. 2, pp. 91–110, 2004.

[21] H. Bay, T. Tuytelaars, and L. V. Gool, "SURF: Speeded-Up Robust Features," in *Computer Vision - ECCV 2006*, vol. 3951 of *Lecture Notes in Computer Science*, pp. 404–417, Springer, 2006.

[22] M. Brown and D. G. Lowe, "Automatic Panoramic Image Stitching using Invariant Features," *International Journal of Computer Vision*, vol. 74, no. 1, pp. 59–73, 2006.

[23] AMD, "AMD64 Architecture Programmer's Manual, Volume 3 : General-Purpose and System Instructions," November 2009.

[24] Intel, "Intel ® SSE4 Programming Reference," July 2007.

[25] G. Tsoumakas and I. Katakis, "Multi-label classification: An overview," *International Journal of Data Warehousing and Mining*, vol. 3, no. 3, pp. 1–13, 2007.

[26] M. Quigley, K. Conley, B. P. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, and A. Y. Ng, "ROS: an open-source Robot Operating System," in *International Conference on Robots and Automation Workshop on Open Source Software*, 2009.

[27] R. B. Rusu and S. Cousins, "3d is here: Point Cloud Library (PCL)," in *International Conference on Robotics and Automation, Shanghai, China*, June 2011.

[28] A. Shishkov. On http://answers.ros.org, March 2011.