ROBUST OBJECT DETECTION FOR SERVICE ROBOTICS

TIM DE JAGER

ICA-3698823

SUBMITTED FOR THE DEGREE OF MSC.

Multimedia and Geometry
Computer Sience
Utrecht University

October 17, 2013

Tim de Jager: *Robust object detection*
for service robotics, with the AMIGO service robot, © October 17, 2013

THESIS CODE:
ICA-3698823

SUPERVISORS:
Dr. Robby T. Tan
Msc. Sjoerd van den Dries

REVIEWER:
Prof. Dr. Remco Veltkamp

LOCATION:
Utrecht

DATE:
October 17, 2013

ABSTRACT

In the near future robotics will become a much larger part of our society, and more robots will be developed to execute a range of general tasks, so that they are able to assist human operators in their day to day work. At the Technical University of Eindhoven (TU/e), the AMIGO robot is designed to perform such general service tasks, with a special focus on health care applications. A general purpose service robot, such as AMIGO, needs a certain skill set to perform tasks autonomously. One of these skills, is being able to perceive objects in the world. Using this information AMIGO can act accordingly, e. g., by bringing the required object back to its operator.

This thesis describes a system for detecting household objects in domestic environments. The object detection system has been developed for AMIGO, and is based on the object detection method, LINE-MOD, presented by hinterstoisser *et al.* [1]. To make the system applicable for both the general service tasks performed by AMIGO and the RoboCup competition, an annual event encouraging robotic research, LINE-MOD is extended by adding extra color modalities to the detection system, testing different normal extraction routines and enabling the use of additional features. The detection module is subsequently included in a framework with a different set of tools, making it of practical use for the AMIGO project and its users, and is integrated with existing systems used currently used in the AMIGO project. The detection module is evaluated with household objects that AMIGO encounters while performing service related tasks and competing in the RoboCup competition.

The system is discovered to be able to recognize most household objects with reasonable accuracy, the use of an added color modality results in the increased accuracy of the detection of household objects at a small performance penalty. However, it may still fail in situations where objects are highly textured, resulting in false positives inside object instances, and when object models are too similar in shape or color a problem which is primarily caused by the use of highly quantized feature types.

Our vision for robotics depends on perception..

— Gary Bradski

ACKNOWLEDGMENTS

This thesis could not have been realized without the support of a number of key people. I would like to thank the Robotics group at the Technical University of Eindhoven for providing an enthusiastic environment for working on projects in Robotics. You are the best! The group includes a number of very talented individuals that are making service robotics possible, some of which I would like to thank specifically.

A special thanks goes out to my supervisor, Sjoerd van den Dries, for fruitful discussions on the late night train back to Utrecht, for having a great number of intuitive ideas to share, and for having a wonderful sense of humor and thus never failing to make people smile. I especially admire the way you can motivate and activate people working with you, as well as your ability to tackle complex projects with ease. Your enthusiasm is infective, I hope you can continue to inspire people and wish you the best of luck with your research.

Another big thanks goes to the other Phd. candidates (Jos, Janno and Rob) in the AMIGO project, without you AMIGO probably would never have come this far, let alone capture the third position at the last Robocup.

I would also like to thank fellow master student Bas Coenen especially. Whom I have have come to know, as an absolutely inspiring, talented and friendly person that is able to express himself and his ideas with absolute clarity. Thank you for the inspiring discussions about our projects that could sometimes span hours, and for always providing a place to sleep after having some beers in Eindhoven. The past year has not been easy for you, but I have seen that you have endured past hardships with such resilience that I know all will be well, I wish you only the best.

I would like to thank my parents and sister for keeping their faith in me, even when my graduation project has taken on much longer than it was supposed to. Without your unending patience, great advice and unconditional support, I would not even have made it halfway. You are my favorite people on this earth, I loved spending the weekends with all of you and will always be grateful that you always welcomed me back into my former home with open arms. I hope that we get to spent many more days together the coming years.

The University of Utrecht also receives my gratitude for providing a great master program, with another thanks going out to my supervisor dr. Robby Tan, I wish you and your future research all the best!

Finally, I would like to thank AMIGO for performing so admirably when I have asked him to, always trying to detect objects to the best of his ability. Thank you, you have been a star!

CONTENTS

Part I

INTRODUCTION

1

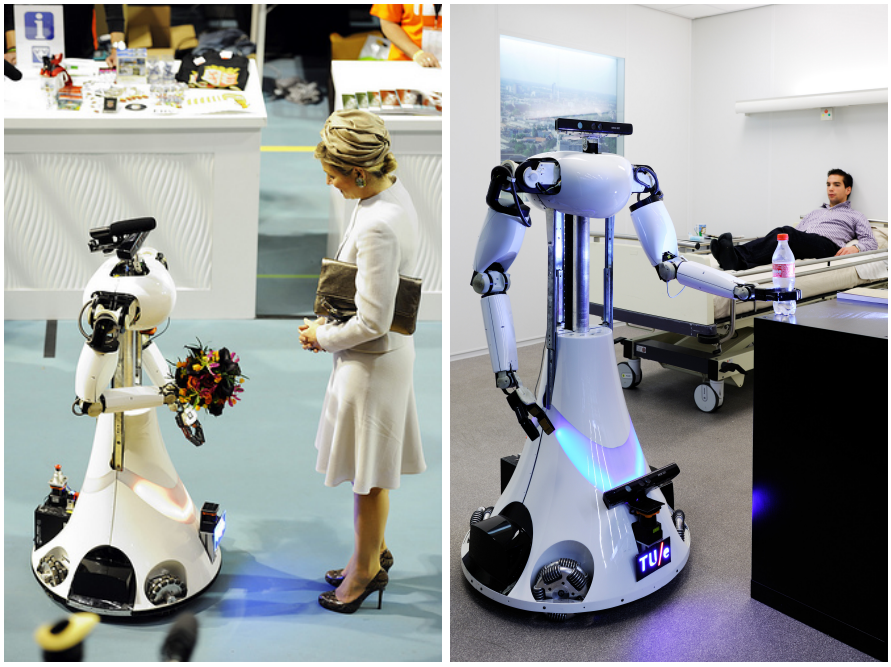
INTRODUCTION

Since the 20th century robotics is becoming an increasingly active field of research. With both the software and hardware becoming more advanced, we will continue to see an increase of robots in our direct environment. The focus in robotic research has always been a combination of both structurally sound hardware and sufficiently advanced software, to make a given task feasible. Industrial robots have been actively for three decades now. However, because robots are now also being applied in commercial, domestic and military applications, the encountered environments are not always as well conditioned as industrial environments, which increasing the need for generic and robust systems. Domestic service robots are required to perform a number of autonomous tasks, e.g. serving food and drinks, which requires a large amount of sophistication in both the hardware and software. Domestic service robots will need to be accurate, safe and fast. Ideally performing a task at least the rate and efficiency of a human. Although a significant amount of steps have been made, we are not quite there yet. Software is still insufficiently robust and accurate, and the costs for producing a robot are high. Furthermore there is some amount of compatibility issues between different robotic systems, research done on a specific robot can often not be applied to another. These issues are being improved in a number of ways: events like the annual RoboCup [2], aim to improve and promote current research into robotics and autonomous behavior. Standard platforms like the PR2 [3] and the Robotics Open Platform (ROP)[1], aim to bring down costs and provide a standard for the hardware. On the other hand, the Robotics Operating System (ROS) [4], facilitates the sharing of knowledge between both scientific instances and commercial ventures. The Technical University of Eindhoven currently participates in the RoboCup and the Autonomous Mate for Intelligent Operations (AMIGO) is the current iteration of the robotic platform for service robotics and makes use of ROS. AMIGO is designed to be able to execute general service tasks with the focus on healthcare applications.

1 http://www.roboticopenplatform.org/

1.1 AMIGO

AMIGO is a custom robot designed by the university of Eindhoven. AMIGO features a holonomic base, human-like arms, movable torso and multiple sensory input. The sensors currently used are multiple Laser Range Finders (LRF) mounted on the base and a Microsoft Kinect [5] serving as the head. A figure of AMIGO is displayed in Figure 1. As can be seen AMIGO has a distinct appearance with a some human-like features.



(a) AMIGO handing over flowers (b) AMIGO grasping a bottle

Figure 1: The Amigo robot in action, performing its task that usually requires interaction with humans.

Because AMIGO is aimed to be used as an autonomous service robot, it is specifically designed to perform tasks in a unconstrained domestic human environment. AMIGO should be able to take orders from humans and execute them correctly. These requirements place a number of demands on the hardware, software and the general appearance of AMIGO. Within the AMIGO project a number of research area's can be identified, which address these demands:

ACTUATION & CONTROL AMIGO should be able to control his actuators robustly and safely. The focus of this research is to remove any inherent instabilities and providing an interface which high level components should be able to access.

TASK PLANNING & EXECUTION When AMIGO is given a task, it should be able to create a plan of execution, potentially splitting

up the task and assigning priorities. When a plan has been generated, AMIGO should execute this plan correctly and safely.

KNOWLEDGE & REASONING AMIGO should be able to keep a certain amount of knowledge available, and should be able to reason about this knowledge, enabling him to answer questions about his environment, e.g. "Are there any drinks nearby" or "Have you seen this human before?"

HUMAN MACHINE INTERFACING AMIGO should be able to communicate with humans, in a correct and natural manner. Provide feedback during task execution and ask for additional information if required.

MOTION COORDINATION AMIGO should be able to plan and execute his movements, which can be required while performing a task. For example moving an arm to a correct position or planning and executing a motion plan to a designated location.

PERCEPTION AMIGO should be able to identify and recognize his environment and potential operators, and translating this seemingly unstructured environment into a model, from which knowledge can be extracted and reasoned.

Most of the work being done on AMIGO, as of writing this thesis, involves one of these subjects. This thesis will focus on the last point, i.e., perception.

1.1.1 *Project motivation*

Perception is a broad concept that can be interpreted in various ways, but in any case it involves the detection of certain elements in a scene. Eventually, the goal of perception on a robot is to get a total understanding of a scene, which means that all information is processed so that irrelevant information can be discarded and relevant information can be used. The robot would then be able to integrate this into a single model, from which it could query the necessary information, which is similar to humans who have ready access to a certain amount of knowledge about an environment. However, this is still infeasible; currently there is a focus on recognizing different types of elements in a scene such as people and objects and presenting this information to higher-level modules. As such, this thesis will present an *object detection* module that will be able to do just that. The need for such a system is high, as a lot of interaction between AMIGO and the environment will consist of the manipulation of various objects. Also in the RoboCup@home league[2], a number of challenges focus on

2 A RoboCup league focusing on service robotics.

interacting with humans by locating, retrieving and presenting various objects. In the rest of the chapter general object detection will be discussed, as well as its relation to robotics and AMIGO.

1.2 OBJECT DETECTION

In this section, general object detection and its history will be discussed, as well as some common techniques used in object detection systems. First a brief overview will be given of past research in the area, followed by a format for general detection frameworks. After this general overview, we will explore the invidual components of an object detection method. Finally, closing off with by applying this knowledge on a specific application, namely AMIGO. This final section will explore the possibilities and the eventual choice for an existing detection method.

1.2.1 *Detection of categories or instances*

However, before the general overview will be given, the distinction of the detection of instances opposed to categories will be discussed, as this choice limits the amount of available methods. The detection of object categories or the detection of specific instances of objects, can be viewed as two distinct problems. Categorizing and detecting an object in an image is generally a much harder problem, object that share the same semantic category may look completely different. The problem remains largely unsolved for difficult categories, e.g. chairs. The PASCAL VOC Challenge [6] focuses on detecting categories in images using state of the art implementations, where the results vary per category. For instance, chairs have a precision score of 19% as opposed to cats which have a 53% precision score[3]. Detection of specific instances in an image, is a somewhat simpler problem, as the appearance of the object will remain similar. The hard task is to make a system robust to variations in the environment, which receives the largest amount of attention in research papers. The trade-off is generality versus accuracy and speed, and the specific applications should decide which goal it finds more important.

1.2.2 *Brief overview*

Generic object detection has been an open problem since the advent of Computer Vision. A multitude of different approaches have been taken to solve the detection problem, some of which will be discussed below. As early as the work done by Roberts [7], there has been an

3 See the website http://pascallin.ecs.soton.ac.uk/challenges/VOC/voc2012/results/index.html for the results of the most recent competition.

interest in detecting objects from images. In subsequent years a lot of progress has been made in detecting objects in images. Thanks to classification techniques like the Hough transform [8, 9], template matching and the Normalized Correlation Coefficient [10, 11] and more general Machine Learning[12, 13, 14] (ML) methods, the possibilities to make practical object detection systems have become significantly larger. In the last two decades an increase can be seen in probabilistic/statistical classification approaches that coincide with the increasing use of Machine Learning methods. These approaches are distinct when compared to older approaches like approaches presented by Roberts [7] or Huttenlocher & Ullman [15], which seem to think of object detection as more of an Image Alignment[4] task, while modern methods regard the problem more as a general classification task. When Lowe *et al.* [16] presented the scale-invariant feature transform (SIFT), a feature description was provided that could be used reliably for a number of Computer Vision tasks, accelerating the research done in Computer Vision to a next level. The two discussed research area's: classification and feature extraction, seem to have the largest impact on object detection research. The current systems could be considered practical applications of the combination of these area's, which can provide new insight into the established theories.

When regarding recent object detection techniques one can extract a general format. In this section such a format will be given, followed by a detailed explanation of each step in which different solutions and methods will be briefly discussed. During training the objective is to create a classifier that will be able to detect either *instances* or *categories* of objects.

1. Obtain input from sensors 1.2.2.1.
2. Extract a (sparse or dense) number of features 1.2.2.2.
3. Repeat first and second steps, until enough data is collected to form an object model 1.2.2.3.
4. Train one or multiple classifiers using the object models. Using positive samples and optionally negative samples 1.2.2.4.

During detection we want to test an image and determine if it contains the object we are looking for. The steps are summarized as follows:

1. Obtain input from sensors 1.2.2.1.
2. Extract the features that we want to test 1.2.2.2, and combine these into a model that the classifier is able to accept.

4 Matching image features to another image defined by a (linear) transformation.

3. Consider multiple image locations to test, perhaps discarding some improbable locations. 1.2.2.5

4. Test the image locations using the classifier(s). A classifier is function that can produce a *probability*, *similarity* or binary *decision*, which determines if an image location is also an object location 1.2.2.6.

5. Post-process the detection results, to make a decision on whether an object is present in the image 1.2.2.7 .

These steps are very general and there are some ways that a method can deviate from the mentioned steps. For example, when employing unsupervised learning or semi-supervised learning, the training and detection steps are often combined and the object model or classifier is able to adapt online. This is opposed to supervised methods the training is done offline, with a stable classifier during online detection that evaluates the input. The Predator tracking and generic detection system [17] is a successful method that takes the former approach. Object detection approaches without tracking capabilities more often take the latter approach [18, 19].

1.2.2.1 *Input*

Up until recently seminal work on object detection have been done on 2D images [20, 11, 21, 16, 22, 23, 19]. Single or multiple calibrated [24] cameras can be used to estimate pose and position, and enable techniques like depth from stereo. Depth was often ignored in object detection, because depth from stereo techniques were relatively imprecise and good stereo cameras were expensive. However, with the release of the Microsoft Kinect, the availability of a cheap sensor with relatively high accuracy was suddenly a reality.The product has been very successful in robotics producing a lot of new research in object detection [25, 26, 27, 1, 28], and in related fields including the very impressive Kinect Fusion 3D reconstruction method [29], a feat that has been made possible, primarily by the introduction of the Kinect.

1.2.2.2 *Features*

Features play a very important role in object detection, as they can change the performance of a detection algorithm significantly. Features should ideally be *descriptive* and *invariant*: descriptive so that a collection of features can describe and object or object class, and invariant so that the detection can handle varying environmental conditions or different object poses. Prince noted in his book [30] that there are recurring ideas in feature extraction to achieve invariance and descriptiveness. Namely, to make a descriptor invariant to intensity changes we can *filter* and *pool* responses over a region. To obtain

descriptiveness and uniqueness, we can maximize the response of a descriptor over different *orientations* and *scales*. To create invariance to small translations, local responses are *pooled*. The creation of a descriptor usually uses a combination of these techniques. A large number of popular 2D descriptors exist, with SIFT [16] being the most common. Since the release of the Kinect there has also been an increase of interest into 3D descriptors. Recent methods are using 3D gradients [1], 3D SURF [31] and 3D Heat Kernel Descriptors [32] and augmenting methods with depth information [28].

1.2.2.3 *Object model*

An object model is a collection of image features that either retain spatial information or discard it, in most detection systems the spatial information is retained. A model is usually given a label like 'car' or 'coke', the objective of a object detection system is to find this model in an image. There are different ways an object can be represented, a few of which will be discussed.

A simple way to represent an object are collections of features in rectangular region annotated with coordinates, this is called a Template model. The Template model often contains multiple templates per object. It has been used in both older and more recent work [11, 20, 18, 17, 1]. The part-based model, which was also recently used by Felzenszwalb *et al.* [19], builds upon the template model it encodes the relative positions of detectable parts as opposed only the absolute feature locations. Bag of Visual Words (BoW) approaches, discard spatial information and combine the features into a visual vocabulary, this visual vocabulary can then be considered to represent the object model.

1.2.2.4 *Training a classifier*

Object detection typically defines a classification scheme to detect an object instance or category in an image. Successful works from the last decade like Lowe's *et al.* object detection system [16], Viola and Jones face detection system [18], Felzenszwalb *et al.* object detection system [19]. Are using Machine Learning (ML) techniques to create and train a classifier to classify an image or image patch. Some ML classifiers have been successfully applied in literature, with boosting [12] being one of the first [18, 33, 34], Support Vector Machines [14, 35] (SVM) also producing successful works [20, 19], and more recently Random Forests [13] (RF) are getting attention with successful applications [5, 36, 37, 22]. It is also possible to evaluate the similarity or probabilities directly. Successful hough detection systems [9, 38] and template matching techniques [11, 23, 1, 17] have been used in this regard. Alternative models, like Bag of visual Words approaches,

use Nearest-Neighbor methods or SVM's for performing the classification.

1.2.2.5 *Exhaustive Image Search*

During run-time when an input image is provided, an object detection system should decide if an object is present in the image. This can be done by doing an exhaustive image search at each location, or using a heuristic/scheme to quickly disregard possible object locations. The Viola-Jones [18] detector uses a cascade of boosted detectors, each with a significant false positive rate, to rule out image patches. The combined cascade results in low false positive rate. This kind approach has also more recently be used in Predator [17], which rules out image patches based on simple thresholding decisions, and uses increasingly complex detectors to detect the correct object. Alternative approaches evaluate the objectiveness of an image patch [39], or rule out image patches based on context [40]. There is also research being done in speeding up the detections directly, by using discrete optimization techniques like branch-and-bound [41] or regarding the problem as a convolution in the frequency domain [42]. Modern hardware can also be utilized to speed up the process directly [23, 1], often requiring the problem to be formulated differently when compared with traditional hardware because these devices are best when processing data in parallel.

1.2.2.6 *Detection*

When suitable locations have been selected, an object detection module typically detects an object by evaluating an image patch with the learned classifier. Binary classifiers typically output a decision, either the object is present or it is not. Classifiers can also output a measure of similarity, which expresses a certain amount of confidence in the outputted decision. Some classifiers can also output a probability, e. g.LDA [43]), which can be either Bayesian or point estimates depending on the classifier. Other classifiers can be adapted to do so even though they did not output a probability originally, e. g.SVM [44].

1.2.2.7 *Post-processing*

In some cases, when the classification has been made some post-processing is required, usually when the output could contain a number of false positives which can be eliminated. One of the more common post-processing steps is Non-Maxima Supression, which involves sliding a window across the image and suppresses all points which are not a maxima for that region. NMS is commonly applied on a Response Map, which is a 2D array that determines the similarity per

location. NMS can be used for suppressing false positives when detecting multiple instances. Hough detection systems [22], for example, often require this operation to perform robustly. Another common operation is the specification of a threshold on the similarity or probability, which can be used to create a binary decision regarding the presence of an object given a location in the Response Map. After performing the necessary operations, the detections can be propagated to a higher level module. This module can subsequently make use of the given information and will be able to update its state of the world.

1.3 OBJECT DETECTION ON AMIGO

In the past section we have looked at object detection in computer vision, as well as the steps that are required when trying to detect an object in an image. Now that we have obtained a better view of object detection, we will discuss object detection in robotics and specifically on AMIGO. In robotics we want to detect objects so that they can be interacted with, and the requirements dictate which object detection module should be selected. If we were to build a industrial robot that should only recognize coins then perhaps a circular hough detector would be sufficient. However, AMIGO will be required to detect objects in different environments as well as detect object robustly and interact with them by picking them up. This will require a somewhat more complex solution. In the next section we will discuss the problem of detecting an object on AMIGO and the requirements that an object detection system will have to meet. These will both be general requirements relevant for robotic solutions and requirements specific for AMIGO.

1.3.1 *Problem description*

The problem that we are aiming to solve is to detect household objects in a domestic environment. A typical use-case would be the following: AMIGO is asked to 'clean up' a room. AMIGO processes this command and proceeds to detect objects in different locations, grasps these objects and brings them to their designated locations. To be able to do this, multiple components will have to work together. Specifically, the perception module will have to recognize and communicate the type of objects and their location in the world to other modules, which enables the arm control to determine a grasping location and subsequently grasp the object. This specific use-case is also a RoboCup challenge. To be able to perform such tasks, a number of different aspects will have to be considered as to decide how to do the detection robustly and accurately. The goal of this project is to provide an object detection module that is both usable as general

object detection system for a service robot and can be used directly in the RoboCup challenges. Therefore, the system will have to meet certain theoretical and practical demands. These requirements are listed below.

1.3.2 *Requirements*

Some theoretical considerations:

ROBUSTNESS The system should be robust in home/office environments. This entails that the detection should function under reasonable (no large illumination changes, enough light available) lighting conditions. Objects can be assumed to be placed on typical locations, e.g. a table or a couch.

ACCURACY The system should be accurate enough to detect and distinguish objects that are typically found in a household environment. Non-deformable objects are not a focus yet so can be considered out of project scope.

DATABASE SIZE The system should be able to recognize enough objects to participate in the RoboCup@home league, normally featuring about 20 to 25 objects.

SENSORS AMIGO is equipped with a Kinect which should be able to provide both RGB and depth images. These images have a VGA resolution, i.e. 640 pixels in width and 480 pixels in height. An LRF is also available if needed.

SPEED The speed of the system should be quite high, ideally being to detect objects in near real-time.

GENERALITY . The system should be general enough to function in a domestic environment. The household objects to be considered can be both textured and untextured, e.g. food packaging or a coffee mug. Ideally the system should be able to handle both.

Some practical considerations:

USABILITY The system should be easy enough to be used by novices. Providing modules for learning and detection based on ROS. Ensuring a fast setup during a RoboCup competition.

INTEGRATION The system should integrate with the existing code base and make use existing modules for communication as well as higher level reasoning.

EXTENSIBILITY The system should be extensible, so that future students can use and further extend the system.

The proposed system will need to meet most of these requirements. In this thesis, a system will be presented which is based on an existing object detection algorithm.

1.3.3 *Advocated method*

The method that is built upon in this thesis is chosen to be the LINE-MOD object detection method, presented by Hinterstoisser *et al.* [1]. LINE-MOD was deemed suitable for our purposes because of the following considerations:

- The method is fast, providing object detection capabilities in real-time.
- The method provides support for multiple modalities, i. e.both RGB and Depth types, as well as being tested on a sensor that is currently being used, viz. the Microsoft Kinect [5].
- The method supports the detection of non-textured objects, which can often be found in household locations.
- The method needs little training time. This is especially advantageous because objects often need to be learned in a small time frame.
- According to the original paper the method does not need a lot of tuning to perform well, which is often hard to do for dynamic environments.

These listed considerations, made LINE-MOD seem a suitable candidate to base the new object detection system on Amigo, and will be used during the course of this thesis.

1.3.4 *Research Questions*

In the work presented by Hinterstoisser, a algorithm is given for the detection of objects. By combining multiple modalities and making use of the new Kinect hardware, a method has been presented that appears to be robust enough to utilize in a Robotics environment. The interest of Eindhoven in the project is mostly the application of this method on AMIGO. Object detection is a task that AMIGO should be able to perform robustly. The following questions remain:

1. LINE-MOD was created for detecting texture-less objects, How well does LINE-MOD generalize to objects used in the RoboCup competition? What are the current deficiencies and how could these be tackled?

2. Is LINE-MOD a suitable candidate for the integration into the AMIGO infrastructure and could it be used for everyday tasks? Additionally, is LINE-MOD suitable for use in general service robotics, and not just the AMIGO project?

1.4 CONTRIBUTIONS

The main contributions of this thesis are the implementation, extension and evaluation of LINE-MOD, as well as the creation of a framework for object detection on Amigo. These can be summarized as the following:

1. Provided a complete object detection system for AMIGO, which provides the following additional features:
   a) Automatic loading/saving of the object database
   b) online loading of configurations for runtime switching
   c) Learning and detection frameworks
   d) Extensible system, automatic loading of different modalities
   e) online visualizations
   f) Integration with ROS and existing systems
2. Inclusion of an extra modality, i. e.color, which includes both a Hue and a Color Name [45] descriptor.
3. Different estimation schemes are provided for 3D gradient (normal) calculation.
4. Small extension: multi-threading of input process and the raising the amount of features that can be used.
5. Small extension: sorting of the gradients by location on the template, reducing cache misses
6. Provided an evaluation of the current system, as an application on AMIGO.
7. Practical Application of the method on the RoboCup.

1.5 THESIS ORGANIZATION

The system described in this thesis is largely based off the existing work done by Hinterstoisser *et al.* [1], a custom implementation is provided that has been integrated into AMIGO as a new system with a number of extensions. In the following chapters we will discuss the implemented system in detail. First, an introduction and overview is provided of the implemented system. This is followed by a detailed

discussion of the steps of the object detection module, as well as a discussion regarding the implementation running on AMIGO. Visualizations are provided, to showcase the different steps of the object detection module.

The Object Detector implemented on AMIGO is described in the subsequent chapters:

CHAPTER 2 will focus on the description of the core detection module, based on the LINE-MOD detection module [1].

CHAPTER 3 explains the additions made to LINE-MOD, to make it more suitable for household environments and the RoboCup challenge

CHAPTER 4 will detail the Object Detector as a whole and how it has been integrated on AMIGO, focusing especially on the AMIGO ecosystem and the integration with existing systems.

CHAPTER 5 will focus on testing the system with objects encountered by AMIGO and will evaluate these results.

CHAPTER 6 will conclude this thesis by answering the research questions and providing future work.

Part II

METHOD

2

CORE DETECTION MODULE

2.1 INTRODUCTION

This chapter will mainly focus on the original implementation as it has been presented [1]. However, some notation will differ as to what has been used in the original paper, so as to accommodate the subsequent chapters. All visualizations have been produced by the custom implementation, but some images have been taken from the original paper to help clarify portions of the text.

LINE-MOD as it is presented in the work by Hinterstoisser [1] is a template matching method at its core. The contribution was the fact that the method can be easily linearized and can therefore make use of low-level optimizations. Also a number of steps can be preprocessed offline which further increases the processing speed. LINE-MOD is an instance based detection method, this entails that the method needs to learn the object instances itself and does not generalize to categories. Another feature of the method is that the it can perform quite well in cluttered scenes, mainly because of the use of multiple modalities, which in the case of LINE-MOD means both 2D and 3D gradient information[1]. If only a single modality is used for the classification, the false positive rate is high. However, the combination of the two proves effective, bringing down the false positive rate drastically.

2.1.1 *Notation*

2.1.1.1 *Pixels and Points*

In this thesis we wil refer to the input image as $\boldsymbol{X}$, $\boldsymbol{X}$ contains 2D points (pixel coordinates) $\boldsymbol{x} = (x, y)$, such that $\boldsymbol{x} \in \boldsymbol{X}$, and $\boldsymbol{x} \in \mathbb{R}^2$ if $\boldsymbol{X}$ is a 2D image. Coordinates in three dimensions (points), are contained in a pointcloud $\boldsymbol{P}$, and are denoted by $\boldsymbol{p} = (x, y, z)$, such that $\boldsymbol{p} \in \boldsymbol{P}$, and $\boldsymbol{p} \in \mathbb{R}^3$.

1 A gradient in a RGB image is related to the intensity change of a Color or Gray-scale channel in the x and y direction, producing a vector in the direction of the greatest intensity change

2.1.1.2 *Objects and Templates*

The Object Detector implemented on AMIGO, concerns itself with detecting an object from a series of input frames. The Object Detector uses multiple inputs streams, called the *RGB* (2D) and *Depth* (3D) frames, to detect and report objects. The input of the Object Detector are these input frames, as well as a number of known objects $\boldsymbol{O}$ that are to be found in that frame. The output of the Object Detector is a series of binary detections, detailing which objects have been found and how similar, denoted by $\mathcal{E}$, these objects are when compared to their model.

The detection modules works with *templates* that are combined into an *Object Model*. A template is a set of features:

$$T = (F_0, \ldots, F_n) \tag{1}$$
$$F_i = (d, \boldsymbol{r}) \tag{2}$$

Where each feature F is a binary feature descriptor d and a 2D position $\boldsymbol{r}$. These positions are relative to the bounding box that defines the template size. The feature descriptor d is binary descriptor that can be captured in a single *byte* i.e $d = \{0, 0, 0, 0, 0, 0, 0, 0\}$. An Object Model is a collection of n templates for each modality m:

$$\boldsymbol{O} = \begin{pmatrix} \boldsymbol{T}^0 \\ \vdots \\ \boldsymbol{T}^m \end{pmatrix} \tag{3}$$

$$\boldsymbol{O} = \begin{pmatrix} T_0^0 & \cdots & T_n^0 \\ \vdots & \ddots & \vdots \\ T_0^m & \cdots & T_n^m \end{pmatrix} \tag{4}$$

Note that original paper by Hinterstoisser *et al.* [1], has no concept of Object Models, only of templates. The extension of templates into an object model is straightforward, and necessary if we want to consider the detection of an object in its entirety.

2.1.2 *Similarity calculation*

During the detection phase, the system loads these models from disk and proceeds to detect these objects in a scene by evaluating the similarity function. For each template $T_i \in \boldsymbol{O}$ and $\boldsymbol{x}$, where $\boldsymbol{x}$ is a number of possible object positions, the following is evaluated:

$$\mathcal{E}(\boldsymbol{O}, \boldsymbol{x}) \tag{5}$$

which is the similarity for object $\boldsymbol{O}$ at image location $\boldsymbol{x}$.

The object detector makes use of a number of quantized input images and optimized representations, that are used for calculating the similarity. The method makes use of two quantized representations of the input images, these are listed below:

- $\mathcal{Q}$ represents the *quantized* input image.
- $\mathcal{J}$ represents the *spread* image $\mathcal{Q}$.

When a feature from either $\mathcal{Q}$ or $\mathcal{J}$, is compared to template T, this is termed a response. The sum of these responses is the object similarity as specified by Equation 5. To optimize the calculation of these responses, two intermediate representations are created that are conditioned on $i \in (0, \ldots, 7)$, which is the bit index of d:

- $\mathcal{S}_i$ represents all possible feature responses for i
- τ_i is a Lookup Table (LUT), which is used during the calculation of $\mathcal{S}_i$

Most of these notations, will not make more than an intuitive sense at this point. However, during the course of this chapter, it will become apparant how these are used and calculated.

2.2 MODULE OVERVIEW

The current detection system is visualized in Figure 2, the figure shows an overview of the core detection module.

The object detection module is divided into a learning and detection phase, similar to those described in Section 1.2.2. The part of the figure that is labeled with 'offline' is concerned with the calculation of static data. This data does not change at run-time and can be calculated once per modality. The goal of the learning phase, is to learn an object $\boldsymbol{O}$ and add this object to the database.

2.2.1 *Module pipeline*

In this section a short birds-eye overview is provided of the detection module, after which we will go into more detail for some of the steps. The goal of this section is to provide a clear overview that will simplify the understanding of the more detailed explanations that are to follow. When running the module, the first step of doing a detection is the preprocessing phase. The primary objective of the preprocessing phase is to present the features in such a way that the detection can be used for the detection of an object. The steps can be summarized as follows:

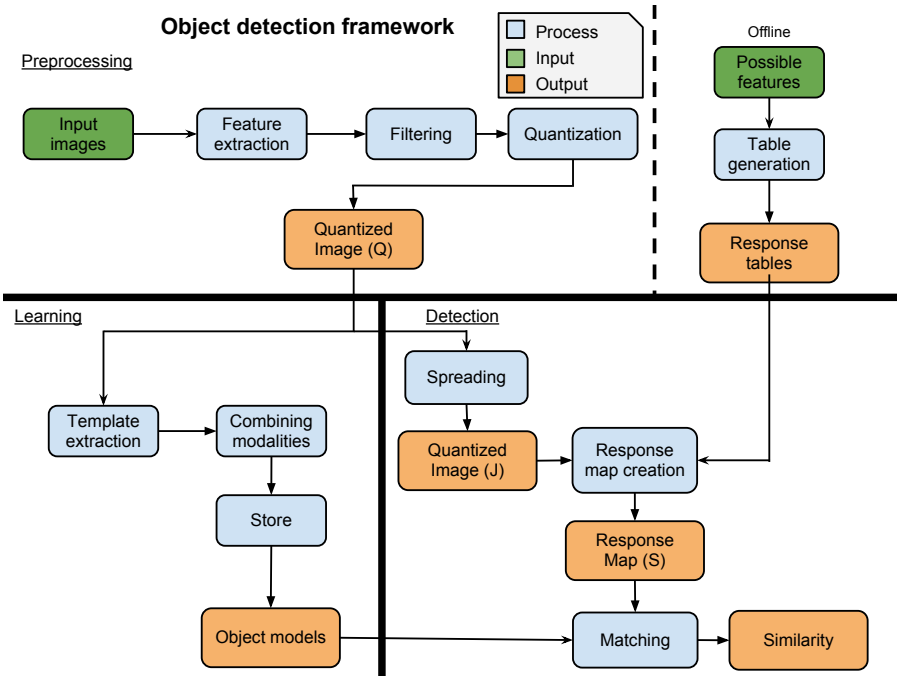


Figure 2: The current detection module. The green nodes represent the *input* used by the system, the orange nodes the processed *output*, and the blue nodes represent *processing* steps.

1. *Feature extraction*. Perform a number of low-level image processing operation on the entire input image, extracting the features F.

2. *Filtering*. Eliminate features F that can be considered noise.

3. *Dimensionality reduction*. Quantize the remaining features into the descriptor d.

4. *Pooling over a region*. Consider a neighborhood and only keep dominant descriptors d. Collect these features into $\mathcal{Q}$

Note that the preprocessing is done for each modality separately, producing multiple binary images. The preprocessing is done for both the *detection* and *learning* phases, which is related to the feature extraction step as introduced in 1.2.2.2. After the image $\mathcal{Q}$ is created, we can move on to the detection phase. During the detection phase we want to use the binary images to decide whether a specific object instance is present. The steps can be summarized as follows:

1. Spread the features in $\mathcal{Q}$ over a region, effectively *pooling* the local features.

2. Creating the Response Maps $\mathcal{S}_i$. Determining the response for each feature descriptor d, by cross-referencing an entry in the image $\mathcal{Q}$ with the precalculated Response Table τ_i.

3. Matching each template $T_i \in \boldsymbol{O}$ by concatenating responses for image locations x, where the number of locations is determined by the amount of spreading. Outputting a similarity map $\mathcal{E}$, which specifies the amount of similarity per location x, and is a discrete approximation of Equation 5.

From the Similarity map it is possible to determine the image location containing the maximum similarity and decide, with a user defined threshold, if an object has been detected.

As described, the detection phase makes use of different object templates T_i, which should be learned offline. Note, that this is not discussed in detail in the paper so a new implementation has been created. During learning the goal is to learn an object model $\boldsymbol{O}$ by collecting templates for each modality. The steps summarized below:

1. Given a binary image $\mathcal{Q}$ and a bounding box, extract a number of features from the binary image $\mathcal{Q}$.

2. Combine the templates for each modality and save this as an object model. Store the object model to disk.

3. Use detection to determine the object model in a new input image, if a detection in which the similarity is between two thresholds t_{low} and t_{high} is found then save to disk. Repeat steps 1 and 2 until enough templates have been collected to describe the object.

We will discuss the features and extraction methods in more detail in Section 2.3. Followed by a discussion of the actual detection in Section 2.4 The process of learning an object model is discussed in Section 4.3.1, because learning templates is not fully discussed in the main paper, the implementation could differ from the original, and will therefore discussed in Chapter 4.

2.3 FEATURE EXTRACTION

LINE-MOD extracts different feature types for different modalities. Because the focus of LINE-MOD is on non-textured rigid objects, the features that it uses should work well with these objects. One of the main intuitions held by the original authors is that combined modalities should outperform a single modality. Figure 3, demonstrates how these modalities can be combined.

2.3.1 *Feature representation*

Recall that the descriptor d is binary descriptor, captured in a single *byte* i.e $d = \{0,0,0,0,0,0,0,0\}$. Although it is possible to capture 256 configurations with a byte, LINE-MOD uses a single value for

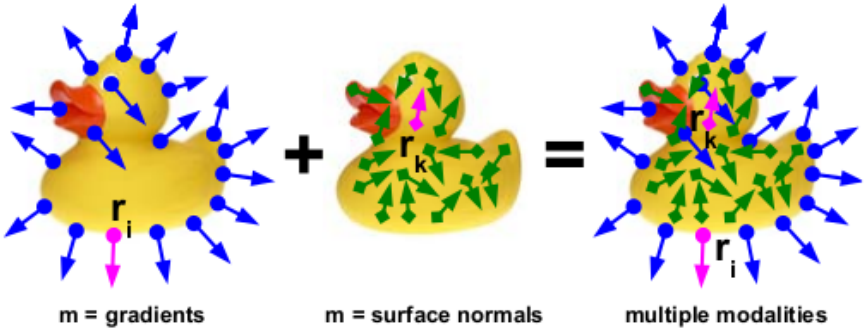


Figure 3: The combination of multiple modalities. The first two images showing both a separate 2D gradient and 3D normal feature representation. The last image on the right showing the combined modalities. *Source* [1]

each bit because of the *spreading* and the low-level optimizations. Implying a total of 8 configurations. Section 2.4 examines the matching operation done with the quantized representation, and will clarify this restriction. This can be seen as a trade-off, as this means that we are forced to *quantize* a potentially high-dimensional feature into just 8 configurations, but this enables a large speed increase.

2.3.2 *Feature types*

In the next sections the different feature types that have been used will be described as well as the routines used to extract the features, for both the RGB and Depth images.

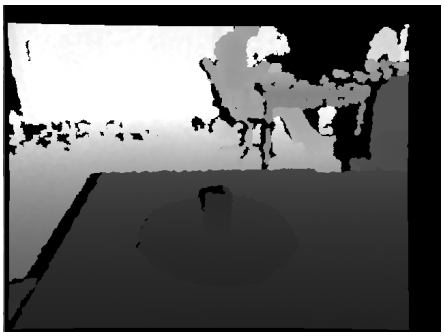
2.3.3 *2D gradient modality*

The intuition behind the color gradient modality is that it should describe the shape of the object, i.e. the *contour*. Edge features tend to describe the boundaries of the object reasonably well. Furthermore, for textureless objects other feature types like SIFT or SURF, are often unavailable as there is simply not enough information on the surface of the object. The choice for this feature type is inspired by the features presented in the Histogram of Oriented Gradients (HOG) framework [20], which have proven effective in practice, and earlier work done by the authors in [23]. In Figure 4 a typical scene is illustrated that is used during the learning of an object. The figure shows the two input images that are processed by the system.

The feature F, as introduced in 2.1.1 is the orientation of the gradient, these can be described in 2D by the following procedure:

- Given pixels positions $\boldsymbol{x} \in \boldsymbol{X}$, where $\boldsymbol{X}$ is the input image, we can define the image intensity $I(\boldsymbol{x}) = I(x, y)$ of the 2D image

(a) RGB Image



(b) Depth Image

Figure 4: A scene with a single coke can. The visualizations in this chapter are all based on these input images. (a) Shows the RGB input image. (b) Shows the Kinect depth map which ranges from $(0.0, \ldots, 5.0)$ meter but are remapped to $(0, \ldots, 255)$ intensity values to display the image.

- The partial derivatives can be calculated from the intensity $I(x, y)$:

$$\nabla I(x, y) = (\frac{\partial I}{\partial x}, \frac{\partial I}{\partial y}) \tag{6}$$

- The function $\nabla I(x, y)_C$ is approximated with a Sobel filter[2], combined with a Gaussian Blur to eliminate noise.
- This can be done for the grayscale intensity $I(x, y)$ or the calculation can be done per color channel c:

$$\nabla I(x, y)_C = (\frac{\partial I_C}{\partial x}, \frac{\partial I_C}{\partial y}) \quad \text{where} \quad c \in \{R, G, B\} \tag{7}$$

  which means that we should apply the sobel filter per channel C.
- Let $\boldsymbol{G}_C$ denote the vector output evaluated at a specific coordinate $\boldsymbol{x}$ for color channel C:

$$\nabla I(\boldsymbol{x})_C = \boldsymbol{G}_C = \left[\frac{\partial I_c}{\partial x}, \frac{\partial I_c}{\partial y}\right]^T \tag{8}$$

  these can be calculated for all $\boldsymbol{x}$
- After which the largest gradient over all color channels is selected per $\boldsymbol{x}$:

$$||\boldsymbol{G}_C|| = \sqrt{G_{Cx}^2 + G_{Cy}^2} \tag{9}$$

$$\boldsymbol{G} = \arg\max_{\boldsymbol{G}_C}(||\boldsymbol{G}_C||) \quad \text{for} \quad C \in \{R, G, B\} \tag{10}$$

2 A Sobel filter is a convolution with a central difference kernel, which calculates numerical derivatives in the x and y directions, enabling an easy calculation of the gradient

The calculation of derivatives on multiple independent color channels C, has been done before for HOG [20] and [19], and seems to enhance discriminative power [1]. The reason that a Sobel filter is used as opposed to a more sophisticated edge detector, e.g. a canny detector, is probably because of the high speed of this filter. To transform $\boldsymbol{G}$ into the quantized descriptor d, and quantize into the appropriate value the following transformations are used:

- Convert the gradient into an angle and calculate the index:

$$\theta = atan(\frac{\boldsymbol{G}_y}{\boldsymbol{G}_x}) \tag{11}$$

$$index = \theta * (16/2\pi) \tag{12}$$

- Only use the indexes that should not be considered noise we make use of the gradient intensity $||\boldsymbol{G}||$ to filter out noise and non-descriptive gradients.
- To map the index to the descriptor d, assign the $index$ to the correct bin and possibly truncate the value into 8 bins:

$$d = 1 \ll (index \ \& \ 7) \tag{13}$$

- This is equivalent to binning the orientation into equally sized bins. The symmetric gradients which have a negative y component are mapped to positive y values, as has been visualized in Figure 5.
- After the quantization we employ a noise reduction step. The non-dominant gradient orientations are filtered, by sliding a small window over the image and filtering out any gradients that occurs less than 5 times in the neighborhood.
- Once the gradient suppression step is completed the binary image $\mathcal{Q}$ is created by assigning $\mathcal{Q}(\boldsymbol{x}) := d$, for all $\boldsymbol{x} \in \boldsymbol{X}$ that passed the filtering steps.

The binary image $\mathcal{Q}$, is further utilized during the detection phase.

The color gradient extraction routine only makes use of Figure 4a. Figure 6, shows the processed gradients, Figure 6a, as well as the intensity image, Figure 6b , as is returned by the Sobel filter. The intensity image is used for the filtering step. Note that the colors in the image correspond to those in Figure 5, also note that symmetric gradients are mapped into their corresponding counterparts.

2.3.4 *3D gradient modality*

The intuition behind the depth gradient modality, is that it should describe the *shape* of the object. 3D gradients or normal vectors give some rough description of the surface of the object and can be easily utilized in the LINE-MOD framework, as will be seen in 2.4. The

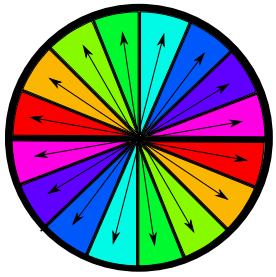
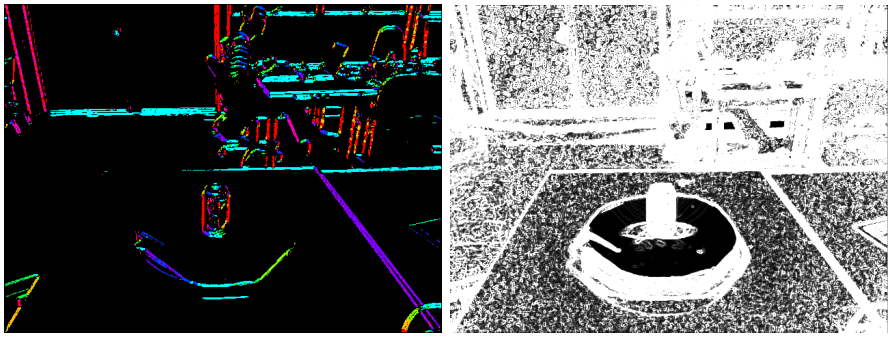


Figure 5: The quantized 2D color gradients, the negative y gradients and mapped to the (mirrored) positive y bins. Each gradient is mapped to a bin of 22.5 degrees or $\frac{\pi}{8}$. The colors represent the quantized gradients as they are visualized by the system, note the mirroring of the colors.



(a) Binary Image (b) Intensity Image

Figure 6: The processed scene, as seen in Figure 4.

depth map as shown in Figure 4b is used as an input. Starting with the depth map we will have to reconstruct a point cloud that is located in the Camera Frame. For each pixel coordinate $\boldsymbol{x}$, and the depth map $D(\boldsymbol{x})$ (provided by the Kinect), which returns the depth at coordinate $\boldsymbol{x}$:

- The point $\boldsymbol{p} \in \mathbb{R}^3$ in the Camera Frame can be reconstructed, by first evaluating $D(\boldsymbol{x})$ and subsequently constructing the point $\boldsymbol{p}_{im} = \begin{bmatrix} x, & y, & D(x,y) \end{bmatrix}^T$.
- These reconstructed values are located in the Image Frame, to reconstruct the original pointcloud $\boldsymbol{P}$ we transform each point $\boldsymbol{p}_{im} \in \boldsymbol{P}_{im}$ into the Camera Frame:

$$\boldsymbol{P} = \boldsymbol{K}^{-1}\boldsymbol{P}_{im} \tag{14}$$

  using the Camera Matrix $\boldsymbol{K}$
- Because the Infrared projector and Infrared camera on the Kinect are translated by a few centimeters, the images will have to be aligned before doing this transformation. Fortunately, the calibration of the Kinect is done during the production of the

camera, this entails that the provided camera matrix $\boldsymbol{K}$ and the translation between the camera's is known[3]

Now that we have obtained $\boldsymbol{P}$ we can continue with the extraction of the 3D gradients, which will also be referred to as *normals*, because the definition is equivalent.

2.3.4.1 *Normal approximation*

After the point cloud is acquired, we can proceed with the extraction of gradients from the point cloud, the normal extraction can be done in a number different ways. Current literature has not yet yielded an accurate, robust and fast method for the Kinect. The problem is the large amount of noise on in the depth values, the values become more inaccurate as the depth increases, and that the measurements are incoherent at object edges.

The approach presented in LINE-MOD is based on a least-squares approximation. An obvious approach would be to use the fact that a *normal* can be used to define a *plane* for two neighboring points $(\boldsymbol{p}_1, \boldsymbol{p}_2) \in \boldsymbol{P}$, i.e:

$$\boldsymbol{v} = \boldsymbol{p_2} - \boldsymbol{p_1} \tag{15}$$

$$\langle \boldsymbol{n}, \boldsymbol{v} \rangle = 0 \tag{16}$$

Using this intuition we can create a neighborhood of points that defines the normal using a system of equations. Unfortunately, this will not work, as the outcome of such a system would have to be $\boldsymbol{An} = 0$, where $\boldsymbol{A}$ is a set of vectors on the plane $\boldsymbol{p}$. Obviously the normal vector should *not* be the null vector. Thus, we know that if the null space of the system contains more than the null vector, then we are dealing with a linear dependent system. This linear dependency is obvious, because we are trying to approximate the normal that is orthogonal to a plane. So there is no least squares solution by applying the normal equations naively. However, another approach is to look directly at the depth function $D(\boldsymbol{x}) = z$. Using this function it is possible to make the following first-order Taylor approximation [1]:

$$\begin{aligned} D(\boldsymbol{x} + \boldsymbol{dx}) &= D(\boldsymbol{x}) + \boldsymbol{dx}^T \nabla D \\ D(\boldsymbol{x} + \boldsymbol{dx}) - D(\boldsymbol{x}) &= \boldsymbol{dx}^T \nabla D \end{aligned} \tag{17}$$

Where $\boldsymbol{x}$ are the (x, y) image coordinates. Now we can create a neighborhood of points from $\boldsymbol{P}$, denoted as $\boldsymbol{A}$ to approximate the gradient ∇D:

3 The details can be found in a number of places, e.g. http://pille.iwr.uni-heidelberg.de/~kinect01/doc/index.html

- Formulate the problem as a least-squares optimization problem, by approximating the new vector of points $\boldsymbol{y}$ with ∇D:

$$\boldsymbol{A}\nabla D = \boldsymbol{y} \tag{18}$$

- Take the inverse to acquire ∇D:

$$\nabla D = \boldsymbol{A}^{-1}\boldsymbol{y} \tag{19}$$

- Rewrite $\boldsymbol{A} = \boldsymbol{X}$ and apply the normal equations:

$$(\boldsymbol{X}^T\boldsymbol{X})\nabla D = \boldsymbol{X}^T\boldsymbol{y} \tag{20}$$
$$\nabla D = (\boldsymbol{X}^T\boldsymbol{X})^{-1}\boldsymbol{X}^T Y \tag{21}$$

- Calculate $(\boldsymbol{X}^T\boldsymbol{X})^{-1}$ using Cramer's Rule, obtaining ∇D which can be considered the gradient at that point.

Smoothing is done automatically, because we are considering a neighborhood of points.

2.3.4.2 *Quantization*

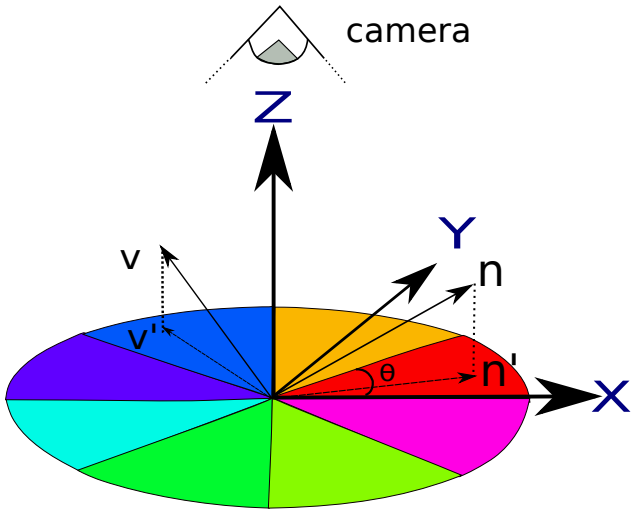


Figure 7: Quantization of the 3D gradients, the two gradients n, v are mapped onto the XY plane, the mappings are denoted as n', v'. The colored bins represent the quantization of range of angles (θ from $(0, \ldots, 2\pi)$) into the XY plane, each descriptor is associated with a unique angle.

The quantization for the 3D modality is similar to the 2D case. However, in 3D we have an extra degree of freedom, meaning that there are 2 angles to consider to parameterize a vector. The quantization is visualized in Figure 7. We only take the half of the sphere into account that faces the camera. Because the point cloud is created with the Kinect the normals should always face the viewer, hence they are located in the $+z$ region of the space in Figure 7. The calculation of

the *index* of d is similar to Equation 12, except that the 8 orientations are mapped over the range of 2π instead of π. The quantization can be seen as mapping a normal from 3D to a 2D plane, this plane is split up into parts defined by $\frac{2\pi}{8}$ angular parts, the angle θ is defined on this plane. The mapping is done on the XY plane because of the three possible planes it is the most descriptive, covering the largest space that can be viewed from the camera.

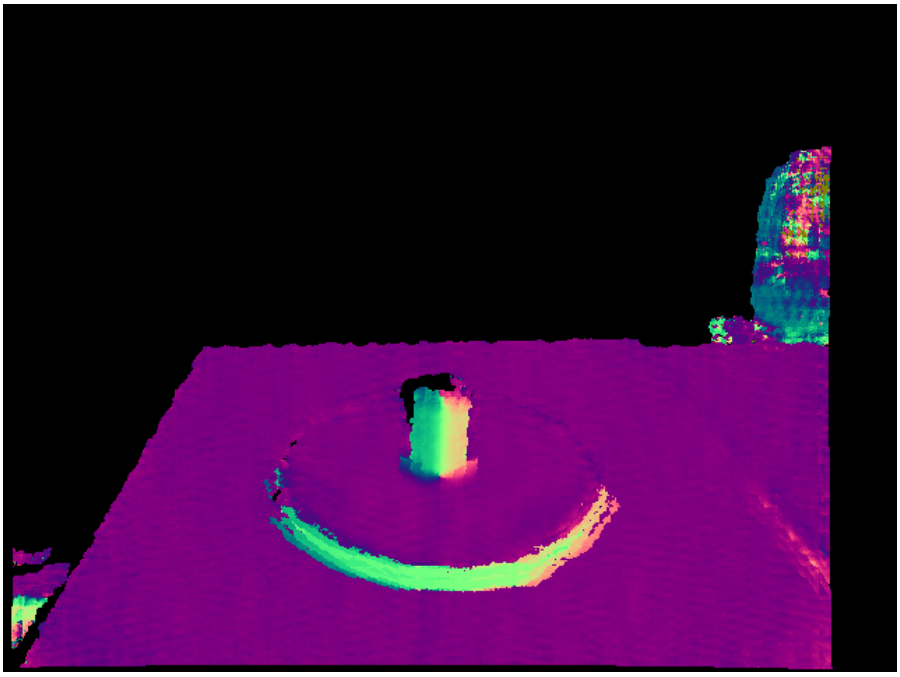


Figure 8: The normal approximation method based on least squares.

After the quantization operation the gradient is mapped onto the descriptor: $d = 1 \ll index$, and is combined into the image $\mathcal{Q}$. Note that a filtering step cannot be applied in 3D, because there is no such measure as gradient intensity in 3D, as every gradient $||G|| = 1$, corresponding to a unit normal vector. This entails that $\mathcal{Q}$ will be more dense in 3D. However, we can still choose a dominant orientation, this done in exactly the same way as in the 2D case, which provides some measure of robustness. To eliminate noise, neighboring points that have a depth difference larger than 0.5 cm are ignored, as well as Not a Number (NaN) values which indicate that no depth information is available.

In Figure 8 the estimated normals are shown, where the normal x, y, z components are mapped to the respective r, g, b components. In Figure 9, the quantized values are shown for the least squares method, the color codes correspond to those introduced in Figure 7.

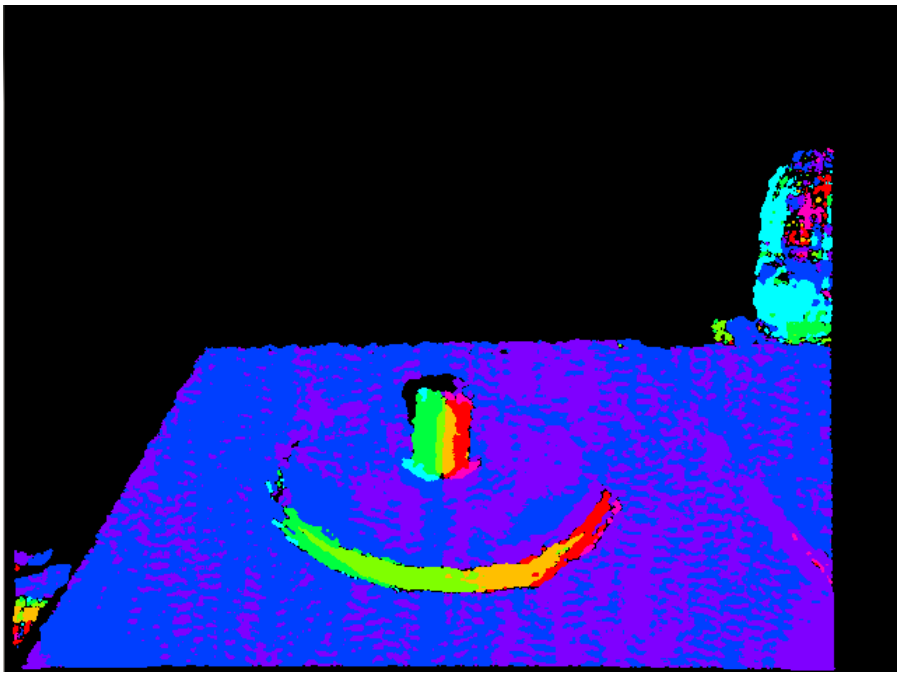


Figure 9: The quantized normals, the orientations have been mapped into 8 bins.

2.4 DETECTING AN OBJECT

After we have extracted the features into the binary images $\mathcal{Q}$, the next step is to perform a matching over the entire input image with an object model O. Naively, one could slide each template in the model over the entire image, which is called a sliding window detector. The problem is that this is inherently slow, because to determine the exact location one would have to perform $(640 \times 480) \times features$ operations for the Kinect. Early detectors would use the Normalized Correlation Coefficient (NCC)[4] to compare pixel values, making optimization hard. However, modern detectors often make use of a sparse template representation to do the actual detection, enabling different optimization options. The latter is the approach taken in the current system.

2.4.1 *Similarity measure*

To perform a detection, we will have to specify a measure of similarity $\mathcal{E}(\boldsymbol{O}, \boldsymbol{x})$, for an object $\boldsymbol{O}$ on an image location $\boldsymbol{x}$. The similarity measure will have to be defined for each modality so that a similarity can calculated:

- We want to determine a similarity per modality m, implying that we need a definition for $\mathcal{E}_m(\boldsymbol{O}, \boldsymbol{x})$

4 See http://tev.fbk.eu/TM/html/tmCodeCompanionse10.html

- Having this definition the individual similarities can be summed per location:

$$\mathcal{E}(\boldsymbol{O}, \boldsymbol{x}) = \sum^{m} \mathcal{E}_m(\boldsymbol{O}, \boldsymbol{x}) \tag{22}$$

- To find the maximimum similarity at this location, only the template T is considered that maximizes the similarity at that location:

$$\arg\max_{T_i^m} \mathcal{E}_m(T_i^m, \boldsymbol{x}) \tag{23}$$

- Allowing us to rewrite Equation 22:

$$\mathcal{E}(\boldsymbol{O}, \boldsymbol{x}) = \sum^{m} \arg\max_{T_i^m} \mathcal{E}_m(T_i^m, \boldsymbol{x}) \tag{24}$$

Equation 22 states that the final similarity for a location x, is the combined similarity over different modalities for the maximum template argument. These combined similarities are represented in a map of similarities $\mathcal{E}$, which is a discrete representation of Equation 22. To be able to use a feature in the object detection framework, the similarity measure $\mathcal{E}_m$ must be defined for all possible values of the descriptor d.

2.4.1.1 *Gradient similarity measure*

The gradient similarity measure currently being used was provided in a paper by Steger [46]. In this paper, Steger provides measure for comparing edge orientations between a 2D gradient found in the image $\boldsymbol{G}^{im}$ and the template $\boldsymbol{G}^{t}$:

$$\mathcal{E}_{2d}(\boldsymbol{G}^t, \boldsymbol{G}^{im}) = \frac{\langle \boldsymbol{G}^t, \boldsymbol{G}^{im} \rangle}{\|\boldsymbol{G}^t\| \|\boldsymbol{G}^{im}\|} \tag{25}$$

Which can be viewed as the $cos(\theta)$, where θ is the angle between the two gradient vectors $\boldsymbol{G}_i^t$ and $\boldsymbol{G}^{im}$. The measure returns a similarity of 1 if gradients are *tangent* and 0 if they are *orthogonal*. So for a single template:

$$\mathcal{E}_{2d}(T_i^m, \boldsymbol{x}) = \frac{1}{N} \sum_{(\boldsymbol{r},d) \in T_i^m} \mathcal{E}_{2d}(\boldsymbol{G}^t, \boldsymbol{G}^{im}) \tag{26}$$

For the N number of features in a template. In our specific case we can retrieve $\boldsymbol{G}'$ which approximates the original gradient $\boldsymbol{G}$ using the descriptor d:

$$\theta = bin\_index(d) * (\frac{\pi}{8}) \tag{27}$$

$$\boldsymbol{G}' = \begin{bmatrix} cos(\theta) \\ sin(\theta) \end{bmatrix} \tag{28}$$

$\boldsymbol{G}'$ can be approximated with a descriptor d taken from either the feature $d \in F_i^m \in \boldsymbol{T}_i^m$ or from the quantized image at $\mathcal{Q}(\boldsymbol{x}) = d$. Steger notes that this measure is robust against local illumination changes and occlusion. Hinterstoisser, extended the measure to 3D by assuming that the similarity holds for the 3D case as well. Note that the π in Equation 27, has to be replaced with 2π in the 3D case. The advantage of the high discretization of the gradient angle becomes apparent, as we can calculate the similarities from Equation 26 offline, this is discussed in the Section 2.4.2.3.

2.4.2 *Modified similarity measure*

In [1], the authors observe that while the similarity measure introduced by Steger is robust to local illumination changes and occlusion it does not respond well to small translations. Which is why a new measure is introduced, based on their prior work [23]. It is derived in the following manner:

- Given a region R of the quantized image where R:

$$R(\boldsymbol{x}) = \begin{bmatrix} x - \mathcal{T}/2, & y - \mathcal{T}/2 \end{bmatrix} \times \begin{bmatrix} x + \mathcal{T}/2, & y + \mathcal{T}/2 \end{bmatrix} \tag{29}$$

- Where $\mathcal{Q}^{\mathcal{T}\times\mathcal{T}} = \{d \in \mathcal{Q}(x,y) | (x,y) \in R(\boldsymbol{x})\}$ for which we define the following similarity based on Equation 25:

$$\mathcal{E}_{2d}(\boldsymbol{G}^t, \boldsymbol{G}^{im}) = \max_{\boldsymbol{G}^{im} \in \mathcal{Q}^{\mathcal{T}\times\mathcal{T}}} (\frac{\langle \boldsymbol{G}^t, \boldsymbol{G}^{im} \rangle}{\|\boldsymbol{G}^t\| \|\boldsymbol{G}^{im}\|}) \tag{30}$$

  Which finds the maximum for the similarity function in the specified region. Note that for ease of notation d has been converted back into $\boldsymbol{G}^{im}$ according to Equation 27 and Equation 28.

By using a region for the modified similarity the number of computations that need to be done is increased. Because of this drawback, another approach was presented by Hinterstoisser to calculate the similarity that is considerably faster, this optimization is described in the next section.

2.4.2.1 *Spreading*

By spreading the descriptor in multiple directions the similarity from the previous section can be calculated more efficiently. To spread a

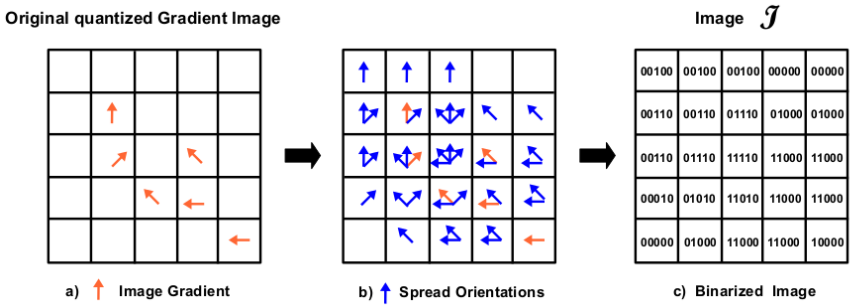


Figure 10: The features are spread into the final image $\mathcal{J}$, by a binary 'OR' operation. In this case, $\mathcal{T} = 2$ is used. *Source* [1].

descriptor, which is a single activated bit in a byte, it is appended into a *final* descriptor with a binary 'OR' operation. This explains the fact that only 8 descriptor values can be used, as only 255 unique combinations are contained in a byte. The process of spreading the features is visualized in Figure 10. The resulting input image $\mathcal{J}$ is the final binary image that is used for the actual matching process. Figure 11, depicts different $\mathcal{J}$ images for different modalities. The colors in the image correspond to their non-spread counterparts. The amount of white of determines the amount of descriptors at that location. A completely white pixel indicates a descriptor value of 255, this entails that a completely white image, would have a maximum similarity for each value of d.

2.4.2.2 *Response maps and Memory linearization*

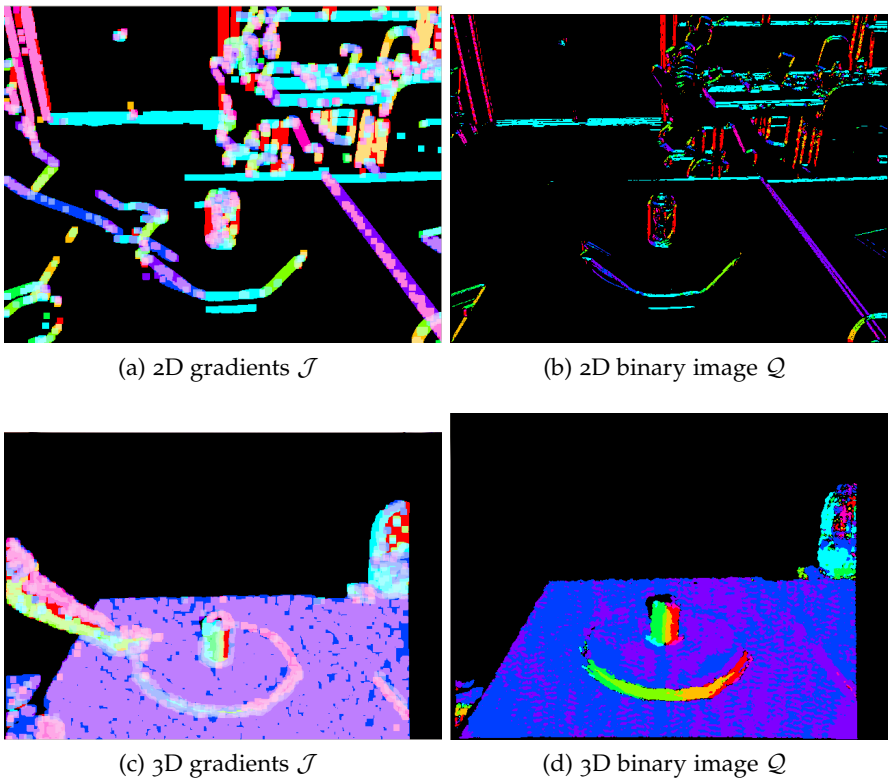
Before the detection and linearization will be discussed, the concept of a Response Map $\mathcal{S}$ will be explained. A Response Map is a map that contains the feature responses (Equation 25), which have been discussed in the previous sections. To speed up the creation of $\mathcal{S}$ we can create the lookup table τ. These contain feature responses that can be calculated offline. Because we know that for each modality there are 8 features, and there are a maximum of 255 unique feature descriptors to be found in the binary image $\mathcal{J}$. It is possible to construct a lookup table τ per feature type (which are 8 lookup tables per modality) offline.

The index of this lookup table τ is the integer value of $\mathcal{J}(\boldsymbol{x})$. The associated value is the maximum similarity for the given feature type in the region $\mathcal{T} \times \mathcal{T}$, this evaluates the Equation 30 efficiently.

To construct $\mathcal{S}$, we can simply evaluate τ at each location of $\mathcal{J}$ for each feature type orientation i:

$$\mathcal{S}_i(\boldsymbol{x}) = \tau_i(\mathcal{J}(\boldsymbol{x})) \tag{31}$$

Figure 12, shows the creation of the lookup table and the associated Response Map. The Response Map can be seen as single lookup table τ_i back projected onto the input image $\mathcal{J}$.



(a) 2D gradients $\mathcal{J}$ (b) 2D binary image $\mathcal{Q}$

(c) 3D gradients $\mathcal{J}$ (d) 3D binary image $\mathcal{Q}$

Figure 11: Two binary images $\mathcal{J}$, are shown. (a) Shows the quantized and spread 2D gradients. (c) Shows the quantized and spread 3D gradients. (b), (d) The unspread counterparts are shown as a reference.

Using these Response Maps, it is possible to calculate the similarity for a template T_i^m on a given location $\boldsymbol{x}$ in the input image, by summing each corresponding response from the correct Response Map:

$$\mathcal{E}_{2d}(T_i^m, \boldsymbol{x}) = \sum_{(\boldsymbol{r},d)\in T_i^m} \mathcal{S}_d(\boldsymbol{x}+\boldsymbol{r}) \tag{32}$$

The use of $\mathcal{S}$ and τ allows us to precalculate the values, and removes the need to evaluate Equations (30, 28, 27) for each template.

LINEARIZATION OF THE RESPONSE MAP The goal of the linearization is to make as much of the data accessible in a single line of memory, which essentially eliminates a large number of cache misses and enables the use of SSE instructions (see the Appendix A for a tutorial). To achieve this, we can make use of the Response Maps that we have created. Normally, one would slide the template over the image, and compare each feature in the template to the features in the input image at location $\boldsymbol{x} + offset$, where the offset is defined by the location of the feature F_i in the template. However, this method is unfeasible because the features F_i need not be adjacent in memory.

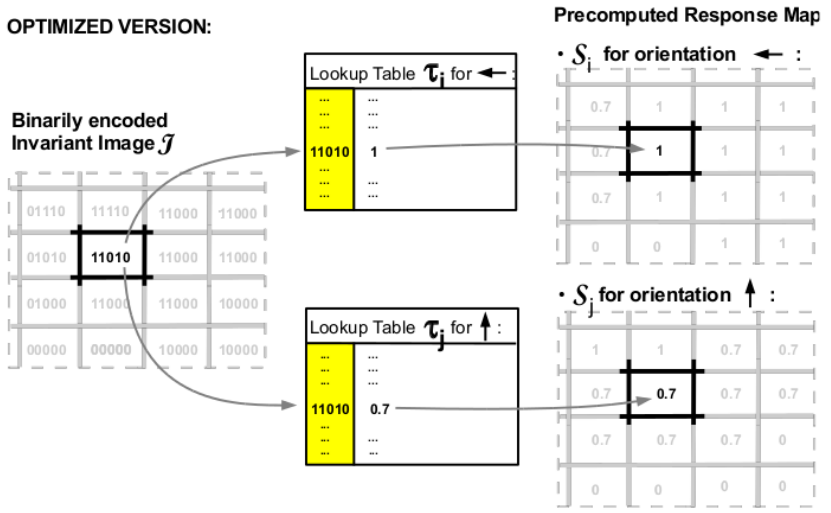


Figure 12: Two lookup tables τ_i and τ_j are shown with the associated Response Maps $\mathcal{S}_i$ and $\mathcal{S}_j$. The binary descriptor is used as an index, and the response (maximum similarity in a region) is stored. The integers i and j denote the feature index. *Source* [1].

Therefore, will have to *transform* the representation of the Response Map, so that we can make this assumption. Because of the spreading procedure $\mathcal{T}$, we will only have to perform a detection at each offset $\mathcal{T}$. Knowing this we can split up the Response Map into $\mathcal{T} \times \mathcal{T}$ linear memories, this process is visualized in Figure 13. To represent these linear memories internally we can employ a two-dimensional array.

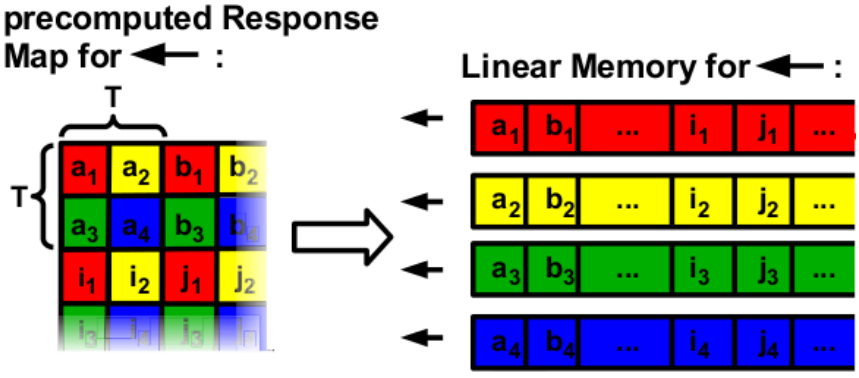


Figure 13: The Response Map $\mathcal{S}_i$ is linearized into $\mathcal{T} \times \mathcal{T}$ linear memories. The linear memories are a collection of one dimensional memory lines which span the rows and the columns of the Response Map. In this case $T := 2$, has been chosen. *Source* [1].

2.4.2.3 *Matching*

The advantage of these linear memories is that the templates can be processed in such a way that we will only have to read a single mem-

ory line per feature, because we have now organized the Response Map by the spreading parameter $\mathcal{T}$. To utilize this we use a different matching scheme as opposed to regular template matching. In our case, we reverse the process and first loop over the features F_i and then the image locations $\boldsymbol{x}$ instead. Listing 1, defines this procedure.

```
for F_i in T_i^m do
begin
  idxMemory := (row_size mod T^2) + r_x mod T;
  offset := (r_y / T) * (im_width/T) + (r_x / T);

  for i:=0 to image_area/T^2 do
  begin
    E[i] := memory[idxMemory][offset + i]
  end
end
```

Listing 1: The matching procedure

The index of the linear memory to be used with F_i, depends on the offset of this feature in the template $\boldsymbol{r} = (r_x, r_y)$. The offset is needed because the feature offset can cross the boundaries created by the spreading parameter T, in which case we should discard the first part of the linear memory. Note that the second loop that iterates over the image locations, is executed by SSE instructions, this enables the parallel processing of 16 values.

Figure 14: The Similarity Image $\mathcal{E}$ visualized. The amount of white is dependent on the amount of similarity at that location. The position in the similarity map is anchored to the top left of the template bounding box.

2.4.3 *Similarity*

This process produces a similarity image $\mathcal{E}$. However, to combine modalities the similarity maps are appended per modality $\mathcal{E}_m$. To be able to do this quickly SSE instructions are used, which processes the additions in parallel. This results in a final similarity image which could contain the detected object. If multiple instances can be found, non-maxima suppression must have to be performed to eliminate false positives. The detection is obtained by finding the maximum (or maxima if regarding multiple instances) and checking if this higher than a user defined threshold. If this is the case we consider the location $(x * \mathcal{T}, y * \mathcal{T})$, where x and y are the locations of the maximum in the similarity image, as a possible instance of the object $\boldsymbol{O}$. The Similarity Image $\mathcal{E}$, is visualized in Figure 14. The original implementation uses a user defined threshold, which considers a maximum a detection, when it exceeds this threshold.

3

METHOD ADDITIONS

3.1 INTRODUCTION

During the evaluation of the implementation of the original method, we noticed a few issues concerning the accuracy of the method. The problem is the large number of false positives that can be found. This is especially apparent for objects of similar shape, this mainly has to do with the quantization of the gradient descriptors and the loss of precision, a problem also touched upon in the original paper. This problem will be further elaborated in Chapter 5. Unfortunately, some objects in household and RoboCup settings are quite similar in shape. Which is why a few additions and changes have been made to the original system to increase the discriminative power of the method. These are summarized below:

- Other normal extraction routines are integrated, to see if another extraction scheme would yield discriminative power or a speed increase. As well as changing the selection criteria for the features.
- Addition of another modality, i.e. a color modality to the current system. Two descriptors have been implemented: a *hue* based descriptor and a descriptor based on the concept of color names introduced by Weijer *et al* [45].
- The original method is limited to 64 features because of the representation of a feature by a byte. An extension is proposed which increases this limit, but retains the high performant matching scheme.
- The addition of a modality slows down the system. Because of this a straightforward optimization is proposed by parallelizing the processing of the input images.

3.2 3D NORMAL EXTRACTION

To determine if other quantization schemes increase discriminative power of the detection method, two different extraction schemes have

been implemented to determine if these yields either an performance or speed increase. These two methods have been implemented next to the original estimation method based on least-squares inspired by the original paper. The first is a simpler albeit faster extraction routine than the original, that has been inspired by Kinect Fusion [29]. This method uses regular cross products, which are also often used in graphics programming, and can thus also be easily outsourced to the Graphical Processing Unit if needed. The second is a method based on the Principle Component Analysis (PCA) [47, 48] of a neighborhood of points, a natural candidate for dense point clouds, discussed in the thesis by Rasu [49].

3.2.1 *Cross products*

A simple but widely used technique in computer graphics to calculate surface normals is using the fact that the cross product of two vectors in plane yield an orthogonal vector. This operation is then subsequently performed on a 2D surface mesh using neighboring vertices. To perform this operation on a point cloud we can use a neighborhood of points:

- Let $\{\bar{\boldsymbol{p}}, \boldsymbol{p}_1, \boldsymbol{p}_2\} \in \boldsymbol{P}^{k\times k}$ be three neighboring points taken from a neighborhood of points $\boldsymbol{P}^{k\times k}$. The center point is denoted as $\bar{\boldsymbol{p}}$. The two neighboring points of $\bar{\boldsymbol{p}}$ are denoted as $\boldsymbol{p}_1, \boldsymbol{p}_2$.
- Let $\boldsymbol{P}^{k\times k}$ be a region containing the nearest points in the point cloud $\boldsymbol{P}$ for a query point $\boldsymbol{p}_q$.
- Because the point cloud is extracted from the Kinect, we know that the point cloud is ordered, so finding these nearest points is trivial. This reduces to $\boldsymbol{P}^{k\times k} = \{(x, y, z) \in \boldsymbol{P} | (x, y) \in R(x, y)\}$, where $R(x, y)$ is a region as defined in Equation 29 with $\mathcal{T} = k$
- The normal vector of these three points can be calculated by calculating two vectors $\boldsymbol{v}_1, \boldsymbol{v}_2$ on the plane and considering the result of the cross product between these vectors:

$$\begin{aligned} \boldsymbol{v}_1 &= \boldsymbol{p}_1 - \bar{\boldsymbol{p}} \\ \boldsymbol{v}_2 &= \boldsymbol{p}_2 - \bar{\boldsymbol{p}} \\ \boldsymbol{n} &= \boldsymbol{v}_1 \times \boldsymbol{v}_2 \end{aligned} \tag{33}$$

- To make the method more robust to noise, the normals are summed and normalized in a small neighborhood of k normal vectors, calculated for all $\{\bar{\boldsymbol{p}}, \boldsymbol{p}_1, \boldsymbol{p}_2\}$ in the region $\boldsymbol{P}^{k\times k}$:

$$\boldsymbol{n}' = \sum_{i=0}^{k} \boldsymbol{n}_i \tag{34}$$

$$\hat{\boldsymbol{n}} = \frac{\boldsymbol{n}'}{||\boldsymbol{n}'||} \tag{35}$$

Where $\hat{\boldsymbol{n}}$ is the final normalized vector. Intuitively each normal $\boldsymbol{n}_i$, is calculated by applying Equation 33 in a $k = (3 \times 3)$ region, using $\bar{\boldsymbol{p}}$ as the region centroid.

Because the same similarity measure, which has been given in Equation 26 will hold, the same quantization scheme (Section 2.3.4) can be used for both new methods. Note that for all these methods, points are discarded that have a depth difference of larger than 0.5 cm, as to eliminate noise and create robustness around edges. Depth values where no information is available, typically identified in the depth map as Not a Number (NaN) are also ignored.

3.2.2 *Principle Component Analysis*

The second extraction scheme is based on the PCA analysis of a neighborhood of points. This transformation is more expensive, than the previously described methods. However, according to Rasu [49] it provides an accurate estimation method for point clouds, which made at an interesting candidate for inclusion in the framework. Essentially, the PCA projects the data into a new basis, this basis can be used to define the normal vector.

- A PCA [47, 48] transformation on a point cloud region $\boldsymbol{P}^{k \times k}$, returns the following:

$$(\lambda_j, \boldsymbol{v}_j) \text{ for } j \in \{0, 1, 2\} \tag{36}$$

  which are three vector components $\boldsymbol{v}_j$ and three scalars weights λ_j.
- The components $\boldsymbol{v}$, are chosen so that the variance:

$$var(\langle \boldsymbol{v}_j^T, \boldsymbol{p} \rangle) \tag{37}$$

  is maximized between the data $\boldsymbol{p} \in \boldsymbol{P}^{k \times k}$ and the new space $(\boldsymbol{v}_0, \boldsymbol{v}_1, \boldsymbol{v}_2)$
- The value of an associated weight λ_j, indicates an ordering of components that maximize the aforementioned variance. To find these components $\boldsymbol{v}_j$ and λ_j, we can look at the eigenvectors and eigenvalues of a covariance matrix.
- For a point cloud the following covariance matrix $C^{3 \times 3}$ is created from $\boldsymbol{P}^{k \times k}$:

$$C = \frac{1}{k} \sum_{i=0}^{k} (\boldsymbol{p}_i - \boldsymbol{\mu}) \cdot (\boldsymbol{p}_i - \boldsymbol{\mu})^T \tag{38}$$

- $\boldsymbol{\mu}$ is defined as the average point in the neighborhood of the covariance matrix:

$$\boldsymbol{\mu} = \frac{1}{N} \sum_{i=0}^{k} \boldsymbol{p}_i \quad \text{where N = number of points} \tag{39}$$

- The eigenvectors and eigenvalues of C correspond to Equation 3.2.2. To calculate the described process, the built in *PCA* function of the OpenCV library [50] is used.
- To obtain the normal vector $\hat{\boldsymbol{n}}$, the component v_j is selected with the smallest corresponding weight λ_j.
- Unfortunately, the normals returned by the PCA method are not ordered consistently. They can either be facing away from the viewer or towards it. However, knowing that the data has been captured by a Kinect depth sensor, we are able to assume that each normal should be located in the half sphere facing the viewer.
- Using this knowledge and if we know the vector from the object to the viewer $\boldsymbol{v}_p$, which is the z basis vector in the Camera Space, we can calculate:

$$\boldsymbol{v} = \boldsymbol{v}_p - \bar{\boldsymbol{p}} \tag{40}$$

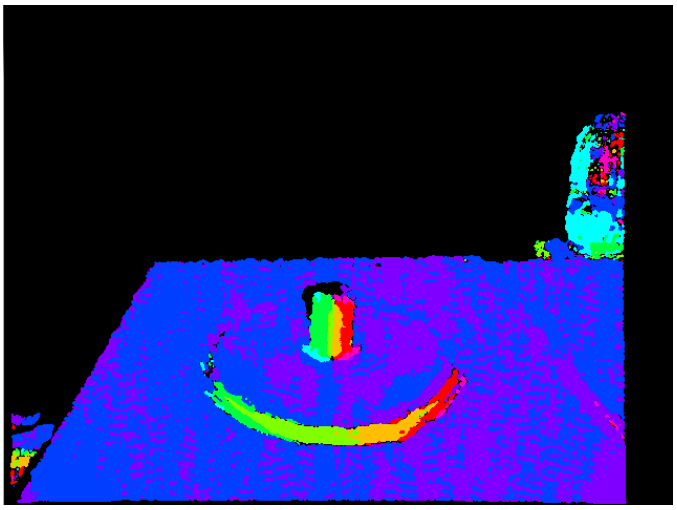
  The normal has to satisfy the equation:

$$\langle \boldsymbol{v}, \hat{\boldsymbol{n}} \rangle > 0 \tag{41}$$

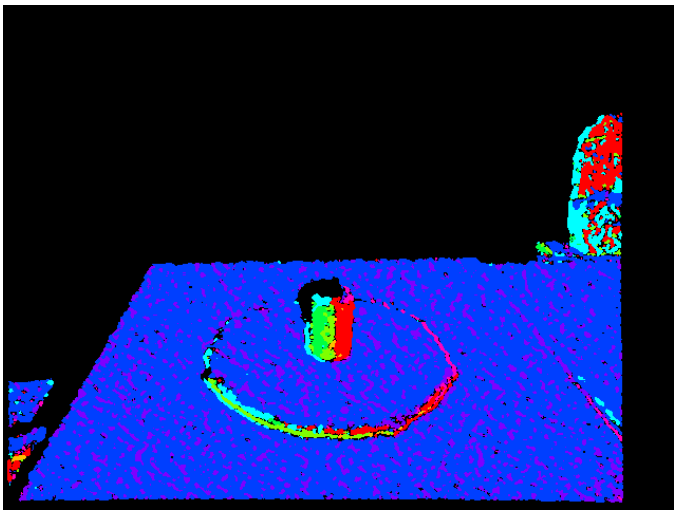
  the inner product will be less than zero if the normal is oriented away from the viewer, in that case the normal can be flipped by changing it sign.

Taking the principle component with lowest *variance* my seem unintuitive, as this is contrary to the component that is normally used in data analysis. However it is actually quite logical, because by doing the PCA transformation we are obtaining the vectors $\boldsymbol{v}_j$ that correspond to the *tangent*, *bitangent* and *normal* of the neighborhood of points. The normal is used in the plane equation to constrain the points, meaning that the points belonging to the *same* plane will share a similar normal vector, so the *variance* in the direction of the normal will be close to zero. Theoretically, the variance would even be zero for a plane, however, this is not possible because of noise and outliers in the point cloud data.

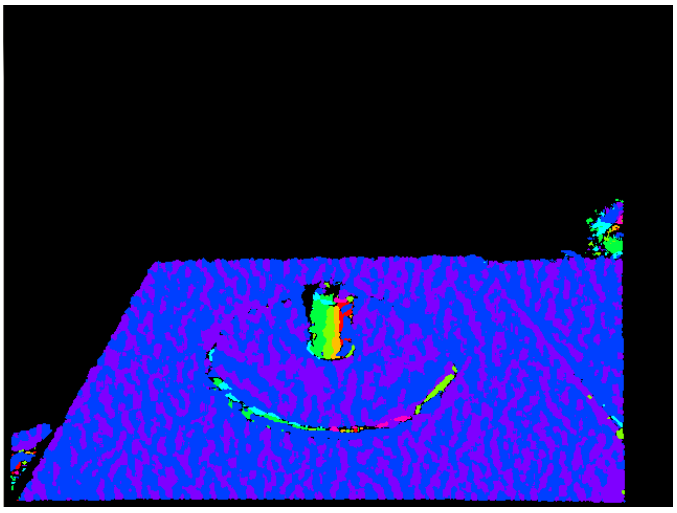
Figure 15, shows multiple normal estimation methods and the corresponding quantized features.



(a) Least Squares method



(b) Cross method



(c) PCA

Figure 15: The quantized image for the three normal approximation schemes as visualized in Figure 8.

3.3 COLOR FEATURE

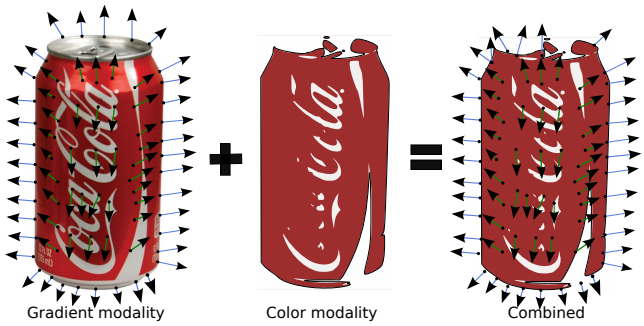


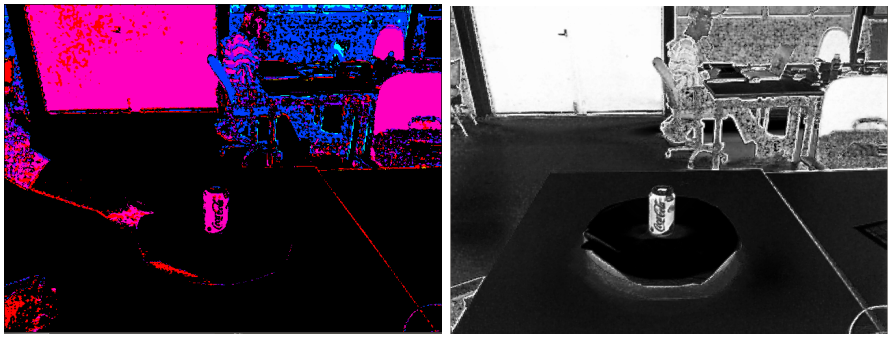
Figure 16: The combination of a color modality with the shape based descriptors.

The original paper limited the features to 2D and 3D gradients, representing the change in intensity and depth respectively. With some small adaptations we can regard LINE-MOD as a system that *transforms* a feature description into an intermediate representation. This intermediate representation, the image $\mathcal{Q}$, can then be used to perform a generic matching routine. This way we can integrate other modalities such as color or perhaps texture representations, while we only need to specify a new extraction routine.

The intuition behind the color feature that has been added to the LINE-MOD framework is that it should describe another modality of the object. As we will see during the evaluation in Chapter 5, some objects may appear similar that are not so to the human eye. One of the ways to distinguish such objects are to include color information. This has been exemplified in Figure 16.

The color modality is employed as an *early fusion* detection step, which means that the color information is used in combination with other feature types. This is opposed to *late fusion*, in which color information is used as a verification step after performing an initial detection. Early fusion was chosen as an approach, because this entails that color can directly be used as a modality in the framework. Templates may also have a different color appearance when regarding different orientations, making it a natural candidate for direct inclusion in the template.

The color modality has been added to the framework with two distinct feature types. The first is a *Hue angle* feature type inspired by the *gradient angle* features that have been discussed above. The other modality is based on *color names* as described in the papers by Weijer and Khan *et al.* [45, 51].



(a) Quantized hue gradient (b) Saturation image

Figure 17: Quantization of the Hue descriptor. (a) Represent the quantized hue values, most colors are located in the red spectrum, dividing the pixels over the last two bins. (b) Saturation image, the whiteness determines the amount of saturation, these values are used during thresholding

3.3.1 *Hue gradient*

The Hue gradient is extracted from the 2D input image. The following extraction routines are used to extract the different color channels give an image pixel in the RGB color space, derivation and notation taken from Hanbury [52]:

- Calculate the angles of the cylindrical color space, α and β, to be able to define the color values:

$$\begin{aligned} \alpha &= (2R - G - B) \\ \beta &= \sqrt{3}(G - B) \end{aligned} \tag{42}$$

- Calculate the Hue H and Saturation S:

$$H = atan2(\beta, \alpha) \tag{43}$$

$$S = \frac{\sqrt{\alpha^2 + \beta^2}}{max(R, G, B)} \tag{44}$$

- The Hue H can be directly used as a gradient angle similar as has been discussed with the 2D and 3D gradients. Equation 12 can be used to generate the index, which is mapped onto the descriptor $d = 1 \ll index$.
- A threshold on the saturation S is used during filtering, to filter any non-discriminative color regions. S is commonly defined as $S = \frac{C}{V} = \frac{\sqrt{\alpha^2+\beta^2}}{max(R,G,B)}$, which is the chroma C divided by the value V.
- Chroma describes the colorfulness relative to the brightness of a similarly illuminated white. When divided by V, the maximum R, G, B component, value we create a mapping for different color ranges that fit into $[0, 1]$ range, making the saturation

a suitable candidate for thresholding, as a single threshold can be selected.

- After the quantization, the feature is filtered by regarding its neighbors as was done for the 2D and 3D gradients in Section 2.3.3, and the descriptors combined into the binary image $\mathcal{Q}$, which is visualized in Figure 17.

After we have defined the image $\mathcal{Q}$, we can use this image in exactly the same way for the matching procedure described in Section 2.4.

3.3.2 *Color names*

color names are presented in the paper 'Learning color names from real-world images' bij Weijer *et al.* [45] (and is further elaborated on in Weijer *et al.* [53]), in which they present the theory that humans give the color of the object a certain name, for example *Green*.

These color names are a mapping from a continuous set of RGB values into a discrete number of labels or topics. These labels don't share an equal volume in the RGB cube; the labels green, red and blue span a much larger volume than either white or black. The distribution of the color names over the Munsell[1] color space can be seen in Figure 18, as can clearly be seen the label area's are unequal. In [45] the authors have used images labeled and gathered by Google as a training set to estimate the probability of a color name.

Thus, given a point $\boldsymbol{z} \in Z$, where z is a point of the CIELAB color space Z, the authors define a random variable containing the color names $C = \{c_0, \ldots, c_{10}\}$, to approximate the probability:

$$P(C = c | Z = \boldsymbol{z}) \tag{45}$$

Which is the probability for a certain color label c given a certain color value $\boldsymbol{z}$.

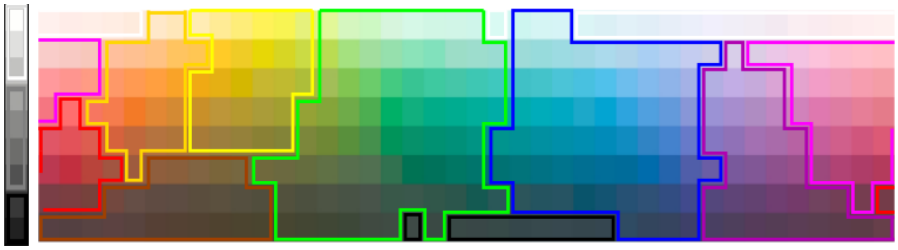


Figure 18: The distribution of color names over the Munsell color array. The colors of the bordered regions correspond to the most likely color label for that region. *Source* [53].

1 Munsell was an american artist, that devised an early color system on which later color systems like CIELAB are based.

The color names have been learned using a variant of the Probabilistic Latent Semantic Analysis (PLSA) [54] technique, which is similar to the popular Latent Dirichlet Allocation (LDA) [43] except that it does not include the Dirichlet prior. The output of the learning process is an $CN = (32 \times 32 \times 32)$ quantized RGB table that contains the multinomial distribution for the point $z \in Z_{RGB}$. So for a given point z the following distribution is returned from the table:

$$CN(z) = P_C(C = c|Z = z) \tag{46}$$

The Color Name attributes have been used in the HOG framework [51], and have also been utilized successfully in the recent Pascal VOC 2012 challenge. An example of a few annotated images are given in Figure 19. These images were taken from the Google Images database.

The advantage of the color names descriptor as opposed to the Hue descriptor is that, according to [51], it should less variant for lighting and more appropriate for real-life scenes, which means provide more semantically correct labels. Another advantage is that, unlike the Hue descriptor, black and white can be represented. To be able to represent these colors in the HSV space, both hue and value are needed. These advantages and the successful application, makes this feature an interesting candidate for a color modality.

3.3.2.1 *Extraction*

To use the color names feature in the framework a couple of choices will have been made. First we assume, like in the original feature extraction, that the pixels in the image and their associated probabilities are independent. Furthermore, in the original implementation by Weijer *et al.* 11 Color name attributes have been used. A previously discussed requirement of our detection method is that we can only use 8 feature types. The choice of the used colors have been determined by experimentation, the colors names that are currently being used are: *Yellow, Blue, Brown, Green, Pink, Purple, Red, Orange*. In other cases the most likely Color Name label is extracted from the table and used instead:

- Given an RGB value $z \in RGB$. The RGB values are *floored* to the closest quantized value in the table, we extract the conditional probability:

$$r = \lfloor z_r \rfloor, \quad g = \lfloor z_g \rfloor, \quad b = \lfloor z_b \rfloor \tag{47}$$
$$CN(z) = CN(r, g, b) = p_C(c|Z = z) \tag{48}$$

- This provides a conditional probability per color name $C_n = c$, for which we define the labeling function:

$$label(z) = \arg\max_c p_C(c|Z = z) \tag{49}$$

(a) Ground truth (b) Pixels are annotated with color names

Figure 19: (a) The ground truth (images before the transformation) is included as a reference. (b) The color names are visualized, where each color corresponds to its Color Name label, the pixels are processed independently. *Source* [53].

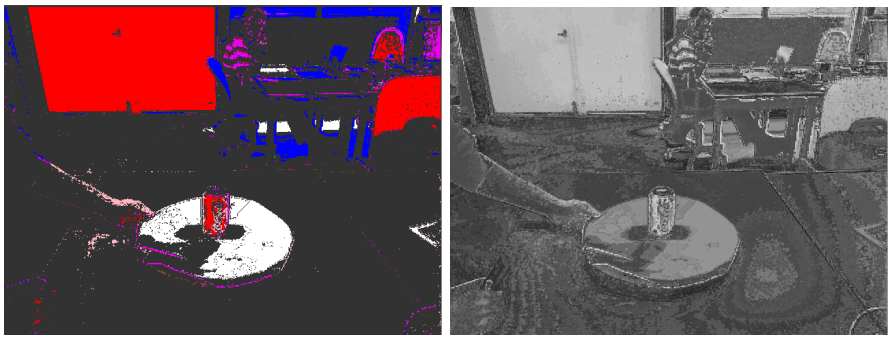
which is the color name which is considered the peak of the multinomial distribution.

- This can subsequently be used to generate an index by activating one of the bits corresponding to the index $d = 1 \ll index$. The index corresponding to an arbitrary ordinal that denotes the index of the color name in the set of used color names.
- To filter the color names the conditional probabilities are used. Such that only pixels are used that adhere to:

$$P(C = c|Z = z) > 0.5 \quad \text{for a known } c \text{ and } z \tag{50}$$

to filter non-descriptive color values.

Figure 20, shows the image $\mathcal{Q}$ for the quantized color names as well as the probability map used for filtering. The conditional probabilities are high around the coke and the red door. Most of the image is filtered out because of the low probabilities, these regions are also more prone to the switching and therefore unstable, switching between labels in during multiple frames.



(a) color names image (b) Conditional probability map

Figure 20: The Color Name feature. (a) Shows the quantized color name image, with white included for clarity. (b) Shows the conditional probabilities, a higher intensity denoting a higher condition probability

3.4 COLOR FEATURE SIMILARITY

For the 2D and 3D gradients, we can take advantage of the fact that most of the features are gradient measures. Thus, we are able to use a single similarity measure for these feature types. The color name feature has a probability measure which we are able to use directly. For the additional feature types introduced in this chapter, new lookup tables τ have to be generated, as the values will naturally be different from the original modalities.

3.4.1 *Hue gradient similarity measure*

Since the Hue is also a 2D gradient in the color space, the angle θ can be calculated similarly as for the other gradient features. Therefore, Equation 26 is used directly. A similar approach can be found in related literature [51, 55]. To make use of the optimizations presented in the previous chapter, the modified Steger similarity, found Equation 30, that is used for the gradient modalities is also applied to the Hue gradient.

3.4.2 *Color names similarity measure*

To use the color names feature in the current framework a measure of similarity has to be specified. To find this similarity we have to regard the labeling function described in Equation 49. Because during the matching process we have only access to the quantized descriptor d, which represents the label of the pixel from the binary images $\mathcal{Q}$ and $\mathcal{J}$. So we are seeking a function that returns a similarity between two labels: the image label l^{im} and the template label l^{t}. Consider the event $l \in L$, where the event is assigning the label of a certain $c \in C_n$ to the RGB values $\boldsymbol{z} \in Z$. The assumption is made that the pixels in the RGB space are uniformly distributed:

$$p_Z(z) \sim U(0, M) = \frac{1}{M} \quad \text{for } M \text{ pixels} \tag{51}$$

With this definition, it's possible to describe the conditional probability for labeling a point z with l, using Equation 49 and the indicator function $\mathbb{I}$:

$$p_L(l|Z = \boldsymbol{z}) = \mathbb{I}(l = label(\boldsymbol{z})) \tag{52}$$

It is also possible to calculate the prior probability for a certain label:

$$p_L(l) = \sum_{z \in Z} p_L(l|Z = \boldsymbol{z}) p_Z(z) \tag{53}$$

$$p_L(l) = \frac{1}{N} \sum_{\boldsymbol{z} \in Z} p_L(l|Z = \boldsymbol{z}) \tag{54}$$

Equation 54, expresses the prior label probability as the amount of pixels $\boldsymbol{z} \in Z$ labeled l. To define the similarity seen in Equation 24 and Equation 25, we need to calculate the probability:

$$P(C_n = c|L = l) \tag{55}$$

which indicates the probability of detecting a color name c given that the a point $\boldsymbol{z}$ is labeled l. Writing this as a similarity function: $\mathcal{E}_{cn}(T_i^m, \boldsymbol{z})$, we obtain:

$$\mathcal{E}_{cn}(T_i^m, \boldsymbol{x}) = \frac{1}{N} \sum_{(\boldsymbol{r}, d) \in T_i^m} P(C = d|L = l) \tag{56}$$

To calculate $P(C = c|L = l)$, we need to calculate the conditional probability distribution:

$$p_C(c|L = l) = \frac{p_{CL}(c, l)}{p_L(l)} \tag{57}$$

To calculate this distribution, consider the joint probability of c and l, conditioned on the point z:

$$p_{CL}(c, l|Z = z) = \sum_{z \in Z} p_C(c|Z = z) p_L(l|Z = z) \tag{58}$$

For which is assumed that $p_C(c|Z = z)$ and $p_L(l|Z = z)$ are conditionally independent. This makes sense considering that both the labeling is solely dependent on the labeling function and the conditional color name is not influenced by the label given to the pixel. Finally, now that all values are known, we can calculate the conditional distribution from Equation 57.

$$p_C(c|L = l) = \frac{\sum_{z \in Z} p_C(c|Z = z) p_L(l|Z = z)}{P_L(l)} \tag{59}$$

The final equation essentially calculates an probability that a pixel is differently labeled. Unfortunately, there is still a problem with using this equation directly as a similarity measure. Similarities, are not probabilities, while similarities do range from 0 to 1 they need to return a value of 1 if the feature is of the same type, the calculated probability need not return 1 when calculating:

$$p_C(c|L = l) \text{ when c = 1} \tag{60}$$

The other similarity measures discussed in this thesis do. To remedy this problem we make use of the likelihood ratio between two color names c_0, c_1:

$$\frac{p_C(c_1|L = l)}{p_C(c_0|L = l)} \tag{61}$$

Where the label l is fixed for both the denominator and the numerator, and c_1 is varied for the different color names that are used. When $c_1 = c_0$, the ratio will be *equal* to 1 and if $c_1 \neq c_0$ the ratio is less than 1. A similar ratio measure is used for naive bayesian spam filtering [56], so this works best when assuming a uniform prior.

Now that we have obtained a measure of similarity between color name features, we can calculate the tabel τ per feature. Enabling the method to provide the spread image $\mathcal{J}$ and the Response Maps $\mathcal{S}$.

3.5 MATCHING ADDITION

Because the addition of an extra modality means that we will require more features, an extension has been made to the current system. The similarity map $\mathcal{E}$ is represented as shorts instead of bytes/chars that the original method used. This allows us to have a higher similarity, which effectively means that we can use 116384 features instead of 64. This proves especially significant in our case because additional modalities have been added, which need to be represented by a certain amount of features. Doing this naively would slow down the algorithm significantly, which is why the appropriate SSE instructions should be used.

Recall that the linear memory is an array of bytes, using SSE we can process 16 bytes in parallel. The similarity is represented as an array of shorts, which means we can process 8 shorts at a time. However, we can make use of multiple registries to load the bytes into two short registries, by interleaving the bytes with zeros into the most significant bytes. If we do this for the upper and lower part of the registry, we create two registers of shorts which we can add to the similarity image.

3.5.1 *Speed increase*

Because each modality has its independent input image $\mathcal{J}$, we are able to speed up the computations by multi-threading the input. Since the input images are processed each frame, this parallelization results in a significant speedup. To make this possible, the images captured from the *RGB* and *Depth* streams should be temporally synchronized. When receiving the images, we can pass copies of the images to the processing functions which can be executed in parallel. The The processing time is generally the time it takes to process the most computation intensive image, which in the current method can range from 60 to 80 milliseconds.

3.5.2 *Cache miss reduction*

The Central Processing Unit (CPU) has different levels of memory which it can quickly access: RAM being the slowest; the on die memories caches L1 and L2, being the fastest respectively. A cache miss occurs when the CPU has to load memory into a higher cache level. The CPU loads the memory into the cache, in multiple bytes dependent on your system architecture. However, the CPU cache size is much smaller that your average RAM size: a few megabytes opposed to a few gigabytes. Thus, it has to load data into the cache multiple times during the execution of a program. A cache miss is when the

data cannot be found in the CPU cache, a further distinction can be made regarding the nature of the cache miss[2]:

COMPULSORY misses are those misses caused by the first reference to a location in memory.

CAPACITY misses are those misses that occur regardless of associativity or block size, solely due to the finite size of the cache.

CONFLICT misses are those misses that could have been avoided, had the cache not evicted an entry earlier.

The linearization discussed in the previous chapter reduces the amount of cache misses when matching the template. To further reduce the amount of Conflict matches we can sort the gradients by their descriptor d and their position in the template. Which means that the Response maps $\mathcal{S}_j$ are accessed in order, which means they are only evicted from the cache when all templates with the descriptor $d = j$ have been processed. Effectively reducing the amount of cache misses by 30%. This result has been verified using cachegrind [57], an open-source cache analysis toolkit, which is part of the valgrind suite [58].

2 distinctions made by Mark Hill, http://www.cs.wisc.edu/~markhill

4

SOFTWARE ON AMIGO

4.1 INTRODUCTION

Because the software will have to run on AMIGO in real-time and the learning and detection should be easy to startup during a RoboCup challenge, demo or when AMIGO has to perform a task, a framework has been developed that contains the detection module and provides an infrastructure for the detection of objects. The detection framework should function within the AMIGO eco-system, which means that it should be able to communicate with the rest of the software running on AMIGO. This chapter contains a description detailing the practical considerations which were regarded during the development of the detection module.

4.1.1 *Third party software*

Whenever AMIGO performs a task, usually one of the steps to ensure a successful completion of a task requires the detection of objects. However, this in it self is often not enough. Even if a successful detection has been made, this has to be communicated to the other systems so that AMGIO can decide and act on this new set of information. Two important third party modules, used on AMIGO, are the Executive and the Reasoner, both of which are described below:

Executive The Executive is a planning and decision making module. AMIGO needs to plan and decide tasks in different situations, these situations arise when performing a challenge or a general service task. The executive provides a high-level interface into the robot, which runs in combination with a state-machine that uses this interface. The executive decides when to activate the perception module to learn more about the environment.

WIRE The WIRE module maintains a model of the current world state and enables reasoning about the state of the world. The details of this system are described in a paper by Elfring and van den Dries [59]. The World Model fuses sensor information into a model using multiple probabilistic hypotheses, and performs tracking using a Kalman Filter [60]. WIRE is able to main-

tain multiple object hypotheses which it can collapse or expand into a single or multiple hypotheses respectively. The detection module communicates directly with this component.

To make the detection module to useful for AMIGO it should be able to communicate directly with these systems, in either a direct or indirect manner.

4.2 SYSTEM GOALS

Another important requirement was to achieve some of the goals that were set in Section 1.3.2, specifically pertaining to: the *Robustness*, *Usability*, *Integration* and *Extensibility*. To achieve these requirements the following goals were kept in mind when the detection software was developed:

- The system should be easy to startup and be used with the rest of the robotics software, providing a seamlessly integrated whole.
- The system should be easy enough for novices to use, and should be quick to setup during competitions like the RoboCup or during a more general execution of a task.
- The system is going to be used by other (novice) users, so tooling should be available to make detection and learning easy.
- The system should be extensible, lowering the threshold for further research. Thus, making it easy to change parameter settings, provide information during detection and make it easy to add different modalities or other extensions.
- The system should be *efficient*, not wasting CPU cycles and should be able to run idly. This is important because other software may need to obtain processor priority.

4.3 DETECTION ON AMIGO

Now that the goals and requirements have been described, a description is given of the integration into the current infrastructure on AMIGO. The entire system has been developed in the ROS [4] framework from the start, which is a framework that makes it easier to develop applications for mobile robotics, and is used throughout the AMIGO project.

Figure 21, illustrates the infrastructure of the detection system, and how it fits in the current robotic infrastructure. The blue modules are modules that are used during detection and the orange depict external modules. The blue modules (and some other offline tools)

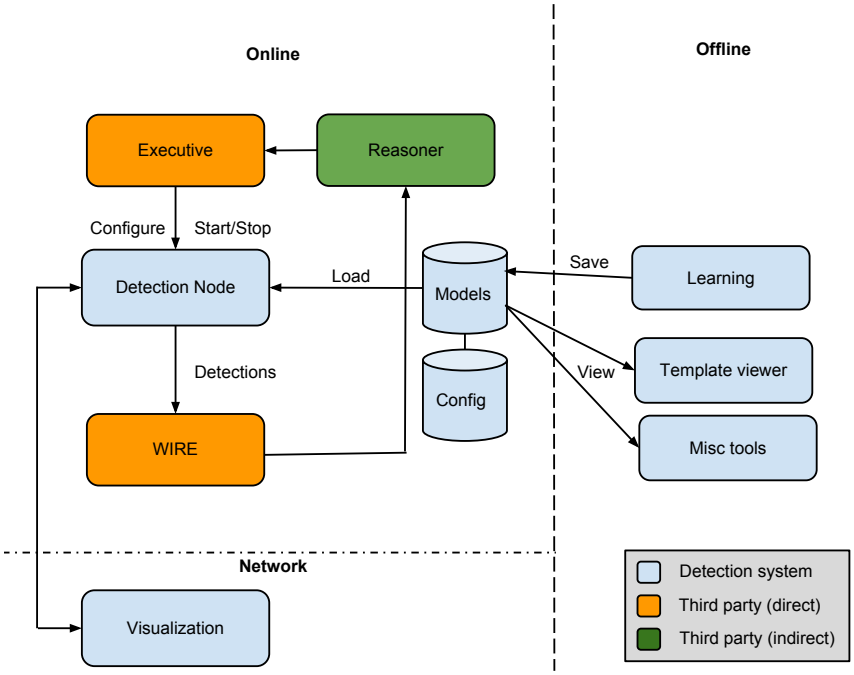


Figure 21: The infrastructure in which the detection module operates. The diagram is divided into three areas: Online, modules running on the robot in real-time; Offline, modules for processing of the templates and viewing these; Network, this module does not need to run on the robot but can also run on a remote computer. The color depicts the dependency in the system: blue is a direct dependency, orange has direct communication with the system and green has no direct connection.

have been developed during the production of this thesis. The green modules are indirectly related to perception, that is, there is no direct communication between blue and green. Because the output is passed to the Executive indirectly (communication is one-way), an independence between perception and executive can be established, which enables the use of different perception modules.

Now that an overview of the detection system has been established, two important processing steps will be discussed. The offline learning of the objects including the corresponding object database, and the online detection phase and its output is discussed.

4.3.1 *Learning an object model*

To learn an object model a file database system was developed to record and load the models. A collection of templates is saved per modality, and these are combined into an object model $\boldsymbol{O}$. To learn these objects an application was developed to record templates in real-time. Because the system is fast enough to record these templates

in real-time, we can reuse the steps described in 2.2.1 for the learning process.

To record a template:

- Start the learning module by providing a *category* and *name* for the object.
- Initialize the *position* and *location* with a user defined bounding box.
- Perform a match to find the object.
- Determine if the template should be added to the database. We will have to make sure that the template is different enough to provide extra information, but should not be too different because the detection could have been a false positive. Currently, this is ensured by checking that the detection falls between the threshold t_{low} and t_{high}.
- Should this be the case than the template is added to the current database. An object model is saved with both a category and a name, which is subsequently used to identify the model when loading from disk, and is used as an object name (label) during detection.

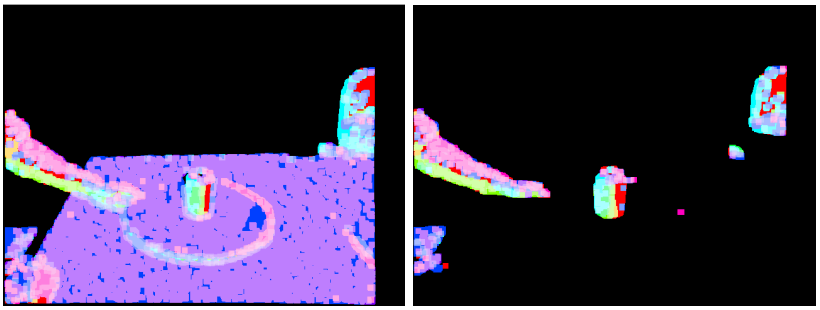
To create the object database the user has to manually learn multiple objects from different angles for the detection to perform optimally. To load the database onto AMIGO, a System Configuration will have to be specified, this is detailed in Section 4.3.3

4.3.1.1 *Plane Extraction*

Because the bounding box contain background information, we should aim to eliminate this information from the template. To do this depth map is transformed into a point cloud $\boldsymbol{P}$, using the inverse camera matrix $\boldsymbol{K}^{-}1$, as was demonstrated in Section 2.3.4. The point cloud is then used as input to a Random Sampling and Consensus (RANSAC) [61] method, which estimates a plane equation $Ax + By + Cz + D = 0$ from a set of points. We use this to eliminate the points belonging to the plane by evaluating the equation for the point cloud, and only regard the object we are trying to learn, essentially masking the input image. The resulting image is shown in Figure 22

4.3.1.2 *Feature selection*

To create the templates, the system must select the appropriate features. Currently, features are selected based on their spreading in the template. To obtain a good response, features should typically be spread evenly. A good spreading ensures more distinctive templates, similar feature types are often clustered in a small region, and it ensures that the position in the template makes for a stronger constraint. Because of the spreading, the position of a feature in a template is



(a) Unmasked 3D binary image $\mathcal{J}$ (b) Masked 3D binary image $\mathcal{J}$

Figure 22: The unmasked (a) and masked (b) processed depth image. The same process is applied in the processed RGB image.

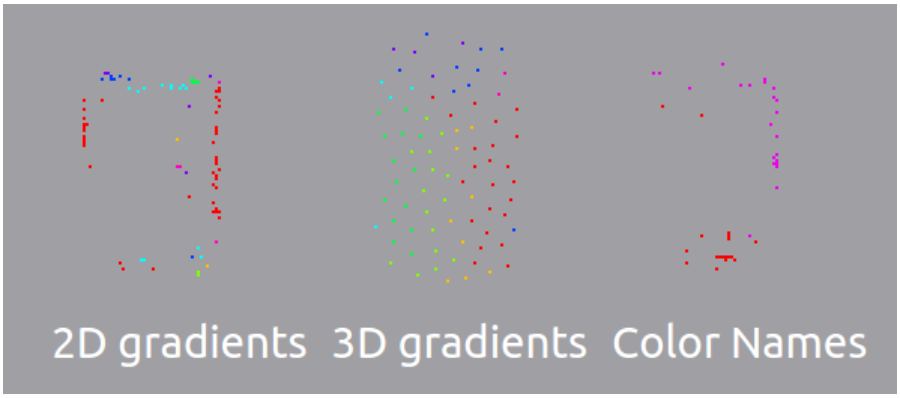
generally less of a constraint. A heuristic is used for the selection of the feature based on the distance to features that have already been selected, and is combined with a selection on the highest certainty, which can be either the *gradient magnitude* (gradient modalities) or the *posterior probability* (color names). In the case of the 3D gradients, only distance is taken into account. Because intensity has no distinctive meaning, as the 3D gradient is equal to a normal vector for which the length equals 1. The effect of different spreading parameters can be seen in Figure 23.

4.3.2 *Learning GUI*

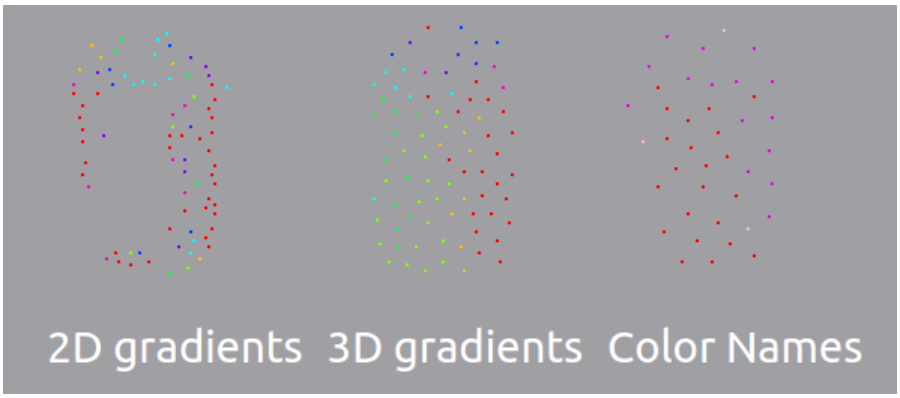
The GUI for learning Object Models is visualized in Figure 24. This GUI is used for training object models, which are then utilized for the detection of objects. A user is able to learn a new model by defining a new bounding box, or add templates to an existing model with a new bounding box. Existing bounding boxes that have been learned for similar objects, can also be used to either create a new model or add templates to an existing model. The learning module automatically creates the appropriate file database for storing and loading the object models. The Learning GUI can easily be used by novices, with minimal instructions.

4.3.3 *System Configurations*

Before the models can be used for the detection of objects during an execution of a task, AMIGO first has to know which objects it should detect. With the current method it is infeasible to use a database of more than 30 objects, an amount which certainly has a large negative influence on the performance, and can have a negative influence on the accuracy as well. Therefore, it is possible to define an Object Configuration, which details the objects to use and the associated pa-



(a) Coke Template: low feature spread



(b) Coke Template: high feature spread

Figure 23: Three modalities with their corresponding templates for a single view, visualized by the Template Viewer application. Note that by changing the heuristic we influence the spreading of the 2D feature types, as these both have a measure for the intensity of a template

rameters: like the modalities (2D, 3D and Color) that are to be used, and the threshold at which the detection module considers a detection a match. These possibility to create these configurations also improves usability, as the models can be reused in multiple situations. This makes an interesting use-case possible, in which the Reasoner could deduce in which room AMIGO is located and make use of this information. For example, the reasoner decides that AMIGO is in the kitchen and changes the configuration at run-time to only detect items that could be found in the kitchen. These kind of situations come up often during the course of a RoboCup competition, as the environments are modeled similarly to human homes, so that each room contains different object types.

Listing 2 shows such a configuration. Object models are identified through a category and a name. The System Configuration name can directly be supplied to detection module to load these object models automatically.

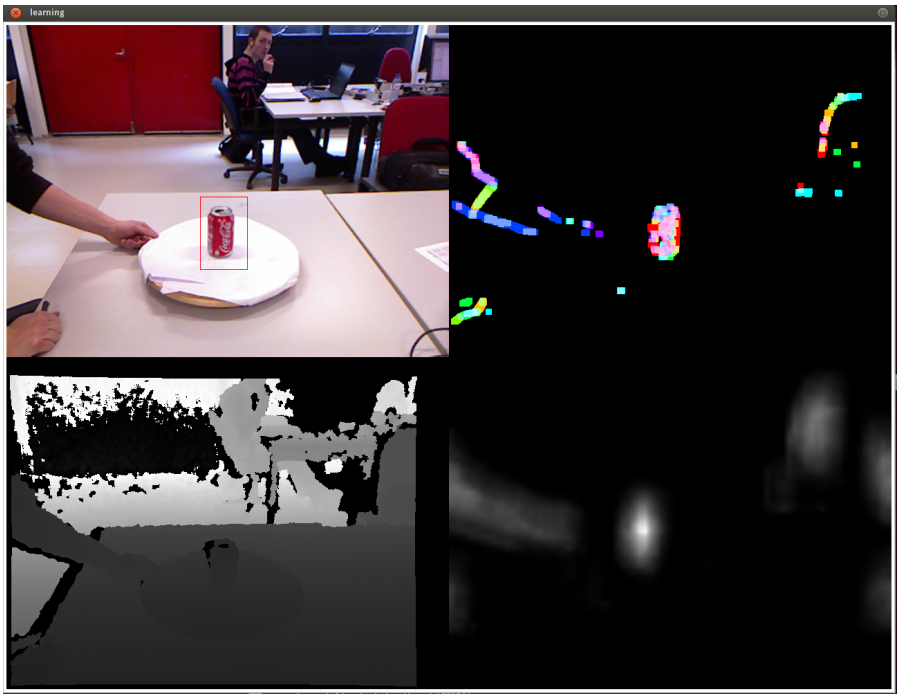


Figure 24: The user interface for learning an object. The left side shows the input images, as well as the bounding box where we have found the latest object. The right part shows the image $\mathcal{J}$, and the combined Similarity Image $\mathcal{E}$. The visualization can be switched during run-time.

4.3.4 *Run-time detection*

The concepts of a Executive and a World Model (WIRE) have been previously introduced. To enable the communication with these described components, the detection module must provide an interface to the former and use the interface of the latter. It should be able to activate and deactivate on command, as well as process detections and publish a higher level description of the detection. The detection module is currently implemented as a ROS Node, which is an executable which runs concurrently with other nodes on the robot.

4.3.4.1 *Networking*

Because the detection software is running on AMIGO, it is hard to debug the process if no visualizations are available. AMIGO, unlike other robots, does not have a screen to display information, everything is done remotely. To provide visualizations and information, the object detector makes use of an Client-Server architecture that is visualized in Figure 21, the server is the *Detection Node* and the client as the *Visualization* module. The client makes connection with the server, which in turn informs the client what visualizations it is able to run. After this initial handshake, the server proceeds to sent

```
- systemConfiguration: eval_cross_cn
  dashboardConfiguration:
    modalities: [1, 2, 4]
  defaultCategory: eval_cross_cn
  objects:
    - name: aa_drink
      score: 80.0
    - name: beer
      score: 80.0
    - name: coke
      score: 80.0
    - name: crackers
      score: 80.0
```

Listing 2: A System configuration example, specifies what objects to use and the corresponding parameters.

raw data to the client regarding current detections and images to visualize, these are visualized on the client side and the detections are shown on-screen. This makes it easy to visualize the procedure with multiple clients and offloads processing time from the server otherwise spent on visualizations. The images sent over the wire, are 33% of the input image size, as they have been discretized into a data line of single bytes.

During run-time the detection module follows the following procedure:

1. The module is started with some specific settings regarding which modalities and normal estimation it should use, and a link is provided to the object database. At this time the existing System Configurations are loaded from a settings file. A System Configuration is collection of objects $\boldsymbol{O}$ and associated thresholds t.

2. After initialization, the detection module is in idle mode, which means that it does do any detections at the moment.

3. When the *Executive* requires the perceiving of objects, it makes a direct call to the detection module. The detection module is then activated with the current System Configuration. Note that these can be switched at runtime, depending on the Configuration required at that moment.

4. The object module continuously perceives objects by preprocessing the input images and performing a detection using the current database. Note that the Depth and RGB images are published separately by the sensor, so we need to make sure that

the images are being temporally synchronized, i.e. using images that are as close to each other in the time domain as possible.

5. After the aggregation of the detections, we pass these on to WIRE. WIRE requires us to give a probability which expresses the confidence of our detection. Similarity cannot be directly converted to a probability, as the probability to detect object O at position $\boldsymbol{x}$, given input I:

$$p(O, \boldsymbol{x}|I) \tag{62}$$

has to take into account the prior probabilities $p(I)$ and $p(\boldsymbol{x})$, which similarity does not do.

6. However, we can make use of an heuristic: $(sim - (max/2))/(max/2)$, to estimate a measure that is constrained between $(0, 1)$. This heuristic is used because the spreading has a side effect that causes highly textured areas to have high similarity. For many objects there is at least part of scene which has around 50% similarity, so we should only consider regions that have a similarity that is at least as high as these regions.

7. The system keeps processing and sending detections to the world model, perceiving each frame as independent.

8. The system is stopped by the executive, which can send a command to return the detection node back into idle mode.

Figure 25, shows the detection module in action, detecting an object in the same scene that was used throughout the last chapters.

4.3.4.2 *Scenario*

To give a clearer picture, let's illustrate a small typical scenario for AMIGO. Recall that the Executive can interface with the world state through a Reasoner, this Reasoner provides an interface which can be used to query the world state, which are different hypotheses contained in the world model (WIRE).

Consider the following scenario, AMIGO wishes to find a coke can at a designated location. AMIGO turns on its object detection system:

- The detection module, processes the scene and publishes the positive detections to WIRE, this proceeds as has been described above.
- WIRE integrates other received information and tracks the potential objects in the world model. WIRE also decides the matter of trust to put in the received detection, and decide to create a new hypothesis or decides to update an existing hypothesis. After each processing step a state of the world is updated.

- The Reasoner is now able to query WIRE, e.g: "The robot is at x, are there any coke cans nearby?". WIRE determines if an hypothesis exists by regarding the received information, and returning the most likely world state. WIRE responds positively.
- The Executive, which activated the Reasoner to acquire this knowledge, now knows of the existence and position of the coke can. It can proceed to actuate AMIGO to grasp and return the object.
- Thus, completing the loop and allows AMIGO to move on to the next task.

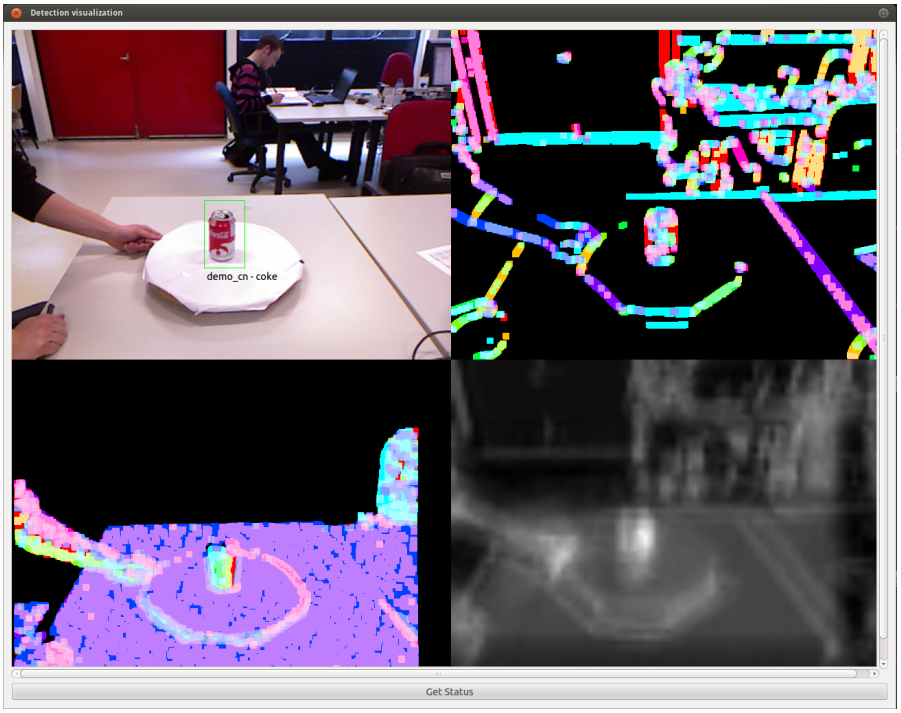


Figure 25: The visualization node, showing a detection of a coke with it associated category, both of which can be configured. The information visualized is provided by a separate detection process.

4.3.5 *Summary*

We have discussed the infrastructure and seen some of the tooling that is available. In the end, an entire object detection module is provided. The system has all the tooling needed to learn and detect objects, including the ability to provide visualization over network on multiple computers. The system is easy to use and fully configurable, with the ability to load multiple object configurations and different estimation methods and modalities are readily available.

Part III

EVALUATION & CONCLUSIONS

5

EVALUATION

5.1 INTRODUCTION

In previous chapters the detection system used by AMIGO has been described. It is currently being used as the general object detection system in the AMIGO project. It was used during the RoboCup, experiments and during demonstrations of AMIGO to the general public. Figure 26 shows AMIGO interacting with objects at the last RoboCup event: RoboCup 2013.

(a) Detection of a coke can (b) Grasping an object from the table

Figure 26: AMIGO in action. (a) AMIGO is detecting a coke can. (b) Amigo has detected the seven up, and continues to grasp the object.

5.1.1 *Overview of the evaluation*

In this chapter, an evaluation will be done of the detection system with custom benchmark. The goal is to provide a measure of accuracy of the developed system, to do this the system described in this thesis will be compared with the method proposed in the paper by Hinterstoisser *et al.* [1], the normal estimation methods and color names will be varied to test which configuration has the optimal performance. To be able to answer the research questions, the performance of the system is evaluated with a custom object dataset. We will also look at what influence the world model has, on the accuracy of the detection method. W

The remainder of this chapter is organized as follows:

- First, the dataset and the benchmark, which is used during the course of the evaluation chapter, will be discussed shortly .
- After which the benchmark will be shown that compares the original method with the method discussed in this thesis.
- The individual color modalities and normal estimation routines will also be benchmarked and discussed.
- These benchmarks are followed by an in-depth discussion of the current system and its drawbacks.
- A final evaluation will measure the processing times of the system and the added modalities.
- Finally, conclusions are drawn with regards to the performance of the system.

5.2 OBJECT DATASET

(a) AA drink (b) Cup of noodles (c) Red cup (d) Garlic sauce

Figure 27: Four of the objects that were used for detecting objects with AMIGO. Demonstrating which objects have been used in the past. Object taken from the Kinect camera, resulting in somewhat low resolution images.

During the RoboCup, the objects are typically bought in a supermarket so that they resemble actual objects that are found in a household environment. The amount of objects that AMIGO has to detect typically ranges from 10 to 25 objects, depending on the challenge. Note that currently the robots in the challenge are only required to detect object instances, so no generalization to categorical based detection is required yet. AMIGO has to detect these instances at different locations in the so called arena, which is the venue where the event takes place. The objects are used in the dataset, are similar or the same as the objects that were used during the last RoboCup events.

Some objects that were used during testing have been visualized in Figure 27, the images were taken from the dataset from which the objects have been learned. The objects have been chosen as to resemble objects used in the different RoboCup scenarios. In total 12 objects have been tested in different locations. To test the detection system, we have used scenes with different amount of clutter at four locations in the lab at the TU/e Eindhoven. The object dataset in its entirety, as well as the locations, are listed in Appendix B.

Figure 28, gives an impression of the arena's where AMIGO is currently deployed, note that the lighting conditions may vary throughout the day. During this evaluation we have made use of different locations in the robotics lab in Eindhoven to test the system, this lab resembles an environment that can be found during a RoboCup competition.

(a) View of the RoboCup arena 2013

(b) View of the German Open 2013 arena

Figure 28: The RoboCup arena's. Different area's are allocated where objects may be located, objects can be set at different positions on multiple surfaces. (a) Shows the latest arena, AMIGO is picking up a coke can from the table. (b) Shows the German Open arena, objects were placed on the tables and near the kitchen.

5.2.1 *Object Annotations*

The dataset has been annotated with object locations, annotations are specified with a bounding box that indicates the (x, y) positions and the $(width, height)$ in image coordinates. The detection node, that was introduced in Section 4.3, has been modified to accept custom annotation files, thus enabling the generation of statistics according to these annotations. The object annotations are created specifically per scene but can be re-used by multiple configurations. This is in line with one of the goals, namely to make the system extensible, with this addition the option is added to recreate the results generated in this chapter by other students working on the AMIGO project.

5.2.2 *Matching criteria*

The Pascal VOC bounding box matching criteria [6] has been used to determine if an detection is a true positive or a false positive. The measure is described below:

Given the bounding boxes: B_{obj}, the object bounding box; and B_{im} the image bounding box. The matching threshold is specified as:

$$\frac{B_{obj} \cap B_{im}}{B_{obj} \cup B_{im}} > 0.5 \tag{63}$$

which places a threshold on the fraction between the intersection and union of the bounding boxes, when a detection meets this criteria than it is considered a detection at the given location. Note that this criteria is evaluated *after* a detection exceeds the threshold specified by the detection system.

5.3 DETECTION BENCHMARK

The dataset described in the previous section has been run using different configurations. Namely, the original method *LINE-MOD* and the method extended with color modalities: *Hue* and *Color Names*. The goal was to see whether the original method generalizes to different object sets, and to what extent the color modalities influence the accuracy of the system. The results have been listed in Table 1. Both the precision: $\frac{TP}{TP+FP}$, and the recall: $\frac{TP}{TP+FN}$, are displayed; which are the true positives (TP) divided, by the true positive plus false positive (FP), and the false negatives (FN) respectively.

During the benchmark, a threshold of 0.85 has been used, and the number of gradients per template was 80 for the gradient modalities and 40, for the color modalities. The color names and hue modalities use the least squares gradient approximation for the scores in table 1. Different scenes have been set up, either containing a single or multiple objects, as this is a situation that AMIGO would encounter. Only

DATASET	CONFIG	UPREC	FPREC	UREC	FREC
Single	LINE-MOD	0.40	0.90	1.00	1.00
Single	COLOR NAMES	1.00	1.00	0.44	0.44
Single	HUE	0.58	1.00	1.00	1.00
Multiple	LINE-MOD	0.41	0.73	0.87	0.84
Multiple	COLOR NAMES	0.71	0.94	0.71	0.69
Multiple	HUE	0.64	0.90	0.97	0.97
Categories	LINE-MOD	0.36	0.71	0.60	0.59
Categories	COLOR NAMES	0.71	0.87	0.40	0.40
Categories	HUE	0.64	0.82	0.86	0.84

Table 1: Detection scores for the dataset using different configurations

small scale changes and occlusions have been included in the dataset as AMIGO is currently not encountering these variations often, as it can position itself at a fixed distance of the object in most cases. Clutter, as well as some objects not in the dataset, have been included in the scenes.

False and True Positive ratios, have been generated by employing a filtering step that is similar to that of the world model used on AMIGO:

- At each object location, if multiple objects are found (which we know from the matching criteria from Equation 63, average the similarity per object over the frames.
- Then look at all possible object locations which object hypothesis seems most likely. Choose this hypothesis as the object at that location.
- If this decision matches the annotation, regard it as a True Positive. Should another object is found at this image location, regard this object as a False Positive. When no object is found at that location, then regard it as a False Negative.

In the columns of Table 1, the tested configuration (which specifies the templates and modalities) is denoted as *config*. The unfiltered precision as *uprec*, which is the true and false positive ratio before the filtering step. The unfiltered recall is denoted as *urec* and their filtered counterparts as *fprec* and *frec*. The configuration named *LINE-MOD* is the combination of the 2D and 3D gradient modalities, which are used by the original method. The *Color names* and *Hue* modalities are the original gradient modalities augmented with the color names and hue color modalities respectively.

We can observe that all configurations do quite well in the dataset containing only single objects (termed *single*). This is probably be-

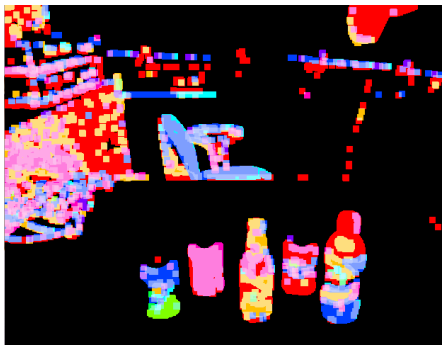
cause it is a simpler dataset, with a relatively clean workspace. The low recall score for the color names has to with the fact that it failed to detect a view from the crackers object, which resulted in a significant amount of false negatives.

The color modalities add precision to the existing dataset when compared with using the original modalities exclusively. The color names modality generally has a higher precision than the hue gradient. However, when recall is considered, the hue modality performs better, because the hue gradients tend to change less when the object pose changes. The addition of the color modalities, and the consequences are evaluated in the next section.

5.3.1 *Color Modalities*

(a) RGB image of five objects

(b) $\mathcal{J}$ image for color names



(c) $\mathcal{J}$ image for hue

Figure 29: A scene from the detection benchmark. (a) The original image. (b) and (c), the processed images. Note that the color labels are somewhat more distinct for (b), but both seem to label the objects appropriately. Also note that (c), retains some of the background from (a) on the left in the resulting image.

Table 1, shows that both color modalities seem to perform reasonably well. The color names modality has a higher precision but often a lower recall than the hue modality. This is because hue seems to be more consistent over the frames, whereas color names can be more fickle even with the thresholding that is currently being done. How-

ever, color names do provide a somewhat better separability of the dataset. Both methods have their advantages and disadvantages. The advantages of Color Names, are listed below:

- Color names seem to work well for different objects, recognizing their color correctly and is also able to discern color changes inside the object. As we can see in Figure 29a, both the transition between Sprite and the AA-drink inside the object are correctly labeled. Thus, also making it easier to detect different object angles, should one want to use this information.
- The background is often filtered out during the thresholding, while the object boundaries are retained, creating a sharper boundary between object and background, when compared with Hue. This is displayed in Figure 29b and Figure 29a, where can be seen that more of the background is retained for the hue modality.
- The color names could be used to represent, black and white, which cannot be done using hue alone. Alas, during testing this did not seem very robust. Because only 8 color names can be represented, black and white have not been included in the evaluation.

Advantages of using the hue as a feature type, are listed below:

- Calculating the hue gradients requires slightly less computation time. Because the color names are part of a large lookup table, cache inconsistencies can make access slow, and the hue calculation is done using optimized OpenCV functions.
- The hue gradients seem less sensitive when compared with the color names feature type, making it less prone to False Negatives. Thus, resulting in higher recall scores that are seen throughout this chapter.
- The hue modality is more consistent in the framework, and can use the similarity measures directly as they have been introduced in Chapter 2 to seamlessly fit in with the original modalities.

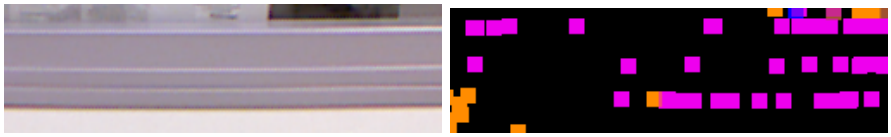
During the evaluation of the framework, and during the use of the object detection framework on the different RoboCup events, some distinct disadvantages of using these color descriptors as modality have been found:

- The color modalities are both more sensitive to illumination changes, as color is less invariant to changes in illumination than the gradient modalities.
- Both modalities, but especially the color names, suffer from debayering[1] artifacts, that result in purple borders in the quan-

1 The process of retrieving the RGB grid from the camera CMOS, using a color filter array

tized images (Figure 30). The Kinect is supposed to have settings that lessen this effect, however, changing these settings had no positive effect. This increases the response for the red and purple color names near the object boundaries.

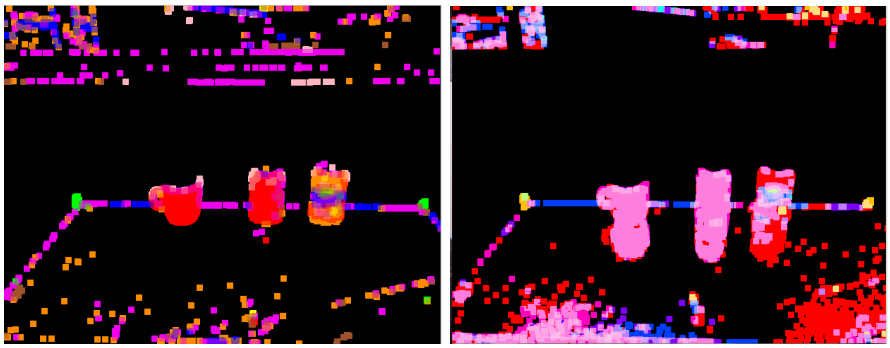
- The color names modality requires more templates, because the color names tend to be more descriptive but less stable. They vary more for changes on object pose.
- The Hue color modality collects a lot of features in the first and the last byte (bin) of the descriptor, which entails that a lot of red features are found, often mistaking colors like orange and purple as red.



(a) Input image, which shows the debayering artifact (b) Result in $\mathcal{J}$ image

Figure 30: Debayering artifact: purple border surrounding edges, which results in purple quantized regions

(a) RGB image of three objects



(b) $\mathcal{J}$ image for color names (c) $\mathcal{J}$ image for hue

Figure 31: A scene from the detection benchmark. (a) The original image. (b) and (c), the processed images. For (c), the background filtering is less successful as the reflections are not filtered from the image.

DATASET	CONFIG	UPREC	FPREC	UREC	FREC
Multiple	HUE	0.13	0.13	0.49	0.47
Multiple	COLOR NAMES	0.40	0.53	0.80	0.79

Table 2: Detection of a single benchmark for the Hue and Color names modality only

5.3.2 *Standalone color modalities*

Table 2, shows the precision and recall scores only using color as modality for the Multiple objects dataset. These modalities alone perform worse than the combined modalities, which is to be expected. The interesting fact is that the color names by itself perform better than the hue. Both suffer from false positives in the background, greatly bringing down the precision scores, but the color names seems less sensitive. This is due to the better descriptiveness and the better thresholding with the background (Figure 31). However, as has been discussed before an important factor is the ability for the detection, to have additional means to differentiate between objects, when the original modalities don't suffice. Hue suffices to provide this extra distinction even though it does not perform too well by itself. On the other hand, the color names can be too strict but generally perform better, when it is used as the a standalone descriptor.

5.3.3 *Summary*

Both hue and color modalities increase the accuracy of the detection method. The difference between using the two modalities, in combination with the original descriptors, is not very large. The trade-off is a increased *precision* against a *decreased* recall. The recall can be somewhat increased by learning additional templates, which has the negative effect of increasing the running time. For objects that AMIGO encounters it is beneficial to add a color modality, as it increases the performance when compared to the original method. Both Hue and color modality objects still have problems identifying the coke can from the Fanta; during learning it was wrongly classified as being red by the color names modality. However, during testing, because of the different lighting conditions it has been classified as orange multiple times, as can be seen in Figure 31b.

5.3.4 *Normal estimation*

In Section 3.2, the question was raised whether other normal approximation schemes would provide better results. Two extra normal approximation schemes have been described and implemented. Table 3

DATASET	CONFIG	UPREC	FPREC	UREC	FREC
Single	LINE-MOD	0.40	0.90	1.00	1.00
Single	Cross	0.36	0.70	0.69	1.00
Single	PCA	0.40	0.70	1.00	1.00
Multiple	LINE-MOD	0.41	0.73	0.87	0.84
Multiple	Cross	0.38	0.60	0.85	0.80
Multiple	PCA	0.41	0.67	0.80	0.75
Categories	LINE-MOD	0.36	0.71	0.60	0.59
Categories	Cross	0.45	0.79	0.77	0.75
Categories	PCA	0.51	0.67	0.76	0.70

Table 3: Detection scores for the dataset using different normal estimation methods

shows the different schemes. These have been combined with the 2D gradients only. LINE-MOD, is using the original least squares method which has been described in Section 2.3.4, the additional cross and PCA methods have been described in Section 3.2.

The processing times of the different extraction methods are listed in Table 5. Note that these running times are on an average per frame basis. It can be seen that the cross method proves to be the fastest and the PCA method the slowest. The running times are listed for 3D modalities alone and combined with 2D. These running times stay relatively equal because of the parallelization of the input frames, the input processing takes as long as the slowest modality, which is an improvement as the processing of the 2D modality takes 41 milliseconds on average.

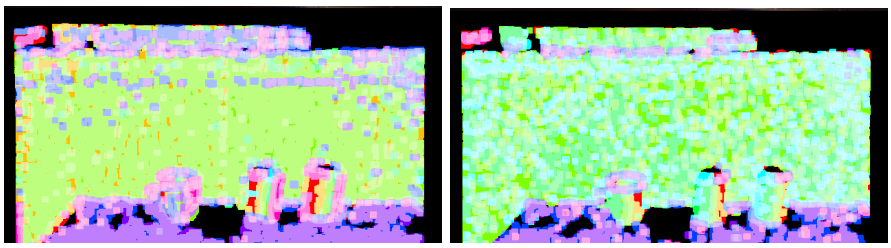
Table 3, shows the detection scores for different normal estimation techniques: *LINE-MOD* is the original method, and *Cross* and *PCA* are the new estimation techniques introduce in Section 3.2. From the table a couple of observations can be made:

The PCA gradient performs equally well when compared with the cross gradient, but is much slower to compute (see Table 6). The problem is that the region that we are using for the PCA computation, is a small 3×3 region, this was chosen mainly because of the high computational costs. Using PCA does not result in better gradients than just using regular cross products or the least squares method. The least square method performs the best on this dataset, albeit with a small margin, the lower precision for the cross gradients is caused by the large number of false positives in the background. In the case of the cross gradients the number of background false positives is 43% of the total, whereas in the least square and PCA method these amount to 30% and 19% respectively. One of the causes is the fact

DATASET	CONFIG	UPREC	FPREC	UREC	FREC
Single	CROSS-CN	1.00	1.00	0.47	0.47
Single	CROSS-HUE	0.72	1.00	1.00	1.00
Single	LS-CN	1.0	1.00	0.44	0.44
Single	LS-HUE	0.58	1.00	1.00	1.00
Multiple	CROSS-CN	0.67	0.94	0.78	0.77
Multiple	CROSS-HUE	0.66	0.90	0.90	0.89
Multiple	LS-CN	0.71	0.87	0.72	0.69
Multiple	LS-HUE	0.64	0.90	0.97	0.97
Categories	CROSS-CN	0.64	0.98	0.72	0.72
Categories	CROSS-HUE	0.61	0.87	0.89	0.87
Categories	LS-CN	0.43	0.93	0.39	0.38
Categories	LS-HUE	0.60	0.82	0.86	0.84

Table 4: Detection scores for combined color, 2D and 3D modalities

that the quantization for normals directly facing the viewer is erratic, often resulting in multiple quantized orientations. This can be seen in Figure 32, where the Least Square normals produce a more stable (more uniformly colored) plane that is facing the viewer, resulting in less false positives.



(a) Least square normals (b) Cross normals

Figure 32: Comparison of the Least square and Cross normals.

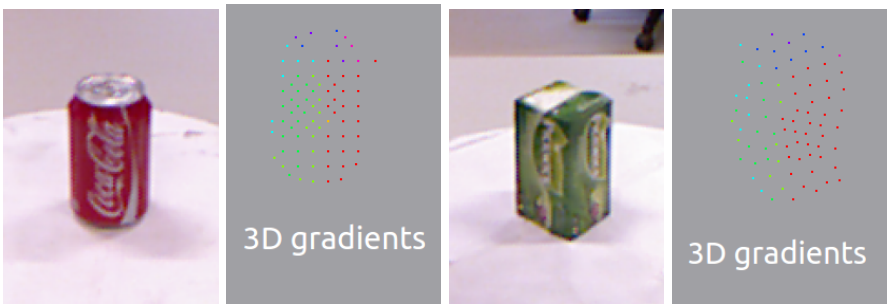
Both the least squares and cross estimation techniques provide a better separability of the data when compared to the PCA method. The filtered precision rates increase more radically, and due to the low number of background false positives of the PCA, we can assume that most of the false positives are generated by false detection at the object positions. For these reasons and the high computational costs, only the cross and least squares estimation techniques will be considered during the combination with the Color modalities. In Table 4, the results of the detection method can be seen with some different configurations. The term before the hyphen in the *config* column de-

termines the estimation method, i. e., cross or the Least Squares (*LS*) approximation. Both these estimation methods have been combined with the *Hue* and color names (*CN*) modalities, to see which datasets produce the best results. The PCA method has not been included because of the reasons stated above, and produced no improvement over the other modalities.

5.4 DISCUSSION AND DRAWBACKS

While testing the dataset, some problems of the method have been discovered. This is partly because we are utilizing the method for object types that were not originally used, i. e., more textured and more similar in shape, and because of the quantization method introduced by LINE-MOD. This section will look at the different factors which influence the results presented in the previous sections. The quantization scheme, feature spreading and feature types will be analyzed and discussed, and some drawbacks of the method will be presented.

5.4.1 *Quantization*



(a) Coke can (b) Coke can template (c) Tea pack (d) Tea pack template

Figure 33: Comparison of Coke and Tea Pack. (a) and (c), are the reference images of the templates. (b) and (d), shows their respective templates in a visualization, note the similarity between the two objects

Objects that appear to be dissimilar can generate false positives using the default modalities and settings. Mainly due to the feature type and quantization scheme. This can be seen from a template example in Figure 33 (the figure uses the same coloring scheme as explained in Section 2.3.3). We can see that for both the Tea Pack and Coke, which appear to be of a different shape, the templates are quite similar. Both in position and feature distribution. To further motivate this

example, we can create a histogram of the template collections. Given a collection of templates:

$$\boldsymbol{T}^m = \left[T_0^m, \quad \ldots, \quad T_n^m \right] \tag{64}$$

the feature indexes can be summed into a bin h_i, while discarding the positions in the template:

$$h_i = \frac{1}{N} \sum_{T_i^m \in \boldsymbol{T}^m} \sum_{\{d, \boldsymbol{r}\} \in \boldsymbol{F}^m} \mathbb{I}(d = i) \tag{65}$$

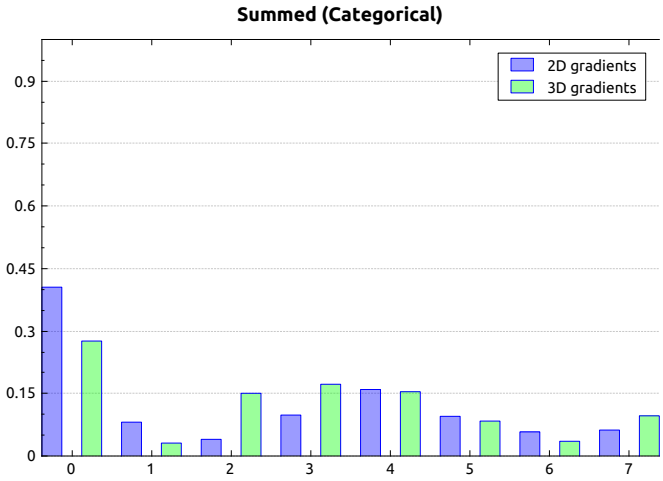
Where N is the *total* number of features and $\mathbb{I}(d = i)$ is the indicator function. The indicator function returns 1 if i and d have the same feature index, and 0otherwise. This provides an estimate for the frequency of a feature in the entire collection of templates per modality m, essentially transforming the set of templates into a histogram distribution. This is visualized in Figure 34.

Figure 34 gives an indication of the *distribution* of features, while 33 gives an indication of the distribution and the positions of the features in the template. Note that because both the 2D and 3D gradient features share a similar distribution over the templates that it makes the position of the feature in the template the differentiating factor. Figure 33, shows that for some templates the position of features in the template is almost equivalent, increasing the chance of a higher similarity between these object instances, thus increasing the chance of a false positive.

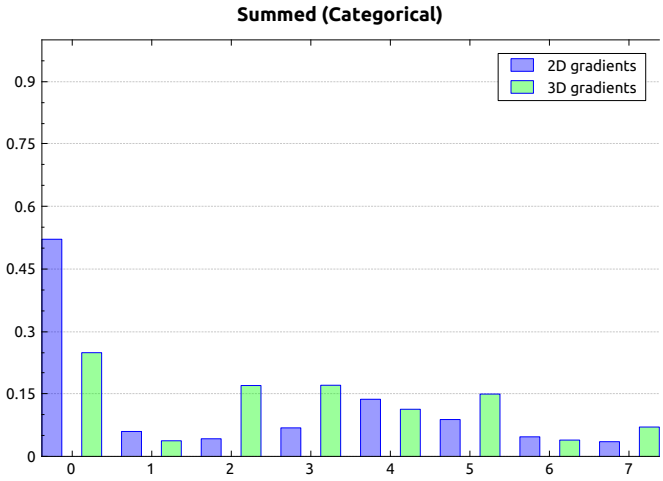
The same problem can occur with background environments, especially when regarding the 3D gradients. Figure 35 shows a cluttered scene with both the Tea Pack object and the Coke. The green boxes indicate wanted behavior, the red boxes indicate unwanted behavior. The quantization of the 3D gradients makes the Coke appear box-like, this is because only 8 orientations are available, which causes the quantization of normals of the cylindrical Coke and the box into mainly 2 bins. On the other hand, this is expected for the Tea Pack and the background box as these are of similar structure. These quantization issues can lead to false positives for structurally similar objects.

5.4.2 *Spreading*

The spreading of features can result in false positives being detected, mainly in highly textured areas. The 3D gradients were specifically chosen for this reason, namely to compensate for the fact that the 2D gradients can appear ambiguous. However, as we have seen from the last section this limitation can also apply to the 3D gradients. Spreading can make the problem more pronounced. For areas with



(a) Coke summed histogram



(b) Tea Pack summed histogram

Figure 34: Histogram of the collection of coke and Tea Pack templates. The x axis indicates the descriptor d bin index. The y axis shows the normalized frequency of the bin.

a lot of 2D gradients we have found that the spreading can cause a high similarity response for multiple objects.

Figure 36, shows that for objects with high textured area's the spreading can become significant. This results in objects typically being detected inside the textured object.

5.4.3 *Modality separability*

To create further insight into the results of the benchmark, we can take a look at how the different modalities influence the similarity between objects. This section will provide a deeper insight into which features differentiate the different objects in the dataset, and the influ-

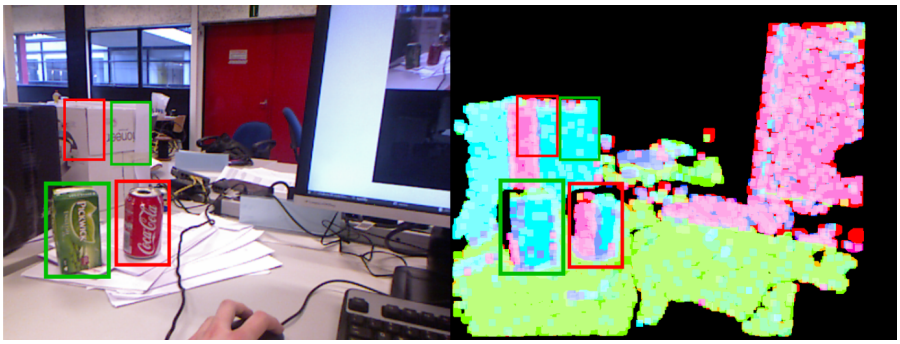
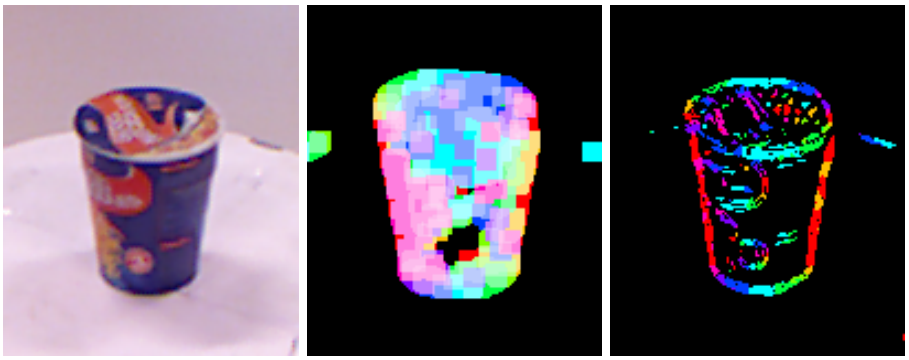


Figure 35: Coke and Tea pack, in a single scene. RGB input shown as reference on the left, 3D gradient $\mathcal{Q}$ image is shown on the right. Red boxes indicate unwanted behavior, which is caused by the quantization, such that round objects tend to look like edges of square objects. Green boxes indicate wanted behavior, both planar objects seem similar



(a) Reference RGB image (b) $\mathcal{J}$ for Noodles, $\mathcal{T} = 8$ (c) $\mathcal{J}$ for Noodles, $\mathcal{T} = 1$

Figure 36: Showing a detection frame for the Noodles object. (a) Reference RGB image. (b) and (c), $\mathcal{J}$ image with the default spreading parameter $\mathcal{T}$ and no spreading. The amount of white determines, the amount of gradients at that location

ence of these modalities on the actual similarity score will be made more clear.

5.4.4 *Cross-similarity*

In the original paper by Hinterstoisser the claim was made that it would be more *likely* that the three dimensional gradients could yield ambiguous similarities. We expect that this would hold for our dataset especially, as the objects are often cylindrical or rectangular. To investigate this hypothesis, we are going to introduce the concept of cross-similarity. After we have learned the objects using the method as described in Section 4.3.2, we can create a corresponding configuration (Section 4.3.3) to detect these objects. We then run the detection on the dataset, which was used to learn the objects. To find the cross-

similarity we consider the similarity of objects between themselves in this dataset, determining with a constrained detection, how similar object are to one another. The following steps are used:

- Filter out plane as was discussed in Section 4.3.1.1.
- Detect objects in the learning scene and determine location of maximum, for all objects in the dataset.
- If the bounding boxes of the annotation and the detection, meet the matching criteria (Equation 63), consider that location for the calculation of the cross-similarity.
- Should the bounding boxes not meet the matching criteria, then the object is not at the location of the maximum, in that case consider a window near the annotation position for the maximum similarity at that location.
- Average the similarity over the frames, essentially averaging for different object orientations.

The output is a matrix indicating which objects are similar to one another. An example is given in Figure 38, which shows the matrix with the cross-similarity of the objects displayed in the cells. This is a similarity measure between objects and provides information regarding the similarity between *objects* and not *object* and *background*, as we are only considering the positions of the annotations.

Intuitively, when looking at a *row* of the cross-similarity matrix, it displays how much the object in the row looks like the object in the column. When looking at a *column* we are considering how likely the other objects in the table will be seen when the object in the column is present in the scene. When these values are low, than the dataset contains objects which appear dissimilar to the detection system and vice versa. The values in the diagonal do not always equal 1 because of the learning process: if there is a loss of information, as is the case for Crackers and this results in a low detections than the template is not recorded. In this case, the object model will not contain the template for that angle, which results in missing templates for orientations, which subsequently causes lower scores during the detection phase for these orientations.

5.4.5 *Separability*

For our dataset it was found that due the nature of the objects, and due to the drawbacks described in the previous section, objects are often incorrectly classified. Figure 39, shows the cross-similarities for both the 2D and 3D gradients. Note, that these look as expected, the 3D gradients show more similarity between objects and are unable to differentiate between the cans, but also have high similarity for other cylindrical objects like peanut butter. The 2D gradients show a better separation of the data, especially for the stapler because the

contour is different from the other objects, as it is wider as opposed to higher. However, some of the ambiguities still remain, cup and peanut butter, for example, are quite similar because they are both small and cylindrical.

The effect that was described in Section 5.4.2 can also be observed from these figures. Looking at the row titled Noodles in Figure 39, the similarity for Noodles when matched with other object instances is shown, Noodles is dissimilar to these other objects. However, when looking at the column containing Noodles, we can see a higher similarity per object, these higher similarity scores are caused in part by the spreading, such that templates of other objects have high feature responses inside the Noodles object.

In some cases the non-symmetry of the cross-similarity can clearly be seen. Take garlic sauce and cup for example, from Figure 39, it can be noted that the cup is similar to the garlic sauce but not the other way around. This is probably because cup has less gradients and is smaller, and garlic sauce is the larger object and has more gradients to spread, increasing the feature response at that location thus increasing the response for objects that fit inside the Garlic Sauce template.

5.4.5.1 *Color modalities*

Figure 37: The stapler object

The cross-similarity can be calculated for the color modalities as well, Figure 40 shows the cross-similarity for both color modalities. We can observe that the Color modalities provide a higher separability for the dataset, as cross-similarity score are lower. Objects appear less similar at the annotation positions for these modalities, which make sense because similar shaped objects in this dataset often have a different color. This holds especially for the cans (Coke, Fanta etc.), because they are almost identical in contour and shape. However, this fails in the case of Fanta and Coke, these remain similar in spite of the added color modality. This is because Fanta was misclassified as being red in the majority of frames during learning, making the deci-

sion boundary between the two objects slim, and resulting in a higher cross-similarity score.

The stapler object, displayed in Figure 37, has been included to specifically show, that objects that have a different shape tend to produce well-separable data-sets, which it does for all modalities (Figure 40, Figure 39, and combined in Figure 38). This is one of the main differences with our objects and the objects used by the authors,

As can be seen in the cross similarity figures, adding an extra color modality provide some extra separability to the dataset. By using these feature types we are able to combine contour, shape and color into a single detector. Enabling the method to differentiate objects that are challenging to detect using just the original gradient modalities. In the next sections, we will explore the effect of the added modality during the actual matching process.

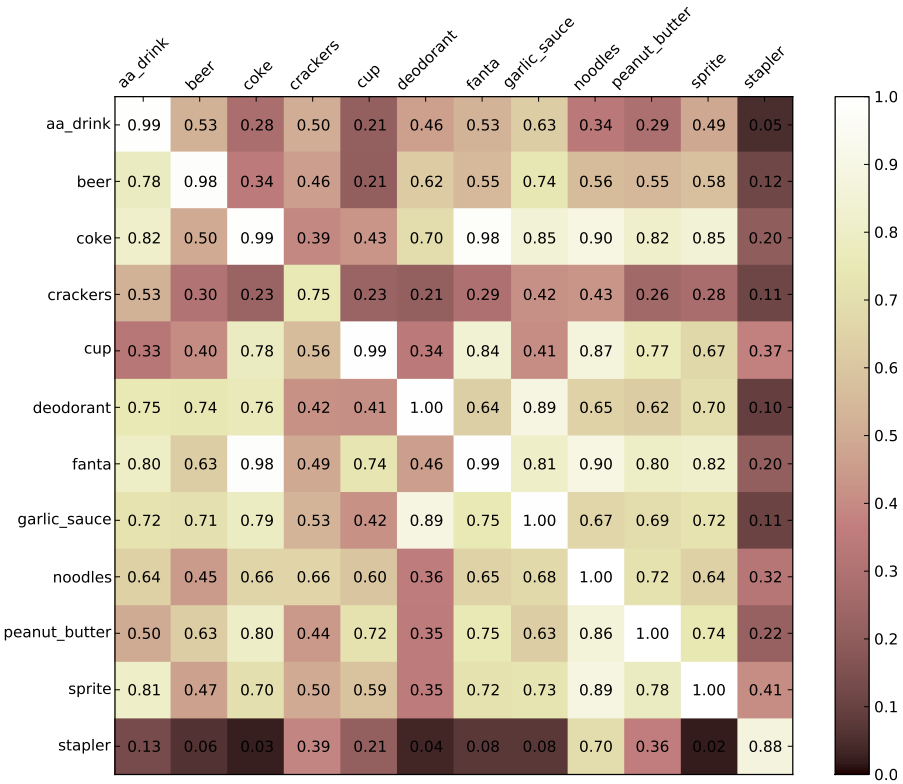


Figure 38: The cross similarity matrix. Darker cells indicate less similarity, lighter cells indicate a higher similarity. The diagonal contains the self similarity. Note that the measure is not symmetric.

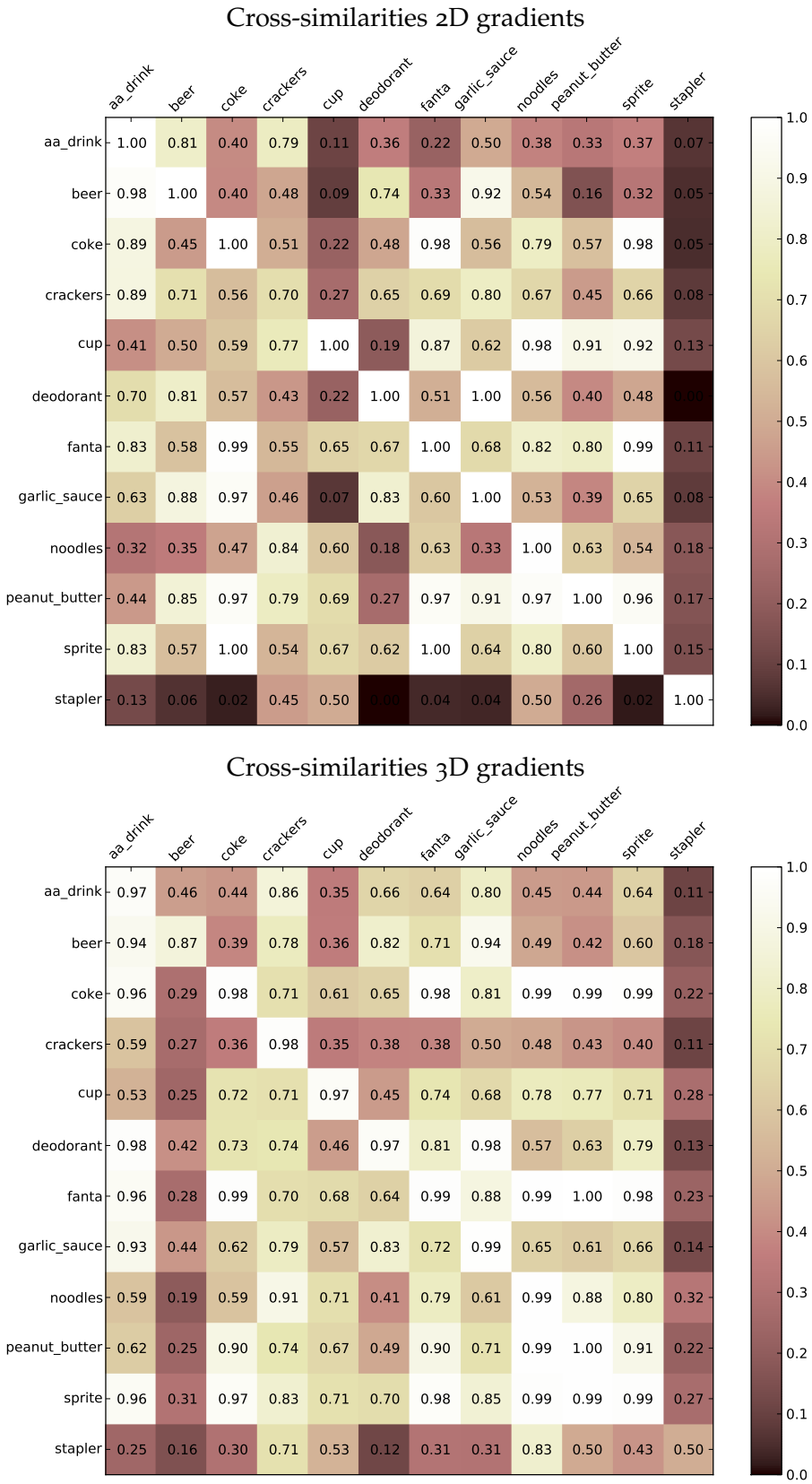
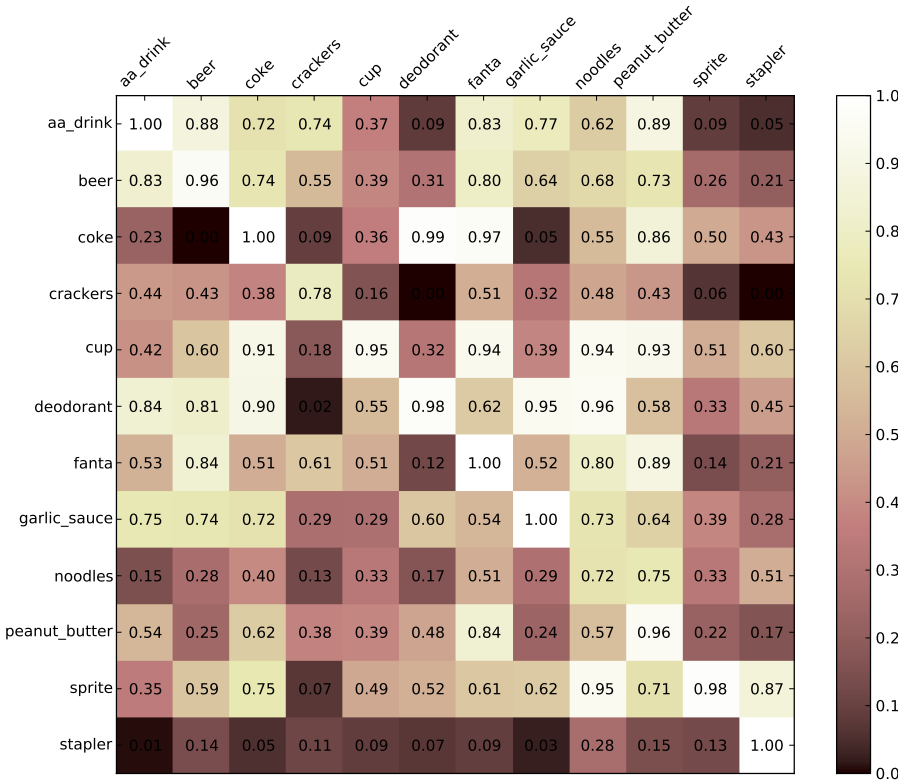
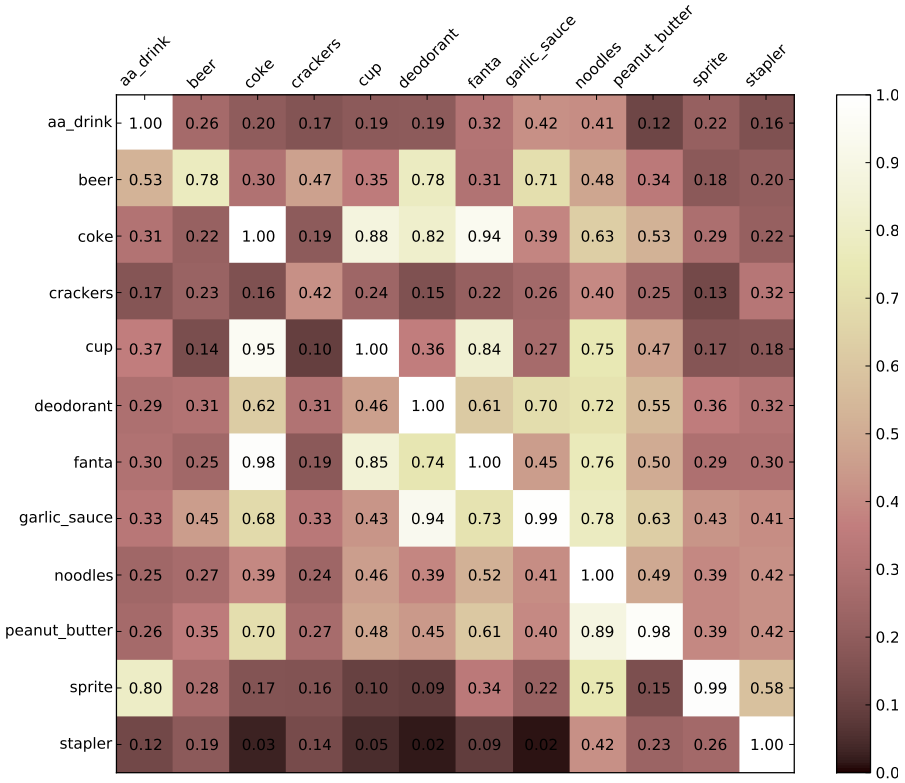


Figure 39: (a) The cross-similarities for the 2D gradient, (b) The cross-similarities for the 3D gradients



(a) Cross-similarities Hue



(b) Cross-similarities color names

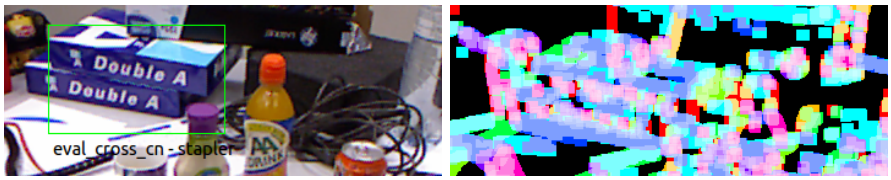
Figure 40: (a) The cross-similarities for the Hue gradient, (b) The cross similarities for the color names

5.4.6 *Influence of background*

In the previous sections results have been shown, which indicate how well the object detector performs on the dataset. Different normal estimation methods have been tested, as well as the two color descriptors introduced in this thesis. During the evaluation, a few other limitations of detecting these specific set of objects with the system were found. The system could not accurately detect the Noodles (Appendix B), with any configuration and was almost always misclassified as a sprite, which is similar in color. The Fanta and Coke, couldn't be discerned from each other as Fanta is incorrectly classified as being red in most of the frames.

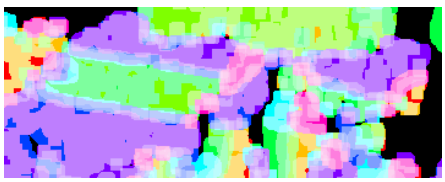
Both the color names and the hue modality have problems with white and black respectively. The color names modality, often detects white as blue or purple if there is some amount of blue in the image. The hue modality, finds large saturation values for black so that it is not filtered, resulting in the spreading of a large amount of features in the background that result in high feature responses in these regions.

Some objects, were falsely identified in the background, this has to do with the problem described in Section 5.4.1. Figure 41, illustrates this problem for an example case. The stapler, is a blue object with 2D gradients pointing in the y direction, and 3D gradients pointing either upwards or towards the camera. Because of the quantization, the gradients will be allocated in two bins. Even though the object is rounded, and does not possess a sharp corner like the background it is still wrongly identified at this location because of the quantization. Using the current quantized information, which provides in an insufficient amount of information, these kind of situations are hard to eliminate completely.

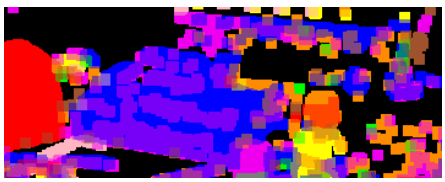


(a) False detection stapler

(b) $\mathcal{J}$ 2D gradients



(c) $\mathcal{J}$ 3D gradients



(d) $\mathcal{J}$ CN

Figure 41: (a) The false detection. (b), (c), (d) The quantized and spread images

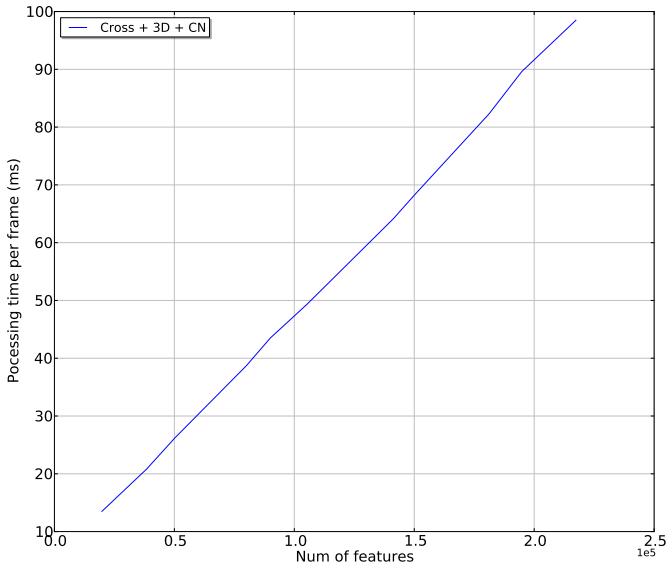


Figure 42: Running time of the detection phase, plots the runtime to feature correspondence, from which can be seen that the relation is almost linear.

5.5 RUNNING TIME PERFORMANCE

One of the main contribution of the original method was that it was very fast. When compared to the original, a performance increase has been obtained because of the parallel processing of the input frames per modality, this extension can reduce the processing time for the input images up to a third, in practice it reduces the input processing time to that of the slowest modality, which is illustrated in Table 5.

AMIGO has four computers, containing either Intel i5 or i7 processors, the more CPU intensive processes are run on the more powerful i7 processors. To test the results presented in this chapter we have used a second generation (Ivy Bridge) Intel i7-3630QM processor running at 2.4 Ghz, with 8 GB of ram.

Table 5 and Table 6 show the running times of the preprocessing of the images. These steps happen each frame, the processing of the 3D gradients generally take the longest time. From Table 6, because of the parallel processing of the input frames, the speed stays the same for the added modalities. The bottleneck in this case, is the modality that takes the longest time to process, which is the 3D modality. Note that both the Least Squares and Cross product approach take an almost equal amount of time. The PCA method takes the longest to process, the calculations involved are more complex than the other two methods. However, the PCA method does not lead to an improvement in the detection.

Figure 42, shows a plot of the processing time required to match a features in a template with an input image. Up to 250,000 features are

processed, the figure shows that the relation between the amount of templates and the detection runtime, is almost linear. This is expected as the detection only involves the summation of feature responses. The addition or removal of an additional modality does not change this linear relationship.

During testing we noted that depending on the dataset (color names requires more templates than hue), we are able to run the detection node at about 6 to 8 frames per second. The detection method, presented in this thesis, was also deployed during the RoboCup, during which 22 objects had to be detected. When using the RoboCup dataset the node runs at around 4 frames per second, using color names as a color type.

The running times of the different configurations are listed in Table 7. The time scales with the amount of templates used. The amount of templates depends on the learning phase, as templates are recorded and saved automatically depending on the threshold. Because color modalities add further distinction to different object angles (orientation of the object), using a color modality will automatically cause more templates to be recorded. The table shows that both color modalities, use more templates than the original method. Color names requires significantly more templates, this is because it change of orientation causes a larger change of color templates, when compared with the Hue modality. The decrease in average running time per frame decreases as the amount of templates increase, because the processing of the input images is a large constant factor.

CONFIGURATION	RUNTIME (MS)
HUE	54
COLOR NAMES	57

Table 5: Running times (ms) of the hue and color names extraction. Displayed running times are calculated an average frame basis for 200 frames.

CONFIG	3D	2D, 3D	2D, 3D, CN	2D, 3D, HUE
LS	72	72	74	73
CROSS	63	65	65	66
PCA	423	419	429	412

Table 6: Running times (ms) of different normal estimations routines. The columns display the normal estimation, and the combined modalities. Running times are calculated on an average frame basis for 200 frames.

CONFIG	# TEMPLATES	RUNTIME (MS)	MS / #TEMPLATE
LINE-MOD	914	36	0.039
CROSS-CN	3261	98	0.030
CROSS-HUE	1215	39	0.032
LS-CN	2352	72	0.031
LS-HUE	1266	41	0.032

Table 7: The processing times for the detection procedure (average per frame basis). Displayed for different configurations. The columns respectively display the configuration, number of templates, and processing time of the detection, and the processing time per template

5.6 CONCLUSIONS

From the results presented in this chapter and the quantitative evaluation done during the course of the project the following conclusions are drawn:

- The addition of the color modalities generally improve *precision*, but does not consistently improve *recall*. This is expected, because the detection system is better at differentiating objects using a color modality, but that does subsequently entail that the number of False Negatives is also decreased.
- The Color Names seem a promising addition to the detection system. However, in some cases the color names perform worse than the hue modality, as we have seen only some measure of distinctiveness between objects is needed, as this increases the discriminative power of the method. In most cases the hue modality provides enough discriminative power, whilst being less prone to false negatives, resulting higher recall scores.
- Adding a color modality is a trade-off, while it does improve precision, it also increases the time necessary to detect objects. For some objects, like a Coke and Fanta can, there may be no alternative in the current system. Because when using only 2D and 3D gradients there is not enough discriminative power.
- The Least Squares normal approximation is the most accurate when only 2D and 3D gradients are used. However, by adding a color modality, this advantage can become less significant. Because false positives in the background, which occur when using the cross estimation, are filtered out. The PCA method seems unfeasible to use, as it takes the longest time to compute and does not subsequently improve on the results.
- The filtering done by the world model is important, because this drastically improves the precision. A limitation of the current method, is that a large amount of objects still seem similar, causing them to pass the threshold, and can only be distinguished when considering both their position and similarity.

6

CONCLUSIONS AND FUTURE WORK

6.1 INTRODUCTION

In this last chapter a short discussion will given about the suitability of the described system for detecting objects with AMIGO. The evaluation and utilization of the system on the RoboCup, has made it possible to form a number of conclusions regarding the system. In this thesis we have presented an object detection module based on an existing detection framework by Hinterstoisser *et al.* [1], the goal was to integrate this system into the robot and find out if it could be utilized successfully. The next sections answers the Research Questions and lists Future Work.

6.2 RESEARCH QUESTIONS

6.2.1 *Research Questions*

LINE-MOD was created for detecting texture-less objects, how well does LINE-MOD generalize to objects used in the RoboCup competition? What are the current deficiencies and how could these be tackled?

LINE-MOD LINE-MOD is especially suited for detecting texture-less objects in relatively controlled environments, one of the problems regarding our application was the fact that textured objects sometimes caused problems because of the spreading. As well as the fact that household objects tend to resemble each other in shape. To alleviate this problem we have introduced a color modality, which helps differentiate objects more strongly. The world model also helps in filtering out the false positives that occur during detection. However, for larger datasets the separability will become more important. As we have seen in the last chapter less separability can cause more false positives. The objects used on the RoboCup are also often of a similar shape, in this case the 2D and 3D modalities will not be sufficient to describe these objects and the method falls short. Although color can provide some benefit, in some cases another approach will have to be taken. Another drawback of the method is that sometimes the background can resemble an object too closely, e. g.there is no way that

the current implementation of the object detector could differentiate between a cube that has been learned with a front view perspective and a similar colored box in the background. Therefore, I think these modalities are insufficient and should be combined with a texture based framework, ideally also outputting similarity maps so that the methods can be combined.

IMPLEMENTED OBJECT DETECTOR The combined gradient and color modalities introduced in this thesis result in better precision score for the dataset, when opposed to the original method. However, color names can be too strict, creating a larger number of false negatives, but usually do result in better precision scores when opposed to the Hue modality. Thus, color names are a promising addition the original modalities, but there should be some research into the best utilization of the color names, as they are more sensitive to the translation and scaling of the objects than other features. The original 3D gradient extraction routine, introduced Hinterstoisser in *et al.* [1], is the most appropriate in most cases. However, the choice of the cross or Least Squares normal routine does not affect the performance by a large amount when combined with the color modality. The implemented system does not include a method that integrates scaling into the infrastructure, in the original method image pyramids were used to incorporate scaling into the detection method. This was specifically not included, because AMIGO does can position himself at objects at a particular angle, and the locations are mostly known. Additionally, the intuition was also held that if there is depth information available, then scaling over the entire scale space should not be necessary. Unfortunately, because of time constraints, a working solution has not been integrated into the system.

Is LINE-MOD a suitable candidate for the integration into the AMIGO infrastructure and could it be used for everyday tasks? Additionally, is LINE-MOD suitable for use in general service robotics, and not just the AMIGO project?

AMIGO LINE-MOD is a suitable candidate for integration into the AMIGO project. However, the system alone will prove insufficient especially because of our applications. It should be considered a low-level fast detection module, that could perhaps eventually be used to generate object hypothesis that a more complex system can verify later. During our experimentations it proved to work better than the existing methods that have been used on the AMIGO project, thus it will continue to be a part of the AMIGO eco-system in the foreseeable future.

GENERAL SERVICE ROBOTS In the general service robot domain, the focus should not only be on the detection of objects, but also

regarding the integration between systems, something that has been elaborated upon on in Chapter 4. I believe it is important to eventually integrate contextual information into perception architectures. In the current method the communication is one-way. However, perception should start utilizing information gathered by the robot to be able to recognize objects more accurately. If the robot has an indication of the location (e.g. a table in an office) and other sources of information, so that the robot has an idea for what and where it should be looking, it would definitely simplify the problem. Thus, I believe the method to be accurate enough to be utilized together with other perception modules on AMIGO. However, future focus should not only be the creation of more accurate stand-alone perception modules, but also on the creation of a more general cognitive architecture.

6.3 FUTURE WORK

- COMBINING PROBABILITIES The current detection method makes use of similarities, these are similar to probabilities but do not take any prior information into account and do not adhere to probabilistic axioms and rules. An interesting question is if these similarities could be replaced by a probabilistic interpretation. Specifying prior probabilities indicating the chance of a detection, as well as posteriors that specify how likely an object hypothesis is. Perhaps, a probability could even be formulated that specifies the probability of a misclassification. Using these probabilities a more informed detection result could be presented to the World Model. This is desirable as this is an inherently probabilistic system.

- METHOD CASCADE LINE-MOD has problems detecting highly textured objects. The addition of a more complex (albeit slower) texture based method, would provide more accuracy for objects found in household settings. An extenstion would be to arrange more methods in a cascade, where LINE-MOD does a first fast detection and generates object hypotheses. Another method could then provide a more accurate measurement without having to do an exhaustive image search, reminiscent of other cascaded architectures [18].

- WEIGHTED MODALITIES The Equation 22, weights the modalities for the Object Models equally. This is not always correct, as in some cases the detections should be more confident for a certain modality. It would be interesting if the modalities could be weighted per object or per scene information. This could be done by defining an inter-class similarity that estimates which modality has the largest chance of a misclassification, or by estimating the chance of a false positive. Perhaps by looking at

the information entropy of the spread image: the larger the spreading, the larger the entropy becomes. So that the probability to find a positive match given a descriptor d with a maximum value (wich means that all gradient orientations are represented), could be seen as a uniform distribution, which maximizes entropy. It would be interesting to see if an adaption could be made to include this information, and if it would provide more accuracy.

- TEMPLATE TREES The current detection method makes use of templates for performing the detection of objects. During the evaluation it has been shown that the method will still run in real-time for up to 20 objects. However, initiatives like the Roboearth project [62], are working on the creation of that databases and methods such that massive amounts of information can be shared between robots. This will place a larger requirement on systems that aim to detect objects in real-time. Previous methods ([63], template tree approach) have made use of tree structures to accelerate the matching of templates. The object detector on AMIGO could make use of these methods, if there is a way of comparing templates with each other a-priori so that a split criterium can be created, or an exemplar template can be specified, a tree based approach becomes feasible. As of now, the amount of templates determines the run-time of the current system, so an optimization in this area seems likely to succeed.

- 3D FEATURES AND QUANTIZATION During the course of this thesis, the 3D features that are used consisted solely of normal vectors. As we have seen in the previous section, these are not always the most descriptive features. Recently, with the advent of the Kinect, research into new feature types has been done [32]. To integrate these into the framework, the biggest obstacle is that most features even small features like Local Binary Patterns (LBP) require more than 8 feature orientations. Because the object detector is limited to a single byte, with 8 unique (unspread) orientations, this places limitations on the feature that can be used. Nevertheless, it will be interesting to research the possibility of employing different features within the framework. The Quantization of the 3D normals is not optimal. Other research based on the compression of normals [64], provide and test different normal quantization schemes that preserve the largest amount of information. An especially interesting scheme uses the geodesic sphere to provide equally spaced regions to perform the quantization. Unfortunately, a problem with this approach is that a half of a geodesic sphere (geodesic dome), cannot be split up into 8 equally spaced regions (number of feature orientations). Still it would be interesting to see if there is

a workaround that would perform better. These limitations are hard to overcome, it could be therefore be the case that a byte will simply not be enough storage space. The SSE instructions operate on single bytes (char) for features, and on double bytes (short) for similarities. It would be interesting to see whether there is an direct improvement when the systems works on two bytes instead. As this would double the amount of unique quantized orientations that the detection module could handle. It could also open the possibility for more complex feature types.

Part IV

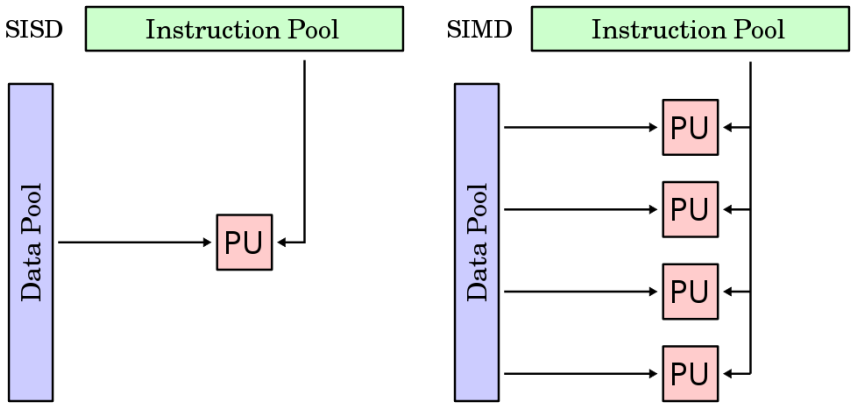
APPENDIX

A

SSE INSTRUCTION SET

A.1 INTRODUCTION

Because a large part of the high performance of the system, is thanks to the use of SSE instruction, it seemed prudent to give an introduction about the use of SSE instructions. This appendix will mainly explain the high level concepts and a small code example which showcases a simple SSE operation. Note that some knowledge is assumed of C++ and the reader should be comfortable with lower level memory operations and pointer arithmetic. Mainly, because the support that is offered by C++ for SSE is targeted at an audience working with high performant systems.



(a) SISD (b) SIMD

Figure 43: The SISD (a) and SIMD (b) architecture are illustrated. *Source*[1]

The Streaming Single Instruction Multiple Data (Streaming SIMD or SSE) is an extension to the standard x86 instruction set. SSE instructions effectively enable the parallel processing of larger amounts of data than was possible with the regular x86 instruction set. SIMD is different from Single Instruction Single Data (SISD), which processes a single unit of data using a single instruction. Parallelization in the

SISD case is provided by multiple processors, and the pre-fetching of data and instructions. The difference is indicated by Figure 43.

The SIMD instructions have multiple Processing Units (PU) which can process the data in parallel. The data to be processed is stored in registers, these registers can store up to 16 bytes or 128 bits. Hence, using C-style nomenclature; up to 16 shorts, 4 floats or integers, and 2 doubles can be processed using a single instruction. These sets of primitives are referred to as SIMD vectors. Different instructions for different vector types are defined, similarly as to regular x86 instructions, which can be directly issued to the processor using assembly. Processors must adhere to the SSEx standard, where x is the version number, which is at 4 as of writing. Most reasonably modern (2003 onwards) processors by both AMD and Intel provide at least SSE2 capabilities. To enhance the performance of an algorithm with SSE instructions, the programmer has to consider the problem on a conceptually different level. Because most algorithms are not easily parallelizable, some work has to be done to convert an existing solution to a parallel or, in SSE terms, a vectorized format. Note that the compiler is able to vectorize simple numeric operations automatically. However, automatic vectorization is still an open problem and currently there is no way to automatically vectorize an entire complicated procedure.

A.1.1 *SSE for image processing*

SSE are especially useful in image processing and therefore also to some extent in computer vision, considering the fact that images are mostly large matrices of data. Processing this data can take up a large amount of time in an algorithm. By utilizing SSE, one can significantly speed up the processing of an image using just low-level optimizations. In the method discussed in this thesis, the matching of a single template T on a single binary input image $\mathcal{Q}$, takes up to 9 milliseconds unoptimized while the SSE version takes just 0.3 milliseconds. Effectively speeding up the operation by 300%. Unfortunately, some problems are hard to convert to an SSE version or would perhaps require a totally different approach. Also some amount of tuning can be required to obtain the most performant version, both these problems also hold for inherently similar GPU computing. Some work is being done to ease this process, for example the Halide language provides a high level interface which selects appropriate schedules and parameters [65].

1 Courtesy of wikipedia: http://en.wikipedia.org/wiki/File:SISD.svg and http://en.wikipedia.org/wiki/File:SIMD.svg

A.2 USING SSE INSTRUCTIONS

This section discussed how to utilize SSE instructions. In the last section, SSE instructions were essentially introduce as assembly instructions. However, compilers like GCC, Visual C, and Clang provide SSE intrinsics; intrinsics are a collection of structures and methods designed that encapsulate the corresponding assembly instructions. Essentially simplifying the production of SSE code. There are intrinsics for loading into SSE registers and performing operations on these registers.

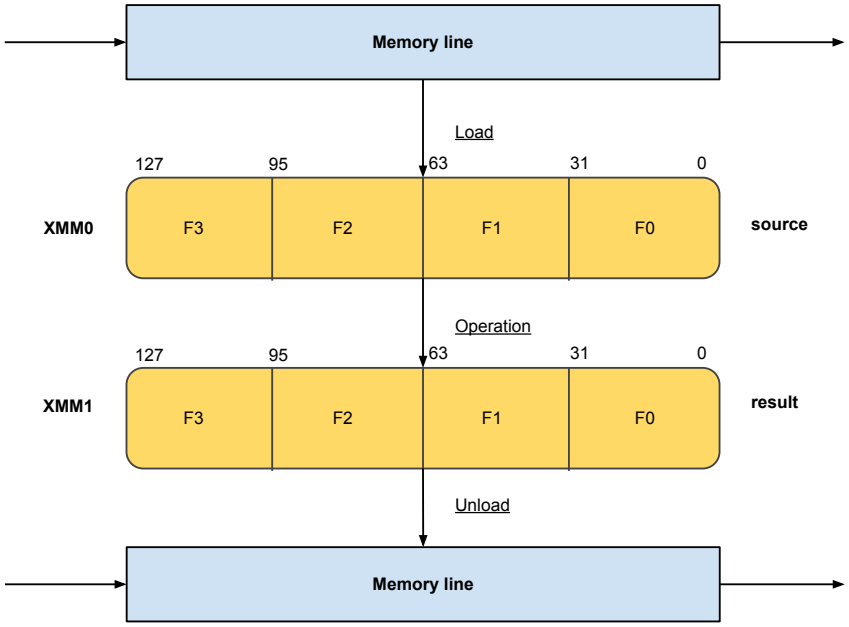


Figure 44: Two corresponding SSE registers, with a source and destination register. SSE registers are typically labeled as XMM registers. Data is loaded from the memory line into a source register, an operation is performed into a resulting register, from which the data can be loaded back into the memory line.

Figure 44 shows the pipeline for processing SSE instructions. Data is loaded into a XMM register by an intrinsic, multiple operations can be performed on these registers, and data can subsequently be extracted from these registers. Data can only be loaded in 16 bytes, in the figure the registers are split up into four blocks, most operations are still defined for floats, as historically the SSE instructions have been used in combinations with float vectors, and a lot of the operations are still defined as such. The elements of the register are processed in parallel, which means processing 16 byte values in parallel.

A.2.1 *Memory alignment*

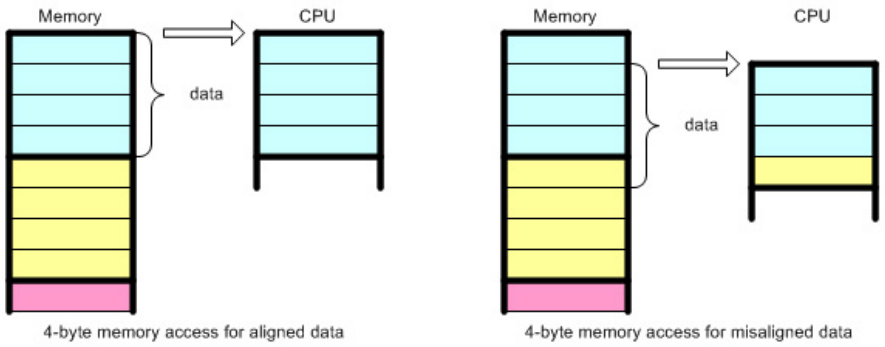


Figure 45: Illustrates how the CPU accesses a 4-byte chuck of data with 4-byte memory access granularity. *Source*[2]

When working with SSE it's best to utilize the memory in blocks of 16 bytes, because of the size of the SSE register. Data that is loaded into the register, from which the source is 16 byte aligned is appropriately named: aligned memory. Most operating systems, including those adhering to the POSIX[3] standard, provide memory aligned allocation options. Compiler intrinsics support both aligned and unaligned loads, but especially in older SSE standards (SSE & SSE2), aligned loads are significantly faster. An unaligned load can essentially be seen as a cache miss, where an extra load an shift have to be done to get the data into the register. This is illustrated in Figure 45.

A.2.2 *Small example*

A small example is given below, in this piece of code we show a simplified version the or'ing of the binary image $\mathcal{J}$, as has been explained in Section 2.4. Two ways of loading memory will be shown, an aligned version and an unaligned version. A short function will be shown that or's a shifted version of the same image.

```
1  #include <pmmintrin.h>
2
3  void orUnaligned(const void* src, void* dst)
4  {
5      //Load into register a, unaligned
6      __m128i val = _mm_lddqu_si128((const __m128i*) (src));
7      //Load into register b, unaligned
8      __m128i dst_ptr = _mm_lddqu_si128((const __m128i*) (dst));
```

2 Courtesy of Song Ho Ahn: http://www.songho.ca/misc/alignment/dataalign.html

3 Portable Operating System Interface, provides a standard between operating systems.

```
9
10      //Perform bitwise or
11      dst_ptr = _mm_or_si128(dst_ptr, val);
12
13      //Store back into memory, unaligned
14      _mm_storeu_si128((__m128i*) (dst) , dst_ptr);
15  }
```

The function specified above, takes two void pointers as input. The

`__m128i`

type, is a placeholder for an SSE register, replaced by C++ compiler to SSE Assembly instruction. By casting the memory pointer to a SSE type, we are saying that we want to refer to this as an XMM registry. Because the data is unaligned we have to load the XMM vector into a registy by calling:

`_mm_lddqu_si128((const __m128i*) (src))`

Which takes an XMM registry pointer as input. The actual XMM vector is returned, and we perform the bitwise OR by calling:

`_mm_or_si128(dst_ptr, val)`

. Finally, we store the vector back in the original memory location.

```
1   void orAligned(const void* src, void* dst)
2   {
3       //Use cast to load into register a aligned
4       const __m128i* src_ptr = (const __m128i*)(src);
5       //Use cast to load into register b aligned
6       __m128i* dst_ptr = (__m128i*)(dst);
7
8       //Perform or, and dereference to store values
9       //into memory
10      *dst_ptr = _mm_or_si128(*dst_ptr, *src_ptr);
11  }
```

The above function, is similar to the first. The big difference is that we do not need to load the values into the registry explicitly, the cast is sufficient and the memory is loaded automatically. The same thing holds for the last line:

`*dst_ptr = _mm_or_si128(*dst_ptr, *src_ptr);`

which stores the value automatically by dereferencing and assigning the XMM vector. So the alignment of the memory becomes crucial either when dereferencing the `src_ptr` and `dst_ptr`, or when performing certain SSE operations that expect aligned memory.

```
1  void orImageJ(J& image)
2  {
3      //Pointer at (x:0, y:0)
4      char* src_ptr =
5          image.pointerAt(0,0);
6
7      //Pointer at (x:0, y:1)
8      char* dst_ptr =
9          image.pointerAt(0,1);
10
11     size_t size = image.size() - image.row_size();
12     for(int i = 0; i < size; i+=16, src_ptr+=16, dst_ptr+=16)
13     {
14         //Check if aligned to 16 memory
15         if((uintptr_t) src_ptr % 16 == 0 &&
16            (uintptr_t) dst_ptr % 16 == 0)
17           orAligned(src_ptr, dst_ptr);
18         else
19           orUnaligned(src_ptr, dst_ptr);
20     }
21 }
```

The final function performs on operation on the image J, this describes a single step in the spreading routine. Suppose that we have a class `J`, which is basically an image/matrix of bytes that provides functionality to obtain a pointer at an image location. In the function, we are going to use the SSE bitwise OR operation on a shifted version of the Image. Note that for simplicity we do not do any bound checking. On line 5 and 9 we obtain the source and destination pointers. Because the data can be either aligned or unaligned we run the following check:

```
(uintptr_t) src_ptr % 16 == 0 && (uintptr_t) dst_ptr % 16 == 0
```

which interprets the pointers as integers, and checks if it falls on the 16 byte boundary. Should this be the case we can use the aligned OR:

```
orAligned(src_ptr, dst_ptr);
```

Otherwise we use the unaligned version:

```
orUnaligned(src_ptr, dst_ptr);
```

A.3 FURTHER READING

Note that the above example is a bit contrived. The intrinsic types support direct pointer operations that can be faster than doing the

cast or load each time. For instance, we could have written the aligned function as:

```
typedef unsigned int uint;

void orAligned(const void* src, void* dst, uint size)
{
    //Use cast to load into register a aligned
    const __m128i* src_ptr = (const __m128i*)(src);
    //Use cast to load into register b aligned
    __m128i* dst_ptr = (__m128i*)(dst);

    for(uint i = 0; i < size; i+=16, src_ptr++, dst_ptr++ )
    {
      //Perform or, and dereference to store values
      //into memory
      *dst_ptr = _mm_or_si128(*dst_ptr, *src_ptr);
    }
}
```

Because we know the memory is aligned, we can use direct incrementation of the pointers to move the next 16 bytes into the registry. Which could speed up the process even further. For the interested reader there are some good resources on SSE. Especially the MSDN guide, which is also suitable for GCC development[4]. Other suitable tutorials can also be found online. An article on general memory architecture for programmers can be found at: http://lwn.net/Articles/250967/

4 http://msdn.microsoft.com/en-us/library/26td21ds(v=vs.100).aspx

B

OBJECT DATA SET

B.1 OBJECTS

The data set listed below is a custom created data set. Mainly to test how well the current detection system generalizes to objects typically encountered by AMIGO.Objects are also selected to be graspable, because AMIGO and other robots in service environment settings need to interact with the objects. The object instances are listed below:

B.1.1 *Object instances*

1. aa_drink
2. beer
3. coke
4. crackers
5. cup
6. deodorant
7. fanta
8. garlic_sauce
9. noodles
10. peanut_butter
11. sprite
12. stapler

The data set which can be made available on request, uses these object labels and corresponding numbers to differentiate the objects in the recordings. Three unidentified objects were added, to see if the detection system would confuse these with the true positives:

- cup_small

- tape
- tea_box

Objects have been categorized in categories, mainly because this is also done during the RoboCup competitions, and because some categories contain more similar objects than others:

- Drinks:
  - aa_drink
  - beer
  - coke
  - fanta
  - sprite
- Food:
  - noodles
  - peanut_butter
  - crackers
  - garlic_sauce
- Miscellaneous:
- deodorant
- stapler
- Unidentified:
  - cup_small
  - tape
  - tea_box

Figure 46, shows the object dataset. All objects, bounded by a green bounding box are required to be detected by the object detector. Objects bounded by a red bounding box are considered unknown.

B.1.2 *Locations*

The TU/e robotics lab is an area where AMIGO currently operates, most experiments are currently being done inside the lab. The different locations inside the lab that have been used during the evaluation phase, are listed below:

1. Left_table
2. Middle_table
3. Cupboard
4. Bed_table

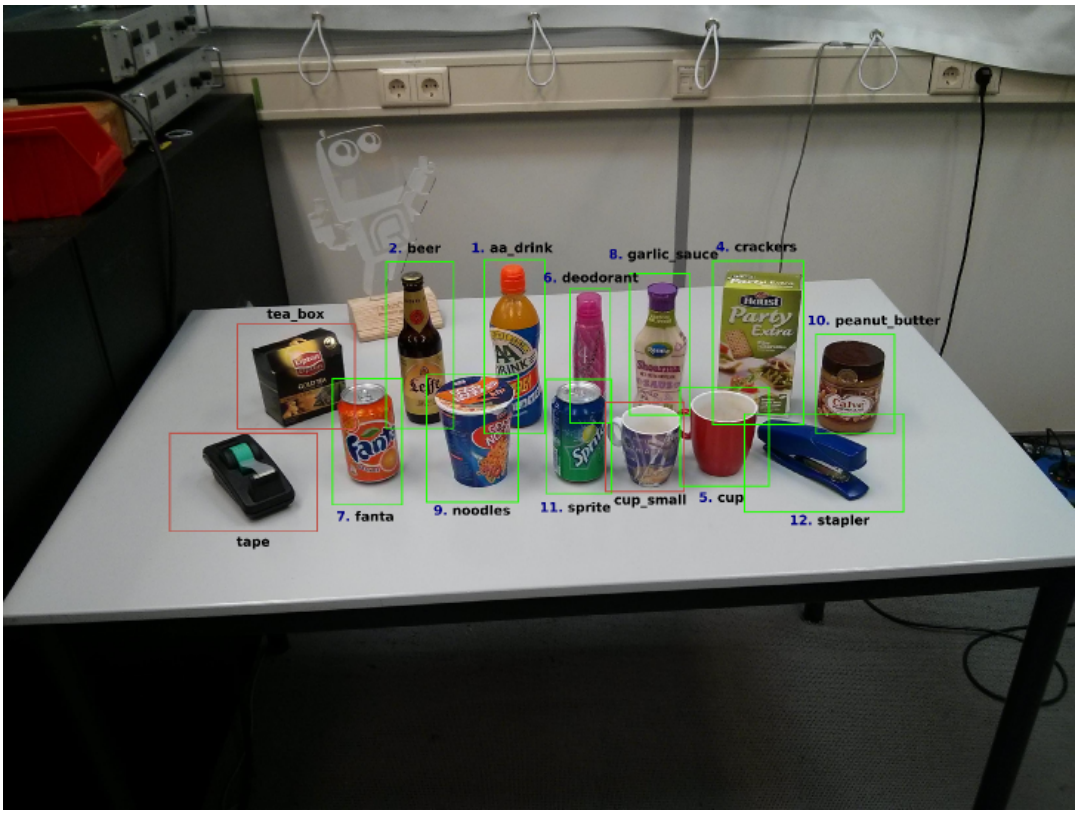


Figure 46: The object dataset, the identified objects are bounded by a green bounding box. The unidentified objects are bounded by a red bounding box.

BIBLIOGRAPHY

[1] S. Hinterstoisser, C. Cagniart, S. Ilic, P. Sturm, N. Navab, P. Fua, and V. Lepetit, "Gradient response maps for real-time detection of textureless objects," *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 34, no. 5, pp. 876–888, 2012.

[2] T. Wisspeintner, T. van der Zant, L. Iocchi, and S. Schiffer, "RoboCup@Home: Scientific competition and benchmarking for domestic service robots," *Interaction Studies: Robots in the Wild*, vol. 10, pp. 392–426, Dec. 2009.

[3] S. Cousins, "Ros on the pr2 [ros topics]," *Robotics & Automation Magazine, IEEE*, vol. 17, no. 3, pp. 23–25, 2010.

[4] M. Quigley, K. Conley, B. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, and A. Y. Ng, "Ros: an open-source robot operating system," in *ICRA workshop on open source software*, vol. 3, 2009.

[5] J. Shotton, T. Sharp, A. Kipman, A. Fitzgibbon, M. Finocchio, A. Blake, M. Cook, and R. Moore, "Real-time human pose recognition in parts from single depth images," *Communications of the ACM*, vol. 56, no. 1, pp. 116–124, 2013.

[6] M. Everingham, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman, "The PASCAL Visual Object Classes Challenge 2012 (VOC2012) Results." http://www.pascal-network.org/challenges/VOC/voc2012/workshop/index.html.

[7] L. G. Roberts, "Machine perception of three-dimensional solids," tech. rep., DTIC Document, 1963.

[8] R. O. Duda and P. E. Hart, "Use of the hough transformation to detect lines and curves in pictures," *Communications of the ACM*, vol. 15, no. 1, pp. 11–15, 1972.

[9] D. H. Ballard, "Generalizing the hough transform to detect arbitrary shapes," *Pattern recognition*, vol. 13, no. 2, pp. 111–122, 1981.

[10] J. Lewis, "Fast normalized cross-correlation," in *Vision interface*, vol. 10, pp. 120–123, 1995.

[11] A. K. Jain, Y. Zhong, and S. Lakshmanan, "Object matching using deformable templates," *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 18, no. 3, pp. 267–278, 1996.

[12] Y. Freund and R. E. Schapire, "A decision-theoretic generalization of on-line learning and an application to boosting," *Journal of computer and system sciences*, vol. 55, no. 1, pp. 119–139, 1997.

[13] L. Breiman, "Random forests," *Machine learning*, vol. 45, no. 1, pp. 5–32, 2001.

[14] C. Cortes and V. Vapnik, "Support-vector networks," *Machine learning*, vol. 20, no. 3, pp. 273–297, 1995.

[15] D. P. Huttenlocher and S. Ullman, "Object recognition using alignment," in *Proceedings of the 1st International Conference on Computer Vision*, pp. 102–111, 1987.

[16] D. G. Lowe, "Object recognition from local scale-invariant features," in *Computer vision, 1999. The proceedings of the seventh IEEE international conference on*, vol. 2, pp. 1150–1157, Ieee, 1999.

[17] Z. Kalal, K. Mikolajczyk, and J. Matas, "Tracking-learning-detection," *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 34, no. 7, pp. 1409–1422, 2012.

[18] P. Viola and M. J. Jones, "Robust real-time face detection," *International journal of computer vision*, vol. 57, no. 2, pp. 137–154, 2004.

[19] P. F. Felzenszwalb, R. B. Girshick, D. McAllester, and D. Ramanan, "Object detection with discriminatively trained part-based models," *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 32, no. 9, pp. 1627–1645, 2010.

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

[21] B. Leibe, A. Leonardis, and B. Schiele, "Combined object categorization and segmentation with an implicit shape model," in *Workshop on Statistical Learning in Computer Vision, ECCV*, pp. 17–32, 2004.

[22] J. Gall, A. Yao, N. Razavi, L. Van Gool, and V. Lempitsky, "Hough forests for object detection, tracking, and action recognition," *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 33, no. 11, pp. 2188–2202, 2011.

[23] S. Hinterstoisser, V. Lepetit, S. Ilic, P. Fua, and N. Navab, "Dominant orientation templates for real-time detection of texture-less objects," in *Computer Vision and Pattern Recognition (CVPR), 2010 IEEE Conference on*, pp. 2257–2264, IEEE, 2010.

[24] Z. Zhang, "A flexible new technique for camera calibration," *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 22, no. 11, pp. 1330–1334, 2000.

[25] K. Lai, L. Bo, X. Ren, and D. Fox, "Rgb-d object recognition: Features, algorithms, and a large scale benchmark," in *Consumer Depth Cameras for Computer Vision: Research Topics and Applications* (A. Fossati, J. Gall, H. Grabner, X. Ren, and K. Konolige, eds.), pp. 167–192, Springer, 2013.

[26] K. Lai, L. Bo, X. Ren, and D. Fox, "Detection-based object labeling in 3d scenes," in *IEEE International Conference on on Robotics and Automation*, 2012.

[27] K. Lai, L. Bo, X. Ren, and D. Fox, "Sparse distance learning for object recognition combining rgb and depth information," in *IEEE International Conference on on Robotics and Automation*, 2011.

[28] M. Sun, G. Bradski, B.-X. Xu, and S. Savarese, "Depth-encoded hough voting for joint object detection and shape recovery," in *Computer Vision–ECCV 2010*, pp. 658–671, Springer, 2010.

[29] S. Izadi, D. Kim, O. Hilliges, D. Molyneaux, R. Newcombe, P. Kohli, J. Shotton, S. Hodges, D. Freeman, A. Davison, *et al.*, "Kinectfusion: real-time 3d reconstruction and interaction using a moving depth camera," in *Proceedings of the 24th annual ACM symposium on User interface software and technology*, pp. 559–568, ACM, 2011.

[30] S. Prince, *Computer Vision: Models Learning and Inference*. Cambridge University Press, 2012.

[31] C. Redondo-Cabrera, R. J. López-Sastre, J. Acevedo-Rodríguez, and S. Maldonado-Bascón, "Surfing the point clouds: Selective 3d spatial pyramids for category-level object recognition," in *Computer Vision and Pattern Recognition (CVPR), 2012 IEEE Conference on*, pp. 3458–3465, IEEE, 2012.

[32] L. Bo, X. Ren, and D. Fox, "Depth kernel descriptors for object recognition," in *Intelligent Robots and Systems (IROS), 2011 IEEE/RSJ International Conference on*, pp. 821–826, IEEE, 2011.

[33] P. Viola, J. Platt, C. Zhang, *et al.*, "Multiple instance boosting for object detection," *Advances in neural information processing systems*, vol. 18, p. 1417, 2006.

[34] J. Shotton, J. Winn, C. Rother, and A. Criminisi, "Textonboost: Joint appearance, shape and context modeling for multi-class object recognition and segmentation," in *Computer Vision–ECCV 2006*, pp. 1–15, Springer, 2006.

[35] C. J. Burges, "A tutorial on support vector machines for pattern recognition," *Data mining and knowledge discovery*, vol. 2, no. 2, pp. 121–167, 1998.

[36] V. Lepetit, P. Lagger, and P. Fua, "Randomized trees for real-time keypoint recognition," in *Computer Vision and Pattern Recognition, 2005. CVPR 2005. IEEE Computer Society Conference on*, vol. 2, pp. 775–781, IEEE, 2005.

[37] J. Santner, C. Leistner, A. Saffari, T. Pock, and H. Bischof, "Prost: Parallel robust online simple tracking," in *Computer Vision and Pattern Recognition (CVPR), 2010 IEEE Conference on*, pp. 723–730, IEEE, 2010.

[38] O. Barinova, V. Lempitsky, and P. Kholi, "On detection of multiple object instances using hough transforms," *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 34, no. 9, pp. 1773–1784, 2012.

[39] B. Alexe, T. Deselaers, and V. Ferrari, "What is an object?," in *Computer Vision and Pattern Recognition (CVPR), 2010 IEEE Conference on*, pp. 73–80, IEEE, 2010.

[40] D. Hoiem, A. A. Efros, and M. Hebert, "Putting objects in perspective," in *Computer Vision and Pattern Recognition, 2006 IEEE Computer Society Conference on*, vol. 2, pp. 2137–2144, IEEE, 2006.

[41] A. Lehmann, B. Leibe, and L. Van Gool, "Feature-centric efficient subwindow search," in *Computer Vision, 2009 IEEE 12th International Conference on*, pp. 940–947, IEEE, 2009.

[42] C. Dubout and F. Fleuret, "Exact acceleration of linear object detectors," in *Computer Vision–ECCV 2012*, pp. 301–311, Springer, 2012.

[43] D. M. Blei, A. Y. Ng, and M. I. Jordan, "Latent dirichlet allocation," *the Journal of machine Learning research*, vol. 3, pp. 993–1022, 2003.

[44] T.-F. Wu, C.-J. Lin, and R. C. Weng, "Probability estimates for multi-class classification by pairwise coupling," *The Journal of Machine Learning Research*, vol. 5, pp. 975–1005, 2004.

[45] J. Van De Weijer, C. Schmid, and J. Verbeek, "Learning color names from real-world images," in *Computer Vision and Pattern Recognition, 2007. CVPR'07. IEEE Conference on*, pp. 1–8, IEEE, 2007.

[46] C. Steger, "Occlusion, clutter, and illumination invariant object recognition," *INTERNATIONAL ARCHIVES OF PHOTOGRAMMETRY REMOTE SENSING AND SPATIAL INFORMATION SCIENCES*, vol. 34, no. 3/A, pp. 345–350, 2002.

[47] L. I. Smith, "A tutorial on principal components analysis," *Cornell University, USA*, vol. 51, p. 52, 2002.

[48] I. Jolliffe, *Principal component analysis*. Wiley Online Library, 2005.

[49] R. B. Rusu, *Semantic 3D Object Maps for Everyday Manipulation in Human Living Environments*. PhD thesis, Computer Science department, Technische Universitaet Muenchen, Germany, October 2009.

[50] G. Bradski, "The OpenCV Library," *Dr. Dobb's Journal of Software Tools*, 2000.

[51] F. Shahbaz Khan, R. M. Anwer, J. van de Weijer, A. D. Bagdanov, M. Vanrell, and A. M. Lopez, "Color attributes for object detection," in *Computer Vision and Pattern Recognition (CVPR), 2012 IEEE Conference on*, pp. 3306–3313, IEEE, 2012.

[52] A. Hanbury, "Constructing cylindrical coordinate colour spaces," *Pattern Recognition Letters*, vol. 29, no. 4, pp. 494–500, 2008.

[53] J. Van De Weijer, C. Schmid, J. Verbeek, and D. Larlus, "Learning color names for real-world applications," *Image Processing, IEEE Transactions on*, vol. 18, no. 7, pp. 1512–1523, 2009.

[54] T. Hofmann, "Probabilistic latent semantic analysis," in *Proceedings of the Fifteenth conference on Uncertainty in artificial intelligence*, pp. 289–296, Morgan Kaufmann Publishers Inc., 1999.

[55] D. Tang, Y. Liu, and T.-K. Kim, "Fast pedestrian detection by cascaded random forest with dominant orientation templates," *Recall*, vol. 1, no. 3, p. 4, 2012.

[56] P. Flach, ed., *Machine learning: The Art and Science that Makes Sense of Data*. Cambridge: Cambridge University Press, 2012.

[57] N. Nethercote, *Dynamic binary analysis and instrumentation*. PhD thesis, PhD thesis, University of Cambridge, 2004.

[58] N. Nethercote and J. Seward, "Valgrind: a framework for heavy-weight dynamic binary instrumentation," *ACM Sigplan Notices*, vol. 42, no. 6, pp. 89–100, 2007.

[59] J. Elfring, S. Van Den Dries, M. Van De Molengraft, and M. Steinbuch, "Semantic world modeling using probabilistic multiple hypothesis anchoring," *Robotics and Autonomous Systems*, 2012.

[60] G. Welch and G. Bishop, "An introduction to the kalman filter," 1995.

[61] 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, no. 6, pp. 381–395, 1981.

[62] M. Waibel, M. Beetz, J. Civera, R. D'Andrea, J. Elfring, D. Galvez-Lopez, K. Haussermann, R. Janssen, J. Montiel, A. Perzylo, B. Schiessle, M. Tenorth, O. Zweigle, and R. van de Molengraft, "Roboearth," *Robotics Automation Magazine, IEEE*, vol. 18, no. 2, pp. 69–82, 2011.

[63] D. M. Gavrila, "A bayesian, exemplar-based approach to hierarchical shape matching," *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 29, no. 8, pp. 1408–1421, 2007.

[64] E. J. Griffith, M. Koutek, and F. H. Post, "Fast normal vector compression with bounded error," in *ACM International Conference Proceeding Series*, vol. 257, pp. 263–272, 2007.

[65] J. Ragan-Kelley, A. Adams, S. Paris, M. Levoy, S. Amarasinghe, and F. Durand, "Decoupling algorithms from schedules for easy optimization of image processing pipelines," *ACM Transactions on Graphics (TOG)*, vol. 31, no. 4, p. 32, 2012.

COLOPHON

This document was typeset using the typographical look-and-feel `classicthesis` developed by André Miede. The style was inspired by Robert Bringhurst's seminal book on typography "*The Elements of Typographic Style*". `classicthesis` is available for both LaTeX and LyX:

http://code.google.com/p/classicthesis/

Happy users of `classicthesis` usually send a real postcard to the author, a collection of postcards received so far is featured here:

http://postcards.miede.de/

Final Version as of October 17, 2013 (`classicthesis` version 4.1).