Hierarchical Object Classification through Probabilistic Common Sense Knowledge Reasoning

Masters' Thesis

Loy van Beek

January 27, 2013

Abstract

This thesis presents a manner for object classification by the use of semantic knowledge and probabilistic reasoning with such knowledge. An ontology of object classes and their context and properties is represented as a Markov Logic Network, which is a method of unifying first-order logic with probabilistic reasoning, developed recently. For each scene, the ontology is combined with symbolic observations of objects observed in the scene. Probabilistic inference is then used to infer the class or a superclass of those objects.

Preface

Ever since being a little boy, I loved and dreamed about robots. I was amazed and inspired by movies as Star Wars, but also by robotic rover missions to the planet Mars that set of during my childhood. While building planetary rovers and other robots rather than cars and houses from my Lego toys, I always knew that I wanted to do one thing when I grew up: robotics.

After studies in the fields of mechanical engineering and mechatronics, I started a study in the field in of Artificial Intelligence at Utrecht University, as I was disappointed by the industrial robots I worked with during my internships. These are programmed by listing the coordinates they should visit, when to enable their welding torch, et cetera. I wanted to make robots that were more intelligent than just following such detailed and fixed program.

The chance to be involved in developing such a more intelligent robot came via Eindhoven University of Technology (TU/e). Although I did not pursue any study at TU/e, I was accepted as a new member of the RoboEarth research group which develops an intelligent robot named Amigo. During the year in which I worked on my thesis, I worked in this very friendly and open group of PhD, masters' and bachelors' students. Besides doing research, a part of the group participates in the RoboCup@Home[1] competitions, mostly as a hobby project. By participating in this project I gradually became the teams' expert on the autonomy and coordination software, developing the software needed to coordinate tasks such as learning faces and names of persons and serving drinks to those. Working in this project involved several robot competitions, one of which took place in Mexico City.

First, I want to thank my supervisors Jos Elfring and Jan Broersen, for the great feedback I received from them. Second, I would like to thank the team for the great year I had with them and finally thank René van de Molengraft of Eindhoven University of Technology for giving me the opportunity of working in this group and living my childhood dream of making robots.

[1] `http://www.robocup.org/robocup-home/`

Contents

Chapter 1

Introduction

Robots are becoming more and more prevalent and are starting to invade our homes. Currently, robots exist for mundane household tasks, such as lawn mowing, vacuum cleaning and even washing a floor. Another development is the aging of the population. The need for care for the elderly is increasing, but the supply of care is not keeping up with this development and it is expected that it will not catch up. One solution could be employing robots to at least assist care workers and the elderly themselves, to lessen the need for care and to make the care workers more efficient at their jobs.

To make this happen, robots need to become smarter than they currently are. It is often said that inventions and new ideas arise due to needs and frustrations. A personal frustration is that robots often fail to recognize an object correctly although there are clear hints, at least to a human, to what class an object might be. The difference, in my opinion, is common sense knowledge. Humans simply know the world better and can infer from their perceptions and common sense what sort (class) of object they are observing. This thesis project attempts to integrate common sense knowledge with perception, with the goal of making robots smarter and to let robots take over more tasks.

Thesis Outline

This report is outlined in the following manner: First, the background of the research is formulated (page 5), after which the main idea (page 7) is stated along with a basic research question and the contributions of this research as well. Then, an investigation into existing methods for object classification is made (page 10), followed by a refined formulation of the main idea (page 17) as motivated by this investigation, together with more refined research questions. Next, the literature to answer these questions and to accomplish the main idea is studied (page 20). This is then followed by the framework in which the implemented system must be placed (page 30) and the design (page 33) and implementation (page 39) of the system. Finally, experiments (page 47), conclusions (page 60) and future work (page 61) are discussed.

Chapter 2

Background

The Eindhoven University of Technology, at which this research was performed, is developing a robot named Amigo, which stands for Autonomous Mate for InteliGent Operations.

2.1 Goal of the Amigo robot

The robot Amigo (Figure 2.1 on the following page) is used in several research projects, as well as in RoboCup[1]. The future goal is to develop a robot that is employable in a care or home environment. For this purpose, robots should be cheap, must be able to perform in complex environments and be able to share knowledge.

RoboEarth

The RoboEarth[2] project aims to create a world wide web for robots, with which robots can share knowledge. Examples of such shared knowledge are listed below:

- Navigation maps of the environments the collaborating robots operate in.
- Action recipes of how to perform actions and how robots with different capabilities can perform those.
- Object models for recognition, classification, manipulation and possibly other tasks.

Bobbie

The Bobbie[3] project aims to accomplish standard interfaces for robot components on both the hardware and software level. This in order to allow easy exchange of robot parts, such as hands, arms, heads et cetera between robots, so that each robot can use parts from different vendors of robot parts.

[1] `www.robocup.org`
[2] `http://www.roboearth.org/`
[3] `http://www.bobbierobotics.nl/`

Figure 2.1: The Amigo robot. Source: `http://www.techunited.nl/nl/photos/72157631289381106`

RoboCup

RoboCup[4] is a set of competitions for robots. The overall goal is to accelerate robotics and AI research by setting an ambitious challenge, which is to have soccer robots play and win against the human world soccer champions by 2050. There is also a competition for household and care robots, called the RoboCup@Home-league. Amigo and the team surrounding it finished 7th at the 2012 world championship in this league, which took place in Mexico City.

2.2 Existing System

The Amigo robot consists of several subsystems, a few of which are of relevance to this thesis. As this thesis deals with perceptions and objects in the robots' environment, the perception software that observes objects and their properties and world model that keeps track of those objects and properties, are of importance. These are discussed in more detail in Section 7.2 on page 30 and Section 7.4 on page 32 respectively.

[4] `http://www.robocup.org/robocup-home/`

Chapter 3

Main Idea

Object recognition and classification in computer vision works on a non-symbolic, numeric level, as will be shown in the literature survey of Chapter 4 on page 10. Recognizing objects and classifying objects are broad topics in the Computer Vision literature, and are two separate tasks. Object recognition is the task of assigning observations of *some* object to a *specific* object, observed earlier. Object classification is assigning a class or category to an object and is one of the hardest problems in Computer Vision [42]. The difference between the tasks is the difference between finding *your* car and finding *a* car.

This problem is hard due to a couple of reasons. First of all, there is the problem that instances of the same class can look very different (intraclass variance), while on the other hand, instances of different classes may appear to be very similar (interclass variance). Then, there is variance in the scene an object is pictured in, with for instance differences in lighting, which makes that the same object looks differently between different scenes. In a household environment, the environment in which service robots are destined to work, objects may be found in the dark of a cupboard or in the bright light next to a sun-lit window. When the class or category 'unknown' or 'something else' is permitted, assigning a correct class label to an object gets even more difficult, as this requires some measure and threshold of dissimilarity on which to base such a conclusion.

Methods to do so exist, but often make mistakes that humans would not make because of their common sense. Common sense, however, is not very common in computers and robots and as such, computer vision methods usually do not take common sense knowledge into account. Equipping robots with some form of common sense knowledge could improve their capability to recognize and classify (household) objects.

The main idea in this thesis is thus to use some form of knowledge to describe, on a symbolic, semantic level, what everyday household items look like. When this knowledge is represented in some way on a robot, it can be used to classify objects in the robots' environment, by combining it with symbolic/semantic, rather than numeric, observations on those objects.

3.1 Research Question

The research question for this final thesis project is thus: **How can inference with descriptions and predicates about color, shape, context and spatial relationships to context be used to classify instances of object classes?** First, a survey of existing computer vision object recognition and classification methods is made, after which the main idea and the research question above will be refined, in Section 5.1 on page 19

3.2 Contributions

In short, the contribution of this research is ontology based classification of objects.

Resulting from this, some interesting properties are obtained:

- Hierarchical classification.
- Recognition framework extensible with new types of observations.
- Common sense in classification.

These properties are elaborated on below.

Hierarchical Classification

Hierarchical classification in this context means that when not enough information is available, as-accurate-as-possible classifications can still be made. Instead of providing a *wrong* classification, it can provide a *less precise* classification. I.e. a more general class can be assigned to the observed object. Based on such a classification, a robot may decide to take a better look at the object in case more detail is needed. Deciding this is future work however (see Chapter 12 on page 61).

Besides the more robust classification this yields, it has other advantages as well. As the approach in this thesis is knowledge based, the hierarchical nature of the knowledge provides a way to compactly describe objects. If a new subclass of objects is to be added to the system, "learning" that new subclass can suffice with stating what superclass the new subclass has, and then stating what distinguishes it from that superclass and from other subclasses of the same superclass (i.e. its "sibling"-classes). The new subclass can derive most of its properties from its superclass. E.g, a *tea* cup may be described by stating what makes it more a specific class than the the more general *cup*, and by stating what distinguishes it from is sibling-class *coffee* cup.

Recognition framework extensible with new types of observations

The second property is that the knowledge, about the classes that the system knows of, is easily extensible. New facts about classes and classification rules concerning those can be added to the knowledge base directly, in a declarative fashion.

Common Sense in Classification

Classification in the domain of Computer Vision is by definition based on the appearance of objects. In some cases however, deciding what class an observed object belongs to can also be helped by *reasoning* about the object and about the context the object is in. For instance, a floor could be defined as the thing that is beneath every other thing in a room. This classification rule is not related to the appearance of the actual floor, and it applies to floors regardless of what they are made of and their appearance.

Chapter 4

Literature Survey on Object Classification

In this chapter, a survey is made into existing methods of object classification. The goal of this survey is to provide some insight into what has already been researched, to identify opportunities for improvement and to refine the main idea stated in Chapter 3 on page 7.

Various methods for recognizing and classifying objects exist and vast amounts of literature have been written on this subject. To constrain the search for literature in this field, a focus has been placed on methods for object classification with 3D-data, e.g. point clouds. From this kind of data, pieces of information such as shape and position of objects can be gathered, which is much more difficult or impossible with only 2D data.

In the past, sensors for acquiring point cloud data have been very expensive. The Microsoft *Kinect*® range camera, a cheap, real-time 3D sensor, has revolutionized robotics by giving even hobbyists and students access to full 3D perception. Papers on 3D recognition and classification written before and after the release of the Kinect can even be distinguished from each other, as the older papers devote more text on the setup of their 3D sensing devices and scanners, whereas later papers often simply mention the use of a Kinect. The Kinect also put a larger focus on real-time algorithms, as data comes in at a 30Hz rate instead of 1Hz or less.

The robot used in this thesis, Amigo, is also equipped with such a sensor, mounted in a anthropomorphic fashion: it uses a Kinect as its head. The Kinect outputs two streams of images: one of coloured images (RGB) and a second one of depth (D) maps, i.e. gray-scale images with the gray-value indicating distance to the sensor, making it a RGBD-camera.

As 2D cameras have existed for much longer than 3D cameras, vastly more research and literature exists for object classification in 2D. Many methods for 2D object classification can, however, be extended to 3D in a rather straightforward manner. First understanding the 2D approaches therefore is a requirement for understanding their 3D extensions, on which the literature survey is focused. An overview of the methods available is given, first introducing the method in 2D and later the extension to 3D, if it exists.

Difficulties of 3D perception

Working with an additional dimension comes with some (inherent) additional difficulties. A major difficulty, depending on the type of 3D sensor, is the result of perspective. In 2D, the laws of perspective dictate that objects further away are perceived smaller, so that for a distant object, a pixel on the camera covers a larger area of that object. As 3D captures an additional dimension, each pixel covers a larger *volume* for distant objects. This larger volume is still represented by a single point, and thus the sampling density decreases with distance to the 3D sensor. In 2D, pixels remain side-by-side, while in 3D, points may be meters apart.

Secondly, consecutive samplings may yield measurements of points that have the same coordinates. This, however, does not imply that they were sampled of the same surface or object. Additional information, like the surface's color or normal can reduce ambiguity, but not eliminate it. To generalize the notion of a point beyond its 3D coordinate, the concept of a point is replaced by a point feature representation (PFR), which may also take the normal, intensity or even color at the coordinate into account [34].

One might argue that voxels, a portmanteau of volumetric pixels, are a closer equivalent of 2D pixels. A 2D image is constructed of pixels, which divide a picture in small, rectangular grid cells. Equivalently dividing a 3D object results in the object being constructed of small cubical cells, called voxels. Such a representation with voxels is dense, whereas pointclouds are sparse.

4.0.1 Feature Descriptors and Keypoints

An important concept in Computer Vision is that of a feature, feature descriptors and of a keypoint. A keypoint is in essence a pixel or 3D point in an image or pointcloud that is distinctive in some way so that it can be detected again in different images or pointclouds containing the same or very similar keypoint [8]. For instance, it can be a feature like the corner of a window, that is sampled to a pixel or 3D point, which is then labeled by some algorithm as a suitable keypoint. A feature descriptor is a numeric description (e.g. a vector of numbers) of such a feature, like that window corner.

2D pixels or 3D point feature representations alone are not unique and not distinctive enough to serve as features. Therefore, higher level feature descriptors are needed, that are more distinctive.

Feature descriptors take the region around a keypoint into account when describing it and ideally, they are very (dis)similar for very (dis)similar features, so that when the same feature is present in different images and labeled as a keypoint, it can still be matched via the corresponding *descriptors* of the feature in the two images. One way to match feature descriptors is to see them as points in a high dimensional space and use, for instance, Euclidean distance between points to see which are close and thus very similar.

In 2D computer vision, such feature descriptors are SIFT [22], SURF [1] and subregion histograms. In 3D there are Point Feature Histograms (PFH), Fast Point Feature Histograms (FPFH), Viewpoint Feature Histograms [34] and many more. New feature descriptors, for 2D and 3D, are researched continuously. SIFT and SURF use gradients to describe the region around a keypoint, whereas (F)PFH uses the local curvature around a PFR.

4.1 Recognition by segmentation

One approach to recognition is by doing recognition simultaneously with segmentation. The problem of segmentation is to decide what measurements (2D pixels or 3D PFRs) of an image or pointcloud belong together, e.g. separating background and foreground in a 2D image. When performing recognition simultaneously with segmentation, this problem is generalized by also labeling segments of the image with what class each segment is. For background/foreground segmentation, each pixel is labeled accordingly. To refine the results, e.g. filtering noise and sharpening boundaries, Conditional Random Fields (CRFs) [41] are often used. CRFs are a probabilistic graphical model, that take context into account for tasks such as classification and segmentation, but can be also applied to other tasks. An example concerning background/foreground segmentation is where a patch of neighboring and observed pixels are noisy and all belong to the background. But due the the noise in the image, one of the pixels *appears* to be foreground. It is known (which is thus common sense) that it is unlikely for a single pixel to be foreground while it is surrounded by all background, so the CRF classifies said pixel to be background as well. One could say that the pixel that appears to be foreground is peer-pressured to be classified as background.

Another approach is to divide the image into label-less subregions (e.g. squares) and then try to match these subregions to parts of models, and then labeling them accordingly [42]. A very different approach is to employ a codebook of visual items (e.g. a wheel), in which each item has a predefined segmentation mask (e.g. a filled ellipse). When a visual item from the codebook is matched in the query image, the item's local segmentation mask is added to the global segmentation mask [42].

4.2 Bag of words

The core of the bag of words method is to compare the frequency with which some features occur between a query image and training images [8]. This is best explained with an analogy to text classification, hence the name. This works as follows: The number of occurrences for a set of preselected words, the vocabulary, is counted, thus creating a histogram of which words occur how often in a document. Note that the order of the words in a document is lost when performing this step. Such histograms are created for each document in the training set for each class of documents, resulting in a set of histograms for each class. Obviously, when the right set of words is selected for the vocabulary, the histograms for documents on sports and medicine, for instance, are very different. When a new, untrained, query document is presented, the word histogram of the document is compared to the histograms of the documents in each class. Several methods to compare (word) histograms exist, however this is out of the scope of this overview.

In computer vision, roughly the same method is used, albeit not with textual words but instead with visual words. Visual words can also be feature descriptors. To create a vocabulary of visual words, feature descriptors gathered from a training set of images or pointclouds are clustered together. This clustering can be compared to "clustering" derivations of verbs to a base form, e.g. "drank" and "drinking" to their base form: "drink". When classifying, descriptors are

matched to a visual word, one of the clustered descriptors, in the vocabulary.

As noted earlier, all structure in a text, image or pointcloud is lost in a bag of words model. Structure and relative position is what, in some cases, distinguishes one class of objects from another. For instance, humans are very good at recognizing faces, but not when they are upside down.

As it is uncommon for faces to be upside down, e.g. the mouth above the nose, it is less likely that such a thing is indeed a face. Thus the classification "face" may be less probable than some other classification. The use of prior knowledge, of what situations are more likely and make more sense, can prevent robots and computers from making mistakes.

4.3 Part-based models

A different approach to classifying objects is the use of part-based models. This is perhaps the oldest method of object recognition [42], and it focuses on what parts an object is comprised of and the relative positions of these parts. To use the example of faces once more: eyes are next to each other, the nose is below the eyes in the middle and the mouth is below the nose. Any other configuration is very unlikely to be a face, or due to a failed recognition of the parts.

The key difficulties in part-based models are the representation and learning of the parts and of the geometrical relationships between parts. Occlusion of parts is a problem as well, as it is in most recognition and classification methods. For the representation of the geometrical relationships, i.e. relative positions of the parts, most approaches use a graph, with various types of topology. Many different types and topologies of graphs have been used [42]. One approach is to interpret the edges connecting the parts as springs and then trying to find a configuration of the parts that results in a state of minimal energy, i.e. the most "relaxed" configuration of parts that best fits a model. The most commonly used topology for this approach is the fully connected constellation model, in which all parts are linked to all other parts. A downside of it is the combinatorial complexity that limits its number of parts to ca. seven at most, as the number of weights to assign to the edges grows quadratically with the number of parts [4].

A recent development is the *sparse flexible model*, which does not take the full graph into account like the constellation model, but only a number of edges between adjacent part locations [5]. Another development is the shared use of parts by multiple classes, instead of the standard approach of a set of parts per class [14]. One approach that seemed particularly interesting was that in [4]. Their method does not seek for relative part locations on a global scale, but first at a lower part and subpart scale, by finding subparts and image features that seems to often appear together, thus creating a hierarchical model (e.g. spoke $\rightarrow$ wheel $\rightarrow$ car, so in a different sense than in this thesis, which has hierarchy in classes instead of in (sub)parts). The geometrical relations between children and parents in the tree with respect to each other are represented by uncertain spatial transformations. This approach is interesting because the use of parts and subparts are envisioned to be of use in this thesis.

The work in [31] is also an example of the part-based approach to object recognition. [31] defines a *numeric shape representation*, around points in a mesh. Connected points with a similar shape representation are grouped into

a *shape-class component*. These shape-class components are assigned a symbol, and the mesh-points they were calculated from are labeled with that symbol. The geometric relationships between shape-class components are described by *symbolic shape descriptors*, which are extracted from the labeled mesh. The essence of their approach is that similarly shaped regions in triangulated pointclouds are assigned the same symbol. A short-coming of [31], in the light of this thesis, is that shape-class components alone are too low-level features, while the full symbolic shape descriptors describe whole objects already, making them too high-level.

A similar approach is taken in [24]. Again, a shape descriptor, Radius-based Surface Descriptors (RSD), is used. As to reduce computational complexity, instead of labeling individual points, their enveloping voxels are labeled. From these labeled voxels, Global RSDs (GRSDs) are computed. GRSDs are based on another global feature descriptor, called Global Fast Feature Point Histograms (GFPFHs) [33], but GRSDs are computationally less expensive. GFPFHs are computed by taking the histogram of voxel label *transitions* in the neighborhood of the voxel that contains the point under consideration.

With GRSD, not the distribution of voxel label transitions is used, but rather the sum of these. Finally, these GRSDs are classified using a Support Vector Machine [48].

An additional classification step taken in [24] is to segment these 3D shapes in a 2D image and using a Bag-of-SURF-features model to further classify the objects. The SVM classification reduces the number of candidates for 2D classification, resulting in a global success rate over 98% on a real-world data test set of 12 common household objects. Note that this approach, also the 3D-part, can be seen as a bag-of-words model, but with some basic, structural geometric information used as well.

The approach taken in [37] also uses two levels of classifiers, but in a very different way. The low-level classifier that comes first identifies vertical and horizontal planes and determines their principal dimensions, which are to be as features. For horizontal planes, distance to the ground is also used as a feature, whereas for vertical planes, additional features are the distances between the plane's top and bottom to the floor and ceiling, respectively.

Regions in the vicinity of these planes, or *furniture face candidates*, may be fixtures, like knobs and handles. On these regions, lines and circles are fit. The numbers of handles and knobs as well as the distance between the fixtures and the plane are used as a features in the second, higher-level classifier. On the second level, a naive Bayes classifier is used in the form of a Conditional Random Field (CRF), that classifies the object based on the weighted joint probabilities between the class and the observed value. The classifier is learned under supervision, in order to estimate the weights for each feature that maximizes the log-likelihood of the probability of a label given an observation for a feature. In [37] the system is trained on noisified synthetic data and achieved an item accuracy of 0.97, 0.98, 0.91 for vertical and horizontal planes and furniture, respectively. This is used to create semantic maps, mapping which class of object is located where, a task of which categorizing objects is only a subtask.

An extension, or generalization, of [37] is the work in [25], which labels furniture, learned from CAD models, in scenes. Instead of dealing with points directly, they use larger regions as basic units for recognition. In their approach, a scene is segmented by growing regions of neighboring pixels up to sharp (40°)

curves and openings larger than 5cm. Region growing is done by randomly picking a point as a seed for the region and adding neighboring points that satisfy the aforementioned condition. For large enough regions, which represent object parts, feature vectors are computed, which are clustered into a vocabulary of parts. When presented with a new scene, the scene is segmented and labeled with part-labels from a subset of the vocabulary. Each part then casts a vote for the location in order to determine the location of the whole object. The method used for voting is called probabilistic Hough voting [21]. The best hypothesis for the complete object's location is verified by fitting its CAD-model to the pointcloud data. This approach is very interesting, but it has a deficiency: learning and verification uses CAD-models, which is infeasible for a lot of very specific object types, as they may not be available at all. This is not to say that learning furniture from real-life examples is more feasible, it is likely to be the reverse. Chairs, tables et cetera do have a pattern to their structure though, which can be used to determine whether some object is a chair or table. A possible improvement may be to use probabilistic reasoning for the verification, by determining if the object's label is in accordance with its context. Using semantic labels and context, it is not even necessary to determine exactly what CAD model is detected, as long as any type of chair is correctly labeled as being a chair, instead of as a table. Using context, if a chair shoved under a table is incorrectly labeled as a couch or table, the verification would suggest that such a combination, of a couch shoved under a table, is unlikely. Note the use of a semantic spatial relation here, "shoved under", or simply "under".

[25] assembles objects from parts, by letting them vote for a best possible match. Parts alone are mere symbols, i.e. words in the vocabulary of parts, without relations to other parts or symbols. In the light of this thesis, it would be much more interesting to have parts with a symbolic label, with relations to other symbols. This would allow a robot to reason which objects the object could be part of or what other properties are likely to be associated with the part.

4.4 Semantics and common sense-based classification methods

The methods below involve methods that can be dubbed semantic or could be said to use some form of common sense. These methods, [40] and [29], could also be listed in the sections on Bag-of-words methods and Part-based methods respectively, but a separate section was deemed more appropriate.

The work in [40] takes an approach involving global and local semantic attributes of a scene as a whole and of objects in it respectively. A Support Vector Machine-classifier (SVM) is trained for each attribute. There are 60 global attributes (e.g. city, landscape, grass, sky, et cetera) and 55 local attributes (e.g. black, circle, door, metal) in groups such as color, shape, part and material. Local attributes for objects are found by determining the average response for 100 patches of an image. The responses of the 115 classifiers are concatenated into a 115-dimensional semantic attribute feature descriptor. [40] states that context often helps in recognizing objects.

An approach that uses some form of common sense more heavily is described

in [29]. In the approach taken, objects seen in range images (which can be converted to point clouds) are segmented into primitive shapes (sticks, plates and blobs). Groups of primitive parts are labeled with a functional symbol, such as Sittable, Back Support and Ground Support for example, that are associated with the sitting plane, the back support and the legs of a chair, respectively. These groups can be part of a more complex part or object in their turn, thus forming a directed acyclic graph (DAG) of parts and primitive parts. By doing so, an object is described in a part-wise hierarchy. This part-wise hierarchy allows different classes of objects to reuse parts, which do not have to be learned for each class that uses it.

Chapter 5

Refinement of the Main Idea

The main idea in this thesis is that object classification can be aided by *reasoning* about perceptions, the properties of perceived objects and the spatial relationships between those objects, using symbolic, probabilistic, prior knowledge about the objects and their classes.

As absolute certainty is hard to obtain from sensor data, probabilistic reasoning is needed, in order to determine what sensors are really observing given their output, as this is not necessarily the same due to errors, sensor noise, etc. While knowledge of measurements is current knowledge, an understanding of how the world works or is likely to be is prior knowledge; known beforehand.

Symbolic knowledge is knowledge concerning symbols, the concepts they refer to and the relations between concepts. The meaning, or semantics, of a concept is mainly defined by the relations it has to other concepts. Concepts can have symbols and names associated with them.

Perhaps an example is in order here. Take the concept associated with the symbol "table", a very common household object. How can such a common concept of "table" be defined? The simplest definition is that it is some form of top surface with ca. four legs supporting it. But then, what is a "top surface", what are "legs", what is "top", "surface" and "supporting"?

The interpretation of symbols and names can vary between individuals and languages. For instance, the concept "pink" refers to some color, but exactly which hues and variations of colors are still labeled "pink" can vary between humans. Some languages used by humans do not even have names for concepts that other languages do have. The same symbol can have a different meaning, or concept associated with it, in a different language.

Of course, in order to classify objects, their appearance must be somehow known, but an hypothesis of this paper is that also context can be used as a clue to which class an object belongs to. As an example of using context, imagine observing some protrusion, a little blob of points in space measured by a 3D sensor. This particular protrusion is circle-shaped and only a few centimeters in both length and diameter. When this is all the available information, no usable conclusion can be made on the class of the object observed. Now, imagine that this blob of points protrudes from a plane. Next, the knowledge is added that

the plane is, with some probability, the door of a closet. The protrusion on that door could now be a handle. The reverse also holds: a plane is more likely to be a door if there is some sort of handle on it, no matter what type of handle.

So, from this, we can assert that doors are likely to have handles, and that handles are likely to be attached to doors. Doors may also be related to other types of objects as well, such as houses, storage containers like closets, refrigerators, etc. This prior knowledge, of which objects often occur together, is thus useful for inferring which objects are being observed.

In the above examples, some different relations between objects occurred: a relation between objects and their parts, and relations between two objects. Such knowledge, and knowledge of the appearance of objects and their parts, must somehow be stored, in a probabilistic manner that is fit for some form of reasoning. Effectively, objects and *classes* of objects are to be described, somehow.

In general, the idea is to describe concepts concerning household objects, their properties and the relations between such concepts. The default method [32] to store and reason with such knowledge, about concepts, is by the use of so called Description Logics. In such a logic, one can express the fact that doors have some relation to handles, for instance. Description Logics, often abbreviated to DL, are a family of logics for describing classes of things (concepts) and relations between those. For this thesis, the idea is then to use a DL that is able to represent uncertain or probabilistic relationships concerning classes of objects.

The work in [37] uses a conditional random field (CRF) to learn and store knowledge about object appearance and their context. In this thesis, I want to store such knowledge in a symbolic, semantic way, by the use of description logics and use more properties and attributes of objects as well. In the work of [24], object parts cast probabilistic votes for the main objects' class and position. For this thesis, I want to make such spatial relations symbolic and attach meaning/semantics to them, by describing the (spatial) relations between objects and possibly parts in a DL as well.

In a description logic, one of the possible relations is the sub/superclass relation. This allows to, for instance, define a spoon to be a subclass of cutlery. I think this can be useful in object classification as well: if it is not sure whether a spoon or fork is observed, then the more general class "cutlery" may suffice. If or when needed, a robot may take a better look at the object to determine which subclass of cutlery is a more accurate labeling.

Another advantage of using a class hierarchy, is the possibility to use that hierarchy for inheritance. Inheritance, in the context of this thesis, means that subclasses inherit relations and properties from their superclasses. Using the cutlery example again, the relation that cutlery is often made of metal can be defined for cutlery in general, along with the fact that cutlery is often found near plates. As a subclass of cutlery, the class "spoon" may inherit those relationships.

How this can be done exactly and what "tools" are needed for the execution of this main idea is the subject of the theoretical framework in Chapter 6 on page 20.

5.1 Refined research questions

With the survey of literature and a detailed main idea, we can now define subquestions of the research question stated in Section 3.1 on page 8. The original, high-level research question is this: How can probabilistic inference with descriptions and predicates about color, shape, context and spatial relationships to context be used to classify instances of object classes?

It can be refined to these **subquestions**:

1. How can knowledge of descriptions of objects and classes be represented? Can a formal system be used for that?

2. How can the uncertainty involved in such descriptions be represented?

3. What kind of inference mechanism or decision making scheme can be employed for the classification of objects, using the represented knowledge? I.e. can perhaps Bayesian Networks be used, or Conditional Random Fields, Markov (Logic) Networks, plain non-probabilistic First Order Logic, or some probabilistic variant of a Description Logic?

Chapter 6

Theoretical Framework

6.1 Semantics and Reasoning

In order to decide what type of logic to use for reasoning in this application, it must first be very clear what type of knowledge is being reasoned with. In essence, some classes of objects will be described somehow, stating the relations between a class's symbol to other symbols. As explained above, a single class of objects can have multiple subclasses with their own appearances. Classes of objects can thus have super- and subclasses, e.g. "Doorhandle" is a subclass of the more general class "Handle". Also, classes can have relations to each other, such as the part-of relation that a door would have with "Handle". Furthermore, there are relations describing the visual appearance of objects, like their shape and color. The default way [32] to represent this type of knowledge is by means of a Description Logic. Description logic is investigated below and finally used in this thesis for knowledge representation, because it is the default method, allows hierarchy in classes and represents knowledge symbolically and declaratively.

Additional measures need to be taken to make a knowledge base in a description logic probabilistic, as it needs to represent knowledge of how likely situations and relations are, instead of making only absolute statements, which is the default in formal logic.

In the next section, an introduction to description logics will be given, and thereafter an investigation of what options are available in the field of probabilistic description logics.

6.2 Description logics

Description logics [20] are, indeed, a family of logics to describe *concepts* and *roles*. More specific, a DL can be used to describe classes and relations between classes, respectively. In the description logic literature, the terms "concept" and "class" are used interchangeably, along with "role" and "relation".

Description logics being a family of logics means that there is a variety of such logics, each with different expressiveness and semantics. Many variants of description logic are subsets of first order logic (FOL) and can be implemented or expressed in terms of First Order Logic. In a description logic, all relations are binary, meaning that relations are made between two items. The more

expressive members of the DL family allow the definition of not only just relations, but also subrelations. For instance, the father-of relation is more specific than the parent-of relation. Other expressive features are the construction of relations by composition, the construction of classes by requiring its members to have certain relations to some classes, restrictions on the number of relations to other classes et cetera. An introduction into description logics is provided in [20].

6.2.1 Example ontology

The type of knowledge needed in this thesis is exemplified below.

- Cutlery can be made of Metal.
- Cutlery can be made of Plastic.
- Cutlery is often seen near Plates.
- Cutlery is often on top of a Table.
- Tables are usually made of Wood.
- Tables usually have 4 table legs.
- A table leg is attached to the underside of a table.
- Spoon is a sort of Cutlery.
- Fork is a sort of Cutlery.
- Forks have Spikes, usually four.
- Spikes are pointy-shaped.
- One end of a Spoon is a Bowl.
- Plates are often on top of a Table.
- et cetera.

When substituting the word "is a sort of" with "is a subclass of" or "is a concept included in", then it becomes clear that such knowledge can be perfectly represented in a description logic. The terms "often" and "usually" are *not* very well suited to be stored in any form of first-order knowledge base, as such a knowledge base assumes that all statements and assertions are absolute. Some *relations* will therefore have to be probabilistic, to express the uncertainty involved. Not all of this knowledge is easily represented by any form of symbolic logic. For instance, describing complex shapes by using symbols is very difficult. Some of the perception routines as described in Section 4 on page 10 can however be used to estimate the "pointiness", for instance, of some part (e.g. a spike of a fork), the output of which can be used when reasoning about perceptions. These methods can thus be used as input, about which the classification scheme of this thesis can reason.

Besides knowledge about classes of objects, the information gathered from the perceptions must also be represented symbolically, in order to enable reasoning.

For instance, the observation that object321 is made of metal, is near an instance of Plate and is that it is pointy on one end must be represented. With that information represented symbolically, the reasoning system may infer that object321 is an instance of the class Fork or Knife, each with some probability.

So, an observation of the environment a robot may find itself in, a scene, is represented as a smaller knowledge base, such as the one exemplified below:

- object321 is likely to be made of Metal.
- object321 is unlikely to be made of Plastic.
- object321 is pointy on one end.
- object321 is near object456.
- object456 is likely to be white, less likely to be gray and very unlikely to be black.
- object456 is likely to be round and unlikely to be square.
- object456 is on top of object789.
- object789 is most likely to be an instance of Table.

Requirements for knowledge representation For the purposes of this thesis, only the simplest semantics of description logic are needed. The type of knowledge stated in example 6.2.1 on the preceding page is the following:

1. Definition of classes (e.g. Handle, Cup, Plate, Table, Shape, etc.).
2. Definition of instances of classes (e.g. fork31, table85, red, cylinder, metal etc.).
3. Definition of binary relations between *classes* and instances/values (has-color, has-height, seen-with, seen-on-top-of etc.).
4. Definition of binary relations between *instances* and other instances/values (has-color, has-height, seen-with, seen-on-top-of etc.).
5. Concept Inclusion, i.e. the definition of subclass-relationships and the transitivity of such relations. This means that if B is a subclass of A, and C a subclass of B, then C is also a subclass of A.
6. Subclasses must inherit the relations of their superclasses.
7. Subclasses must inherit the assignments made to relations of their superclasses and be able to make assignments to the same relations, making the description more specific.
8. Preferably, both discrete and continuous values for relations (has-height, has-diameter) (This preference has not been satisfied, see Section **??** on page ??).

A knowledge base describing concepts, i.e. a set of description logic facts, is commonly referred to as an *ontology*.

6.2.2 OWL and RDF

A well defined, standardized format to store the type of knowledge above exists already, and is called Web Ontology Language, abbreviated OWL [3]. This language is based on RDF, the Resource Description Format [43]. An RDF-document, in essence, is a set of triples, each consisting of three items: two entities and one relation between those entities. OWL extends RDF by providing some predefined relations as well, such as relations for subclasses and assigning properties, but also poses some restrictions on what relations can be expressed. Unfortunately, standard OWL and RDF have no way defined in which to express statements of probability. There is some research on creating such extensions, but before we look into these, let us first discuss what type of probabilistic knowledge and reasoning is needed in this thesis.
A list of reasoners for OWL ontologies can be found on the World Wide Web Consortium (W3C) website [46]

6.3 Probabilistic reasoning and probability in description logics

For classification of objects based on their visual properties alone, as described above, simple inference can be used. For instance, from the facts that an object is made of metal, has a bowl on one end is seen near a plate it can be inferred that the observed object is a spoon. That is, when there is absolute knowledge of the observed properties. Absolute knowledge of the environment is hard to obtain, however. As seen in the example of section 6.2.1 on page 21, unfortunately no absolute statements are given about any of the objects, but rather words such as "usually" are employed, which indicate uncertainty.

Instead of absolute certainty, a degree of belief can be assigned to statements, for instance that some class of object is made of metal. Another degree of belief can be assigned to the statement that the same class of objects is made of wood. These uncertain statements can then be used in *probabilistic* reasoning, that also takes the degrees of belief for other properties of objects and their context into account. As seen in the example of Section 5 on page 17, some objects are observed together more often than other combinations of objects. From this, we can conclude that perhaps joint probabilities of objects being observed together need to be used, but also joint probabilities between object classes and values for properties thereof.

Many approaches and frameworks for probabilistic reasoning and probabilistic inference exist. The knowledge described above can become quite complex, with tens to hundreds of statements already for even the simplest ontology. The used method of inference must be able to handle relatively large domains, i.e. hundreds of classes and facts about those. In the next section, an investigation is made of what the inference method must be able to do, thus listing the requirements for an inference method.

6.3.1 Requirements for probabilistic reasoning

In the investigation, two types of requirements are identified: functional and non-functional requirement, The non-functional requirements are:

- A fully functioning implementation should be readily available.
- Integrability with ROS [51] for eventual deployment on Amigo. This implies that the implementation must be able to run on Linux.
- Preferably, some of the knowledge must be able to be learned. Learning is not a focus of this thesis, but allows some practical future work to be more easily performed.
- Preferably, be able to work with standardized knowledge representation formats, such as OWL.

The functional requirements are best stated in terms of what kind of knowledge must be represented, and what kind of inferences it must be able to make. The types of knowledge to represent are shown in the enumeration in Paragraph 6.2.1 on page 22. The inferences to be made are the following: From measurements on the observable properties of some objects, infer what the class of these objects is. More specifically, a measurement consists, for each object and for each observable property, of some degree of belief that the object is having some value for that property. For example, a measurement may report that there is high degree of belief that object A is cylinder shaped and a low degree of belief that it is rectangularly shaped. Object B is believed highly to be white and to be a large box. Further, there is also some degree of belief that object A is inside object B, which is inferred from their estimated positions.

Note that this assumes that the data association problem, of which measurements and observations to assign to which object, has already been solved. The world model [12] incorporated on the Amigo robot performs the task of data association.

The task of the inference system is then to compute degrees of belief about the classes each observed object belongs to. It could output a degree of belief that object A is a bottle or some piece of tubing, as there is no more information. Object B might be a closet, a refrigerator or maybe the enclosing of a boiler for heating the house. As bottles are often found inside refrigerators and tubing may be found near a boiler, additional degrees of belief can be computed for each of these *combinations* of objects.

In short, both the knowledge base itself and the queries posed convey information on degree of beliefs, and the inference mechanism must be able to use this information about uncertainty to output a number of possible conclusions, each associated with a degree of belief in that conclusion.

In this section, some options for probabilistic reasoning will be reviewed. As stated earlier, the knowledge described above can be represented in a description logic. Therefore, first some probabilistic variants of description logic are reviewed.

6.3.2 Existing probabilistic description logics

Some forms of probabilistic description logics (abbreviated PDL) already exist. One of the requirements is that there is an implementation readily available. A probabilistic description logic named P-CLASSIC [19] was immediately dismissed as there was no implementation available.

BayesOWL BayesOWL [38] is an extension of OWL with probabilistic relations between (distinct) classes. An implementation is available, but according to its manual [2], it can only deal with binary variables. As this thesis will deal with both arbitrary discrete and continuous random variables, BayesOWL cannot be used.

PR-OWL PR-OWL [9] is an extension of OWL, that provides ways to express probabilistic statements in an OWL ontology. UnbBayes [6] can reason with PR-OWL ontologies. Not much information is available on how to use PR-OWL and UnbBayes and PR-OWL requires elaborate, exponentially growing conditional probability tables to be defined.

Log Linear Description Logics Log Linear Description Logics [26] are based on Markov Logic Networks [10]. In [26], a description logic called $\mathcal{EL}^{++} - LL$ is defined, which is based on a $\mathcal{EL}^{++}$ also known as $\mathcal{ELRO}$ description logic, but without nominals and concrete domains. $\mathcal{EL}^{++} - LL$ however is not directly suitable for the type of queries that are to be answered in this thesis. It is intended for queries concerning the probability that one concept is included in another. As the LL in $\mathcal{EL}^{++} - LL$ denote the fact that it cannot deal with concrete domains, i.e. *instances* of objects and not only classes, it is unsuitable for use in this thesis. However, the main point to get from this paper is that it is very well possible to implement the rules of a description logic in a Markov Logic Network.

6.3.3 Custom implementation

A decision was made to use a general framework for probabilistic reasoning with first-order logic and to implement a simple probabilistic description logic in that, which would allow maximal control over the implementation. The most general framework to do this with is Markov Logic [10], and more specific *Hybrid* Markov Logic Networks [11]. The hybrid variant allows for both discrete and continuous domains. Continuous domains are desired to describe continuous properties of objects with, such as height and diameter.

Implementing the needed rules and semantics in first-order logic is straightforward, and creating a working implementation of the rules in a logic programming language such as Prolog is moments of work. From this, it was taken that embedding the same rules in a *probabilistic* first order logic would deliver exactly the needed system.

The standard implementation of (Hybrid) Markov Logic Networks is Alchemy [17], developed by a team surrounding the inventor of Markov Logic. Some advantages come from taking this approach:

- Markov Logic Networks scale linearly in the number of relations between entities, instead of the exponential scaling of Bayesian Networks.
- Markov Logic Networks are the most general of probabilistic first order logics. The use of first order logic is required to write the needed rules for description logic.
- Choice of several probabilistic inference algorithms implemented in Alchemy.

- Alchemy is actively developed, in contrast to some probabilistic description logics.

It also fulfills the other non-functional requirements:

- The intended platform for Alchemy is Linux.
- Alchemy has the ability to learn the degrees of belief in the knowledge base already built-in, thus easing possible future work.
- Alchemy is able to learn structure from a knowledge base as well.
- As Alchemy is not a description logic implementation, it cannot handle OWL directly. Converting an ontology in a subset of OWL to a Markov Logic Network for use in Alchemy is very well possible though, and can easily be automated.

Markov Logic Networks will be given a detailed review in Section 6.4. The semantics and logic rules derived from the functional requirements are elaborated in Section 8.2.1 on page 34.

6.4 Markov Logic Networks

Markov Logic Networks (MLNs) [10] are an attempt to unify *logical* artificial intelligence and *statistical* artificial intelligence. The logical side of AI behaves well under complexity, but not under uncertainty. Statistical AI is the opposite in that regard, as it cannot cope with complexity very well, but it can handle uncertainty. The core of the concept of MLNs is to assign each logical sentence with a weight, which can be obtained by learning them from a (large) data set of ground clauses, or by setting them manually.

In a Markov Logic Network, formulas in first order logic are considered as constraints on possible worlds, with possible worlds being the set of all possible assignments to all variables in a set of query predicates. In classic first order logic, inconsistent worlds cannot exist, but in a MLN they are assigned a low probability.

A MLN consists of a set of tuples, each of which consists of a first-order clause and a real-valued weight. Together with a finite set of constants, this forms a template for a Markov Network. From this template, a ground Markov Network is formed by grounding the template, which is done by grounding each variable in each formula with the possible values it can take. Markov Logic Networks are typed, which limits the amount of groundings. Formally, the Markov Network $M_{L,C}$ as defined by the aforementioned set of tuples, named L, and the set of constants, named C, consists of a node for each ground predicate and a feature for each ground formula. A feature has a weight, which is that of the formula of which it is a grounding. Nodes and features both have a value, which can be 1 or 0 according to whether the grounding is true or not, i.e. is the corresponding constraint satisfied or not in a possible world. Nodes are connected via an edge if they appear together in a ground formula.

The concept of a *feature* as used above, it not clearly defined in [10]. In [18] however, a feature is described as a node which' final value, state or output, is interesting. Features also provide an easy mechanism for specifying certain types

of interactions (in a network) more compactly [18]. Features, in this context, represent formulas in a Markov Network. For inference, we are interested in the final state of some formulas, specifically formulas or rules that give the class of an object.

A ground Markov network, or Markov Random Field (MRF) defines a probability distribution over the set of possible states X. The probability of a possible world $x \in X$ is the sum of number of true groundings n_i of each formula F in that world multiplied by that formula's weight w_i, exponentiated and normalized:

$$P(X = x) = \frac{1}{Z} exp(\sum_{i=1}^{F} w_i n_i(x)) \tag{6.1}$$

Markov Logic Networks allow formulas to contradict each other, resolving contradictions by evaluating the weights of each possibility, weighing the evidence as it were.

In the case of this thesis, one of the predicates will be the relation of classes to objects. The ground Markov network contains nodes and features for each combination of possible objects and classes. The most probable possible world then includes the most probable assignment of classes to objects, and thereby performs classification.

6.4.1 Relation to other probabilistic graphical models

Probabilistic graphical models [18] are a way to represent the relations between random variables, in which the random variables are nodes and the relations and dependencies they have among each other are represented by edges. Many other probabilistic graphical models, such as Conditional or Markov Random Fields, Bayesian Networks (of which Naive Bayesian Classifiers in their turn are a special case), Hidden Markov Models and others can be seen as spacial cases of Markov Logic Networks [10]. The advantage of Markov Logic over them is that independence can be assumed while dependence can be incorporated very easily, in a declarative manner. It is very well possible that the final Markov Logic Network developed in this thesis resembles one of the approaches above, although stated in a declarative manner.

6.4.2 Inference

Multiple methods of inference are possible: Maximum A Posteriori (MAP) and marginal inference. MAP inference is finding the possible world with the highest probability, maximizing Equation (6.1) by finding a truth assignment x that best satisfies the weighted formulas. Multiple algorithms to find this world of maximal probability exist; the original paper on Markov Logic Networks [10] mentions MaxWalkSAT [16], which is a satisfiability solver that also takes weights into account.

Several methods for either type of inference in Markov Logic Networks exist:

- MC-SAT [30]
- Gibbs Sampling [7]
- Simulated Tempering [23]

- Belief Propagation [28], and its lifted variant: Lifted Belief Propagation [39].

To evaluate the performance of a Markov Logic Network for classification though, it is not only useful to know the most probable worlds, i.e. classifications, but also to know less probable classifications. This, so it can be observed in more detail if the classification results are completely wrong, or almost right and needs some refinement. With MAP inference, the result is either right or wrong, and no evaluation can be made of *how* wrong the result is, as the correct result may be just slightly less probable.

Marginal inference is capable of finding the marginal probabilities for a (set of) possible world(s), enabling aforementioned evaluation. As the probabilities of each possible world are normalized, marginal inference yields a probability mass function over possible assignments for each query predicate with one variable. If we then "ask" the Markov Logic Network what the class of an object is, given some evidence, it outputs a probability mass function over the possible classes for that queried object. Marginal inference can be performed by the MC-SAT algorithm [30], as mentioned in [10], and is the only algorithm performing marginal inference that is also implemented in Tuffy (Section 6.4.3), as well as Alchemy.

MC-SAT MC-SAT works by first satisfying the hard constraints and setting this assignment of variables as the possible world $x^{(0)}$, where (0) indicates the current iteration, 0, of the algorithm. It then performs the following procedure for a predefined amount of iterations: Initialize an empty set M that is to contain satisfied clauses, each of which should remain satisfied in the next iteration. Then, for each clause c_k that was satisfied in the previous iteration $x^{(i-1)}$, add it to M with probability $1 - exp(-w_k)$, where w_k is the weight of clause c_k. After adding all the selected clauses, sample a new possible world uniformly from the set of possible worlds that satisfy the clauses in M. This assumes that all weights w_k are positive, which can be achieved negating all negative-weight formulas and their weights.

The pseudocode for MC-SAT is shown in Algorithm 1.

Algorithm 1 MC-SAT(*clauses*, *weights*, *num_samples*)

$x^{(0)} \leftarrow$ Satisfy(*hard_clauses*)
for $i \leftarrow num\_samples$ **do**
  $M \leftarrow \emptyset$
  for all $c_k \in clauses$ satisfied by $x^{(i-1)}$ **do**
    With probability $1 - exp(-w_k)$ add c_k to M
  Sample $x^{(i)} \sim \mathcal{U}_{SAT(M)}$

6.4.3 Tuffy

There exist multiple implementations of some of the algorithm(s) discussed in the previous section. The reference implementation of Markov Logic is named Alchemy [17], as mentioned earlier in Section 6.3.3 on page 25. A different implementation of Markov Logic Networks called Tuffy [27] claims to be three

to five orders of magnitude faster. Tuffy only implements MC-SAT for marginal inference.

The input-files for Alchemy and Tuffy are compatible, although Alchemy accepts implications while Tuffy requires the knowledge base to be written in clausal form. The claim that Tuffy's authors make is also valid in the cases of this research, as observed in the experiments: Tuffy produces results in under 20 seconds for an MLN where Alchemy takes roughly 20 hours, which is indeed a speed-up of three-orders of magnitude . It achieves this tremendous speed-up through three improvements on Alchemy:

- Grounding through SQL-queries, using a relational database which optimizes these queries automatically, as a black box.
- Partitioning the ground MRF into smaller components, yielding a smaller search space for the lowest-cost possible world; components which can also be processed in parallel.
- Using a so-called hybrid architecture for inference.

The hybrid architecture for inference is the result of employing the right tools for each job. A relational database is very suitable for grounding but not for inference, as inference involves random sampling, which in turn requires random access to data storage in a computer. Relational databases usually employ disk-based storage, which cannot perform random access efficiently. The architecture is hybrid in the regard that after grounding, the now ground Markov Random Field is copied from disk into the computers' main Random Access Memory (RAM). RAM *is* efficient for the random memory access used in random sampling. As the MRFs produced by Alchemy become very large, they can potentially not fit in RAM anymore, which results in very inefficient memory swapping to disk. Another advantage of the partitioning that Tuffy performs is that the components are smaller than the whole MRF and will generally fit into RAM, also in cases where the whole MRF is too large for RAM.

Tuffy is the implementation used for the final experiments, although Alchemy was chosen first because of its support of Hybrid Markov Logic Networks. This discrepancy is elaborated on in Section 9.3 on page 40.

Chapter 7

Software Framework

In order to integrate with the other software modules in the Amigo-robot, the system developed in this thesis is built on ROS, the robot operating system. It integrates with the world model [12, 45], a reasoning component for that world model and other knowledge and finally with a perception pipeline. Each of these are briefly discussed below.

7.1 ROS

ROS, in recent years, has been adopted by robotics researchers worldwide, as it provides a large library of robotics algorithms and other software, communication services, and software building infrastructure. For instance, it provides off the shelf navigation and localization algorithms, libraries for 2D and 3D perception, inverse kinematics, speech synthesis and much more. The Amigo robot also works with ROS, and in order to be valuable to the team, the developed system should preferably work with ROS.

Software packages, called Nodes in a ROS system, communicate with each other through message passing. This is based on a publish-subscribe architecture, which means that the publisher of information is not directly tied to the subscriber of it, and that multiple publishers may publish on the same stream of messages. Message streams, or topics in ROS terminology, can also have multiple subscribers. Thus, there needs to be at least one publisher of information about each object property. ROS nodes can also provide services to each other, for example to perform some calculation on request.

7.2 Perception Pipeline

As stated earlier, the robot is equipped with a Kinect RGBD camera. This data is processed by the perception pipeline, which includes ROS [51] packages for segmentation, detecting objects on a table and color recognition. Software to recognize object shapes is present as well, based on separate 2D matching with several methods and 3D shape matching using VFH [35]. The perception software packages do not output a definite conclusion, but rather they output the probability of a match to each model. A key deficiency of this pipeline is that it lacks a ways in which to combine all the parallel classification methods

for a final classification. The output information is then fed into the World Model (Section 7.4 on the next page). Several of these systems are subject or ongoing research, which is briefly mentioned below.

7.3 Research Context

The research group in which this thesis is performed in part of a Mechanical Engineering department and has its focus accordingly. Research is also performed on the topic of advanced control theory, motion planning, state estimation and many more topics. Two theses from the group are relevant to this thesis and are elaborated on below.

Object Appearance Prediction and Active Object Search Using Probabilistic Object Relations [15]

The work in [15] is very much related. It investigates the use of probabilistic relations between objects for two different purposes: predict object appearance given the context and to search for particular objects more efficiently. The relations are obtained by learning the joint probabilities of which objects are often seen together in a scene. Then, when a robot is asked to retrieve some item, which often involves searching for the item, the robot plans a path that maximizes the probability of encountering the desired item as fast as possible. Predicting object appearance is useful when performing object recognition with a large database. Matching the observed object against every model in the database is a computationally very expensive operation, and this work can provide means to first match against objects that are most likely to be observed, given the context.

The method used for probabilistic inference is the weighted sum approach. To calculate the probability for a queried item given the context (evidence) items, the weighted probability for the query item Q given each evidence item E is calculated. The weight, for each context item, is the normalized probability for that item having a particular class label. Put mathematically:

$$P(Q|E_1, ...E_n) = \sum_{i=1}^{n} P(Q|E_i)w_i$$

where the weight

$$w_i = \frac{P(E_i)}{\sum_{j=1}^{n} P(E_j)}$$

There is a major difference between the work in [15] and this work though: in [15], the class labels for the evidence items are known, and moreover, the class labels are definitive. In this thesis, class labels are *not* definitive and the point of my thesis is that they are unknown at first.

The second application of the probabilistic relations, active object search, is less relevant to this thesis.

Combined 2D+3D segmentation (Ric Jacobs)

Before objects can be classified, they have to separated, or segmentated, from the background and other objects. One such approach for doing so is shown

in [25] and uses geometric information alone. Ric intends to combine both 2D and 3D information with a Conditional or Markov Random Field for the segmentation of objects, possibly also splitting objects into parts. This project is still underway, so the results cannot be used or evaluated already.

If this research would segment object from their environment and further split objects into parts, outputting multiple hypotheses on splits and parts, it could be very useful.

7.4 World model

One of the systems with which the classification system of this thesis will heavily communicate is the world model. The world model is the result of a master's thesis [45] (a resulting article is [12]) and subject of continuous research. This system is implemented and running on Amigo and performs the task of keeping track of object identities and their properties, such as position, motion, color, shape, et cetera. It can store arbitrary discrete and continuous properties in probability mass and density functions, and uses models, when available, to predict values of those properties. The models used for prediction depend on the class of an object. For example, household objects usually cannot move by themselves and are thus assumed to remain in the same place, while e.g. humans can move around and are assumed to move with a low speed. A research goal concerning this system is to also include common sense knowledge, so that the behavior models do not predict that persons move into walls but take a turn before that happens, for instance.

The perception software's output is pipelined into the world model as input, which then probabilistically matches perceptions to object identities, by keeping a set of hypotheses about which perceptions (a form of data) associate to which objects. By doing so, it attempts to solve the data association problem. The world model uses the information from previous time steps to formulate a more accurate conclusion on the identity and, for instance, class and name of objects and faces.

In short, the world model is able to provide the probabilistic data that will serve as input for the software resulting from this final thesis project. The output of the classification scheme of this thesis will also be input for the world model, in order to label object with their classifications. The world model uses probability distributions for properties of objects where it can, in order to maintain multiple hypotheses about each property.

7.5 Non-probabilistic Reasoner

To access the information in the world model, a first-order reasoning component named `tue_reasoner` was created which can also take other knowledge into account, stated as a Prolog knowledge base. The `tue_reasoner` can be said to act as a query front-end to the world model. It can attach a probability to its output, based on hypotheses in the world model about the queried information. Given that such information is available in the world model through the perception pipeline, it can answer queries like "What is the color of Object1", and yield a probability mass function over the possible colors Object1 has in each hypothesis.

Chapter 8

System and Language Design

8.1 Framework Integration

The system developed in this thesis must be deployed on the Amigo robot, and can preferably also be used in a broader context. This is achieved by splitting up the classification method in two parts:

- A ROS service[1] for Markov Logic reasoning, named `mln_reasoner`.
- A ROS node[2] that extracts the relevant information, i.e. observations, from the running system and passes it to the reasoning service. This node is named `mln_classifier`.

Using this architecture, classification is only one application of Markov Logic and is open to others.

The service for Markov Logic reasoning is mainly a wrapper around the actual implementation of Markov Logic Network inference algorithms. The service loads a Markov Logic Network when it starts or by external request. It then waits for the classification application to send it information on which inference can be performed, after which the results are sent back to the classification application. This node then processes the classification results and publishes these to the rest of the robot system, so that the robot can act on the inferred information by finally grasping a can of coke, for example.

8.1.1 Input and Output

The information that the `mln_classifier` gathers from the system are the properties that are associated with each observed object. For the Amigo robot, a perception pipeline has been developed which outputs a part of the needed information. It performs some classification tasks itself as well, but is lacking a proper method for combining classification results from multiple steps in the pipeline, as mentioned earlier. The perception pipeline outputs information over the following properties:

[1] `http://www.ros.org/wiki/Services`
[2] `http://www.ros.org/wiki/Nodes`

- Color of an object
- Shape of an object, classified using VFH[34] and not into primitive shapes.
- Location of an object
- Dimensions of an object

For the spatial relationships, reasoning about object locations is required. Currently, no implementation of such reasoning is implemented on the Amigo robot, but the reasoning components and so-called computables in KnowRob [44] could provide this functionality when it is integrated with Amigo's `tue_reasoner`-package. The names of spatial relations as defined later in this thesis are taken form the KnowRob ontology with this purpose in mind.

VFH classifies object shapes into more specific classes than just primitive shapes: it can be any set of shapes, such as the shape of a coke can or a fork. The method developed in this thesis does assume primitive shapes, but there is no reason why such more specific shapes could not be used instead. Performance could possibly improve using this sort of shape-symbols, but for the purposes of this thesis the decision was made to remain more generic.

8.2 Embedding Description Logic in Markov Logic

In 6.3.3 on page 25, the choice was made to create a custom implementation of a very simple probabilistic description logic. In this section, the design and implementation of the description logic is described.

8.2.1 Formal Semantics and Syntax of Description Logics

In this section, the needed DL formal syntax and semantics are stated using formal logic. The core of any DL is the Top concept $\top$, or the class which is the superclass of all things. Reversely, there is the Bottom concept $\bot$, or the class which represents no thing at all.

The semantics of a DL are defined by an interpretation $\mathcal{I} = (\Delta^{\mathcal{I}}, \bullet^{\mathcal{I}})$, in which $\Delta^{\mathcal{I}}$ is the domain of discourse, i.e. the set of all classes, instances and relations considered in the model. $\bullet^{\mathcal{I}}$ is the interpretation function, which maps symbols to interpretations. In the model, there is a set of class names N_C, a set of relation names N_R and a set of instances, or objects, N_I. $\bullet^{\mathcal{I}}$ then assigns:

- each class name $C \in N_C$ a subset of classes $C^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}}$,
- each relation name $r \in N_R$ a relation $r^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$
- every instance $a \in N_I$ an element $a^{\mathcal{I}} \in \Delta^{\mathcal{I}}$

In general individuals are denoted with a, b, classes with C, D and roles with r, s. As said, there exists a large family of description logics, that each define different further semantics. For the purposes of this thesis, only a relatively small subset of the semantics and features of description logics is used. The type of knowledge that needs to be represented is the following:

- That some class is a subclass of another.

- That some class has some relation to another class or value.

For classification of objects, i.e. instances of classes, the knowledge about such objects consists of simply the knowledge that some instance has some relation to some value or class. The rules of inference are then merely of the form: "If an instance i has a relation R to A and class C also has relation R to A, then the class of i *may* be C". It is possible to implement such a general rule directly in a logic programming language such as Prolog. A key feature of description logics, however, is that relations can be specified to only apply to a certain domain and range of classes and types of values. While it is possible to specify rules that would enforce such typing, predicates in Markov Logic Networks are already typed, as opposed to Prolog. The distinction between classes and different types of values is very important.

For the purposes of this thesis, the semantics needed and syntax that are defined are listed below. The rules that were to be implemented in Markov Logic have evolved over time, as the capabilities syntax-wise were not familiar to me enough at the time of design. For Markov Logic, that what is presented below are the final forms of the concerned rules, after several iterations described in Section 9 on page 39.

In Prolog, variable names start with capitals and names of constants start with a lowercase letter, as do predicates. In Markov Logic Networks, it is the opposite: names of constants and predicates start with a capital; variables start in lowercase.

Top Concept

Syntax : $\top$

Semantics : $\Delta^{\mathcal{I}}$

Meaning : The Top concept is the set of all concepts, i.e. the class of all things, and represents everything, thus every thing.

Prolog : `superclass(Class, top).`

MLN : `Superclass(class, Top)`

Bottom Concept

Syntax : $\bot$

Semantics : $\emptyset$

Meaning : The Bottom concept is the subconcept of all concepts, and represents nothing, i.e. no thing at all.

Prolog : `subclass(Class, bottom).`

MLN : `Subclass(class, Bottom)`

Concept assertion

Syntax : $c : C$

Semantics : $c^{\mathcal{I}} \in C^{\mathcal{I}}$

Meaning : "c is an instance of class C".

Prolog : `instance_of(instance, class).`

MLN : `Class(instance, Cls)`

Role assertion for instances

Syntax : $(a, b) : R$

Semantics : $(a^{\mathcal{I}}, b^{\mathcal{I}}) \in r^{\mathcal{I}}$

Meaning : "Instance a and b are related by relation R".

Prolog : `instance_relation(instance, role, value).`

MLN : `Role(instance, value).` Please note the difference here. Prolog uses triples, whereas the MLN version uses a binary relation. This also means that a new predicate must be defined for each relation.

Role assertion for classes

Syntax : $(A, b) : R$

Semantics : $\forall a : a \in A^{\mathcal{I}} \wedge (a^{\mathcal{I}}, b^{\mathcal{I}}) \in r^{\mathcal{I}}$

Meaning : "Class a and b are related by relation R".

Prolog : `class_relation(a, role, b).`

MLN : `ClsRole(a, b)`

Concept Inclusion, i.e. subclassing

Syntax : $C \sqsubseteq D$

Semantics : $C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$

Meaning : C is a subclass of D.

Prolog : `subclass(Class, Superclass).`

MLN : `Subclass(cls, supercls)`

Concept equivalence

This rule has not been implemented in Prolog nor Markov Logic as the need for it has not occured.

Syntax : $C \equiv D$

Semantics : $C^{\mathcal{I}} = D^{\mathcal{I}}$

Meaning : Concepts/classes C and D are the same, or equivalent.

Transitivity of Concept Inclusion

No special rules are needed for this in either Prolog or Markov Logic.

Syntax : $C \sqsubseteq D \wedge D \sqsubseteq E \rightarrow C \sqsubseteq E$

Semantics : $C^{\mathcal{I}} \subseteq D^{\mathcal{I}} \wedge D^{\mathcal{I}} \subseteq D^{\mathcal{I}} \rightarrow C^{\mathcal{I}} \subseteq E^{\mathcal{I}}$

Meaning : The concept C is a subclass of D, and D is a subclass of E, so C is also a subclass of E. For example: a fork is some kind of cutlery and cutlery is a form of tools, thus forks are some kind of tools.

Existential Restriction

This rule has not been implemented in Prolog nor Markov Logic.

Syntax : $\exists r.D$

Semantics : $\{x \in \Delta^{\mathcal{I}} | \exists y \in \Delta^{\mathcal{I}} : (x, y) \in r^{\mathcal{I}} \wedge y \in C^{\mathcal{I}}\}$ and $C \equiv \exists r.D$

Meaning : C is defined as the class of objects that have relation r with some object in D. For example: the relation r is one that means "has color", and D is the concept of color. Then C is defined as the class of colored things.

Inheritance of relations from superclasses to subclasses

This rule is also not required in Markov Logic.

Syntax : $C \sqsubseteq D \wedge D \equiv \exists r.E \Rightarrow C \sqsubset \exists r.E$

Semantics : $C^{\mathcal{I}} \subseteq D^{\mathcal{I}} \wedge \{x \in \Delta^{\mathcal{I}} | \exists y \in \Delta^{\mathcal{I}} : (x, y) \in r^{\mathcal{I}} \wedge y \in D^{\mathcal{I}}\} \rightarrow \{x \in \Delta^{\mathcal{I}} | \exists y \in \Delta^{\mathcal{I}} : (x, y) \in r^{\mathcal{I}} \wedge y \in C^{\mathcal{I}}\}$

Meaning : C is included in D, while D is defined as the set of things that have a relation r with E. This implies that C also has that relationship.

Subclasses inherit assignments made to superclass relationships

Syntax : $(d, e) : r \wedge c \in C \wedge d \in D \wedge e \in E \wedge C \subseteq D \Rightarrow (c, e) : r$

Semantics :

$$(d, e) \in R^{\mathcal{I}} \wedge c \in C^{\mathcal{I}} \wedge d \in D^{\mathcal{I}} \wedge e \in E^{\mathcal{I}} \wedge C^{\mathcal{I}} \subseteq D^{\mathcal{I}} \rightarrow (c, e) \in R^{\mathcal{I}}$$

Meaning : If d is related to e through the relation r and if c is a C, d is a D, e is an E and C is a subclass of D, then also c has an r-relation to e. For example: Soda cans are cylindrical. Coke cans are a subclass of soda cans and thus inherit the property of being cylindrical.

Prolog : `class_relation(Sub, Rolevalue) :- subclass(Sub, Super), class_relation(Super, Superrole, Rolevalue)`

MLN : `Subclass(sub, super) ^ ClsRole(super, superrole) => ClsRole(sub, superrole)`

Not all description logic semantics described above are needed, as is stated above. They are still mentioned to show what is needed for this thesis and what is not needed.

These rules, as derived from a formal description logic, are enough to describe objects with knowledge that is certain. However, this thesis does not deal with *certain* knowledge, but with *uncertain* knowledge. In a Markov Logic Network, uncertainty is expressed by assigning a weight (comparable to degree of belief) to each fact or rule, by prefixing it with a floating point number. Rules postfixed with a dot '.' are hard constraints and must always be satisfied. The formal semantics of the description logic itself are not made probabilistic, but rather the rules derived from it are given probabilistic annotations.

8.3 Probabilistic Annotations in the Description Logic

Probabilistic reasoning in description logics can have several meanings, each implying a different part of the knowledge being probabilistic. In this research, concept inclusion is not probabilistic: it is certain that for instance coke cans are a sort of soda can. Only the properties, the role assertions, that define a class of objects are probabilistic, in the sense that a relation between a class and a value only hold with some weight. A high weight for a role assertion, for example `10.0 ClsColor(Coke_can, Red_color)` means that it is very likely that coke cans are red. Please note that weights in themselves are of no particular meaning: it is only in proportion to other weights that one rule is more important than another.

Chapter 9

Development of the Markov Logic Network

9.1 Representing Appearance

The classification scheme of this thesis must classify objects based on their appearance and their context. The appearance of objects must therefore be described, represented in some way.

An attempt was made to use RDF-style triples, as discussed briefly in Section 6.2.2 on page 23. This resulted in triples such as

```
Property(Obj1, Color, Red)
```

and

```
Property(Obj1, Shape, Cylinder)
```

This has a major downside however. As both values for Color and Shape are used in the same predicate, Alchemy puts Cylinder and Red under the same type. Alchemy could thus infer that the Shape of an object was Red, which is incorrect of course. It is possible to write hard constraints to prevent this, but the added complexity was considered to be not worth the effort, not to mention that the amount of possible worlds would still be much larger than with the alternative, which is to repeat rules for each property. Furthermore, in other good software engineering practice, it could be beneficiary to automatically generate such rules from a template.

Even more important on a conceptual level, is the fact that properties of *instances* of classes (objects) are different from properties of classes. I.e. there must be a difference between the knowledge that a class is generally of some color, or that an instance has a specific color.

Following the two observations above, binary predicates are introduced for each property: one to assert that an object has some property and one to assert that a class has some property. These are `ClsProperty(Class, PropertyValue)` and `Property(Object, PropertyValue)` respectively. Such predicates are repeated for each property, so the Markov Logic Network contains facts such as

```
ClsColor(CokeCan, Red)
```

```
ClsShape(SodaCan, Cylinder)
ClsMaterial(SodaCan, Metal)
```

et cetera.

Descriptions of actual objects, instead of classes, use similar facts:

```
Color(Object23, Red)
Shape(Object23, Cylinder)
Material(Object23, Metal)
```

9.2 Inheritance or Concept Inclusion

With the predicates defined in the previous section, it is finally possible to express rules that express inheritance, or subclassing or concept inclusion, via a `Subclass`-predicate. These rules are repeated for each predicate. The inheritance-rule for a predicate such as `Color` is then:

```
Subclass(sub, super) ^ ClsColor(super, supercolor) => ClsColor(sub, supercolor).
```

This states that if `sub` is a subclass of some superclass `super` and that superclass has some particular color `supercolor`, then the subclass also has that color. The above is a template for property-inheritance rules and is repeated for each property.

9.3 Continuous-valued properties and Dimensions

Alchemy, the implementation of Markov Logic that has been chosen earlier in this thesis for this reason, has the capability to deal with continuous values, instead of discrete ones. Using continuous values, an attempt was made to define predicates for representing appearance descriptions and to write rules for classification and inheritance in the same fashion as for the discrete properties. This resulted the following rule:

```
ClsHeight(cls)=clsheight ^ Height(inst)=instheigth ^ instheight=clsheight =>
Class(inst, cls).
```

This simply states that when an instance has some height, and a class has some height, and those heights are equal, then the instance belongs to said class. Unfortunately, Alchemy rejects this rule with a syntax error. In trying to fix this, a closer look at Alchemy's user guide was taken, specifically the tutorial[1] on Hybrid Markov Logic Networks.

The semantics and syntax for defining rules that include continuous properties are not quite clear. E.g. in the rule

```
Class(CokeCan, id) * (Height(id) = 0.1)
```

[1] `http://alchemy.cs.washington.edu/tutorial/11Hybrid_Domains.html`

, as adapted from the tutorial, what is the meaning of the $*$? Other ways of writing the rule were explored, but none seemed to work.

Eventually, a decision was made to *not* use continuous properties using Hybrid Markov Logic Networks and thus to discretize the range of all properties that concern spatial dimensions. This is in conflict with the preference for continuous properties, as listed in the requirements for knowledge representation of Section 6.2.1 on page 22.

A choice was was made to make steps of one centimeter, which is at a scale fit for most household objects: not much household objects are smaller than a centimeter and not much are larger than a few hundred centimeters, thus keeping the number of atoms needed relatively small.

Stepping away from HMLNs allows to explore other implementations of Markov Logic as well, as Alchemy is the only implementation of Markov Logic that is able to handle HMLNs. For example, Tuffy, as mentioned earlier in Section 6.4.3 on page 28, is another Markov Logic implementation that claims to be faster and to scale better to larger MLNs.

The example of this section deals with the property of height, but more dimensions for objects and classes have the potential to yield a more accurate description and result in more constraints on the set of possible worlds and are thus expected to yield a smaller set of plausible answers. Some more philosophical than technical questions and problems arise directly. What exactly is height? What exactly are length, depth and width? In which direction are they measured? Do cylinders and spheres also have depth and width or just diameter? Height, for instance, is the distance between the bottom and top of an object. But, when an object is on its side, what is the top and what is the bottom?

Clearly, some definitions are needed. A difficulty here is that humans intuitively know what the front, back and other sides of an object are, without being able to tell why they think so. The dimensions that are to be used must be clear for robots, which for example do not know the intended usage of an object, certainly not before they know what class an object is, as is the case in this thesis.

Below is a list of some dimensions with their interpretation by humans:

Length The longest dimension of an object [47, 52].

Height The distance between the lowest and highest points on an object.

Breadth or width The distance between two opposing sides of an object, measured in a direction perpendicular to the object's length-axis.

Depth "The distance between the front and the back, as the depth of a drawer or closet." [49]

Diameter Twice the radius, where radius is the greatest distance between the axis of a cylinder and its surface.

These definitions of dimensions depend on the following notions:

Side "A flat surface of a three-dimensional object; a face." [50]

Top The side of an object delimiting it in the vertical dimension at the highest point, opposing the bottom.

Bottom The side of an object delimiting it in the vertical dimension at the lowest point, opposing the top.

Front The side of an object closest to the observer, or the "business-side" of an object, on which it is operated or otherwise used. This side is parallel to the height-axis of an object.

Back The side of an object opposing the front.

These definitions indirectly all depend on the insight and prior knowledge of a human observer, e.g. to indicate what the front of an object is. Such prior knowledge and insight on this level are not yet available in robots and can therefore not be used.

In order to decide what dimensions *can* be used, it it first necessary to asses what can be measured by a robot, specifically the Amigo-robot, which is equipped with a Kinect 3D camera and software to analyze pointclouds based on PCL [36]. The capabilities of the analysis software related to the estimation of the dimensions described above are listed below:

- Fitting an orientated bounding box to a cluster (a cluster is a set of points measured on the same object).
- Of such an oriented bounding box, list its dimensions along the edges of the box.
- Fit an oriented cylinder to a cluster.
- Estimate the diameter of a fitted cylinder and length of its axis.
- Fit planes (i.e. sides) to clusters.

Ideally, the dimensions can be measured irregardless of the orientation of the object, so that an object on its side still has the same dimensions. Dimensions such as height depend on orientation and can therefore not be used. In some cases though, the length and height of an object are equal. Length is not dependent on orientation and can therefore be used.

For the most basic shapes that occur in household environments, the following conventions are used:

Shape	Length	Diameter	Width	Depth
Cube	Longest dimension	Undefined	2^{nd} longest dimension	3^{rd} longest dimension
Cylinder	The distance from end to end along the axis of the cylinder.	Distance between axis and surface.	Undefined	Undefined
Sphere	Undefined	Distance between center and surface.	Undefined	Undefined

In the perception pipeline, some logic will have to included to route measurements to the correct interpretation.

These properties are stored in the knowledge base and evidence respectively in this form:

```
ClsLength(Sodacan, 12).
```

and

```
Length(Obj9, 12)
```

9.4 Context & Spatial Relationships

Classifying objects by their appearance and dimensions is only a part of the investigated approach. Context is also to be used as a clue to the class of an object.

That one object or class is, or usually is, in the context of another object or class can can be expressed using the `Neighbor-` and `ClsNeighbor`-predicates, which were used in early stages of the implementation. The meaning of these predicates is that the two objects or classes in this relation are (usually) spatial neighbors of each other, and that thus are standing very close together, that they are on top of each other, or that one is inside the other, etc. This relation of "being in context of" is refined into predicates such as `OnTop` and `IsInside`. But with this relation of Neighbor already, spatial relation graphs of an environment can already be created, with the objects as nodes and the `Neighbor`-predicate as edges between the objects. An example of this is (Figure 9.1)

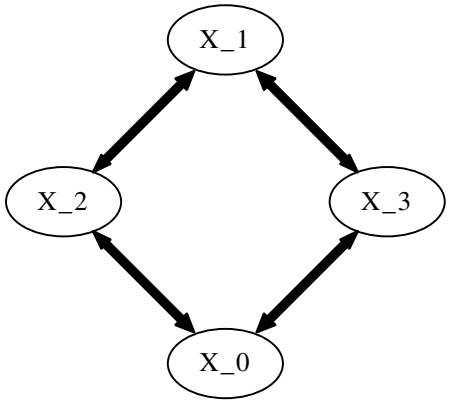

Figure 9.1: Graph showing neighbor-relations between objects. X_i is the class of an object

On each object, however, we observe some properties, such as Color, Shape etc, as already mentioned before. Per object, this yields a graph (Figure 9.2 on the following page) in which observations are also linked to the class of the object.

When these two graphs are combined, we obtain a graph quite similar to a Conditional Random Field with multiple observations, i.e. Figure 9.3 on the next page. Exact inference in CRFs is intractable, but there exist methods that can yield approximate solutions. For each situation, a graph like that in Figure 9.3 on the following page is instantiated with a structure depending on the relative positions of the observed objects and the observable properties of those objects.

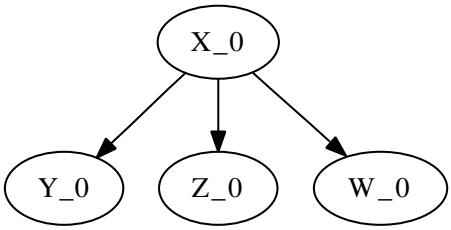

Figure 9.2: Observations on an object X_0. Using the notation commonly used in literature on graphical models, X is the hidden variable that is to be inferred, whereas Y is the observed variable, which could represent Color, Shape etc. As multiple observations are made, also W and Z are observed variables. Strictly speaking, this classification scheme does not deal with observations directly, but instead with the symbolic output of lower-level classifiers and the world model mentioned earlier.

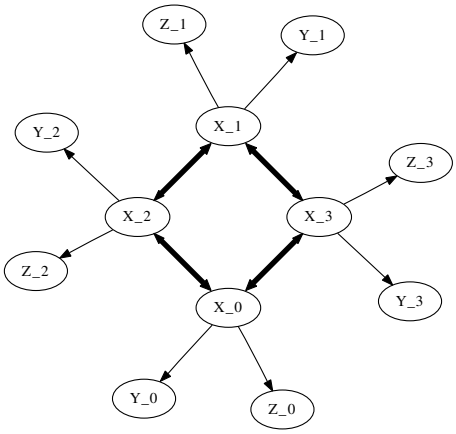

Figure 9.3: Conditional random field with multiple observations per object

As mentioned above, the (Cls)Neighbor-predicates can be refined into much more precise spatial relationships. The following spatial relations are defined:

- AboveOf
- BelowOf
- InCenter
- InFrontOf
- InBehindOf
- IsInside
- IsOnPhysical (note the specification "Physical", to specify that an object is on top of another, instead of the "On"-state of devices and machines. A TV set can be *switched on* while also being on top of a closet, for instance)

- ToTheSideOf
  - ToTheLeftOf
  - ToTheRightOf
- IsNear

Some of these spatial relations are the inverse of each other, such as InFrontOf and InBehindOf, AboveOf and BelowOf, ToTheLeft/RightOf.

9.5 Classification

Classification rules can be written in multiple ways. During exploration of the syntax and possibilities of Markov Logic, rules were defined this form:

```
ClsColor(cls, clscolor) ^ Color(inst, clscolor) => Class(inst, cls)
```

This takes only one property into account. It seems natural that performance improves when multiple properties are taken into account, so that e.g. both shape *and* color should match. This would result in a rule of the following form:

```
ClsColor(cls, clscolor) ^ Color(inst, clscolor) ^
ClsShape(cls, clsshape) ^ Shape(inst, clsshape) => Class(inst, cls)
```

We can then also increase the weight of such rules, as more matched properties naturally lead to more sure classifications. This results in relatively verbose and unclear code, certainly when more than two properties are to be taken into account, and even more so when rules are written in clausal form. This is circumvented taking an intermediate step, using a rule of of this format:

```
ClsColor(cls, clscolor) ^ Color(inst, clscolor) => Colormatch(inst, cls)
ClsShape(cls, clsshape) ^ Shape(inst, clsshape) => Shapematch(inst, cls)
```

These can then be combined in a slightly cleaner fashion:

```
Colormatch(inst, cls) ^ Shapematch(inst, cls) => Class(inst, cls)
```

which is

```
!Colormatch(inst, cls) v !Shapematch(inst, cls) v Class(inst, cls)
```

when converted to clausal form. As there are many properties, ideally all of the combinations of properties are used as rules. All these combinations are indeed used, automatically generated, and weighted proportional to the amount of properties they combine. Several different Markov Logic Networks are created, each with increasing length/complexity of rules, i.e. a MLN with rules of only 1 property up to a MLN with rules for each combination of 6 properties. Which amount of properties yields the best success rate is evaluated in the experiments, in Section 10.4.1 on page 53.

9.6 Ontology

In order to demonstrate that the system works and to perform experiments, a large ontology is needed that encompasses all objects, their properties and context classes and also their properties. A non-probabilistic ontology, manually created in OWL [3] during the literature study-phase of this thesis, was therefore taken and converted to an ontology in Markov Logic, utilizing the predicates developed in previous sections.

This translation from OWL to MLN is fully automatic, with each predicate name in OWL mapped onto a predicate name in MLN, along with a function that assigns a weight to each fact concerning that predicate. Automatically generating MLNs from OWL-ontologies requires to split the factual knowledge and the rules for inference into two parts, listed in Appendix A on page 69 and Appendix B on page 72.

The translation program can selectively include or exclude facts concerning some predicate. When converted from OWL to an MLN, including only Color-, Shape-, Material- and Class-related predicates, the generated MLN consists of ca. 600-700 weighted facts about ca. 115 classes and values. Combined with the rules, this results in a ground Markov Random Field with ca. 800.000 ground clauses, according to Alchemy's output.

9.7 Final Markov Logic Network

The experiments, as discussed in the corresponding chapter, are performed with the MLN as discussed up to this point. Much more knowledge can be added, more rules, other weights; there is a multitude of possibilities for further improvement. It is relatively easy to add new facts and rules and thus the current (variants of the) MLN can be seen as a proof of concept.

The final MLN, as it is used in the experiments, is included in the appendices. It is split into two parts, one containing only the rules (Appendix A on page 69) and a second only containing class knowledge (Appendix B on page 72). In this MLN, the weights of rules are determined mainly based on experience developed during this thesis project. The weights of the facts are randomized and thus not all set to the same value. This has been done in order to emulate the effect of learning the weights, which also would result in different weights for each fact. A final important note regarding the description of classes is that only positive evidence is given, since listing all negative evidence (i.e. what a class of objects does *not* look like) is not reasonable to do.

Chapter 10

Experiments

The experiments to select the best performing variant of the final MLN, to validate the approach and to test the performance of the selected MLN are discussed below. First, the system is evaluated in scenes where all information that could be available is indeed available. Later, it is tested on situations where some information is missing or wrong.

10.1 Experiment setup

In order to test the method, a number of scenes was developed, 16 to be exact. A scene is simply a description of a situation as it could occur in the real world. It lists the knowledge that is available about a situation. A simple example is this (scene 1):

```
Color(Obj12, Red_color)
Shape(Obj12, Cylinder_shape)
Material(Obj12, Metal_material)
Length(Obj12, 12)
Diameter(Obj12, 6)
//Class(Obj12, Coke_can)

Color(Obj22, Brown_color)
Shape(Obj22, Cylinder_shape)
Material(Obj22, Wood_material)
Length(Obj22, 74)
Diameter(Obj22, 100)
//Class(Obj22, RoundTable)

IsOnPhysical(Obj12, Obj22)
```

It lists the properties of a coke can (Obj12) and a round table (Obj22) and the fact that Obj12 is physically on Obj22. The Markov Logic Network, developed in the previous chapter, contains knowledge about classes; scenes contain knowledge about instances of classes. The statement of the class of each of the objects is in comments, so that the inference mechanism cannot take this information, the ground truth, into account. Such a scene is combined with the Markov Logic

Network as listed in Appendix A on page 69 and Appendix B on page 72. The Tuffy program is then ran with these files and the predicate we are interested in (i.e. Class) as its arguments and it outputs a result-file, which contains a probability distribution of possible answers for each query predicate. A possible output of the program is listed below:

```
0.3100  Class("Obj12", "Sodacan")
0.1900  Class("Obj22", "RoundTable")
0.1900  Class("Obj12", "Cookiejar")
0.1700  Class("Obj22", "Closet")
0.1600  Class("Obj22", "PepperShaker")
0.1500  Class("Obj22", "Beer_bottle")
0.1400  Class("Obj22", "Chair")
0.1200  Class("Obj12", "Pan")
0.1100  Class("Obj12", "Juice_carton")
0.1000  Class("Obj12", "Coke_bottle")
0.1000  Class("Obj22", "Table")
0.1000  Class("Obj12", "Teapot")
0.0900  Class("Obj22", "Bottle")
0.0700  Class("Obj12", "PepperShaker")
```

The analysis software developed for this thesis takes a set of such result-files and parses and processes them to finally produce the figures shown below.

In the example above, the ground truth class for `Obj12` is `Class("Obj12", "Cokecan")` and for `Obj22` it is `Class("Obj22", "RoundTable")`. The most probable (rank 1) class for `Obj12`, however is `Sodacan`, which is a *superclass* of `Cokecan`. For `Obj22`, the most probable class is indeed `RoundTable`. The second most probable (thus ranked 2) class for `Obj12` is a `Cookiejar`.

How the results are further processed to determine performance is subject of the next section.

10.2 Validation method

The output of the default Maximum A Posteriori inference is the most probable state of the world. In effect, it yields the most probable class of each object, given a set of observations as listed above. Performing *marginal* inference results in a number of hypotheses with a probability for each, which allows for a much deeper analysis than just observing only which is the best class. The result-files as described in the previous section are the result of marginal inference and used for the validation method described here.

Marginal inference takes some more time than MAP inference, but this difference is relatively small. Results from marginal an MAP inference are often different but similar, as the used algorithms are different. The results of MAP inference do not allow for a deeper analysis of individual results, while those of marginal inference do. Furthermore, the world model, that will receive the output of this classification scheme, can take probability distributions over the class of an object and use that for later reasoning. The world model tracks the values properties, and also the class of an object, over time. The fact that the most probable class is different when more information is available can be

valuable to the world model or to the software that uses it. In this thesis, therefore marginal inference is used, as this provides the most valuable output.

With the output of marginal inference, we can see if the possible world that is inferred to be the the second most probable world, i.e. assignment of classes to objects, was perhaps the correct one. It is then also possible to get the index or rank of the ground truth in the results, when they are sorted by probability: if the ground truth is listed as the most probable class, then the ground truth is at rank 1, the ground truth has rank 2 if it is the second most probable result, et cetera. Ideally, the ground truth class is always the most probable class, but because the Markov Logic Network is used to infer the class of an object, the ground truth class and the most probable class of an object are certainly not always the same, as misclassifications may occur.

The success rate is then how often, in a set of objects in scenes, the ground truth is listed at rank 1, or rather the fraction of cases in which this happens. The method developed in this research can however also classify something as the superclass of the ground truth class, e.g. classify a coke can as a more general soda can. The rank of the superclass in the results is also taken into account, and the total success rate is the sum of these two fractions. For the total success rate, both ground truth and superclass of ground truth are weighed equally; one is not weighed heavier than the other for the total success rate. This measure of total success rate will be used as the primary measure of performance.

Rankings can be summarized into a histogram, an example of which is shown in Figure 10.1 on the following page. The bars in such a graph show the fraction of cases in which the ground truth, superclass thereof or a sibling class of the ground truth was at rank n. The stepped plots in the graph show cumulative histograms for each labeling, along with a total cumulative histogram of the ground truth and superclass cumulative histograms added together. The value of a cumulative histograms at rank n tells in what fraction of cases, the ground truth, superclass or sibling was listed at a rank lower than or equal to n.

From the plot of the cumulative total success rate, another performance measure can be defined. Namely, the rank n at which this plot hits the 1.0 mark. For any n, it can then be said that the classification scheme has the ground truth or a superclass thereof in the first n results.

Ideally, the classification scheme always list the ground truth class as the most probable class for an object, in which case the bar for the ground truth at rank 1 would stretch to 1.0, i.e. all cases. Less ideal but still very good would be the case in which the bars for ground truth and superclass of the ground truth add up to 1.0 at the first rank.

In the example graph of Figure 10.1 on the next page, the total cumulative histogram hits the 1.0-mark at rank 4.

In 2D object classification research, it is common practice to compare the developed classifier to other using the PASCAL Visual Object Classes Challenge [13]. This challenge currently only deals with 2D data, in ca. 20 classes in 4 broad categories and is therefore not an interesting or usable benchmark for use in this thesis.

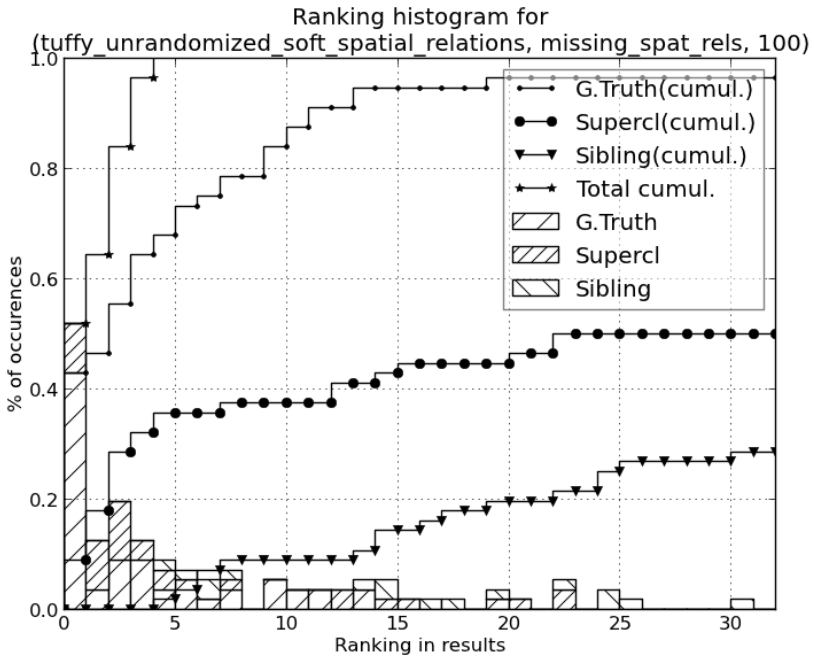

Figure 10.1: Example histogram of rankings. While the bars show a normalized histogram, the lines show the steps for a *cumulative* normalized histogram. The graph marked with stars (as opposed to bullets or triangles) shows the total cumulative histogram, for both ground truth and superclasses of the ground truth

10.3 Parameters for convergence

Before performing further experiments, it is important to tune the performance of Tuffy, in order for it to yield the best results. The key parameter that heavily influences accuracy is the amount of samples that MC-SAT is set to take, as described in Paragraph 6.4.2 on page 28. This is evaluated by performing inference on the same Markov Logic Network on the same scene multiple times for different settings of this parameter. Per setting of the number of MC-SAT samples, 10 inference runs were performed.

In Figures 10.2 and 10.3 on page 52, the average probability with the error over 10 inference runs are shown, for the default 100 and 50.000 samples respectively. The scene used in this evaluation contains four unknown objects of which their class is to be inferred. Although not relevant for what the figures must show, in them, a Pan is confused by the classification scheme for a Teapot. These objects are relatively alike, as both are cylindrical, made of metal and related to stoves. The same goes for the confusion between KitchenStove and ComputerCase, as both can be the same colors and are boxes made of metal. Mainly the dimensions vary, in both (actually: three) cases.

For each of these query objects, and for each setting, the sum of the errors of all possible answers to that query is plotted in Figure 10.4 on page 53. This measure of summed error is not a meaningful measure for classification, but it

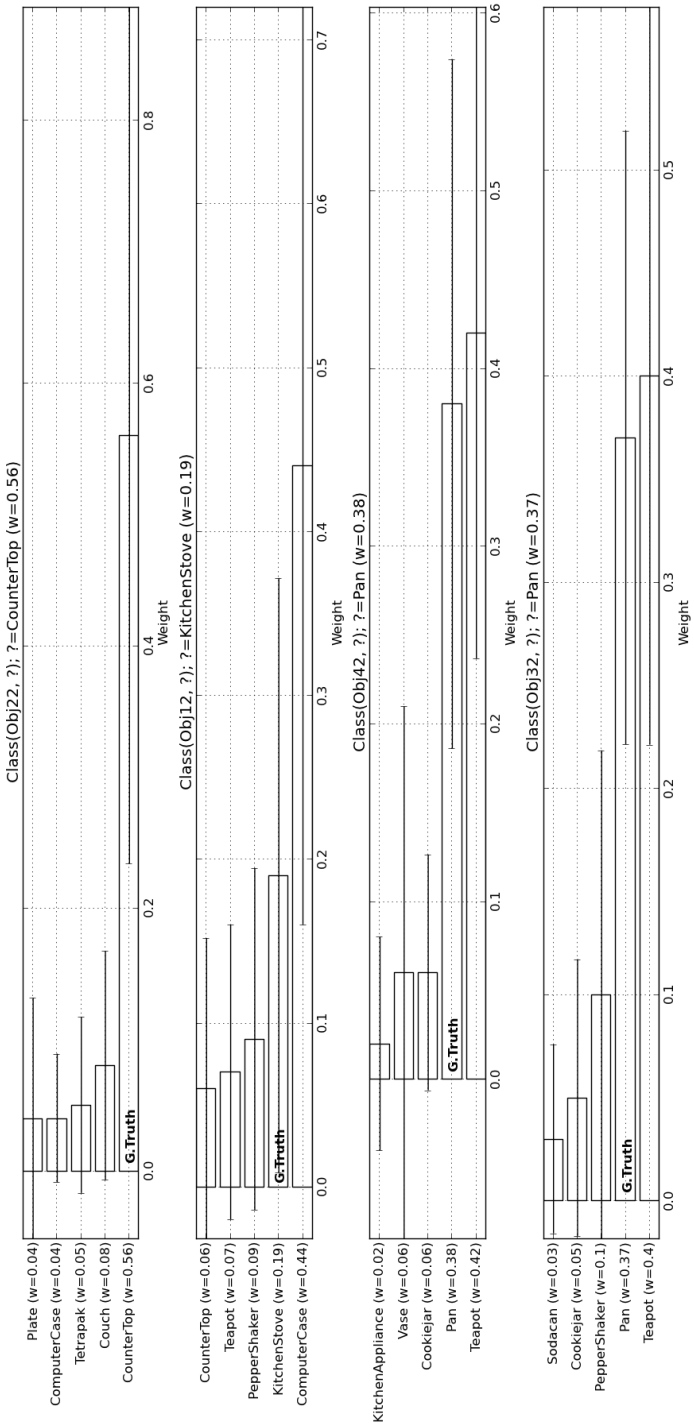

Figure 10.2: Average probabilities with errorbars for MC-SAT with 100 samples (default)

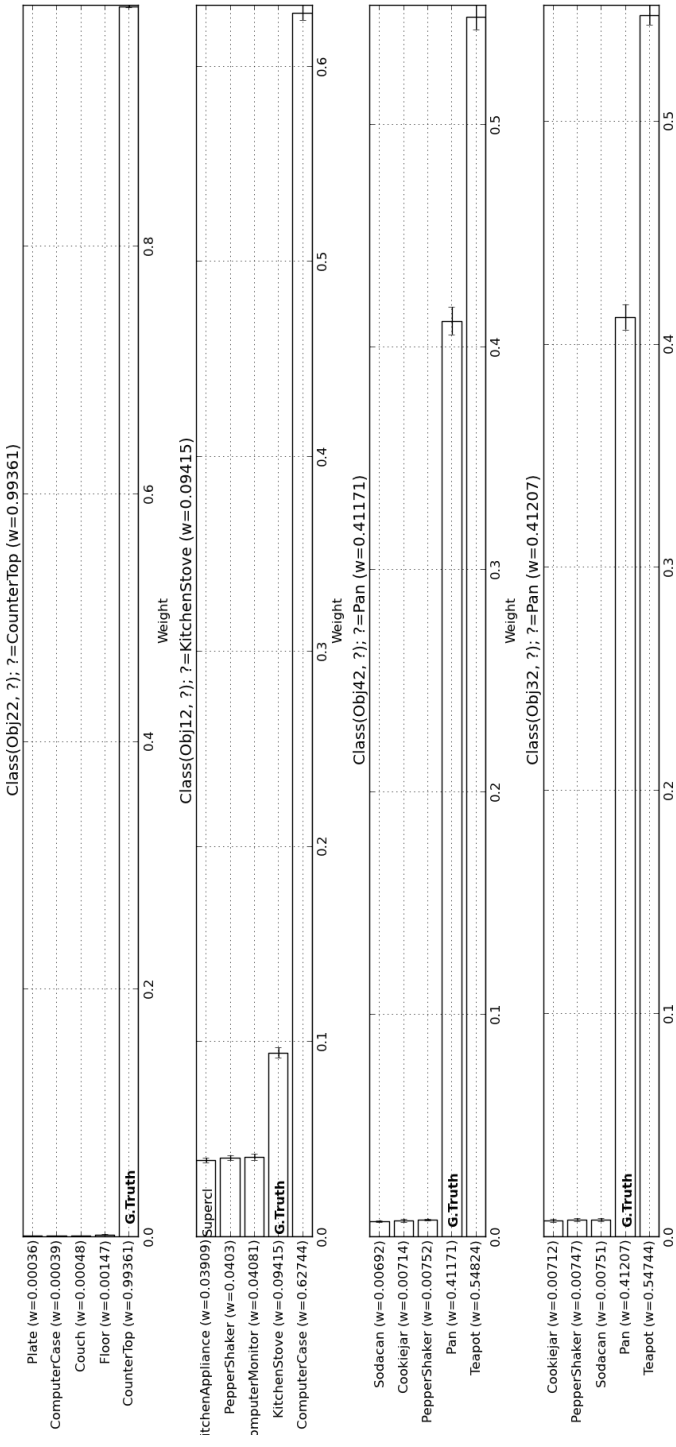

Figure 10.3: Average probabilities with errorbars for MC-SAT with 50000 samples

is a sufficient measure for convergence, i.e. to show the trend of decreasing error with increasing sample count. Unfortunately, taking more samples inherently

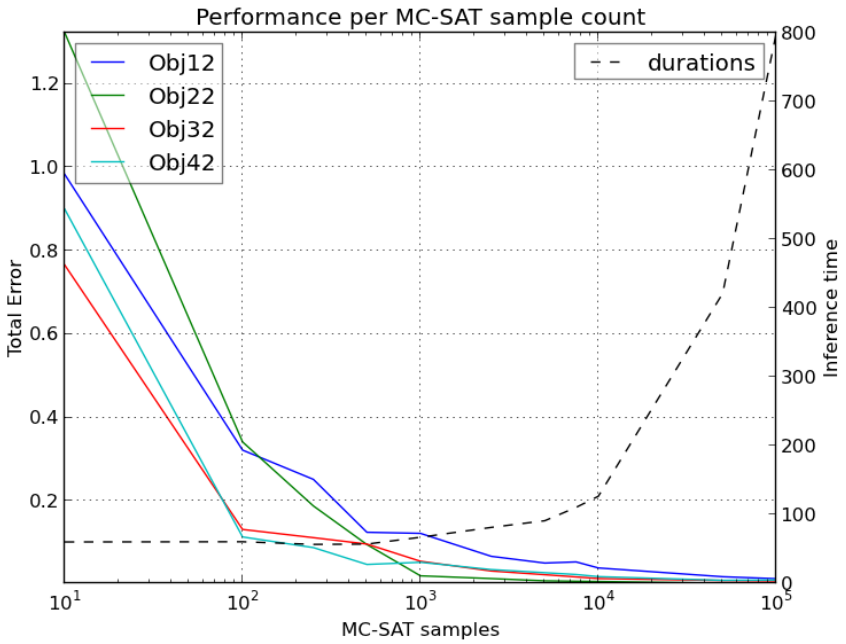

Figure 10.4: Convergence and time consumption of MC-SAT with increasing numbers of samples

takes more time as well, which is also shown in the graph of Figure 10.4.

From this analysis, the parameter for the amount of MC-SAT samples was set to 50.000, as this yields errors smaller than the variations between first and second rank results, and thus the variations between inference runs can be said to least affect the ranking of results, as used in the performance measure described in Section 10.2 on page 48.

10.4 Nominal performance

To evaluate nominal performance, inference is performed once (since convergence is reached) on the 16 scenes mentioned earlier. The MLN that performs best under these nominal conditions, with all information except about class available, is used in subsequent experiments.

10.4.1 Rule Complexity

During the development of the Markov Logic Network, several variants were developed, each variant with a different complexity of rules. In this experiment, the performance for each rule complexity is evaluated.

From the data in Table 10.1, it can be observed that a rule complexity of two properties combined performs best, with a success rate of 73%, which is

Combined properties	Total Success rate	Inference time on scene 1 [s]
1	68%	118
2	73%	80
3	54%	53
4	48%	53
5	34%	52
6	11%	41

Table 10.1: Success rate for varying rule complexity.

reasonable.

If only 73% of the objects are classified correctly, i.e. as their ground truth, then what happens in the other 27% of cases? These are described below.

Scene 1,2 A RoundTable is misclassified as a Chair. Both are made of the same material and have the same color and material and are both related to the floor.

Scene 3 A Knife, Spoon and Fork are present, which are all misclassified as a ComputerCase. This is rather odd, as the only common property between utensils and computer cases is that they are made of metal.

Scene 5 A Book is misclassified as a remote control. Both are box-shaped, can be black and on top of tables. Dimensions and material do not match however.

Scene 6 A Tea cup is misclassified as a pepper shaker in scene 6. Only their diameter is different, so quite understandable.

Scene 7 The confusion between teapots and pans arises again, as was seen investigating the convergence (Figure 10.2 on page 51 and 10.3 on page 52).

Scene 9 The results of Scene 9 contain a confusion between SoupPlate (a subclass of Plate) and a Vase. The only difference there is in dimensions and context. The ontology does not contain flowers; if they were in the context of the vase, possibly, the confusion could have been resolved.

Scene 15 A FruitBowl is confused for a Coffeemaker, with a small difference in probability. These classes of objects however are quite different.

Scene 16 In scene 16, only the DrinkingGlass is classified correctly, the other objects are confused in the same way seen in other scenes.

Further, it can be seen that the performance actually decreases with increasing rule complexity. This is due to the fact that for a rule complexity of 2 and 3, more combinations of properties, i.e. rules can be created. As the weight of a rule is set to be proportional to its complexity, the result is that for complexity 2, the total weight of the matching rules is the greatest.

The expectation is that performance would increase with greater rule complexity. With a different weight-assignment function, other than a proportional one, the MLN can likely be adjusted to compensate for this effect and to match the expectation. This tuning and tweaking of the used MLN, however, is regarded as future work.

10.4.2 Scene complexity

The scenes used in the evaluation have varying amounts of objects in them. It is interesting to note that the success rate decreases with increased scene complexity, i.e. more objects per scene. For low-complexity scenes with only 2 objects, the success rate is 75%, while for scenes 6 objects, the success rate is just over 50%.

Scene complexity	Success rate	Scenes complexity occurrences	Avg. Inference time [s]
2	6/8	4	80
3	13/15	5	122
4	13/16	4	107
5	2/5	1	155
6	7/12	2	153
55	26/55	1	1687

Table 10.2: Success rate for varying scene complexity.

The last row in Table 10.2 shows results for the integration of all scenes into one large scene of 55 objects (in 16 groups). The time needed for inference on this larger scene is ca. 1700 seconds, which is less than the total time needed for the 16 scenes separately. Reducing the needed inference time was the motivation for trying to integrate all scenes into one. The success rate, however, for this integrated scene is significantly lower than the success rate of the separate scenes combined and thus scenes were kept separate.

A possible explanation as to why the success rate for more complex scenes is lower, could be the increased amount of possible worlds and the resulting search space, which is inherently much larger. Potentially, this problem could be mitigated by increasing the amount of samples taken. The set of possible worlds however grows exponentially with the amount of objects in it. Getting the same sampling ratio of the possible worlds, i.e. the ratio of the amount of samples divided by the amount of possible worlds, would require an enormous amount of needed samples and would result in an impractically long time needed for inference.

10.5 Context Makes the Difference

The use of context (i.e. spatial relations to other objects) is expected to be a useful clue for classification. In order to evaluate this expectation, some experiments are performed. In the first, the spatial relations to other objects in a scene are omitted in the evidence. For a second experiment regarding context, the evidence stated the exact class of the context objects.

10.5.1 Influence of spatial relations

The scenes used for evaluating nominal performance are altered for this first experiment, by removing all spatial relations from the evidence.

As can be seen in Figure 10.5 on the following page, the omission of the spatial relations does not affect the success rate at all. This, however, is the case when all other possible information is available. What happens when all the

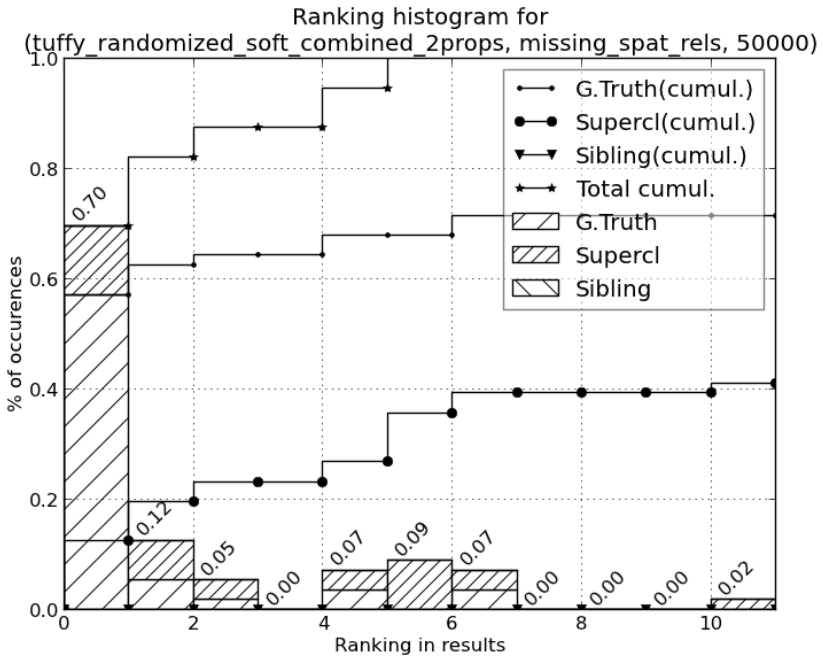

Figure 10.5: Performance with missing spatial relations

information is shape and color? The success rate is only 21% in this case. When information about spatial relations is also included, the success rate slightly increases to 23%. Thus, in the used MLN, the context does in fact not have a large effect and but does benefit classification in cases where there is only little information. The histograms for these cases are listed in Appendix C on page 78, in Figure C.1 on page 78 and Figure C.2 on page 79 respectively.

10.5.2 Known context classes

In previous experiments, no class was known for all objects in the scene. But what happens when only one class in unknown, and all other classes are known? The resulting success rate is 81%, with 3 misclassified objects. The histogram for this case is shown in in Figure C.3 on page 80 in Appendix C. In the first failing scene, scene 8, the ground truth (Floor) has probability 0.4994, while the most probable class (CounterTop) has probability 0.5006, which is a minute difference. The confusion is understandable: A floor and countertop can both be made of stone and are both flat (box-shaped), as can be seen in Appendix B on page 72 In the second failing scene, the difference in probability between the superclass (Plate) of the ground truth (Soup Plate) and the most probable class (Cup) is only 0.003, which is again a minute difference. Plates and cups both have a lot of similarities, both in appearance and context. In each failing result, where the ground truth class is a Spoon, the differences in probability for the superclass (Utensil) and the most probable class (FruitBowl) is larger, over 0.1. The confusion in this case is also less understandable, because the shape and

dimensions of a Spoon and FruitBowl are quite different. Their context and material however are similar. Increasing or otherwise tuning the weights for the inference rules concerning context could possibly improve the results for at least the first two failing cases.

10.6 The Influence of Dimensions

Household objects come in all sorts of shapes and sizes. A goal of this thesis is to generalize knowledge about objects. One of the generalizations it must be able to make are those concerning dimensions of objects. For example, tables can be of various heights, plates can have varying diameters and so on. This experiment shows that the classification scheme developed here still classifies tables correctly as tables even when the dimensions vary very much.

10.6.1 Dimensions Omitted

Let us first see what the effect of not including dimensions is, i.e. only information about color, shape, material and spatial relations is available. This results in a success rate of 30%, which is much lower than the nominal 73% success rate. This is due to the fact that there are multiple dimensions, and account for a large part to a class' description and thus, a lot of information is missing.

10.7 Dealing with Incorrect Observations

The classification scheme depends on lower level classifiers in a perception pipeline. These lower level classifiers, for instance for color, may mis-classify and incorrectly indicate that some object is blue when it in fact is red. Preferably, the developed system still makes correct classifications under the condition that other evidence is correct, similar to the "peer-pressure" discussed earlier.

Multiple variants of this experiment were performed: one for each property and finally one where a random 5% of the evidence facts (i.e. one in twenty) was perturbed. In the cases where facts concerning a single property are perturbed, the value of *each* fact concerning this property was replaced by a different value for that property. E.g. when perturbing the property Color, a coke can that is in reality red could be observed to be orange, blue or any other color equally likely.

As shown in Figure 10.6 on the following page, classification on these incorrect, perturbed observations is still quite successful. This holds for most properties, as can be seen in Table 10.3. Only perturbing observations about all dimensions affects the success rate severely. As discussed earlier, this is due to the fact that dimensions are a large part of a class' description. In the 70% of object where the dimensions are observed wrong and are misclassified due to that, the classes of the objects are arguable quite similar, if the dimensions are ignored or wrong.

10.8 Comparison to other methods

Above, the performance of the classification scheme is discussed. But, how does it compare to some other methods for object classification?

Incorrect property	Success rate
Nominal/All correct	73%
Colors	73%
Dimensions	32%
Materials	66%
5% of observations	70%
Spatial Relations	70%

Table 10.3: Success rate for incorrect properties.

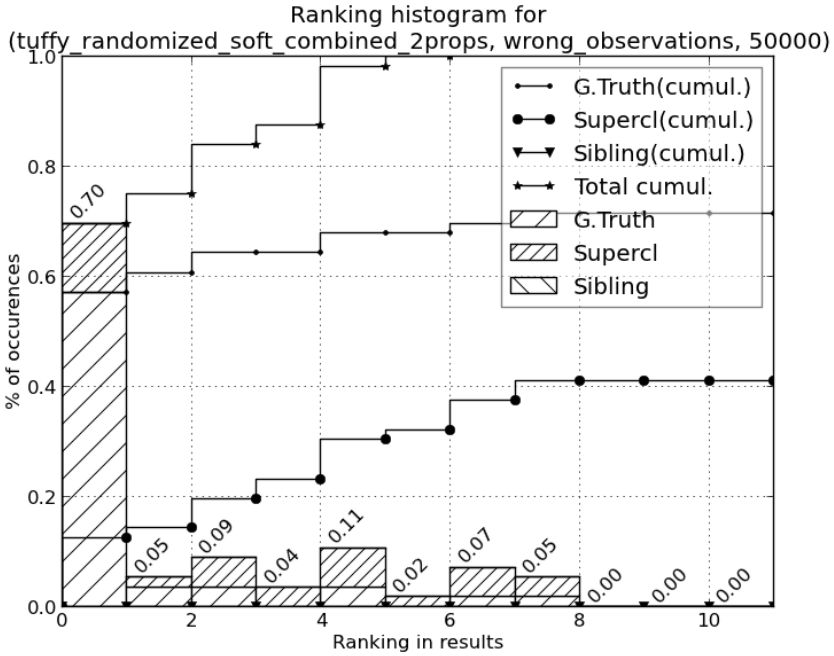

Figure 10.6: Performance with 5% incorrect observations

10.8.1 Naive Bayes Classifier

Naive Bayes Classifier are a relatively simple method of classification, that assumes independence of the features f_i and that the output classes c are mutually exclusive and exhaustive [18]. It tries to find the most probable class using the expression below:

$$argmax_c P(C = c) \prod_{i=1}^{n} P(F_i = f_i | C = c)$$

At least one of the assumptions, however, does not hold for the case of this thesis: the hierarchy of classes make them mutually not exclusive. Further, independence does not always hold either, for instance because the property of color has a relation to that of material. In the final MLN, this relation is not included but can be added with great ease, while this is not possible in a Naive Bayes Classifier. In short, independence in assumed in a MLN but can easily added. However, it can be said that the combined rules somewhat resemble a Naive Bayes Classifier or a template of one. Thus, comparing the developed scheme against a Naive Bayes Classifier is not possible without harming the contributions of this thesis.

10.8.2 Conditional Random Fields

As can be seen in Figure 9.3 on page 44, a scene can be regarded as resembling an instance of a conditional random field, in which for each hidden node, i.e. the class of an object, is conditioned on observed other properties of an object and its neighboring objects. As such, one reason to not use conditional random fields is that CRFs also cannot perform hierarchical classification, in which one class subsumes another and are not mutually exclusive. Furthermore CRFs are not knowledge or logic based and as such cannot symbolically incorporate common sense knowledge and reasoning. As for Naive Bayes Classifiers, also comparing against CRFs is not possible while respecting the contributions of this thesis.

Chapter 11

Conclusion

This thesis presented a method for performing object classification using common sense knowledge, by performing probabilistic inference on such a knowledge base in combination with symbolic observations. As demonstrated above, it can be used for object classification with some success.

In short, the approach taken consists of a common sense knowledge base, stating weighted facts about the appearance of ca. 115 household object classes and their context. The knowledge base also includes weighted inference rules, which are used to probabilistically infer the class of observed objects, given some observations on those objects represented in a symbolic rather than numeric fashion. The knowledge and rules are expressed in Markov Logic and thus it has been shown that Markov Logic can be used to express common sense knowledge and reasoning rules and to classify objects. Hybrid Markov Logic Networks seemed very useful for this thesis, but was hindered by insufficient documentation regarding the implementation and related syntax in the Alchemy package.

This results in a 73% success rate on a small test set of 16 scenes. When some observations (concerning color, material, spatial relations, or a random 5%) are omitted or wrong, the classification scheme still continues to perform similar to the nominal case. Observations concerning the dimensions of objects heavily influence classification results, omitting or perturbing them leads to a drastic drop in the success rate. The influence of context is also significant in this method: when the class of only one object is unknown with others known, the success rate increases significantly. With some tuning, it is expected that the performance can be greatly improved. This is however left to future work.

Further, it has been shown that multiple algorithms can be used to perform inference on Markov Logic Networks in the case of this thesis. Different implementations, namely Alchemy and Tuffy, of the same algorithm (MC-SAT) vary greatly in the runtime they require for the same task. The implementation of Markov Logic provided by Tuffy requires a runtime that is in the order of a few minutes, whereas Alchemy already takes tens of hours for relatively simple cases.

Another conclusion is that there are no strict formal systems for probabilistic description logics available that can perform the task of classification of instances of classes. Instead, a custom simple description logic implemented in Markov Logic is defined.

Chapter 12

Future work

As mentioned in Chapter 9.7 on page 46, the approach taken is mainly a proof of concept. It can be improved in many ways, most notably by also including knowledge about object parts and by learning some of the knowledge and weights. Other ideas for improvement or for further research are mentioned below.

Tuning and Tweaking

The Markov Logic Network rules, the knowledge the MLN contains and Tuffy together have a multitude of parameters. Each rule has a weight, the possibility of tuning each already yields an large space to explore. Furthermore, other rules can be added. For instance, currently, color and material are unrelated, while in practice, some relations between those certainly exist. Tuning these weights is a very tedious task and as such, it is better left to be learned automatically.

Distributions over Dimensions

The used knowledge base lists one value for each dimension of each object. Many objects however come in various sizes. To use the example of soda cans once more, those are manufactured in multiple sizes. The knowledge base can be extended to reflect this. Also, furniture for instance also comes in varying dimensions. In the knowledge base, a distribution over dimensions could be included, that states that for example tables are on average 74 centimeters high, but tables also exist that have different dimensions. A step in this direction has already been made, but is not included in this thesis due to time constraints.

Parts

As noted in some of the examples given and in the literature survey, the use of parts could help classification of the whole object. The ontology used in this thesis does in fact list some parts of household objects, such as handles and buttons, but are not used in the developed classification method due to time constraints.

Learning

No emphasis is put in this thesis on learning. There are several things can could be learned automatically in the light of this thesis. The simplest of these is to learn the weights assigned to individual axioms. This assumes that the axioms themselves are already defined. A field of study is dedicated to structure learning. Structure learning, often directed to graphical models, attempts to learn what random variables exist and influence each other. If this was fully developed and employed to the fullest, that would allow a robot to, over time, learn to semantically recognize objects.

Learning new (sub)classes

If a superclass has a few very specific subclasses, it would be very useful to learn different subclasses automatically, possibly aided by human guidance.

Enhanced spatial reasoning

The integration of enhanced spatial reasoning, combined with world hypothesis evaluation/verification could be useful to the world model system as mentioned in 7.4 on page 32. Consider two objects, A and B, and a situation in which B is on top of A. If the z-coordinates of B's bottom and of A's top are uncertain to some degree, common sense knowledge can provide some additional insights. All situations in which B's bottom is lower than A's top are practically impossible in the case both A and B are rigid objects. Such a situation would be a collision, where B is inside of A, and should be assigned a low probability.. Spatial reasoning can also be of help when reasoning with different parts of objects.

Extension to other properties and integration of other ontologies

In the thesis, only quite basic properties are used. Other properties can easily be added, such as the text on an object or the contents of an object. For instance, imagine an ontology of brands and their products. A transparent cylinder with the text "Water" or "Spa" on it, is then likely to be a bottle of water.

Also, ontologies of other types of knowledge could be integrated. If there is a knowledge base of actions or operations and what objects can perform each action, some interesting inferences could be made. For instance, a dishwasher is a device that cleans dishes, so the dishes that go in are supposed to come out clean. An advanced robot could perceive dirty dishes going in at some time and later observing them coming out clean. As there is only one class of objects that performs such an action, the object could be classified as a dishwasher.

Inferring what to inspect

If an object is not classified in a leaf class (not a superclass of any other classes), the robot needs to perform more/other perception techniques to gather

more information on the object. A system could be devised that infers what perception best to perform in order to determine the leaf class of an object.

Bibliography

[1] H. Bay, T. Tuytelaars, and L. V. Gool. Surf: Speeded up robust features. In *In ECCV*, pages 404–417, 2006.

[2] BayesOWL Project. Bayesowl manual, July 2012.

[3] S. Bechhofer, F. van Harmelen, J. Hendler, I. Horrocks, D. L. McGuinness, P. F. Patel-Schneider, and L. A. Stein. Owl web ontology language reference. `http://www.w3.org/TR/owl-ref/`, Feb. 2004.

[4] G. Bouchard and B. Triggs. Hierarchical part-based visual object categorization. In *Computer Vision and Pattern Recognition, 2005. CVPR 2005. IEEE Computer Society Conference on*, volume 1, pages 710–715 vol. 1, june 2005.

[5] G. Carneiro and D. Lowe. Sparse flexible models of local features. *Computer Vision–ECCV 2006*, pages 29–43, 2006.

[6] R. Carvalho, K. Laskey, P. Costa, M. Ladeira, L. Santos, and S. Matsumoto. Unbbayes: modeling uncertainty for plausible reasoning in the semantic web. *Semantic Web, IN-TECH Publishing, ISBN*, pages 978–953, 2010.

[7] G. Casella and E. I. George. Explaining the gibbs sampler. *The American Statistician*, 46(3):pp. 167–174, 1992.

[8] G. Csurka, C. R. Dance, L. Fan, J. Willamowski, and C. Bray. Visual categorization with bags of keypoints. In *In Workshop on Statistical Learning in Computer Vision, ECCV*, pages 1–22, 2004.

[9] P. da Costa, K. Laskey, and K. Laskey. Pr-owl: A bayesian ontology language for the semantic web. *URSW. LNCS*, pages 88–107, 2008.

[10] P. Domingos, S. Kok, H. Poon, M. Richardson, and P. Singla. Unifying logical and statistical ai. http://www.cs.washington.edu/homes/pedrod/papers/aaai06c.pdf, 2006.

[11] P. Domingos and J. Wang. Hybrid markov logic networks. http://www.cs.washington.edu/homes/pedrod/papers/aaai08b.pdf, 2008.

[12] 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*, 61(2):95–105, February 2013.

[13] 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.

[14] L. Fei-Fei, R. Fergus, and P. Perona. One-shot learning of object categories. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 28(4):594–611, 2006.

[15] S. Jansen, J. Elfring, and M. van de Molengraft. Object appearance prediction and active object search using probabilistic object relations. Master's thesis, Eindhoven University of Technology, July 2012.

[16] H. Kautz, B. Selman, and Y. Jiang. A general stochastic approach to solving problems with hard and soft constraints. In *The Satisfiability Problem: Theory and Applications*, pages 573–586. American Mathematical Society, 1996.

[17] S. Kok, P. Singla, M. Richardson, P. Domingos, M. Sumner, H. Poon, and D. Lowd. The alchemy system for statistical relational ai. *University of Washington, Seattle*, 2005.

[18] D. Koller and N. Friedman. *Probabilistic graphical models: principles and techniques*. MIT press, 2009.

[19] D. Koller, A. Levy, and A. Pfeffer. P-Classic: A tractable probabilistic description logic. In H. Shrobe and T. Senator, editors, *Proceedings of the Thirteenth National Conference on Artificial Intelligence and the Ninth Innovative Applications of Artificial Intelligence Conference, Vol. 1*, pages 390–397, Menlo Park, California, 1996. American Association for Artificial Intelligence, AAAI Press.

[20] M. Krotzsch, F. Simancik, and I. Horrocks. A description logic primer. *CoRR*, abs/1201.4089, 2012.

[21] B. Leibe, A. Leonardis, and B. Schiele. Robust Object Detection with Interleaved Categorization and Segmentation. *International Journal of Computer Vision*, 77:259–289, 2008.

[22] D. G. Lowe. Distinctive image features from scale-invariant keypoints. *International Journal of Computer Vision*, 60:91–110, 2004. 10.1023/B:VISI.0000029664.99615.94.

[23] E. Marinari and G. Parisi. Simulated tempering: a new monte carlo scheme. *EPL (Europhysics Letters)*, 19(6):451, 2007.

[24] Z.-C. Marton, D. Pangercic, R. B. Rusu, A. Holzbach, and M. Beetz. Hierarchical object geometric categorization and appearance classification for mobile manipulation. In *Proceedings of 2010 IEEE-RAS International Conference on Humanoid Robots*, Nashville, TN, USA, December 6-8 2010.

[25] O. M. Mozos, Z. C. Marton, and M. Beetz. Furniture Models Learned from the WWW – Using Web Catalogs to Locate and Categorize Unknown Furniture Pieces in 3D Laser Scans. *Robotics & Automation Magazine*, 18(2):22–32, 2011.

[26] M. Niepert, J. Noessner, and H. Stuckenschmidt. Log-linear description logics. *Proceedings of IJCAI*, 2011.

[27] F. Niu, C. Ré, A. Doan, and J. W. Shavlik. Tuffy: Scaling up statistical inference in markov logic networks using an rdbms. *PVLDB*, 4(6):373–384, 2011.

[28] J. Pearl. *Reverend Bayes on inference engines: a distributed hierarchical approach.* Cognitive Systems Laboratory, School of Engineering and Applied Science, University of California, Los Angeles, 1982.

[29] M. Pechuk, O. Soldea, and E. Rivlin. Learning function-based object classification from 3d imagery. *Computer Vision and Image Understanding*, 110(2):173–191, 2008.

[30] H. Poon and P. Domingos. Sound and efficient inference with probabilistic and deterministic dependencies. In *Proceedings of the National Conference on Artificial Intelligence*, volume 21, page 458. Menlo Park, CA; Cambridge, MA; London; AAAI Press; MIT Press; 1999, 2006.

[31] S. Ruiz-Correa, L. G. Shapiro, and M. Meila. A new paradigm for recognizing 3-d object shapes from range data. In *In Proc. of the Int. Conf. on Computer Vision (ICCV*, pages 1126–1133, 2003.

[32] S. Russell, P. Norvig, E. Davis, S. Russell, and S. Russell. *Artificial intelligence: a modern approach.* Prentice hall Upper Saddle River, NJ, 2010.

[33] R. Rusu, A. Holzbach, M. Beetz, and G. Bradski. Detecting and segmenting objects for mobile manipulation. In *Computer Vision Workshops (ICCV Workshops), 2009 IEEE 12th International Conference on*, pages 47–54, 27 2009-oct. 4 2009.

[34] R. B. Rusu. *Semantic 3D Object Maps for Everyday Manipulation in Human Living Environments.* phd, Tecnische Universitatet Muenchen, Munich, Germany, 10/2009 2009.

[35] R. B. Rusu, G. Bradski, R. Thibaux, and J. Hsu. Fast 3d recognition and pose using the viewpoint feature histogram. In *Proceedings of the 23rd IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Taipei, Taiwan, 10/2010 2010.

[36] R. B. Rusu and S. Cousins. 3D is here: Point Cloud Library (PCL). In *IEEE International Conference on Robotics and Automation (ICRA)*, Shanghai, China, May 9-13 2011.

[37] R. B. Rusu, Z. C. Marton, N. Blodow, A. Holzbach, and M. Beetz. Model-based and Learned Semantic Object Labeling in 3D Point Cloud Maps of Kitchen Environments. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, St. Louis, MO, USA, October 11-15 2009.

[38] SemanticWeb.org. Bayes owl, 2009. [Online; accessed 9-July-2012].

[39] P. Singla and P. Domingos. Lifted first-order belief propagation. *Proceedings of the 23rd national conference on Artificial intelligence*, 2:1094–1099, 2008.

[40] Y. Su, M. Allan, and F. Jurie. Improving object classification using semantic attributes. In *Proceedings of the British Machine Vision Conference*, pages 26.1–26.10. BMVA Press, 2010. doi:10.5244/C.24.26.

[41] C. Sutton and A. McCallum. *An introduction to conditional random fields for relational learning*. Introduction to statistical relational learning. MIT Press, 2006.

[42] R. Szeliski. *Computer Vision: Algorithms and Applications*. Springer, draft edition, September 3 2010.

[43] J. Tauberer. What is rdf, 2 2012.

[44] M. Tenorth, L. Kunze, D. Jain, and M. Beetz. KNOWROB-MAP – Knowledge-Linked Semantic Object Maps. In *Proceedings of 2010 IEEE-RAS International Conference on Humanoid Robots*, Nashville, TN, USA, December 6-8 2010.

[45] S. van den Dries. World modeling in robotics: Probabilistic multiple hypothesis anchoring. Master's thesis, Eindhoven University of Technology, 2011.

[46] W3C OWl Working Group. Implementations, July 2012. [Online; accessed 18-July-2012].

[47] Wikipedia. Length — wikipedia, the free encyclopedia, 2012. [Online; accessed 24-October-2012].

[48] Wikipedia. Support vector machine — wikipedia, the free encyclopedia, 2012. [Online; accessed 13-June-2012].

[49] Wiktionary. depth — wiktionary, the free dictionary, 2012. [Online; accessed 24-October-2012].

[50] Wiktionary. side — wiktionary, the free dictionary, 2012. [Online; accessed 24-October-2012].

[51] Willow Garage. Ros: Robot operating system. `http://www.ros.org`, june 2012.

[52] WordNet. Length — wordnet 3.1, 2012. [Online; accessed 24-October-2012].

Appendices

Appendix A

Implemented Rules

```
// Evidence
*Subclass(class, class)
Class(instance, class!)

ClsColor(class, color)
*Color(instance, color!)

ClsShape(class, shape)
*Shape(instance, shape!)

ClsMaterial(class, material)
*Material(instance, material!)

ClsNeighbor(class, class)
Neighbor(instance, instance)

IsAboveOf(instance, instance)
ClsIsAboveOf(class, class)

IsBelowOf(instance, instance)
ClsIsBelowOf(class, class)

InBehindOf(instance, instance)
ClsInBehindOf(class, class)

InCenterOf(instance, instance)
ClsInCenterOf(class, class)

InFrontOf(instance, instance)
ClsInFrontOf(class, class)

IsInside(instance, instance)
ClsIsInside(class, class)

IsOnPhysical(instance, instance)
```

```
ClsIsOnPhysical(class, class)

IsNear(instance, instance)
ClsIsNear(class, class)

ToTheSideOf(instance, instance)
ClsToTheSideOf(class, class)

ToTheLeftOf(instance, instance)
ClsToTheLeftOf(class, class)

ToTheRightOf(instance, instance)
ClsToTheRightOf(class, class)

ClsLength(class, discretedimension)
Length(instance, discretedimension!) //Instance can only have 1 Length

Diameter(instance, discretedimension!) //Instance can only have 1 Diameter
ClsDiameter(class, discretedimension)

Width(instance, discretedimension!) //Instance can only have 1 Width
ClsWidth(class, discretedimension)

Depth(instance, discretedimension!) //Instance can only have 1 Depth
ClsDepth(class, discretedimension)

Colormatch(instance, class!)
Shapematch(instance, class!)
Materialmatch(instance, class!)
Lengthmatch(instance, class!)
Diametermatch(instance, class!)
Widthmatch(instance, class!)
Depthmatch(instance, class!)

Neighbormatch(instance, class!)

// Rules:
!Class(i1, c1) v !Class(i1, c2) v c1 = c2.

//Subclass(sub, super) ^ ClsColor(super, supercolor) => ClsColor(sub, supercolor)
!Subclass(sub, super) v !ClsColor(super, supercolor)        v ClsColor(sub, supercolor).
!Subclass(sub, super) v !ClsShape(super, supershape)        v ClsShape(sub, supershape).
!Subclass(sub, super) v !ClsMaterial(super, supermaterial)  v ClsMaterial(sub, supermaterial).
!Subclass(super, sub) v !ClsNeighbor(super, neighCls)       v ClsNeighbor(sub, neighCls).
!Subclass(super, sub) v !ClsLength(super, neighCls)         v ClsLength(sub, neighCls).
!Subclass(super, sub) v !ClsDiameter(super, neighCls)       v ClsDiameter(sub, neighCls).
!Subclass(super, sub) v !ClsWidth(super, neighCls)          v ClsWidth(sub, neighCls).
!Subclass(super, sub) v !ClsDepth(super, neighCls)          v ClsDepth(sub, neighCls).

// Rules - Clues: (These could be hard constrainst)
```

```
1.0 !ClsColor(cls, clscolor)        v !Color(inst, clscolor)        v Colormatch(inst, cls)
1.0 !ClsShape(cls, clsshape)        v !Shape(inst, clsshape)        v Shapematch(inst, cls)
1.0 !ClsMaterial(cls, clsmaterial)  v !Material(inst, clsmaterial)  v Materialmatch(inst, cls)
1.0 !ClsLength(cls, clsmaterial)    v !Length(inst, clsmaterial)    v Lengthmatch(inst, cls)
1.0 !ClsDiameter(cls, clsmaterial)  v !Diameter(inst, clsmaterial)  v Diametermatch(inst, cls)
1.0 !ClsWidth(cls, clsmaterial)     v !Width(inst, clsmaterial)     v Widthmatch(inst, cls)
1.0 !ClsDepth(cls, clsmaterial)     v !Depth(inst, clsmaterial)     v Depthmatch(inst, cls)

20.0 !Colormatch(inst, cls) v !Shapematch(inst, cls) v Class(inst, cls)
20.0 !Colormatch(inst, cls) v !Materialmatch(inst, cls) v Class(inst, cls)
20.0 !Colormatch(inst, cls) v !Lengthmatch(inst, cls) v Class(inst, cls)
20.0 !Colormatch(inst, cls) v !Diametermatch(inst, cls) v Class(inst, cls)
20.0 !Colormatch(inst, cls) v !Widthmatch(inst, cls) v Class(inst, cls)
20.0 !Colormatch(inst, cls) v !Depthmatch(inst, cls) v Class(inst, cls)
20.0 !Shapematch(inst, cls) v !Materialmatch(inst, cls) v Class(inst, cls)
20.0 !Shapematch(inst, cls) v !Lengthmatch(inst, cls) v Class(inst, cls)
20.0 !Shapematch(inst, cls) v !Diametermatch(inst, cls) v Class(inst, cls)
20.0 !Shapematch(inst, cls) v !Widthmatch(inst, cls) v Class(inst, cls)
20.0 !Shapematch(inst, cls) v !Depthmatch(inst, cls) v Class(inst, cls)
20.0 !Materialmatch(inst, cls) v !Lengthmatch(inst, cls) v Class(inst, cls)
20.0 !Materialmatch(inst, cls) v !Diametermatch(inst, cls) v Class(inst, cls)
20.0 !Materialmatch(inst, cls) v !Widthmatch(inst, cls) v Class(inst, cls)
20.0 !Materialmatch(inst, cls) v !Depthmatch(inst, cls) v Class(inst, cls)
20.0 !Lengthmatch(inst, cls) v !Diametermatch(inst, cls) v Class(inst, cls)
20.0 !Lengthmatch(inst, cls) v !Widthmatch(inst, cls) v Class(inst, cls)
20.0 !Lengthmatch(inst, cls) v !Depthmatch(inst, cls) v Class(inst, cls)
20.0 !Diametermatch(inst, cls) v !Widthmatch(inst, cls) v Class(inst, cls)
20.0 !Diametermatch(inst, cls) v !Depthmatch(inst, cls) v Class(inst, cls)
20.0 !Widthmatch(inst, cls) v !Depthmatch(inst, cls) v Class(inst, cls)

10.0 !ClsIsAboveOf(clsA, clsB)  v !Class(instA, clsA) v !Class(instB, clsB) v IsAboveOf(instA, instB)
10.0 !ClsIsBelowOf(clsA, clsB)  v !Class(instA, clsA) v !Class(instB, clsB) v IsBelowOf(instA, instB)
10.0 !ClsInBehindOf(clsA, clsB) v !Class(instA, clsA) v !Class(instB, clsB) v InBehindOf(instA, instB)
10.0 !ClsInCenterOf(clsA, clsB) v !Class(instA, clsA) v !Class(instB, clsB) v InCenterOf(instA, instB)
10.0 !ClsInFrontOf(clsA, clsB) v !Class(instA, clsA) v !Class(instB, clsB) v InFrontOf(instA, instB)
10.0 !ClsIsInside(clsA, clsB) v !Class(instA, clsA) v !Class(instB, clsB) v IsInside(instA, instB)
10.0 !ClsIsOnPhysical(clsA, clsB) v !Class(instA, clsA) v !Class(instB, clsB) v IsOnPhysical(instA, instB)
10.0 !ClsIsNear(clsA, clsB) v !Class(instA, clsA) v !Class(instB, clsB) v IsNear(instA, instB)
10.0 !ClsToTheSideOf(clsA, clsB) v !Class(instA, clsA) v !Class(instB, clsB) v ToTheSideOf(instA, instB)
10.0 !ClsToTheLeftOf(clsA, clsB) v !Class(instA, clsA) v !Class(instB, clsB) v ToTheLeftOf(instA, instB)
10.0 !ClsToTheRightOf(clsA, clsB) v !Class(instA, clsA) v !Class(instB, clsB) v ToTheRightOf(instA, instB)
```

Appendix B

Class Knowledge

Each fact below that is not postfixed by a dot '.' is suffixed with a random floating point number between 0 and 1. They are omitted for brevity.

```
//Class Knowledge

ClsColor(Apple, Brown_color)
ClsColor(Apple, Red_color)
ClsColor(Apple, Green_color)
ClsDiameter(Apple, 8)
ClsShape(Apple, Sphere_shape)
Subclass(Apple, Fruit).

ClsColor(Banana, Brown_color)
ClsColor(Banana, Yellow_color)
ClsDiameter(Banana, 3)
ClsShape(Banana, Toroid_shape)
Subclass(Banana, Fruit).

ClsColor(Beer_bottle, Brown_color)
ClsDiameter(Beer_bottle, 7)
ClsIsInside(Beer_bottle, Refridgerator)
ClsLength(Beer_bottle, 20)
ClsShape(Beer_bottle, Cylinder_shape)
Subclass(Beer_bottle, Bottle).

ClsColor(Book, Brown_color)
ClsDepth(Book, 2)
ClsIsInside(Book, Closet)
ClsIsOnPhysical(Book, Table)
ClsLength(Book, 20)
ClsMaterial(Book, Paper_material)
ClsShape(Book, Box_shape)
ClsToTheSideOf(Book, Book)
ClsWidth(Book, 12)

ClsColor(Bottle, Green_color)
ClsColor(Bottle, Brown_color)
ClsIsOnPhysical(Bottle, Table)
ClsMaterial(Bottle, Glass_material)
ClsShape(Bottle, Cylinder_shape)
Subclass(Bottle, Closable_container).

ClsIsBelowOf(Chair, Table)
ClsColor(Chair, Brown_color)
ClsDepth(Chair, 50)
ClsIsNear(Chair, Table)
ClsIsOnPhysical(Chair, Floor)
ClsLength(Chair, 90)
ClsMaterial(Chair, Wood_material)
ClsMaterial(Chair, Fabric_material)
ClsMaterial(Chair, Metal_material)
ClsShape(Chair, Complex_shape)
ClsWidth(Chair, 50)
Subclass(Closable_container, Container).

ClsColor(Closet, Brown_color)
ClsDepth(Closet, 60)
ClsIsNear(Closet, Refridgerator)
ClsIsNear(Closet, Dishwasher)
ClsIsOnPhysical(Closet, Floor)
ClsLength(Closet, 150)
ClsMaterial(Closet, Wood_material)
ClsShape(Closet, Box_shape)
ClsWidth(Closet, 90)
Subclass(Closet, Closable_container).

ClsDiameter(Coffee_cup, 5)
ClsLength(Coffee_cup, 11)
Subclass(Coffee_cup, Cup).

ClsColor(Coffeemaker, Gray_color)
ClsDepth(Coffeemaker, 20)
ClsIsNear(Coffeemaker, Coffee_cup)
ClsIsOnPhysical(Coffeemaker, Table)
ClsIsOnPhysical(Coffeemaker, CounterTop)
ClsLength(Coffeemaker, 30)
ClsShape(Coffeemaker, Complex_shape)
ClsWidth(Coffeemaker, 15)
Subclass(Coffeemaker, KitchenAppliance).

ClsColor(Coke_bottle, Red_color)
ClsDiameter(Coke_bottle, 10)
ClsIsInside(Coke_bottle, Refridgerator)
ClsLength(Coke_bottle, 25)
ClsMaterial(Coke_bottle,
    Plastic_material)
ClsShape(Coke_bottle, Cylinder_shape)
Subclass(Coke_bottle, Bottle).

ClsColor(Coke_can, Red_color)
Subclass(Coke_can, Sodacan).

ClsIsBelowOf(ComputerCase, Table)
ClsColor(ComputerCase, Gray_color)
ClsColor(ComputerCase, White_color)
ClsColor(ComputerCase, Black_color)
ClsDepth(ComputerCase, 18)
ClsIsOnPhysical(ComputerCase, Table)
ClsLength(ComputerCase, 47)
ClsMaterial(ComputerCase, Metal_material)
ClsShape(ComputerCase, Box_shape)
ClsWidth(ComputerCase, 43)
```

```
ClsColor(ComputerMonitor, Gray_color)
ClsColor(ComputerMonitor, White_color)
ClsColor(ComputerMonitor, Black_color)
ClsDepth(ComputerMonitor, 10)
ClsInBehindOf(ComputerMonitor, Keyboard)
ClsIsOnPhysical(ComputerMonitor, Table)
ClsLength(ComputerMonitor, 40)
ClsMaterial(ComputerMonitor,
    Plastic_material)
ClsShape(ComputerMonitor, Box_shape)
ClsWidth(ComputerMonitor, 30)

ClsColor(Computermouse, Black_color)
ClsColor(Computermouse, Gray_color)
ClsColor(Computermouse, White_color)
ClsDepth(Computermouse, 4)
ClsIsNear(Computermouse, Keyboard)
ClsIsNear(Computermouse,
    ComputerMonitor)
ClsIsOnPhysical(Computermouse, Table)
ClsLength(Computermouse, 10)
ClsMaterial(Computermouse,
    Plastic_material)
ClsShape(Computermouse, Complex_shape)
ClsToTheRightOf(Computermouse, Keyboard)
ClsWidth(Computermouse, 5)

ClsColor(Cookiejar, Red_color)
ClsDiameter(Cookiejar, 25)
ClsIsInside(Cookiejar, Closet)
ClsIsNear(Cookiejar, Teapot)
ClsIsOnPhysical(Cookiejar, Table)
ClsLength(Cookiejar, 12)
ClsMaterial(Cookiejar, Metal_material)
ClsShape(Cookiejar, Cylinder_shape)
Subclass(Cookiejar, Closable_container).

ClsColor(Couch, Black_color)
ClsColor(Couch, White_color)
ClsColor(Couch, Brown_color)
ClsColor(Couch, Gray_color)
ClsDepth(Couch, 90)
ClsIsNear(Couch, Table)
ClsIsOnPhysical(Couch, Floor)
ClsLength(Couch, 200)
ClsMaterial(Couch, Fabric_material)
ClsShape(Couch, Box_shape)
ClsWidth(Couch, 85)

ClsColor(CounterTop, Black_color)
ClsColor(CounterTop, White_color)
ClsDepth(CounterTop, 64)
ClsIsOnPhysical(CounterTop, Floor)
ClsLength(CounterTop, 200)
ClsMaterial(CounterTop, Stone_material)
ClsMaterial(CounterTop, Metal_material)
ClsShape(CounterTop, Box_shape)
ClsWidth(CounterTop, 82)

ClsColor(Cup, Black_color)
ClsColor(Cup, White_color)
ClsIsOnPhysical(Cup, CounterTop)
ClsMaterial(Cup, Earthenware_material)
ClsShape(Cup, Cylinder_shape)
Subclass(Cup, Open_container).

ClsColor(Dishwasher, White_color)
ClsDepth(Dishwasher, 84)
ClsIsOnPhysical(Dishwasher, Floor)
ClsLength(Dishwasher, 86)
ClsShape(Dishwasher, Box_shape)
ClsWidth(Dishwasher, 60)
Subclass(Dishwasher,
    Closable_container).
Subclass(Dishwasher, KitchenAppliance).

ClsDiameter(DrinkingGlass, 5)
ClsIsInside(DrinkingGlass, Dishwasher)
ClsIsInside(DrinkingGlass, Closet)
ClsIsNear(DrinkingGlass, Plate)
ClsIsOnPhysical(DrinkingGlass, Table)
ClsIsOnPhysical(DrinkingGlass,
    CounterTop)
ClsLength(DrinkingGlass, 15)
ClsMaterial(DrinkingGlass,
    Glass_material)
ClsShape(DrinkingGlass, Complex_shape)
Subclass(DrinkingGlass, Open_container).

ClsColor(Fanta_can, Orange_color)
Subclass(Fanta_can, Sodacan).

ClsDiameter(Flat_plate, 27)
ClsLength(Flat_plate, 2)
ClsToTheLeftOf(Flat_plate,
    Knife_utensil)
ClsToTheRightOf(Flat_plate,
    Fork_utensil)
Subclass(Flat_plate, Plate).
```

```
ClsMaterial(Floor, Wood_material)
ClsMaterial(Floor, Stone_material)
ClsShape(Floor, Box_shape)

ClsIsOnPhysical(Food, CounterTop)

ClsToTheLeftOf(Fork_utensil, Flat_plate)
ClsWidth(Fork_utensil, 3)
Subclass(Fork_utensil, Utensil).

ClsIsInside(Fruit, FruitBowl)
Subclass(Fruit, Food).

ClsColor(FruitBowl, Gray_color)
ClsDiameter(FruitBowl, 30)
ClsMaterial(FruitBowl, Metal_material)
ClsShape(FruitBowl, Sphere_shape)
Subclass(FruitBowl, Open_container).

ClsLength(Frying_pan, 5)
Subclass(Frying_pan, Pan).

ClsColor(Juice_carton, Orange_color)
ClsDepth(Juice_carton, 3)
ClsLength(Juice_carton, 12)
ClsWidth(Juice_carton, 4)
Subclass(Juice_carton, Tetrapak).

ClsColor(Keyboard, Gray_color)
ClsColor(Keyboard, Black_color)
ClsColor(Keyboard, White_color)
ClsDepth(Keyboard, 4)
ClsInFrontOf(Keyboard, Computermouse)
ClsIsOnPhysical(Keyboard, Table)
ClsLength(Keyboard, 40)
ClsMaterial(Keyboard, Plastic_material)
ClsShape(Keyboard, Box_shape)
ClsWidth(Keyboard, 15)

ClsMaterial(KitchenAppliance,
    Plastic_material)
ClsMaterial(KitchenAppliance,
    Metal_material)
ClsShape(KitchenAppliance, Box_shape)

ClsColor(KitchenStove, White_color)
ClsColor(KitchenStove, Black_color)
ClsDepth(KitchenStove, 10)
ClsIsInside(KitchenStove, CounterTop)
ClsIsOnPhysical(KitchenStove, CounterTop)
ClsLength(KitchenStove, 80)
ClsMaterial(KitchenStove, Metal_material)
ClsShape(KitchenStove, Box_shape)
ClsWidth(KitchenStove, 80)
Subclass(KitchenStove, KitchenAppliance).

ClsLength(Knife_utensil, 16)
ClsToTheRightOf(Knife_utensil,
    Flat_plate)
ClsWidth(Knife_utensil, 2)
Subclass(Knife_utensil, Utensil).

ClsIsAboveOf(Lamp, Floor)
ClsIsAboveOf(Lamp, CounterTop)
ClsIsAboveOf(Lamp, Table)
ClsColor(Lamp, Yellow_color)
ClsColor(Lamp, Gray_color)
ClsDiameter(Lamp, 5)
ClsMaterial(Lamp, Glass_material)
ClsShape(Lamp, Sphere_shape)

ClsColor(Milk_bottle, White_color)
ClsDiameter(Milk_bottle, 10)
ClsIsInside(Milk_bottle, Refridgerator)
ClsIsOnPhysical(Milk_bottle, CounterTop)
ClsLength(Milk_bottle, 24)
ClsShape(Milk_bottle, Cylinder_shape)
Subclass(Milk_bottle, Bottle).

ClsColor(Milk_carton, White_color)
ClsDepth(Milk_carton, 9)
ClsIsOnPhysical(Milk_carton, CounterTop)
ClsLength(Milk_carton, 20)
ClsWidth(Milk_carton, 9)
Subclass(Milk_carton, Tetrapak).

ClsColor(OliveOil_bottle, Green_color)
ClsDiameter(OliveOil_bottle, 8)
ClsIsInside(OliveOil_bottle, Closet)
ClsLength(OliveOil_bottle, 24)
ClsShape(OliveOil_bottle, Box_shape)
Subclass(OliveOil_bottle, Bottle).

Subclass(Open_container, Container).

ClsColor(Orange, Orange_color)
ClsDiameter(Orange, 10)
ClsShape(Orange, Sphere_shape)
```

```
Subclass(Orange, Fruit).

ClsColor(Pan, Gray_color)
ClsIsInside(Pan, Dishwasher)
ClsIsNear(Pan, Pan)
ClsIsOnPhysical(Pan, Table)
ClsIsOnPhysical(Pan, KitchenStove)
ClsLength(Pan, 20)
ClsMaterial(Pan, Metal_material)
ClsShape(Pan, Cylinder_shape)
Subclass(Pan, Closable_container).

ClsColor(Pear, Green_color)
ClsColor(Pear, Brown_color)
ClsDiameter(Pear, 7)
ClsLength(Pear, 20)
ClsShape(Pear, Cone_shape)
Subclass(Pear, Fruit).

ClsColor(PepperShaker, White_color)
ClsDiameter(PepperShaker, 3)
ClsIsInside(PepperShaker, Closet)
ClsIsOnPhysical(PepperShaker, Table)
ClsLength(PepperShaker, 7)
ClsMaterial(PepperShaker, Wood_material)
ClsMaterial(PepperShaker,
    Earthenware_material)
ClsMaterial(PepperShaker, Glass_material)
ClsMaterial(PepperShaker, Metal_material)
ClsMaterial(PepperShaker,
    Plastic_material)
ClsShape(PepperShaker, Cone_shape)
ClsShape(PepperShaker, Cylinder_shape)
ClsShape(PepperShaker, Box_shape)
Subclass(PepperShaker, Container).

ClsColor(Plate, White_color)
ClsIsOnPhysical(Plate, Table)
ClsIsOnPhysical(Plate, CounterTop)
ClsMaterial(Plate, Earthenware_material)
ClsShape(Plate, Cylinder_shape)

ClsLength(RectTable, 200)
ClsWidth(RectTable, 100)
Subclass(RectTable, Table).

ClsColor(Refridgerator, White_color)
ClsDepth(Refridgerator, 84)
ClsIsOnPhysical(Refridgerator, Floor)
ClsLength(Refridgerator, 150)
ClsWidth(Refridgerator, 84)
Subclass(Refridgerator,
    Closable_container).
Subclass(Refridgerator,
    KitchenAppliance).

ClsColor(RemoteControl, Black_color)
ClsDepth(RemoteControl, 2)
ClsIsOnPhysical(RemoteControl, Table)
ClsLength(RemoteControl, 20)
ClsMaterial(RemoteControl,
    Plastic_material)
ClsShape(RemoteControl, Box_shape)
ClsWidth(RemoteControl, 5)

ClsDiameter(RoundTable, 100)
Subclass(RoundTable, Table).

ClsColor(Sodacan, Orange_color)
ClsColor(Sodacan, Green_color)
ClsColor(Sodacan, Red_color)
ClsColor(Sodacan, Black_color)
ClsDiameter(Sodacan, 6)
ClsIsInside(Sodacan, Refridgerator)
ClsIsOnPhysical(Sodacan, Table)
ClsIsOnPhysical(Sodacan, CounterTop)
ClsLength(Sodacan, 12)
ClsMaterial(Sodacan, Metal_material)
ClsShape(Sodacan, Cylinder_shape)
Subclass(Sodacan, Closable_container).

ClsDiameter(Soup_plate, 23)
ClsIsOnPhysical(Soup_plate, Flat_plate)
ClsLength(Soup_plate, 4)
ClsToTheLeftOf(Soup_plate,
    Spoon_utensil)
Subclass(Soup_plate, Open_container).
Subclass(Soup_plate, Plate).

ClsToTheRightOf(Spoon_utensil,
    Soup_plate)
ClsWidth(Spoon_utensil, 3)
Subclass(Spoon_utensil, Utensil).

ClsColor(Table, Brown_color)
ClsDepth(Table, 74)
ClsIsOnPhysical(Table, Floor)
ClsMaterial(Table, Wood_material)
ClsShape(Table, Box_shape)
```

```
ClsDiameter(Tea_cup, 5)
ClsLength(Tea_cup, 7)
Subclass(Tea_cup, Cup).

ClsDepth(Teabox, 4)
ClsIsInside(Teabox, Closet)
ClsLength(Teabox, 8)
ClsMaterial(Teabox, Paper_material)
ClsShape(Teabox, Box_shape)
ClsWidth(Teabox, 5)
Subclass(Teabox, Closable_container).

ClsColor(Teapot, Gray_color)
ClsDiameter(Teapot, 15)
ClsIsNear(Teapot, Tea_cup)
ClsIsOnPhysical(Teapot, Table)
ClsLength(Teapot, 20)
ClsMaterial(Teapot, Metal_material)
ClsShape(Teapot, Cylinder_shape)
Subclass(Teapot, Closable_container).

ClsColor(Tetrapak, White_color)
ClsIsInside(Tetrapak, Refridgerator)
ClsIsNear(Tetrapak, DrinkingGlass)
ClsIsOnPhysical(Tetrapak, Table)
ClsMaterial(Tetrapak, Paper_material)
ClsShape(Tetrapak, Box_shape)
Subclass(Tetrapak, Closable_container).

ClsColor(Utensil, Gray_color)
ClsDepth(Utensil, 1)
ClsIsInside(Utensil, Closet)
ClsIsOnPhysical(Utensil, Table)
ClsIsOnPhysical(Utensil, CounterTop)
ClsLength(Utensil, 14)
ClsMaterial(Utensil, Plastic_material)
ClsShape(Utensil, Complex_shape)

ClsColor(Vase, White_color)
ClsColor(Vase, Gray_color)
ClsDiameter(Vase, 10)
ClsIsOnPhysical(Vase, Table)
ClsLength(Vase, 15)
ClsMaterial(Vase, Earthenware_material)
ClsShape(Vase, Cylinder_shape)
Subclass(Vase, Open_container).

ClsColor(WineGlass, Gray_color)
ClsLength(WineGlass, 25)
Subclass(WineGlass, DrinkingGlass).

ClsColor(Wine_bottle, Green_color)
ClsDiameter(Wine_bottle, 9)
ClsLength(Wine_bottle, 26)
ClsShape(Wine_bottle, Cylinder_shape)
Subclass(Wine_bottle, Bottle).
```

Appendix C

Plots and Figures

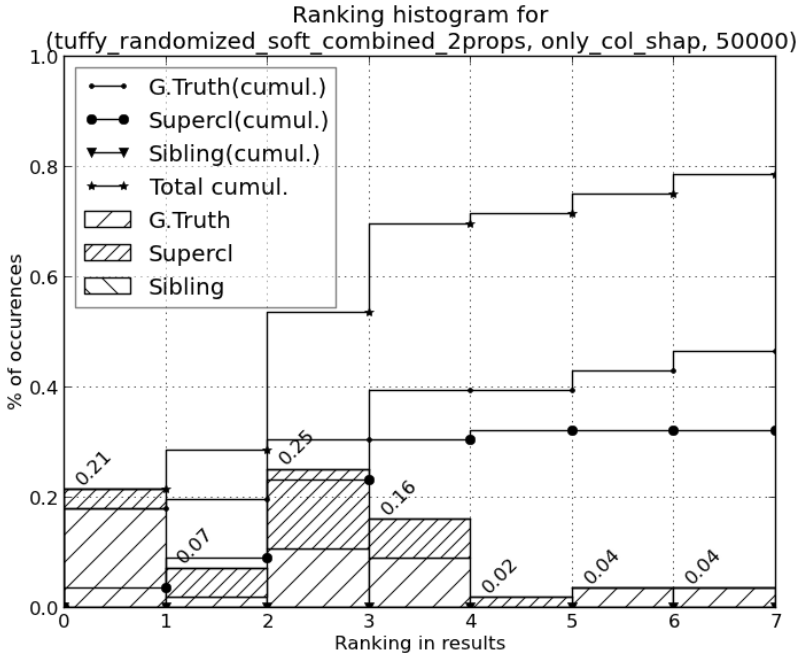

Figure C.1: Histogram for classification with only color and shape

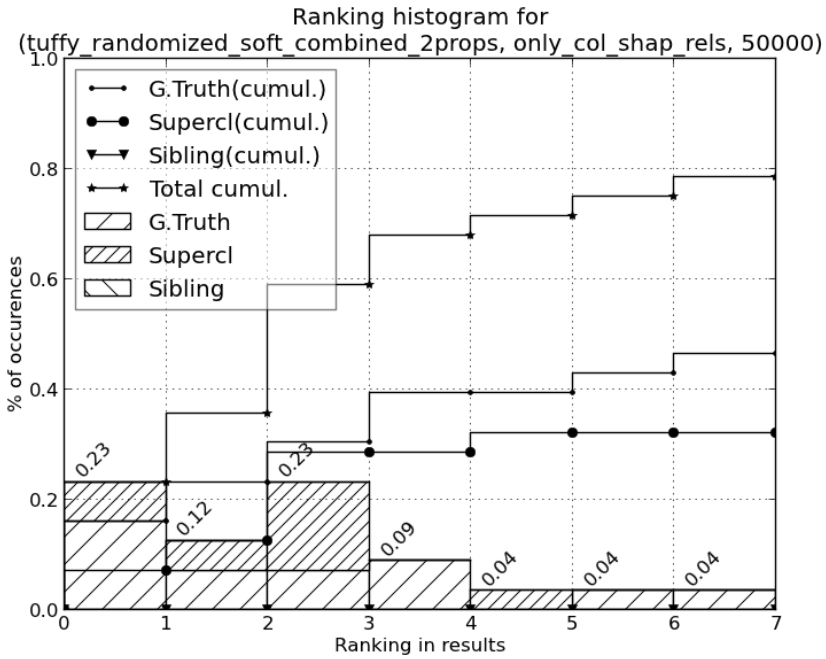

Figure C.2: Histogram for classification with only color, shape and spatial relations

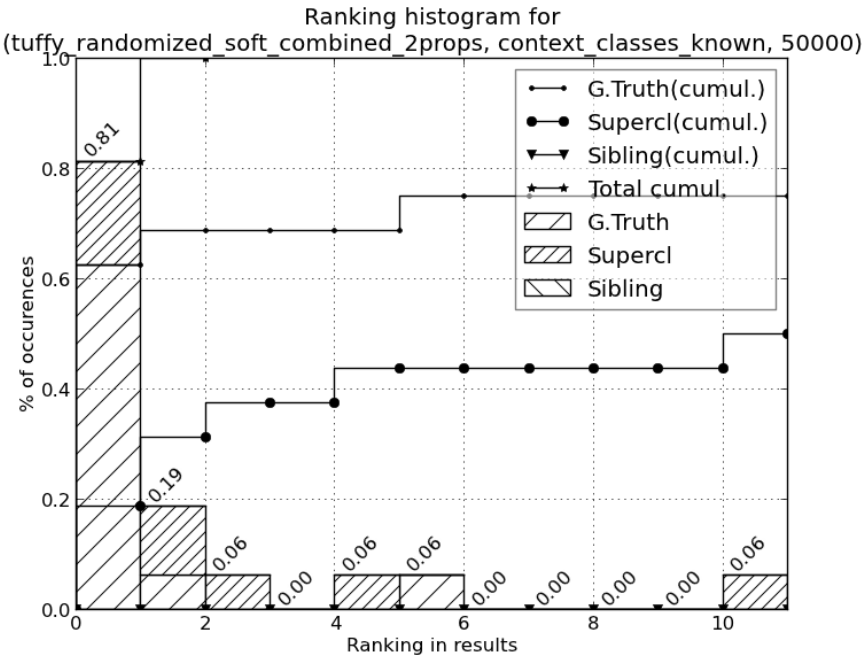

Figure C.3: Histogram for classification with the context classes known