UNIVERSITY OF GOTHENBURG

DEPARTMENT OF PHILOSOPHY, LINGUISTICS AND THEORY OF SCIENCE

KILLE

Learning Objects and Spatial Relations with Kinect

Erik Wouter de Graaf

Master's Thesis:	30 credits
Programme:	Master's Programme in Language Technology
Level:	Advanced level
Semester and year:	Spring, 2016
Supervisor	Simon Dobnik
Examiner	Richard Johansson
Keywords	grounding, spatial relations, object learning

Abstract

In order for humans to have meaningful interactions with a robotic system, this system should be capable of grounding semantic representations to their real-world representations, learn spatial relationships and communicate using spoken human language. End users need to be able to query the system what objects it already has knowledge of, for more efficient learning. Such systems exist, but require large sample sizes, thus not allowing end users to teach the system more objects when needed.

To overcome this problem, we developed a non-mobile system dubbed Kille, that uses a 3D camera, SIFT features and machine learning to allow a tutor to teach the system objects and spatial relations. The system is built upon the ROS (Robot Operating System) framework and uses Opendial software as a dialogue system, for which a ROS support was written as part of this project. We describe the hardware of the system, the software used and developed, and we evaluate its performance.

Our results show that Kille performs well on small learning sets, considering the low sample size it uses to learn. In contrast to other approaches, we focus on learning by a tutor presenting objects and not by providing a dataset. Recognition of spatial relations works well, however no definitive conclusions can be drawn. This is largely due to the small number of participants and the subjective nature of spatial relations.

Acknowledgements

This thesis has been written as the final part of my studies in the MLT programme at the University of Gothenburg in Sweden. The whole thesis and accompanying software have been written at home in the Netherlands. Simon Dobnik has done a terrific job at being my supervisor over video chat and e-mail, and has done a very good job at showing his enthusiasm, which has kept me excited to keep working hard on the thesis.

My interest for the combination of language and vision started with a course project that I worked on together with Mehdi Ghanimifard. I am thankful for his enthusiasm and hard work during this project.

I owe thanks to Pierre Lison, the author of Opendial. Firstly, because he has created great software, and secondly, because he has been very supportive and has been very quick when I was in need of help or a bug fix.

I would like to thank my girlfriend Louise Martens, who has listened to my reflection on troubles during the programming process and offered me her thoughts on the matters at hand. She has also been really helpful reading through my thesis and pointing out where things could be improved.

I also want to thank the other people that proof-read my thesis and provided valuable feedback.

In addition to reading my thesis, I also invite you to try out the system for yourself, or to send me any questions about the system or my experiences building it. Installation instructions are available in the appendices.

Contents

1 Introduction

My bachelor's education has been in Artificial Intelligence and my bachelor's thesis was in the field of language technology, about the detection of multi-word expressions. It is therefore only fitting that my master's thesis is in the same area, where language technology and artificial intelligence meet. This project is about recognising objects, learning names for those objects, and the spatial relations between them. In contrast to machines, humans do not generally learn about new objects or spatial relations from a list of examples, but instead learn incrementally by seeing examples over time. Learning happens interactively through dialogue. This thesis describes work on learning objects and spatial relations with a computer in a similar manner.

A very important feature of robots that are to interact with humans is that the language that is used is situated. The language relates to the real world and the objects and entities in it. This means that these robots have to know about the real world, to be able to have a useful conversation or to follow instructions from a human, whether it is a human tutor or a human wanting the robot to accomplish a certain task. Children make the connection between words and what they perceive at a very early age, but making this connection is not an easy task for automated systems. Allowing these systems to link the real world they perceive with their cameras to their semantic representation is very valuable, and is currently a hot topic in both academia and the commercial world. It is a step that needs to be taken towards the goal of meaningful interaction, as without this so called *grounding* a system would not be capable of grasping the true meaning of a word. Being able to refer to objects by using words and the relationship between these words and the real world form the foundations of linguistic communication (Roy, 2005). Grounding is defined as connecting symbolic descriptions of the real world to non-symbolic properties (Harnad, 1990). Symbolic descriptions are concepts, like names of objects or other entities. Non-symbolic properties are pixel values, but also the higher level things that can be combined from these pixel values. The SIFT (Scale-Invariant Feature Transform) (Lowe, 1999) features used for this work are a good example of can be abstracted from combining pixel values. SIFT features are descriptions of interesting areas or points in a picture. The features obtained by combining pixel values can be processed even further. The features are compiled into models. Non-symbolic properties can become symbolic when a name is attached to them. This happens when a name is attached to something in the real world (tangible objects), but also when a name is attached to something that is not tangible, e.g. spatial relations and other things that have a linguistic representation. By grounding, sensory perception is linked to the real world. The necessity of grounding becomes more evident when we think of an utterance like *put the glass of water next to the jar*. The person or robot spoken to will need to grasp the meaning of *glass*, of *water*, and of *jar*. In addition, it needs to understand which glass of water and which jar are referred to (Kennington et al., 2015).

Presently, building of models of objects is heavily researched. The vast majority of this research uses large databases of pictures to apply deep-learning algorithms to, in order to train neural networks that are akin to a biological brain. This approach is highly successful, but only works when a large database of images is available. Not only well-known objects for which lots of images are available need to be grounded. The world contain lots of items that are unique, or otherwise not found in many other places. The world is constantly changing and systems that operate within the real world will often encounter novel situations that they have not been programmed for and will need to learn new things or concepts and work with those. This means end users will have to be able to teach the robots new objects. As the vast majority of users are computer language-naïve, end users will only be able to do this when the robot speaks a human language (Lauria et al., 2002). Interactive learning with human language is an active research field (Cooper & Larsson, 2009; Larsson, 2013; Dobnik, 2009). There is also a lot of interest in communicating with computers through natural language in its spoken form. This is demonstrated by commercial successes such as the Google Now and Apple Siri software and the Microsoft Kinect hardware. Spoken language-steered applications, as well as applications that can perceive the real world through vision or touch are a step towards incorporating machines into our society even further.

System	Hello, I want to learn some objects!
Presents	a book.
User	Do you know what this is?
S	I do not know what this is.
U	This is a book.
S	Okay, this is a book. Could you show me more objects?
P	a bottle.
U	This is a bottle.
S	Okay, I learned a 'bottle'.
P	the bottle next to the book.
S	I see a bottle and a book. Could you tell me where these things are?
U	The bottle is to the left of the book.
S	Okay.

Table 1: Example conversation of the ideal system.

Robotic systems will not only have to be able to handle known and novel objects, they will also need to be able to work with spatial relations. Spatial relations are not as straightforward as many that are not familiar with the field may think (Dobnik, 2009). A phrase like *to the left of* does not simply mean the object would have a lower x coordinate if it was drawn in a graph, but is much more nuanced and complicated in human language and depends on many factors such as the direction an object is facing. There is extensive research done on this topic (Coventry et al., 2001; Regier & Carlson, 2001), but most of this research does not attempt to combine geometric representations with other sources. Camera imagery introduces the issue of inaccuracy, as especially consumer grade cameras show a lot of noise. This has been improved over the last couple of decades, but inaccuracy related issues still exist.

We want a robotic system that can perceive the world and learn objects and spatial relations. We want a system that can be asked what it already knows, but that can also take the lead and ask for more samples when it does not have sufficient knowledge of an object in its database. In addition, it would need to be able to learn a large number of objects, with only having to see each object a few times. The system would have to interact with humans, so it has to be using human languages in spoken form. An example of a conversation with such a system can be seen in Table 1.

During the first semester of the second year of the MLT programme a system was built that can perform some of these tasks, for the Embodied and Situated Language Processing course (Ghanimifard & de Graaf, 2015)[1]. During this project we worked on software that allows a computer to learn new objects that are situated in the real world, using a RGB-d (Red, Green, Blue - depth) camera, the Microsoft Kinect. This system is able to learn new objects and recall those. This master's thesis addresses some of the issues and shortcomings of that course project.

In this work we describe the workings of the Kinect Is Learning LanguagE framework, abbreviated as the Kille framework. This framework represents a learning robot that is using a passive supervised learning setting for the largest part, which means that the tutor will pick what to teach and what labels to attach to what it teaches. The framework also incorporates some active learning; it can ask for more samples of objects it has already learned when it is unsure about them. It is using active learning as a form of transparency about its knowledge. This form of active learning allows robots to achieve more accurate and faster learning, but it makes it harder to balance dialogue control, i.e. who gets to take a turn in a conversation. Interactive learning offers great perspectives on learning improvements. The system does not exclusively use active learning, as fully active learning has the downside of tutors getting less excited and engaged because they

[1]Wall-e: https://github.com/mmehdig/wall-e

are bombarded with questions (Cakmak et al., 2010). Software contributions are the Python application Kille and the Java application/plugin ROSDial.

We proceed as follows. In Section 2 we look at the hardware involved, in Section 3 we examine the Kille framework, in Section 4 interactive learning is elaborated and in Section 5 we evaluate the recognition of a small number of objects, as well as spatial relation recognition. The thesis concludes with a discussion of current work and suggestions for future work in Section 6.

This thesis and accompanying work is based on the situated learning agents proposal by Simon Dobnik from the Dialogue Technology Lab at the University of Gothenburg. Preliminary work for this thesis has been presented during the SemDial 2015 conference hosted in Göteborg (De Graaf & Dobnik, 2015). All software is available under the BSD license on their respective websites[2][3][4][5].

[2]Opendial: https://github.com/plison/opendial
[3]ROS: http://www.ros.org/
[4]ROSDial: https://github.com/masx/ROSDial
[5]Kille: https://github.com/masx/Kille

2 Hardware

The primary goal of the setup used in the experiments is to have a robust, repeatable way of setting up the system. The secondary goal is to simulate an environment that would be used when a user teaches someone or something the name of an object or a spatial relation. The setup consists of a computer, a Kinect camera, two glass pedestals, a table and various objects to test on, that range in size from apple-sized to shoebox-sized. The camera is mounted stationary to a table. In the experiments, objects are placed on glass pedestals that are not picked up by the depth sensor as the glass does not reflect the infrared pattern emitted by the Kinect camera. This makes it so that, for Kinect, after applying the depth-based background removal the objects appear to be floating in the air. The same situation could be achieved by a person holding the objects, if the person is just out of Kinect's view - further than a meter away from the camera. This thus simulates a normal learning environment, where a tutor actively presents an object by holding it forward. A person holding the objects in front of them is the ideal situation for learning objects in most cases, but in more complicated cases other methods have to be used to determine where and what the object of focus is, such as movement of the object, eye gaze or pointing gestures (Kennington et al., 2015).

The robot of the system is represented by Kinect, an RGB-d (red, green, blue, depth) camera module, connected to a computer running Ubuntu 12.04. As the camera is mounted to the table, it always perceives the same region, thus objects have to be actively presented to the device. It is however possible to pair the camera with a moving robot. The camera module used is Microsoft's Kinect, a camera developed for the game computer Xbox 360. For this research a Kinect with version number 1414 has been used, because this version is best supported by ROS. The depth camera of this version can perceive objects that are between 70 and 300 centimeters removed from it, though a wall limits it from seeing further than 150 centimeters in this situation. Newer versions of Kinect do have a better range [6] , but are not as well supported outside of the official Microsoft SDK. Both the RGB and depth sensors have a resolution of 640x480. The Kinect images are later processed to exclude the background, which is defined as anything more than a meter away from the camera. This is further described in Section 4.

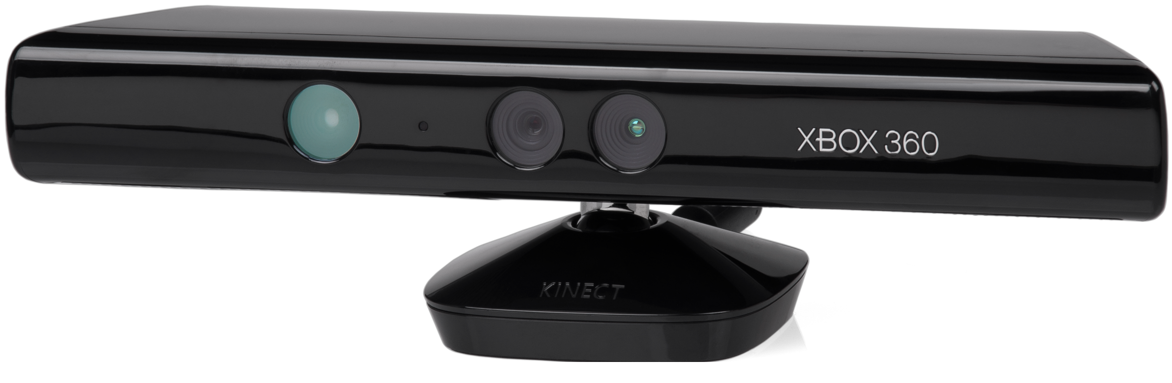

Figure 1: A Kinect camera. From left to right: infrared projector, RGB sensor, infrared sensor.

Driver support for non-Windows systems is provided by freenect, developed by the OpenKinect group [7]. Kinect combines a speckle infrared pattern projector and an infrared camera to sense depth. As it has only one 'eye' it does not perceive the world in stereo, in contrast to humans. Depth information is directly available and does not have to be abstracted from two RGB frames.

[6]https://msdn.microsoft.com/en-us/library/hh438998.aspx

[7]http://openkinect.org/

Figure 2: A frame from the depth camera of Kinect.

3 The Kille framework

The Kille framework consists of several modules that communicate with each other using the ROS operating system. The system is built modularily, meaning the modules in the system can be replaced. The dialogue manager, Opendial, can be replaced by a different dialogue manager and the Kinect system can be switched with a different camera or robot, should the need arise. Each of the modules in the Kille framework are discussed in this section, starting with the communication network, ROS. ROSDial was written in Java to interface with Opendial, while Kille was written in Python, to make use of the OpenCV libraries which are used for visual processing.

Figure 3 gives an overview of how the modules in the Kille framework work together. The arrows indicate data flow. Input from the outside world comes from both Freenect and from user utterances or input to Opendial. The nodes are Roscore nodes, through which communication takes place. The *system output* denotes human text input into the Opendial system as a system utterance.

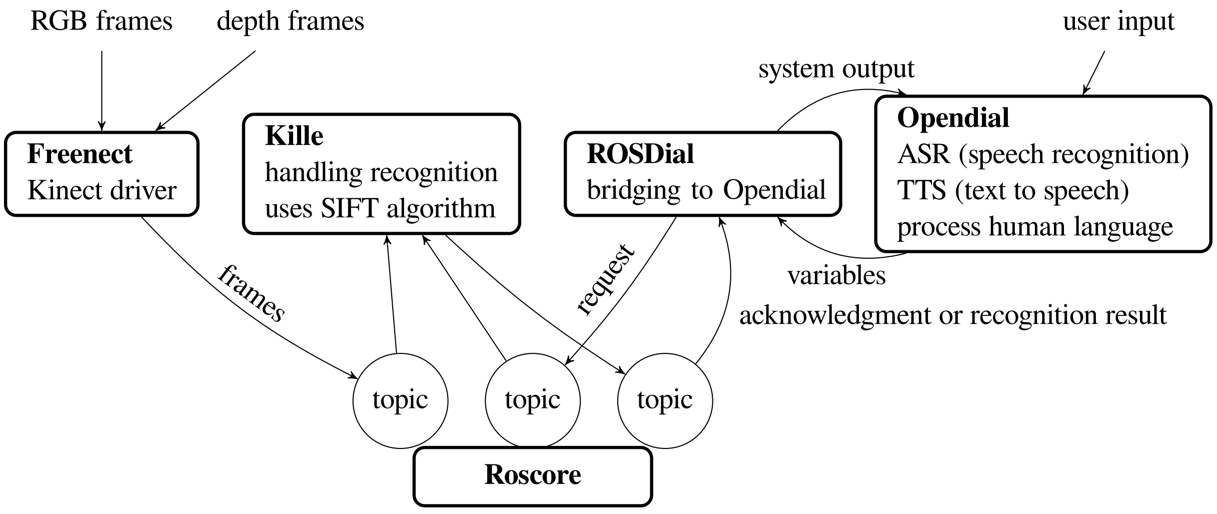

Figure 3: Graphical representation of the modules.

3.1 Robot Operating System

Robot Operating System (Quigley et al., 2009) (ROS), is an open source operating system for robots that is commonly used in the robotics community. ROS is natively written in C and C++, but also provides an interface for various other programming and scripting languages, including Python and Java. Many robots are supported by the various modules that exist for this communication framework. These modules are mostly built in a modular way themselves, allowing users to use only certain parts of a module, for example only the camera of a certain type of robot. In this way, the ROSDial module written for this project can be used in different robots. The used freenect_stack that drives the Kinect camera is another example of modularity, as the module was written to be used for the Kinect used in the well-known Turtlebot. ROS was chosen over other frameworks for several reasons. Firstly, it is free and open (Quigley et al., 2009). Secondly, it has a very elaborate documentation wiki. And thirdly, it has support for Kinect. This Kinect support is provided by the freenect_stack, which in turn makes use of freenect, created by the OpenKinect community.

The modules communicate with each other in a simple way. Each module of the system sets up a node, on which multiple topics can be started. In addition to setting up topics, nodes can also subscribe to the topics of other nodes and receive or publish data over them. This project contains nodes for Kille, ROSDial and the freenect_stack. This all works on top of roscore, the part of ROS that contains the core functionalities of ROS, most notably communication. Kille subscribes to multiple topics created by freenect_stack, in order to receive the depth frame and the RGB frame from the Kinect camera. Kille also subscribes to ROSDial, to listen to the questions asked through Opendial. ROSDial, in turn, subscribes to Kille to receive Kille's answers to these questions. Everything sent through the topics is expected to be answered with at least a special message that denotes acknowledgement. The details of this are in the next section.

3.2 Opendial & ROSDial

The dialogue manager used is Opendial (Lison, 2014), a free, open, and domain independent dialogue system. Opendial was chosen because it is a freely available dialogue system and has support for external modules. These external modules were a necessity, as Opendial has to connect to ROS. In addition, Opendial comes pre-packaged with several other NLP tools, like the MaltParser, speech recognition and plugins for robots. It is written in Java, and therefore an appropriate Java plugin to connect to ROS had to be designed.

Opendial handles all human language input and output, and keeps track of what internal variables to set depending on what the user uttered, and what has happened with the system before that. These variables correspond to the information that are extracted from human utterances and are used in further processing. At this moment the dialogue system supports three different text to speech (TTS) engines, two of which are included with this project, namely Nuance and MaryTTS. Nuance is capable of speech recognition, but when MaryTTS is used it has to be coupled with Sphinx ASR to be able to use speech recognition. Every user utterance received by Opendial is parsed with MaltParser to get a dependency parse, and then run through an XML file that will set the appropriate variables that are read by ROSDial and Opendial itself. ROSDial is a module in Java that was written for this project to allow Kille and Opendial to communicate using the Robot Operating System (ROS) (Quigley et al., 2009) framework. ROS is a large modular framework that can be used for communication between the various devices and modules a robotic system has.

Three plugins are used at any time. MaryTTS is used for speech synthesis, Sphinx ASR for (local) speech recognition and MaltParser for part of speech tagging. Sphinx ASR works with a local JSGF grammar file, which has been written based on the expected sentences. This does however limit the user to object names and spatial relations that are in the grammar, which means a developer needs to provide the names of objects and relations that can be learned ahead of running. MaryTTS and Sphinx ASR can be switched with Nuance, which provides better recognition and a slightly more realistic pronunciation, but Nuance does not work locally and is not open sourced. MaltParser, based upon the well-known Stanford PoS Tagger, is used to parse the incoming sentences.

Opendial interprets domain XML files for its handling of human language, and for setting internal variables. A variable *to_do* is set after every user utterance understood by the domain. A user-input model is triggered every time a part-of-speech parse can be made from the input received from the chosen ASR (Automatic Speech Recognition) module. This model then runs through a series of conditions, which check either just the user input, or the parse of the user input followed by the user input. The parse is checked in some cases to extract the name of the object. The noun phrase is chosen as object name, and any adjectives and subordinate clauses are discarded. In some cases a variable *last_step* is checked in addition to parse and user utterance, to set the correct new *to_do*. If the user utterance was *That is correct*, the new value for the *to_do* variable can become either *reinforce_object*, *reinforce_relation* or both, depending on what the previous step was. Valuable information, such as the name of the object shown is parsed and passed through to Kille by setting a variable. In addition to a user input model, there is a machine input model. This model is much simpler, taking care of (re-)setting variables when the machine utterance is received from Kille via ROSDial.

ROSDial is a Java executable written for this thesis project that starts up a ROS node and an instance of Opendial. As mentioned in the previous section, ROSDial serves as a bridge between Kille and Opendial, translating everything to a format that is readable for the recipient. In the direction of Opendial to Kille, this means grabbing data from different fields parsed by Opendial and putting it into an easily parseable format for Kille before sending it over. ROSDial periodically polls the *to_do* variable of the Opendial system, to take action if it has been updated. Because ROSDial starts a new instance of Opendial, and is not a plugin, triggering ROSDial through Opendial without periodic polling on the ROS side is not possible at this point. In the direction from Kille to Opendial this means translating a parseable format to human-readable text that is set as machine output in Opendial. The parseable format is of the form

```
command:argument-1:argument-2:argument-3:argument-4~object/relation:argument
```

where everything but *command* is optional. An example is

```
detected:2:unknown:gnome,0.402:book,0.1875:~object:box
```

This lets ROSDial know that Kille detected two objects, their spatial relations are unknown, one of the objects is a gnome, and the other is a book. Those have been recognised with reasonable certainty, namely with 0.402% and 0.1875% of the matches with known objects respectively. Kille would like to learn more examples of the object 'box'. ROSDial is in charge of deciding whether to ask the user for more objects or spatial relations or not. It sometimes may decide not to, as it may degrade the user experience when the system takes the lead too often.

An example of a dialogue can be found in the appendices.

3.3 Kille core

Kille is the main module, written in Python. It relies heavily on the OpenCV module, and on the ROS modules. Kille keeps track of objects and spatial relations it can recognise and also handles all the recognising. ROS cannot communicate directly between the image callback - the module that is called when a new frame from the camera is received - and the module that receives messages from ROSDial. This is why Kille works by setting to-go-to stages when receiving input instead of directly moving to these stages. When Kille is started, it will first check whether it receives frames from the freenect node. These frames are processed with help of the OpenCV library.

Users of the software can either teach the machine the objects themselves, or they can use an existing model that they have built previously or that another user has built. However, as spatial relations are partly subjective, one would need to fit them to their own view by instructing the system to forget about learned relations and subsequently re-teaching them. Alternatively, more examples could be added to outnumber existing examples. Models could be saved in the cloud, where anyone can contribute.

When a new learning or recognition session is started, Kille will try to recover objects and relations from previous sessions, if those exist. These objects are recovered from files that were saved (pickled) by Kille. Because of technical limitations[8] the SIFT keypoints and frames have to be taken apart to simpler data structures such as (lists of) lists before saving, and reassembled when loading, but this does not affect performance.

Overall, Kille can handle the following requests:

- Learn an object. Kille will take the currently seen frame and store the SIFT features.

S	Hello, I want to learn some objects!
Presents	a book.
U	This is a book.
S	Ok, I learned book.

- Learn a relation.

Presents	A book next to a box.
S	One of the objects is book, the other is box. Please tell me where things are.
U	The book is to the left of the box.
S	Okay, this relation is called 'to the left of'.

- Recognize an object.

S	Hello, I want to learn some objects!
Presents	a book.
U	What is this?
S	The object is thought to be book.

- Reinforce the last recognised object. Kille will receive this command when the user confirms that recognising an object was successful. It will then store keypoints and features it is seeing with the name of the object it had recognised, just like it would normally learn an object.

S	Hello, I want to learn some objects!
Presents	a book.
U	What is this?
S	The object is thought to be book.
U	Correct.
S	Ok, I learned book.

- Unlearn the last learned object. Unfortunately, ASR (Automatic Speech Recognition) is not perfect. Sometimes this leads to Kille learning names for objects that are incorrect, such as learning *boss* when it is supposed to learn *box*. This can also be used when the item was not presented yet when Kille was instructed to learn the object.

Presents	a book.
U	This is a gnome.
S	Ok, I learned gnome.
U	That is not what I said.
S	Ok, sorry. I forgot about it.

- Unlearning a specific object. This can be used when the user wants to teach Kinect a more specific name for different objects of the same class, such as the names of different books, instead of calling

[8]The pickle library, as well as libraries like it, cannot save keypoints.

them all book, or to forget about specific objects that were loaded from a file at the startup of Kille. When this request is received everything Kille knew about the object will be lost.

Presents	a book.
U	This is a gnome.
S	Ok, I learned gnome.
U	Forget gnome.
S	Ok, I forgot all gnomes.

- Re-learn an object. This can be used instead of unlearning the last object. It is used to attach a new name to a previously learned object.

Presents	a book.
U	This is a gnome.
S	Ok, I learned gnome.
U	I said a book.
S	Ok, I learned book.

4 Interactive learning

OpenCV (Open Source Computer Vision) is a library that deals with computer vision and machine learning of vision objects. It is natively written in C and C++, but has interfaces for various other languages including Python (Bradski & Kaehler, 2008). It is optimized for real-time applications. Upon receiving a new frame from the video feed from the Kinect camera in ROS' format, the frame is converted to the OpenCV format, which is compatible with Numpy arrays. This is done for the frames coming from the depth sensor of the camera as well as the frames coming from the RGB sensor. This allows for very fast array manipulation in the background removal step. The RGB picture is transformed to exclude the background and foreground so the recognition algorithms can focus only on the object that is in focus, i.e. presented in the middle between these spaces. Foreground is defined as anything that is closer than 70 centimeters to the camera. This limit is set because it is a hardware specific limitation of the Kinect. The background is defined as anything further away than 100 centimeters. This leaves 30 centimeters of space to present objects in, which turns out to be plenty for the current experiments. In further research this limit is likely not workable, and a moveable robot has to be used to bring the object within this range. Alternatively a different method of background extraction can be used, but these have disadvantages as well. An alternative that only works with stationary robots is frame differencing, comparing frames to find out what the background is (Patel & Patel, 2012). Figure 2 shows a plush gnome standing on top of a box, with the background removed on the depth feed, creating a silhouette. In the far left of the image a part of a radiator can be seen in black. Kinect can not properly detect depth when things are not reflective enough or over-reflective, as this interferes with the infrared projected pattern. The radiator is reflecting too much of the infrared light for an accurate depth estimation. The black border around the gnome is an infrared shadow caused by the design of the Kinect as well as a small miscalibration between the cameras. The Kinect has the infrared projector apart from the infrared sensor. This casts an infrared shadow in the viewpoint of the infrared sensor. Figure 5 shows a colour frame from the RGB camera after laying it over the shadow of the depth frame. Recognition is started on this frame, by finding recognisable points: SIFT features.

4.1 Learning of objects

Object recognition in both computers and humans is achieved by combining the perceived points resulting from impact of light on the a sensor into higher order features. These features are subsequently stored in a database, to later see which stored features match the perceived object best. One of the very first steps is line and corner detection, which is then used to compute more complicated features and shapes (Serre et al., 2005; Lowe, 1999). The algorithm used in this work is the well-known SIFT (Scale-Invariant Feature Transform) (Lowe, 1999) algorithm. SIFT was chosen because recognition performance (with few samples) was favoured over computational speed. SIFT is similar to some of its best-known competitors,

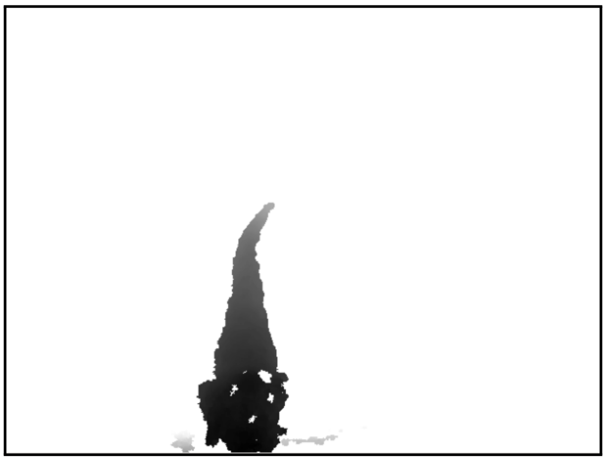
Figure 4: depth frame with background removed.

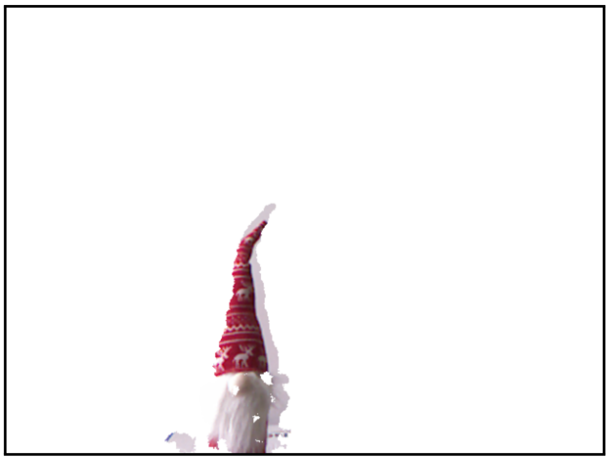
Figure 5: RGB frame with background removed.

such as SURF (Bay et al., 2006) and ORB (Rublee et al., 2011), but provides better recognition with few samples (Agrawal et al., 2008). The downside of SIFT compared to these other algorithms is that it is relatively computationally costly (Agrawal et al., 2008). For this project background removal was applied before computing features. Everything that is left after background removal is considered the object to be learned or recognised.

The SIFT algorithm works by smoothing an image multiple times, subtracting these smoothed images and picking the stand-out points from there. These resulting stand-out points are groups of pixels where sudden changes in colour happened. Elimination of weaker stand-out points is applied, which results in so-called keypoints. An orientation is assigned to each keypoint to make the algorithm robust against rotation. This is done by taking the orientation of sub-areas around the keypoint and combining them into a single vector. These keypoints are invariant to location, scale and rotation. Histograms of the rotations from the previous step of the areas neighbouring the keypoint are combined into a high dimensional vector called the descriptor. The descriptor of the keypoint holds information of the image at the keypoint and is used to detect similar descriptors and thereby keypoints. The keypoints of SIFT are scale-invariant, and thus robust against changes that are introduced by a change of distance from the camera and scaling. In addition, with descriptors, the algorithm can handle rotation and small changes in illumination (Lowe, 1999). The algorithm is similar to how primates process visual information (Serre et al., 2005). SIFT makes descriptors, which algorithms like HARRIS do not (Harris & Stephens, 1988). The SIFT keypoints and their descriptors are stored in an array paired with their given names.

Figure 6 shows SIFT features on an RGB frame before a background is removed. Every circle indicates a keypoint. The line(s) in the keypoint show(s) its orientation. The size of the circle indicates the size of the keypoint. Different sized keypoints result from the different smoothing levels. Very thorough smoothing results in larger keypoints, as only the very large differences that stretch over a larger area remain. The colours of the drawn sift features do not indicate anything, and are purely to allow one to distinguish between keypoints. The background removal is not very precise, but leaves the contour of the object. This contour is roughly detected by the SIFT algorithm.

No model is created for the full set of objects, but each object is learned separately and added to a database. For each object added to the database, it is calculated how well it matches against the perceived, background-removed, frame. No model of all objects together is created, until the comparing at the time of classifying. This delaying of inter-comparing of samples in a model is called lazy learning (Atkeson et al., 1997). The comparing is done in sequence, so the different object models do not interfere with each other. The SIFT descriptors of the perceived frame are calculated using the before mentioned SIFT library and the SIFT descriptors of the model are obtained from the database. Currently observed descriptions may not faithfully represent the reality, because of camera noise and environmental factors such as changes in lighting, which

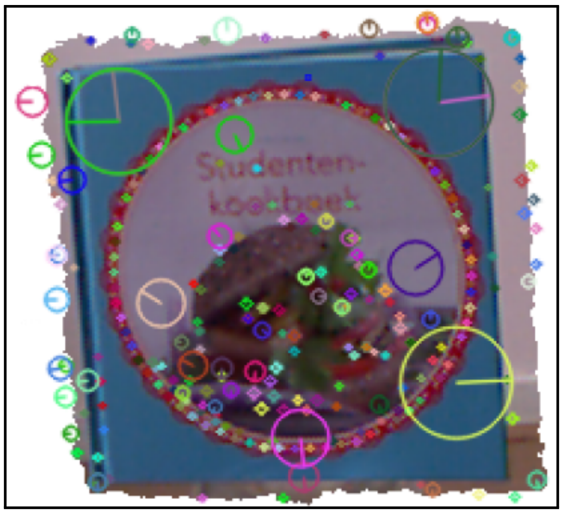

Figure 6: View of the book, with SIFT features shown.

makes the algorithm choose different descriptors of the object. The longest descriptor list is taken (whether this is from the perceived frame or the model), and for each of the descriptors in this list the two closest matches in the shorter list are found using a nearest-neighbour search algorithm provided by the FLANN library (Muja & Lowe, 2009). This is further explained later, in Section 5.1.1, on evaluating the system. The result is a list of tuples of matches. As most of these matches will be incorrect matches, matches where the descriptors are not much alike, a filter has to be applied. This filtering is done using a ratio test (Lowe, 2004). The Euclidian distance between the two matches is calculated. If the distance between the two matches is small, it is likely that both of the matches are not correct matches, but instead are noise. If the first match is sufficiently large, the match is likely a correct one and is counted. This ratio test requires a threshold, for which 0.75 has been chosen. As shown by Lowe (2004), a ratio of 0.75 offers a good balance between keeping correct matches and incorrectly including incorrect ones.

The matches that pass the ratio test are counted, and used for two measures. The first measure estimates how well the perceived frame matches against the model, by dividing the number of matches by the number of descriptors in the model. The second measure estimates how well the model fits in the perceived frame, by dividing the number of matches by the number of descriptors in the perceived frame. The first measure shows a strong bias towards objects that have relatively few features in the model, as it is easier to find a match for a larger percentage of those. The second measure has a bias towards frames with relatively many features, as it is hard for a model with few descriptors to do well against a frame with many of them. To avoid these biases, the harmonic mean of these two measures is used as the final score for the model. Algorithm 1 demonstrates pseudocode for this algorithm.

During normal runs, only a single camera frame is processed. In evaluation mode the number of inspected frames is increased to 10, improving recognition performance slightly, but reducing recognition speed.

If multiple objects with the same name are considered as a match, their results are averaged. The location of the recognised object is estimated by taking the twenty matches of descriptors with the lowest distance. Occasionally points will not fall between the bounds of 70 to 100 centimeters away from the camera, because of shadow introduced by the combination of the infrared projector and the infrared camera. These points are then not considered as they are not part of the object itself.

Algorithm 1 Scoring algorithm.

```
$SIFTperceived \leftarrow$ calc-descriptors$(perceivedframe)$
**for all** occurence $in$ database **do**
    $SIFTmodel \leftarrow$ get-descriptors$(occurence)$
    $matches \leftarrow$ FLANN.match$(SIFTmodel, SIFTperceived)$
    **for all** tuple $in$ matches **do**
        **if** tuple[0].distance < 0.75 * tuple[1].distance **then**
            $good \leftarrow good + 1$
        **end if**
    **end for**
    $measure1 \leftarrow \frac{good}{\text{length}(SIFTmodel)}$
    $measure2 \leftarrow \frac{good}{\text{length}(SIFTperceived)}$
    $score \leftarrow \frac{2 \cdot measure1 \cdot measure2}{measure1 + measure2}$
**end for**
**return** object with the highest score
```

4.2 Learning of spatial relations

Spatial relations are expressions that describe the spatial location of objects. Basic spatial expressions consider only a single object, while non-basic expressions (relations) consider an object relative to another. An example of sentence with a non-basic spatial expression is "The chair is to the left of the couch.". In this example the chair is positioned relative to the couch, of which the position is assumed to be known. The chair is called the figure, trajectory or target object, the couch is called the ground or landmark (Talmy, 1975). In this work, only non-basic spatial expressions, called spatial relations, are discussed.

Spatial relations are built up with a few concepts. The clearest is geometry; angles and distances between objects. Another concept is the frame of reference. Spatial relations with reference frame are called intrinsic spatial relations, and spatial relations without one are called deictic spatial relations (Logan & Sadler, 1996). A reference frame is a mapping between concepts (i.e. *to the left of*) and perception. Typically, reference frames are either viewer-based or object-based. In an object-based reference frame it matters which way the object is faced. Anything in the direction of the backrest of a chair will be considered *behind*. Object size can also play a role, for example when considering if something is *near*. It is very difficult to construct rules based on this (Maillat, 2012), and constructing these rules is beyond the scope of this work.

The third concept is object function (Coventry & Garrod, 2004). For example, the *back* of a chair may either be the backrest of the chair in an object-based reference frame, but it might also mean other sides, depending on where the observer is located (Dobnik et al., 2014), in a viewer-based reference frame. In addition to the difference between basic, deictic and intrinsic relations, spatial relations can be directional, e.g. *to the left of*, or topological, e.g. *near* (Dobnik, 2009).

Spatial relations are vague. This means that their reference, the location in the real world, is not precise. Spatial relations exist as regions, where different places within the region have different degrees of applicability to the spatial relation. This is represented in the form of a spatial template (Logan & Sadler, 1996). An object that is 30 cm to the left but also 10 cm up might still be considered *to the left*, but is not as typical *to the left* as it would be if the object was 2 cm up. How many cm is allowed for this spatial relation to still be considered *to the left* depends on many factors, including the size of the landmark and target object, and the person judging the situation. Each spatial relation has its own spatial template, but similar relations have similar templates. There are two basic components of space that appear to be important for spatial relations, namely proximity and direction (Regier & Carlson, 2001).

Kille's job is to learn and recognise spatial relations without being explicitly instructed about spatial templates and reference frames. Kille's first step to accomplish this is to recognise the objects that are presented. Then, if both of the objects are reliably recognised, Kille takes the coordinates of the top twenty best recognised features of both objects. The ranking of features is based on the Euclidian distance between the SIFT vector of the feature in the model and the SIFT vector of the feature in the perceived object. It then averages those to obtain a center point for the object. This center point of the landmark is subtracted from the center point of the target object to obtain relative coordinates. The averaging of the twenty keypoints gives a good estimation of the center of the object, and is a simpler alternative to the Attentional Vector-Sum (AVS) model by Regier & Carlson (2001).

By only training on the centroid of the two objects and not their edges, the method used in this work cannot detect the shape of the objects like the AVS model can. The shape of the object is effectively ignored. This could raise issues when the dimensions of objects are significantly different, e.g. tall and thin versus cubes. The AVS model was not favoured because the twenty keypoints can be all over the object instead of at the edges, which the AVS model requires. Applying AVS without having detected edges leads to poor results, because the distance to the closest point could vary much. Finding only the keypoints that are at the edge of the object, so that AVS can be applied, is not possible with SIFT, although a bounding box around the keypoints could be calculated and used as an alternative to true edges.

By taking twenty points, a cut is made in the computational resources required to process spatial relations. Substracting the coordinates results in three numbers, one for the x-, y- and one for the z-coordinate, e.g. $x = 125$, $y = 40$, $z = 14$ for the relation *to the left of.* As an example, from the point of view of a person presenting items, for *to the left of* only x would be non-zero; x would be a negative value. For *on the right of* there would be a positive value for x. The y and z coordinates would be at 0. The other coordinates are not zero due to different sizes of objects, imperfect learning of objects, camera noise, and difference in the chosen 'best keypoints'. Another possible issue is that relative distances in three dimensions are represented instead of angles and distance between objects. The system has some representation of distances, but those representations may not be the ideal ones to represent some of the spatial relations. This might make it harder to distinguish between directional and topological relations. In the evaluation (Section 5) is shown to what extent these problems affect performance.

Algorithm 2 SVC training algorithm.

```
for all spatial-relations $in$ database do
    $locationobject1 \leftarrow$ average$(x, y, zof20bestmatchesobject1)$
    $locationobject2 \leftarrow$ average$(x, y, zof20bestmatchesobject2)$
    $relativelocations \leftarrow locationobject1 - locationobject2$
end for
$SVC \leftarrow$ train-svc$(relativelocations)$
```

Kille learns spatial relations using a Linear Support Vector Classifier (SVC), also known as Support Vector Machines (SVM). An SVC is a supervised machine learning classifier, that performs well with relatively little data compared to other classifier algorithms, which was the main criterion for using this classifier. Support Vector machines place samples on a map and attempt to draw lines that separate the different possible outcomes by the greatest margin possible (Cortes & Vapnik, 1995). With linear SVC as applied, only linear lines are drawn. This is scaled to the number of input dimensions, three in the case of Kille. This means that the SVC will try to create hyperplanes that separate the samples as well as possible in a three dimensional space, consisting of x, y, and z coordinates, respectively width, height and depth. As there is no online learning (learning where samples can still be added after the classifier has gone through initial training), the SVM is retrained every time a sample is added. This does entail more computing, but this has no significant effect on the performance of Kille because the domain that is worked in is small and manageable. This is due to the computational cost of training a SVC with such little samples being dwarfed by the computational cost of recognizing the objects and obtaining the coordinates, and immediate responses not being required in a conversational setting.

5 Evaluation

The performance of learning single objects is evaluated, as well as the performance of learning spatial relations. Both are tested without the use of ASR to avoid learning names or phrases that were not intended to be learned at that point.

In most research, a bias in the data is bad, but a slight bias both reflects a realistic learning scenario, as well as increases the rate of learning. Skočaj et al. (2011) showed that the order and way objects are presented has a big influence on the rate of learning. Therefore a slight bias in the data has been introduced. This is done by presenting items in a normal way, instead of using the same pixel frame each time for learning. The system learns from a different frame showing roughly the same thing, but not exactly the same thing. These things would be perceived as the same thing to a human, but are not a perfect pixel copy and therefore not perceived as the same thing to a computer. Thorough testing has been done, to test the bias the system has for training data. Bias is the degree to which the system is tuned towards training data, instead of real world data. It is important to test for re-substitution, errors that appear when testing on training data, as well as testing on new data. In this experiment this means testing the recognition of objects if they are presented in a different manner or if they are different objects altogether than the original learned objects. Three tests are conducted to test object recognition, one on the training set, one on a set with noise introduced in the form of rotation of the training objects, and one where the training objects have been substituted with objects of the same category. Spatial relations are tested for the system's capability to learn them, as well as their fuzziness and the personal preferences of two human testers. Because spatial relations are vague concepts depending on several contextual factors (e.g. preference, the way the observer is raised, point of view) the notion of spatial relations differ between people. This makes it important to have more than one tester.

5.1 Object recognition

5.1.1 Task

The way item names are learned by Kille allows the system to reliably recognise objects with only few learning samples. Often only a single sample is needed, because a nearest neighbour algorithm is used to match current views with the saved sets of features. Nearest neighbour search is an effective way of finding the most similar features in a set. The low number of samples required is in contrast to other object recognition algorithms that require a large number of samples to build reliable recognisers (Skočaj et al., 2011). The trade-off of the nearest neighbour analysis is that classification is slower and requires a lot of memory as a dataset has to be memorised. This is in contrast to classification where a more abstract model of the features is created. Nearest neighbour analysis is not the slowest or most accurate method available. As an example, Figure 7 shows a green circle (the object to be classified) and two possible classes to classify to. This green circle can be classified to either the blue squares or the red triangles. If $k = 3$, the three closed samples are observed (the solid line circle). In this case this circle includes two red triangles and one blue square, which means the green circle is classified as a red triangle. If $k = 5$ (dashed circle) it is classified to the blue square class, as the blue squares are in the majority (Altman, 1992). In Kille this works slightly differently, but is based upon the same idea. Distance to the sample feature is calculated using the euclidean distance over the multi-dimensional vector a feature consists of. Instead of immediately classifying using the class that occurs most within the k closest neighbours of the sample feature, the features are taken and used for further calculation. This is done with $k = 2$.

Another alternative for searching through the candidate features is exhaustive search. Exhaustive search entails searching through the full search space to find the best possible match. This alternative is by far the slowest, and is beyond practical use on current hardware because of the time it takes.

The recognition performance is tested on ten different categories of objects. The objects that were chosen are typical household objects of appropriate size for the Kinect sensor. Due to the design of the Kinect,

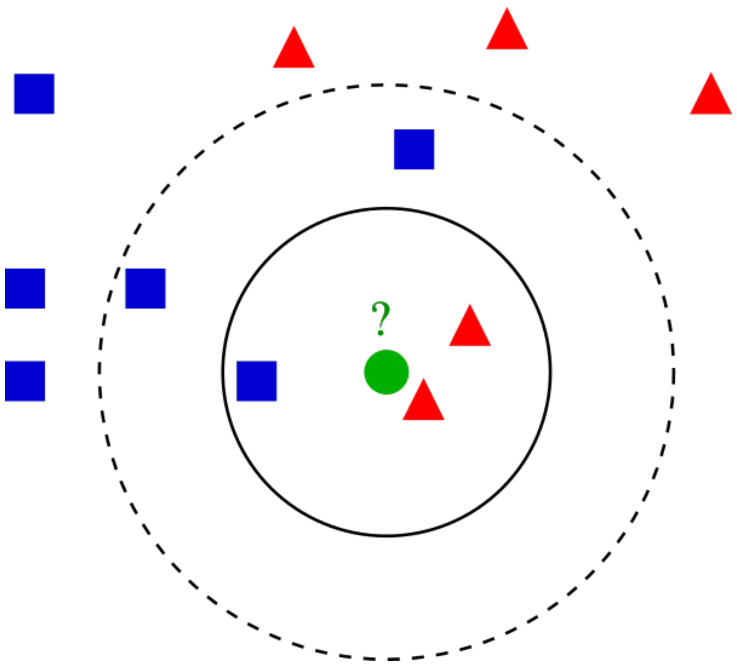

Figure 7: Visual aid for KNN classification. By Antti Ajanki AnAj - Own work, CC BY-SA 3.0, https://commons.wikimedia.org/w/index.php?curid=2170282.

reflective objects do not result in a proper depth reading, making Kille only see parts of the object after removing the background. The objects are placed upon a glass pedestal, which is transparent enough to not get picked up by the depth sensor of Kinect. This ensures that only the object is in view. The used categories are:

- apple
- banana
- bear
- book
- cap
- car
- cup
- paint can (later referred to as 'paint')
- shoe
- shoebox

5.1.2 Data representation

When a lot of data is produced by testing a system, it can be difficult to show this data in a manner that reflects the data both honestly as well as accurately. This subsection demonstrates how to avoid giving a skewed view of the data, and introduces a measure that allows for easy observable but still accurate representation of testing data.

Testing data is represented in multiple manners, to provide a clear view of the performance of the system. With the way Kille scores objects, every model receives a positive score for the object perceived. If an apple is shown, there will be a positive score for the model for apple, but also a positive score for the models for ball, cup, car, box, etc. The difference is the degree in which the models match the object. A better match is a match that maximises the distance from the second match. A larger distance generally means less similarity between objects, and a better match. When the data is represented in a standard way, e.g. showing data for each model for each object, the graph becomes cluttered and it becomes impossible to read how well the system performs.

In order to represent the information in one graph or table, a new measure is presented. The second goal is, in addition to representing more information in a clear way, to eliminate this skewed view that measures such as accuracy can give, as is demonstrated in the previous subsection. Optimally, a measure would include information about

- whether the recognition was correct or not.
- if it was: how close the recognition was to incorrect.
- if it was not: how close the recognition was to correct.

Here a measure is presented that succeeds in including these pieces of information and that is also easy to read. This measure is referred to as *Correct - next*, taking the score (as calculated by Algorithm 1) that would be the correct recognition, and subtracting the highest score that is not this correct recognition.

An example of correct recognition by an agent is shown in Table 2. An apple is presented to the system. The system has models for apple, banana and car. The system gives the apple a confidence score of 0.8, the banana a score of 0.2 and the car a score of 0.7. This results in a $\underline{\text{correct}}$ - next score of $0.8 - 0.7 = 0.1$.

	Apple	Banana	Car
Apple	0.8	0.2	0.7

Table 2: Example of possible test results of a correct recognition. The $\underline{\text{correct}}$ - next score is $0.8 - 0.7 = 0.1$.

Table 3 contains an example of incorrect classification. Again, an apple is presented to the system, and the system has models for apple, banana and car. In this case the system gives the apple a lower confidence score, of 0.5. The banana receives a score of 0.7 and the car a score of 0.4. The system will incorrectly claim the object observed is a banana. The $\underline{\text{correct}}$ - next score in this case is $0.5 - 0.7 = -0.2$.

	Apple	Banana	Car
Apple	0.5	0.7	0.4

Table 3: Example of possible test results of an incorrect recognition. The $\underline{\text{correct}}$ - next score is $0.5 - 0.7 = -0.2$.

With this measure, positive scores mean recognition was successful, but they also indicate how easy the object was to distinguish from others. A higher score means a more distinct recognition. Negative scores indicate incorrect recognition, where the score also indicates how close the system was to classifying correctly. Higher (closer to zero) means the system was closer to being right. In every case a higher score means better recognition.

To demonstrate how this measure makes data easier to read, baseline recognition for the apple is demonstrated in a more traditional way, as well as with the $\underline{\text{correct}}$ - next measure. The traditional way is demonstrated in Figure 8. The graph contains data only for one object. Ideally, all tested objects are represented in one graph, so they can also be compared with each other. Doing this in the traditional way results in a graph with one hundred lines, 10 lines for each of the 10 objects. Such a graph would not be readable. From Figure 8 one can tell that recognition of the apple has been successful three out of four times, and that it was a relatively close call when recognition failed. When recognition was successful, the difference between the apple and the banana is much more pronounced.

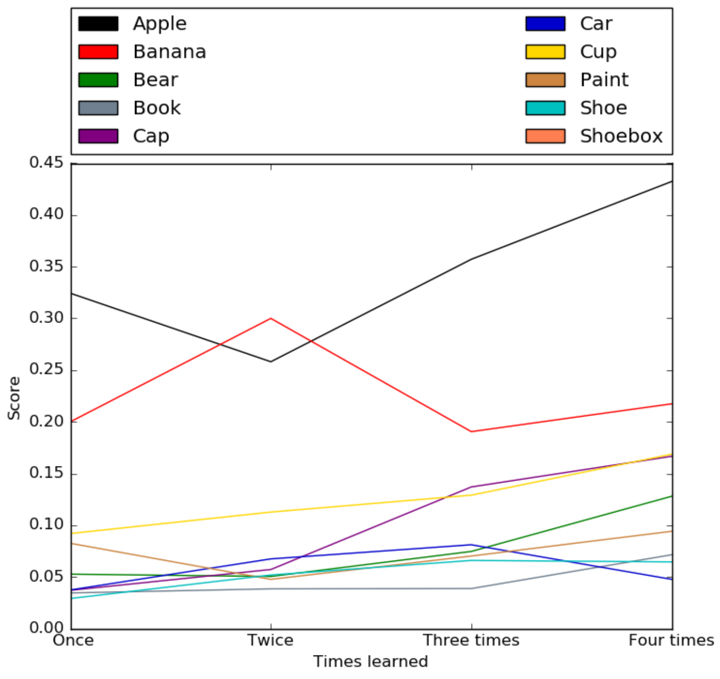

Figure 8: Recognition scores over four learning rounds. An apple was shown to the system.

Figure 9 shows the same data, but in a 'compressed' form, into the correct - next measure. The same data can be extracted from this graph, except for information about what objects were close matches, and what label was incorrectly classified with negative scores. This is a trade-off, as the graph is much easier to read. Recognition was successful three out of four times, but when it was not successful the classifier wasn't far off. A major disadvantage of this measure is that it does not represent the performance of objects that were not the best incorrect recognition (banana) or the correct object (apple). If the scores for the other objects are close, it tells us that the model for the correct object may not be good enough, or that it simply looks a lot like the other objects and is therefore hard to classify. If the scores for the other objects are not close (like in the example in Figure 8), it suggests that the model for the correct object is good, but that its model has much in common with another model.

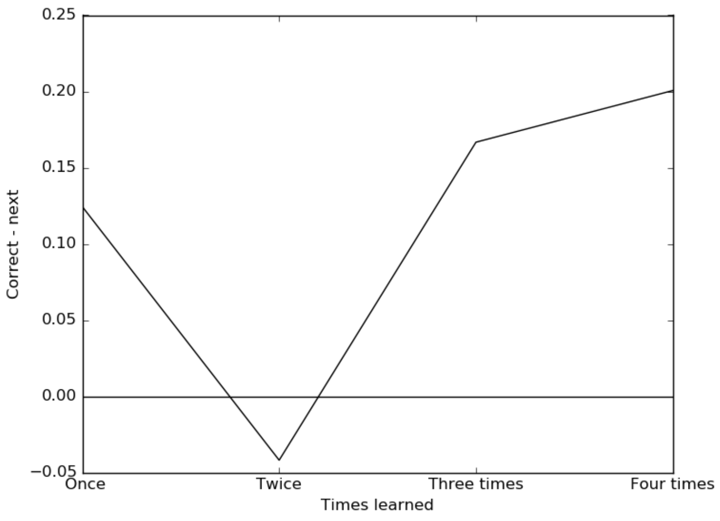

Figure 9: Recognition scores for the apple over four learning rounds using correct - next.

It is important to note that the system can not calculate correct-next, as it does not know what the correct object is. The introduced measure is solely for post-testing analysis.

Three sets of object recognition evaluation were performed, one as a baseline, and two incorporating noise, in the form of rotation and in the form of showing similar objects from the same category, respectively.

5.1.3 Procedure

The first evaluation of object recognition uses only the base object for a category, tested in four learning rounds. This evaluation provides the baseline recognition performance, i.e. the performance of the system at optimal conditions. The objects are shown almost exactly - as accurately as possible - as how they were taught. Note that even when objects are not moved, there are still differences between successive scans. In this experiment, however, objects have been moved in between learning and recognising. The experiment consists of four rounds. In each of these rounds the objects are learned and the recognition performance is recorded. This means, first object 1 is shown and learned, then object 2, then object 3, up to object 10. After this, object 1 is shown (but not learned) and the recognition scores are tested and recorded. Then follows object 2, all the way to object 10 again, only testing for recognition scores. The third and fourth round are identical to the first and second round, starting by learning object 1 again. The learning is cumulative, meaning that for the second round it is possible that the model for the item as it was shown in the first round is matched, instead of the one shown in the second round. Scores are calculated as in Algorithm 1 in Section 4. The highest score of all models of items learned up to the testing moment is picked as winning score. This tests the ability to learn objects and reinforce knowledge about them.

The second and the third test incorporate noise. Evaluating the classifier in this way is important, because it is not only valuable to recognize objects in the ideal situation, but also in more difficult situations. These evaluations do not offer an overly optimistic view of the true performance of the system, like the baseline tests do. The second evaluation tests the recognition of the ten base objects when they are rotated. Each object is observed in eight different rotations, 0 degrees (facing the camera), 45 degrees turn, 90 degree turn, 135 degree turn, 180 degree turn (with the backside (if applicable) of the object to the camera), 225 degrees turn, 270 degrees turn and 315 degrees turn. The objects are the same as the objects used for the baseline testing, thus the 0 degree evaluation is the same evaluation as the fourth round from the first set of evaluations. This tests whether the data is representative of data the system may encounter in real world situations.

5.1.4 Results

	apple	banana	bear	book	cap	car	cup	paint	shoe	shoebox
apple	**.343**	.227	.076	.046	.099	.058	.126	.074	.053	.166
banana	.201	**.357**	.058	.035	.085	.087	.148	.066	.046	.124
bear	.080	.121	**.260**	.074	.089	.091	.120	.099	.074	.136
book	.142	.233	.074	**.496**	.114	.197	.246	.130	.085	.220
cap	.122	**.208**	.076	.049	.146	.096	.103	.083	.061	.114
car	.104	.183	.053	.067	.077	**.414**	.119	.076	.069	.149
cup	.099	.145	.063	.066	.091	.052	**.330**	.094	.054	.120
paint	.119	.140	.075	.076	.083	.147	.121	**.221**	.062	.111
shoe	.078	.123	.070	.056	.079	.116	.124	.076	**.319**	.103
shoebox	.190	.332	.099	.188	.145	.305	.313	.166	.111	**.376**

Table 4: Baseline test results, averaged over the four testing rounds. The first column indicates what item has been shown, whereas the first row indicates what the scored item is. The correct guess, the diagonal, is underlined, and the object that received the highest score (so the one that would be picked as recognized object) is printed in bold face. Values that are both bold face and underlined are correct guesses.

Table 4 shows the recognition scores for the ten objects in the first test, the baseline test. The scores indicate that out of the ten objects, only the cap was recognized incorrectly, as the system thought it was seeing a banana. This is on average over the 4 rounds of learning. The apple was also recognized incorrectly, but only once. The full data is shown in Table 11 through Table 14.

	average	standard deviation	$\underline{\text{correct}}-\text{next}$
apple	.127	.099	.116
banana	.121	.101	.155
bear	.114	.059	.123
book	.94	.122	.250
cap	.106	.053	-.062
car	.131	.110	.231
cup	.111	.088	.185
paint	.115	.051	.075
shoe	.114	.076	.196
shoebox	.222	.115	.044

Table 5: The average column indicates the average score, a high score shows the object is generic and looks much like the other objects. Standard deviation is calculated over the values of each of the four tests, and thus are not the standard deviation over the values in the table, as those have been averaged over the four rounds. The $\underline{\textit{correct}}-\textit{next}$ column shows the correct (underlined) value minus the highest value that is not correct. These values indicate how hard an item is to recognize, its uniqueness. These are plotted in Graph 10.

Table 5 gives a clearer view of the true performance of the system on the baseline objects. The average column shows the score each object gets, averaged over all models. A high score is better than a low score. Some objects are harder to recognize than others, such as the cap and the shoebox. This can have multiple explanations. The possible explanation that the score calculation method using the geometric mean of the two matching scores as explained in Section 4 still favours objects with more features than others, such as the shoebox, is ruled out. This is the case because the apple, with the second lowest number of features, does also score quite high, as well as the car. It is likely because the objects with lower scores are more similar to other objects, i.e. relatively many of their features also match features in other objects.

The $\underline{\textit{correct}}-\textit{next}$ column indicates how hard an item is to recognize, given that other items are presented as well. It shows uniqueness, how likely it is that the shown item is confused with the next most highly ranking item. This data is the received score for the correct object (the score for the best model of apple when an apple is shown) minus the highest score given to an incorrect object (in the example of the apple, the score for the best matching object that is not the apple). This means that for objects that score above zero, the object was recognized correctly. Distinct objects have a high score, i.e. there is a large distance between the matched object and other objects. The system is more likely to correctly recognize objects if they are distinct. This score is also represented in Figure 10, for each of the rounds. In the graph this represents both if the object was recognized correctly and if there were any other candidates close to being recognized. Of the total of $4 \times 10 = 40$ guesses, the graph contains 5 incorrect recognitions, one for the apple and 4 for the cap.

Figure 10 shows how the recognition performance looks like with more learning iterations, from only once to four times. One can observe that learning the object an additional time does not always improve recognition. This can be because the distractor objects (the other 9) were also learned an additional time. The objects differ in recognition over the rounds - with the exception of the banana, which may have just by coincidence received a good score in the first round, or a bad score in the fourth - but show a clear average.

As seen in Figure 11, the system performs significantly worse when objects are rotated. There is a difference between objects. This is expected, as many of the objects have very differently looking sides. A book is square or rectangular and large from the front, but appear as a small rectangle when looking at it from the side. Apples look very much alike from every side, which explains why they are recognized well after rotation. 315 degrees rotation is the same as 45 degrees, but then rotated in the opposite direction. This is the reason that the graph is quite symmetric, 45 degrees having roughly the same scores as 315 degrees, 90 degrees scores similar to 270 degrees, and the scores for 135 degrees rotation are close to those of 225 degrees rotation. The scores at 180 degrees rotation (seeing the back of the object) are better for some of

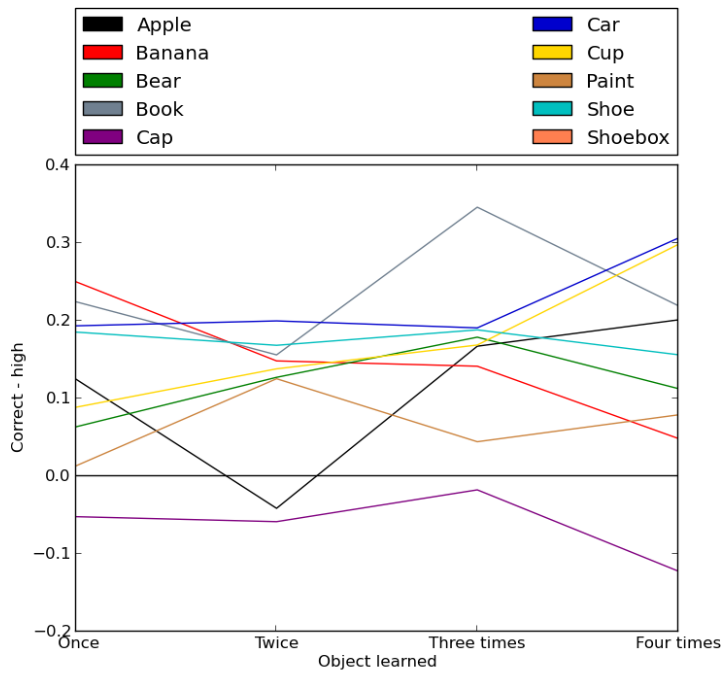

Figure 10: Baseline performance of object recognition, averages over four rounds. This table shows incremental learning. These scores are the correct-high scores before averaging took place. The averaged correct-high scores are in Table 5.

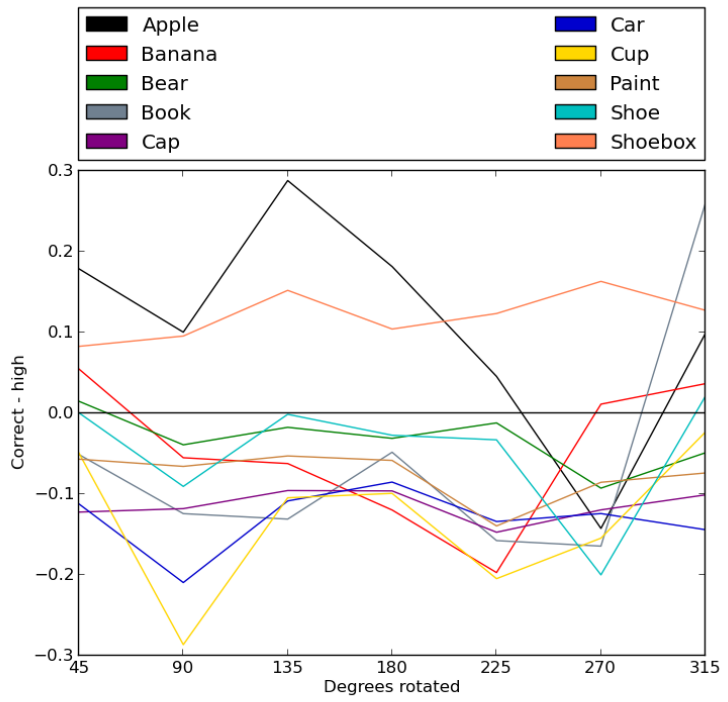

Figure 11: Performance of object recognition after object rotation. The scores are calculated using the correct-high measure. The raw data is in Table 15 through Table 22 in the appendices.

the objects than the scores at 135 and 225 degrees rotation. This is the case for the book, the car, the cup and the paint can. This is a result of the back of the object being similar to the front.

The third evaluation tests performance with other objects belonging to the same category. Each category consists of four different objects. This tests how well objects from the same category (e.g. apples, books, shoes) can be recognized after having only learned four iterations of a base object from the category. Of

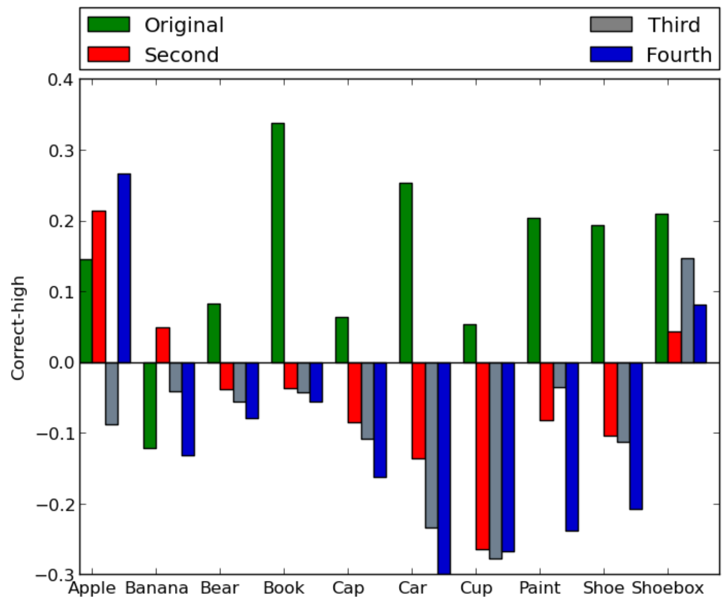

Figure 12: Performance of object recognition with the baseline object, that was used for training, and three other objects that the system was not trained on.

course, some categories consist of more similar objects than others. Different apples look much more alike than different shoes and are therefore expected to be recognized more reliably. Results of these tests are in Figure 12. This graph shows the original (baseline) score, as well as the scores for three other objects of the same category as the original object. Considering some objects *second*, some *third* and some *fourth* is merely an artefact of labelling, and does not indicate any sorting or further reasoning.

The system is not capable of reliably recognizing objects from the same category from which it has learned items previously without learning an instance of those items first. Only the apple and the shoebox in the evaluation were recognized well, where the shoebox is responsible for the majority of false recognition of other objects.

A couple of things can be concluded from the object evaluations. Firstly, there is a limited effect of using more samples on quality of recognition with the chosen SIFT approach. Performance on trained objects with the same angle is high, given the small number of samples, but does not improve significantly given more samples. Secondly, it can be concluded that objects are not recognized well when they are shown from a different side than they were trained on. This is expected behaviour, as the algorithm only looks for patterns on the object, which are different on different sides for many objects. Finally, it is observed that objects from the same category, but that look a-similar to the trained object, are not recognized well either. This is also expected, for the same reason, the algorithm only looks for patterns, which may be absent on other objects, even from the same category.

5.2 Spatial relations

This subsection describes how spatial relations were taught to the system, and how testing took place. The purpose of the test is not only to evaluate the performance of the system, but also to evaluate how spatial relations can be learned. The locations tested can be found in Figure 13. They are spaced out in sixteen places in a two dimensional plane, not utilizing the y-axis (height). The y-axis is still enabled in Kille. This allows for testing that is more akin to the real world, as a human would not be able to disable their height sensors either. The results will demonstrate whether Kille learns to ignore the, in this case, meaningless data from the y-axis.

5.2.1 Procedure

The test took place only in the lateral plane, to keep testing thorough, but still within reasonably size, given the time constraints. Testing in a 3-dimensional volume, which would include also descriptions such as *below* and *under*, will be done in future work. Locations of the object were controlled by dividing an imaginary circle around the landmark (the car from the baseline experiments) into eight directions, 45 degrees apart. One location close, and one further away from the center, for each of the directions. This totals to the 16 placements seen in Figure 13.

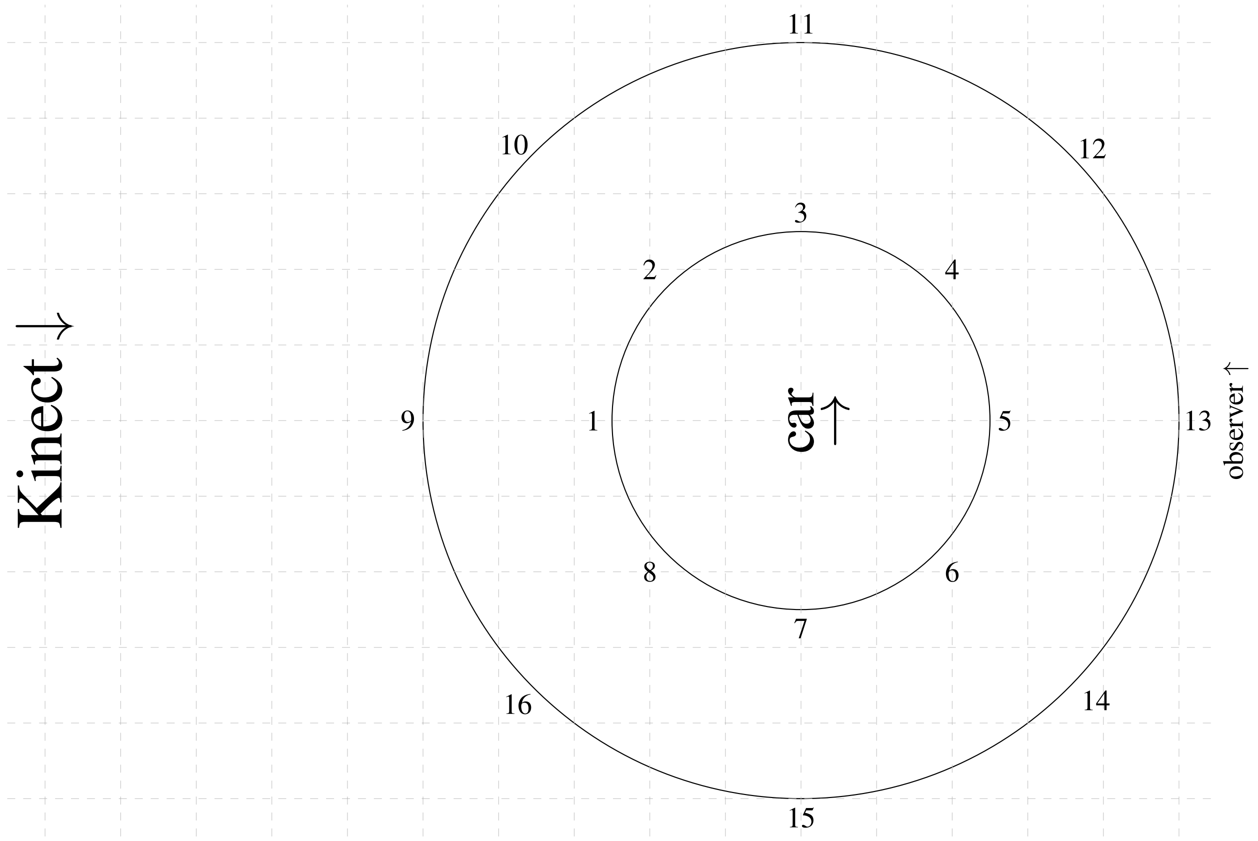

Figure 13: Figure showing the placement locations of the objects. 1 and 9 have been slightly relaxed, as Kille would not be able to see the car behind the book otherwise. The avoidance of occlusion is necessary because the system can not know which item is occluded if it is completely occluded. The car is facing towards 3. The human observer is standing behind 13, facing towards the car and the Kinect.

The placement of points 1 and 9 was slightly relaxed, because Kille will not be able to recognise the object in the back if they overlap too much from its point of view and will therefore not be able to judge that there are two objects. Theoretically, it would be possible to judge that one of the objects is hiding behind the other, but this is not implemented in Kille for a couple of reasons. The main reason is that the system would not be able to tell whether the tutor or user really is showing two items, or simply asked the wrong question, i.e. *where are these?* instead of *what is this?*. Voice recognition and object recognition errors need to be considered. Secondly, it would be impossible to know what object is hiding behind the other, unless item tracking is implemented, and item recognition is run even when the tutor or user does not ask about the objects.

At these 16 places the target object (the book from the baseline experiments) is placed, and the correct spatial relation is taught to the system by a human. The car is placed in the center. These objects were chosen because they are the least likely to cause bad relation classification as a result of bad object recognition, as they are the most distinct. The volunteer is instructed to stand behind the objects, with the objects between Kille and them the volunteer.

A total of 48 placements, by repeating the teaching three times. The order of placement is randomized by assigning a number to each of the placements and running these through a randomisation tool. Random order is necessary, as spatial relations are fuzzy and judgement of them may depend on judgement of the previous placement. A main goal for the system is to learn to ignore the third, irrelevant, y-dimension. Six spatial relations were used, namely *to the left of*, *to the right of*, *in front of*, *behind of*, *near* and *close to*. These spatial relations are frequently used in literature (Landau, 1996). Both *near* and *close to* have been chosen to investigate if there is any perceived difference between them. This bundles two different kinds of relations together: projective and topological relations. Projective relations (*to the left of*, *to the right of*, *in front of* and *behind of*) work with two dimensions; orientation in context of the reference frame and proximity. Topological relations (*near* and *close to*) only work with a single dimension; proximity (Dobnik, 2009). The test setup gives both features. The 16 positions capture distance from the center at 2 degrees and at 8 different angles. They are discretised to fixed units, rather than continuous. The classifier is not instructed about these differences, and has to decide for itself whether the parameters are important.

5.2.2 Task

The test was performed by two evaluators. It was not feasible within the time frame of this work to test with more evaluators. By using more than a single evaluators there is some measure of agreement in their judgement. Spatial relations are fuzzy, but the extent of this fuzziness differs between the spatial relations. If the two evaluators do not agree which spatial relation is presented, it is not fair to expect the system to perfectly predict the spatial relation either. It shows that different people have different reference frames and/or different spatial templates. These concepts are explained in Section 4.2.

The two volunteers give their judgements individually, so as to not coordinate on the meaning and consciously or subconsciously reach agreement. Disagreement also indicates how hard it is to learn that specific spatial relation. If the evaluators do agree, the system and spatial relation are judged not to be tuned specifically to a single the person (at least, not tuned to an extent that the system agrees consistently only with one person and not with the other). It is also judged that the spatial relation is easy to learn, in the specific context of the test.

During the evaluation, the volunteer and a second volunteer tested the learned spatial relations. They did this by taking the same 16 placements, in randomised order, and judging what they would each call the spatial relation. They did this independently of each other, and wrote down their judgement without sharing them. The system was then queried what it thought of the placement. The system's guess was written down, as well as whether the two volunteers agreed with the system or not.

As the relations are fuzzy, the volunteers can agree with the system even if they think a different spatial relation was more suitable. For example, the volunteer can consider position 4 in Figure 13 to best fit the description *near*, but they could agree with the system if it judges the position as *behind*, *close* or *to the right of*.

5.2.3 Results

Figure 14 visualises the spatial relations as they are learned by Kille. The data points shown are the same data points Kille used to train its models. The data points can be found in Table 25 in Appendix 7. Because it is hard to draw conclusions when the data points are clustered together, Figure 15, Figure 17 and Figure 18 show the important data separately. The same data is averaged and input into a bargraph in Figure 16. This graph is useful to see the data in the scatterplots in only a few points and serves as a figure to reference to, as it can be hard to tell actual averages from a 3d scatterplot.

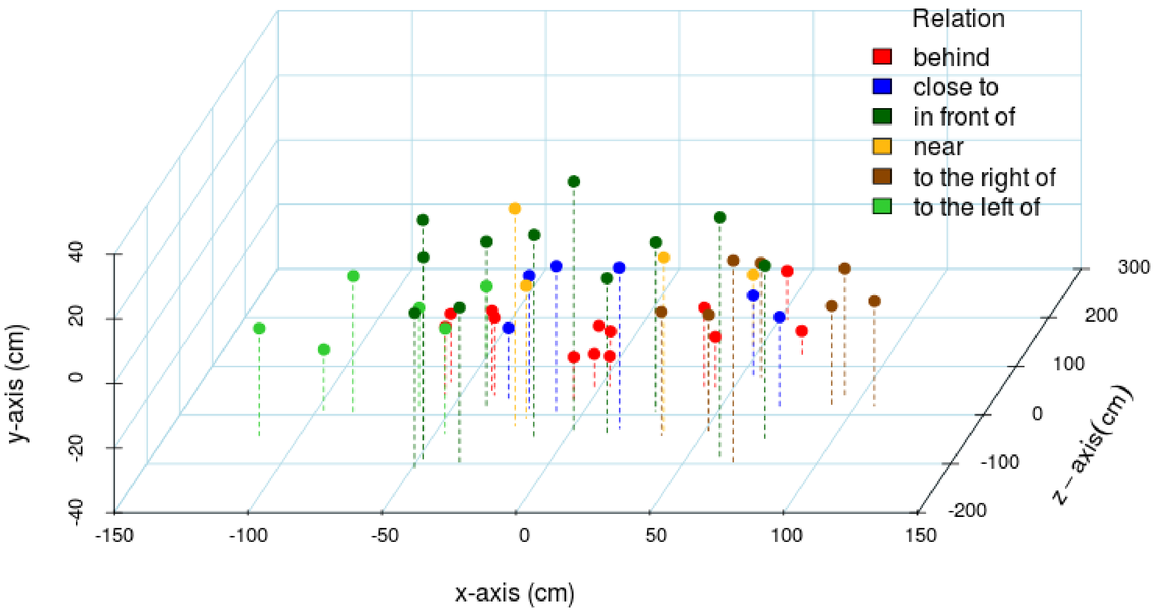

Figure 14: 3d scatterplot with the data from Table 25, found in Appendix 7. The spatial relations that the system has learned are shown in 3d, in a way that shows how Kille learns them. This corresponds with an inverse view point of the teacher, which is standing at the opposite side of Kille. Vertical drop-lines are drawn to aid the reader in understanding the 3d aspect on 2d paper.

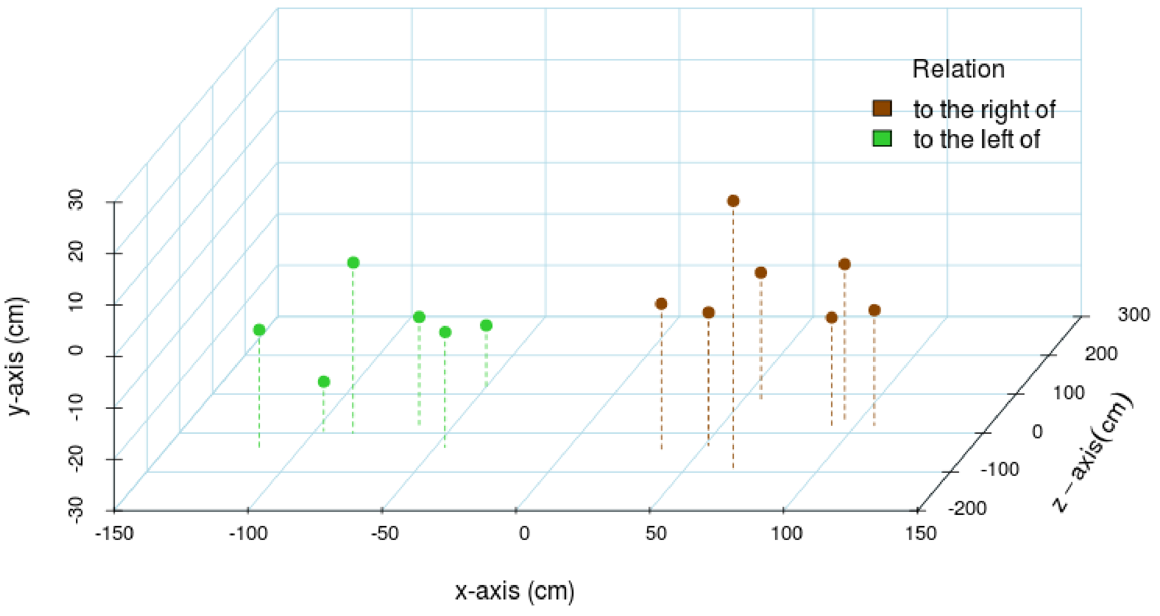

Figure 15: 3d scatterplot with the data from Table 25, found in Appendix 7. This plot shows only the relations *to the right of* and *to the left of*

.

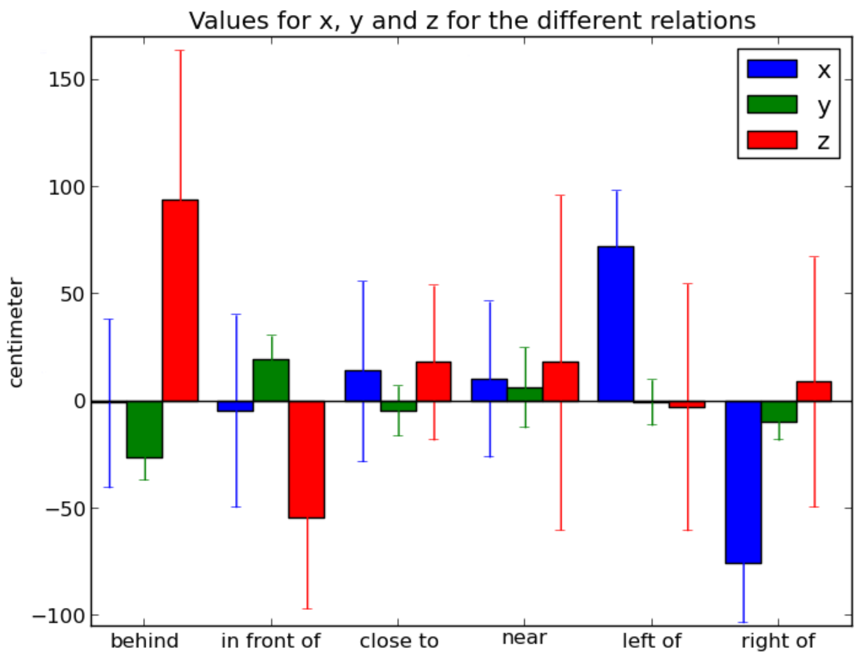

Figure 16: Average x, y and z values for the tested spatial relations.

Figure 15 contains the data points for the spatial relations *to the right of* and *to the left of*. One can observe that the learning of these spatial relations functioned well, as there is a clear cut between negative x and positive x. The data points are not exactly at 0 on the z-axis or y-axis, but are close. They average closely to $z = 0$ and $y = 0$ as seen in Figure 16.

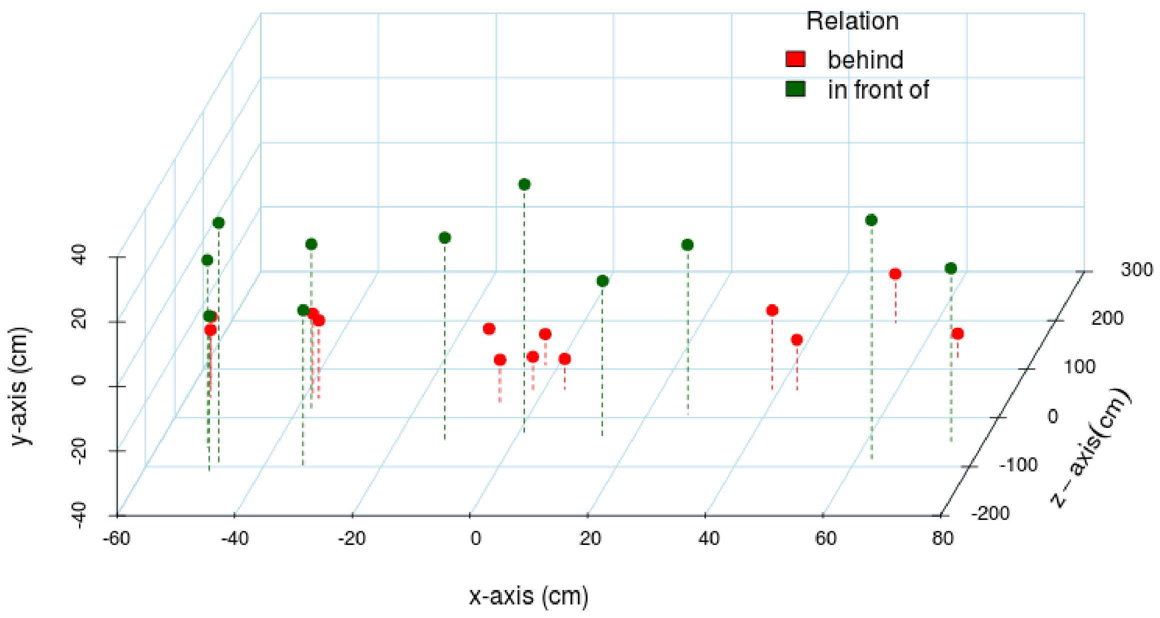

Figure 17: 3d scatterplot with the data from Table 25, found in Appendix 7. This plot shows only the relations *behind of* and *in front of*.

Figure 17 contains a 3d scatterplot with only the spatial relations *in front of* and *behind* shown. There is a clear distinction between front and behind. *in front of* only has data points on the negative side of the z-axis, in the direction of Kille, away from the human tutor. *behind* only has points with positive z, away from Kille, closer to the human tutor. One can observe that the data points for *behind* have lower y-values, that *behind* happens to be closer to the floor of the room. This is also visible in Figure 16. This is a result of Kille being positioned on a slightly higher level (from the floor, so in the y-plane) than the objects. It is looking down upon the objects, which results in objects closer appearing as being higher.

The data points for *close to* and *near* in Figure 18 are not as easy to interpret. The volunteer does not have a clear concept of the difference between *near* and *close to* in this spatial setting. There may not be a difference in this setting, but this may be different when objects have a (different) function. In Section 4.2 was discussed that spatial descriptors differ depending on the function of an object. This may also apply to the spatial relations discussed in this thesis. This is unlikely, however, as the book and the car are not functionally related. Finally, the difference between *near* and *close* is found to be rather personal (Fisher & Orf, 1991). The volunteer does not show signs of having a clear concept of the difference, but this does not mean that other people would not have judged differently.

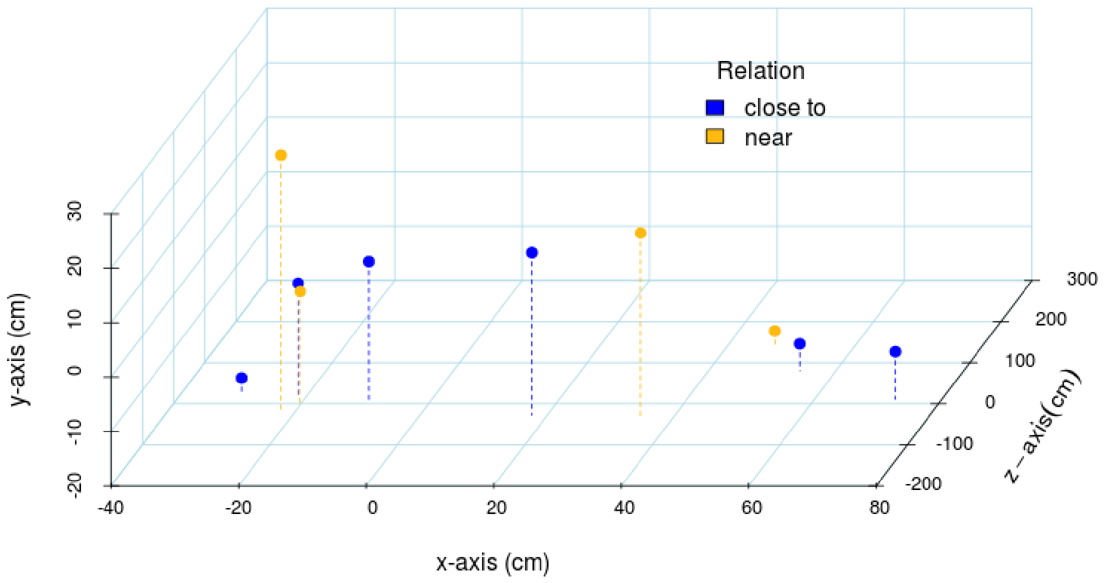

Figure 18: 3d scatterplot with the data from Table 25. This plot shows only the relations *close to* and *near*.

All objects have been shown on the same point in the y-axis, on a lateral plane, yet the y-values are not all equal. This is a result of the camera not being in the exact same horizontal plane as both objects are (i.e. it looks slightly down upon the objects), and random differences in the twenty SIFT features that are picked to calculate an object's centroid. The method of calculating centroids and subtracting their coordinates also leads to non-zero values for the Y-axis. An example is in Figure 19, where two objects are shown. These objects are standing on the same Y-plane, but Kille would assign a higher Y-value to the right object than it would to the left object, because the right object is much taller.

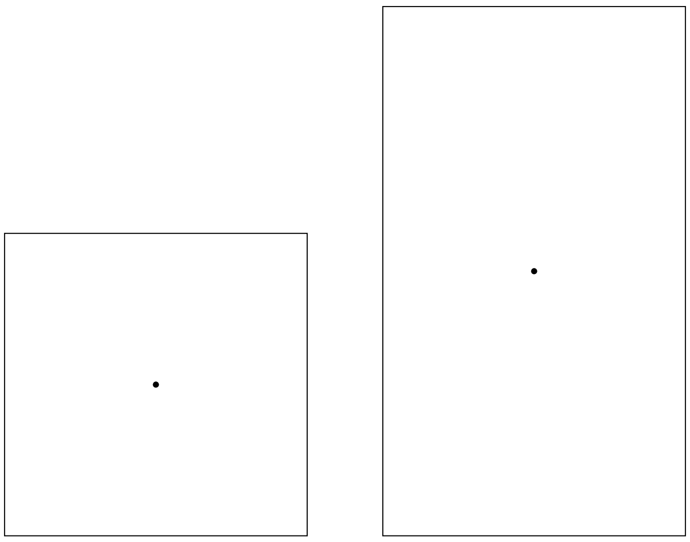

Figure 19: Two imaginary objects standing next to each other and their centroids. The objects are standing on the same y-plane, but subtracting the coordinates of their centroids will give a non-zero result.

To test the recognition performance, the volunteers were instructed to choose the best spatial relation for each of the 16 locations. The order in which the locations were judged was picked at random. Each location was judged twice. Table 6 gives insight in the agreement and disagreement between the two human evaluators. The first column indicates the choices of the first human evaluator (later also called *Volunteer 1*), and the first row indicates the choices of the second human evaluator (*Volunteer 2*). The agreement is indicated with its own row. This row shows how often the two evaluators agreed about what spatial relation was displayed. The *By chance* row indicates the odds that the agreement was a result of chance. This is calculated by taking the number of times each volunteer picked the spatial relation, and dividing it by the total number of judgements made by this volunteer. The resulting values from both volunteers are subsequently multiplied. To exemplify, the *By chance* value for the spatial relation *behind* is $\frac{4}{32} \cdot \frac{5}{32} = 0.01953$. These *by chance* values are a necessary part in calculating Cohen's κ. Cohen's κ is a statistic that measures agreement between judges (Artstein & Poesio, 2005).

	behind	front	left	right	close	near	total
behind	**3**	0	1	1	0	0	5
front	0	**5**	1	2	1	0	9
left	1	0	**2**	0	0	0	3
right	0	0	0	**3**	0	0	3
close	0	2	0	1	**4**	0	7
near	0	1	0	2	2	**0**	5
total	4	8	4	9	7	0	32
Agreement	**3**	**5**	**2**	**3**	**4**	**0**	17
By chance	.01953	.07031	.01172	.02637	.04785	0	.17578

Table 6: Agreement between the human evaluators.

κ is a more robust alternative to the simpler percentage agreement of total guesses, because it takes into account agreement that occurs by chance. Calculation is performed by dividing the number of chance-corrected agreements by the number of chance-corrected guesses. This results in a Kappa of $\frac{17-0.17578}{32-0.17578} = 0.52866$ for exactly correct guesses. This is a reasonable agreement for a task like this, where there are several alternatives a volunteer can choose from.

	behind	front	left	right	close/near	total
behind	**3**	0	1	1	0	5
front	0	**5**	1	2	1	9
left	1	0	**2**	0	0	3
right	0	0	0	**3**	0	3
close/near	0	3	0	3	**6**	12
total	4	8	4	9	7	32
Agreement	**3**	**5**	**2**	**3**	**6**	19
By chance	0.01953	0.07031	0.01172	0.02637	0.08203	0.20996

Table 7: Agreement between the human evaluators. Close and near have been combined into a single spatial relation.

Table 7 is based around the same data as Table 6, but *close* and *near* have been collapsed into a single spatial relation. This is achieved by adding all guesses for those spatial relationship together. The κ can be calculated in the same way, resulting in the slightly higher value of $\frac{19-0.20996}{32-0.20996} = 0.59107$, for exactly correct guesses.

Table 8 shows the number of incorrect guesses (guessed relation is not applicable to the presented situation), the number of correct guesses (guessed relation is applicable, but not the best guess possible), and the number of exact correct guesses (where the volunteer and the system make the exact same judgement). As a result of these definitions, *Exactly correct* is included in the number for *Correct*, because any guess that is exactly

	Volunteer 1	Volunteer 2	Percentage
Incorrect	13	12	39%
Correct	19	20	61%
Exactly correct	9	9	25%
Guesses	32	32	100%

Table 8: Table indicating the number of correct guesses. *Exactly correct* is included in *Correct*, and only counts when the guess the system made is exactly the same as that of the volunteer. This is in contrast to *Correct*, which also counts when the volunteer considers the system's guess correct although not necessarily their first choice.

correct must be at least applicable to the situation. Given the specific constraints of the experimental context and setup of this project, it is hard to compare the results with other research. The correct recognition seems promising, however, given the difficulty of spatial relation recognition and that each relation has only been taught on average $\frac{48}{6} = 8$ times.

Confusion Table 9 displays which spatial relations are often confused with one another. The system frequently confuses *close* and *near* with other relations like *in front of* and *to the right*. Likewise, other spatial relations are often unjustly classified as *near* or *close*. This does not mean that the model of these spatial relations is incorrect. Topological and projective descriptions are non-mutually-exclusive, which means a spatial relation could be described with the same degree of accuracy to *to the right* and *near*. Even within the topological or projective descriptions non-mutual-exclusiveness takes place. As an example, volunteer 1 may classify a layout as *to the left of*, while volunteer 2 may classify it as *behind*. Volunteer 1 may still agree that *behind* is a good description, even though it was not their first choice.

The agreement confusion matrix for volunteers versus Kille is in Table 9. The guesses volunteers have made have been added together, resulting in a total of 64 guesses. Cohen's κ is $\frac{16-0.19385}{64-0.19385} = 0.24772$, much lower than the κ for intra-human agreement, but still positive.

	behind	front	left	right	close	near	total
behind	**4**	2	0	1	0	2	9
front	0	**5**	3	3	6	0	17
left	0	6	**1**	2	0	0	9
right	4	1	0	**4**	0	1	10
close	1	9	1	0	**1**	2	14
near	1	1	1	0	1	**1**	5
total	10	24	6	10	8	6	64
Agreement	**4**	**5**	**1**	**4**	**1**	**1**	16
By chance	.02197	.09961	.01318	.02441	.02734	.00732	.19385

Table 9: Confusion matrix of human guesses and Kille guesses. The left column indicates the human guess, the top row indicates Kille´s guess. Bold face indicates that the human volunteer and Kille made the same guesses. The data is for both evaluators combined.

The agreement where *close* and *near* have been collapsed together is presented in Table 10. The κ coefficient is $\frac{19-0.22412}{64-0.22412} = 0.29440$. This is much lower than the agreement within the human evaluators, but still reasonable.

	behind	front	left	right	close/near	total
behind	**4**	2	0	1	2	9
front	0	**5**	3	3	6	17
left	0	6	**1**	2	0	9
right	4	1	0	**4**	1	10
close/near	2	10	2	0	**5**	19
total	10	24	6	10	14	64
Agreement	**4**	**5**	**1**	**4**	**5**	19
By chance	.02197	.09961	.01318	.02441	.06494	.22412

Table 10: Confusion matrix of human guesses and Kille guesses. This table combines close and near to one spatial relation.

The test results show reasonable agreement on things that are hard to learn, as discussed in Section 4.2. At a κ of 0.53, agreement between human evaluators is quite high. Agreement between human evaluators and Kille is acceptable at a κ of 0.29. This demonstrates that Kille is performing better than chance, even with the relatively small number of training instances. Despite the reasonable performance, tests on a larger scale are necessary to arrive at definitive conclusions.

6 Concluding remarks

We designed and implemented a system that uses Kinect and a dialogue system that is able to do situated learning by example from a human tutor. The system has been built modularly, allowing for easy replacement of parts of the system if the robot setup changes or new demands for the system arise. The parts chosen for the system are

- the SIFT algorithm, because of its performance with small number of samples.
- the Kinect camera, because it is a cheap 3d camera, and good open source drivers (freenect) are available.
- the ROS framework, because it interfaces well with Kinect.
- OpenDial, because it makes it easy to write dialogue, and because it is easy to interface with ROS.

The Kille framework built for this thesis combines these modules. A plugin for OpenDial, that has been dubbed ROSDial, had to be written to do this.

Several issues in the field of embodied and situated language processing are discussed in this work. The main issue is the grounding of objects and of spatial relations. Grounding (linking semantic representation to perceived objects) is very important in conversation (Roy, 2005; Kennington et al., 2015). The system is able to ground both objects and spatial relations as they are being demonstrated by a tutor, using human language in spoken form. Examples of the conversations the system has with tutors are displayed in Table 1 in the introduction and in the table in Appendix 7. The system remembers objects across sessions.

Object learning is achieved by combining pixels caught by a light sensor into high order, SIFT, features. These features are stored, and retrieved during the recognition stage. A nearest neighbour algorithm is applied to find the best matching set of stored features to the features calculated from the pixels of the perceived object. The harmonic mean of two measures is taken, to reduce biases towards perceived frames with few features, and towards frames with many features.

To learn spatial relations, first object recognition is run, to find the landmark and the target object. The top 20 SIFT features between these objects are taken and their location (x, y and z values) are averaged. This gives a good estimation of the respective center of the objects. Only twenty points are used to reduce the computational resources required. The average location of the target object is subtracted by the average location of the landmark. The result is a triplet, of x, y and z values. These values are input into a Linear Support Vector Classifier (SVC). SVC work relatively well with little data, compared to other classifier algorithms. The same method is applied to recognise spatial relations, but instead the trained SVC is asked what spatial description fits best.

We test the system on these two applications, object learning and spatial description learning. Our contribution here is the application of tutor learning in these domains.

A large problem the system in tackles object learning is the need of many other systems, usually build upon neural networks, for vast amounts of training data when learning to recognize objects and spatial relations. The presented system can successfully learn objects and spatial relations, using very few samples. A single learning sequence provides great recognition. Providing the system with more samples does not improve recognition results significantly. When a large number of samples is available, it is preferred to switch to a neural-network based system, as they are much more successful when data is available. In addition, having a lot of samples slows down a SIFT based approach significantly (Chesney & Koelewijn, 2015).

We also tested the system's recognition performance of items that were presented from a different side than they were trained. We found that recognition does not perform well. This is to be expected, as SIFT only has a limited tolerance for rotation. This problem can be dealt with relatively easily, by asking the tutor to rotate the object when teaching, and storing the feature sets of multiple of the visible frames. Given Figure 11, capturing these feature sets needs to happen more often than 8 times per full rotation, as at 45 degrees (so 8 total for 360 degrees) performance already drops by a lot. A problem with this approach is that it will add ambiguity for objects, as one object viewed from the front may look like another object viewed from the side.

Furthermore, we tested recognition of different objects that fall into the same category as the ones that were taught. With the exception of categories that have objects that look alike, recognition was poor during the category tests. Improving performance for category learning does not have a (relatively) easy solution like the solution we have seen to improve recognition of objects that have been rotated. To achieve knowledge of categories, instances of each object to be added to the category will have to be presented. In situations where this is not an option, such as situations where the number of objects in a category is too large, the traditional learning methods have to be applied, such as the system described by Skočaj et al. (2011). These systems can also be combined with the system as presented in this thesis. The advantage of doing this is that the traditional learning method can take care of common items, of which it has a database, while the method described in this work can take care of unique items.

The spatial relations tests took place on z-axis (depth) and x-axis (left, right), but not on the y-axis. This choice was made to allow for good testing without requiring too many samples and thus testing time. Tests took place with two participants, and therefor no definitive conclusions about the fuzziness of spatial relations or about the effort it takes a machine to learn the spatial relations can be drawn from the tests. The non-mutual exclusiveness (in many cases multiple spatial relations can be matched to a single situation) makes it even harder to draw conclusions. The test results have a reasonable agreement between evaluators, though the evaluators often did not agree on (the best) choice for the observed spatial relation. The tutor and testers have been instructed to use the system's point of view as a reference frame. When more reference frames are taken into consideration, like in a more realistic situation, the agreement would be lower.

In conclusion we show that our situated dialogue system can successfully learn objects and spatial relations from a tutor. It is able to learn using spoken human language, and give feedback about the knowledge it has acquired.

6.1 Future research

A few issues need to be addressed in future research. Firstly, the system can not recognise a spatial relation when the landmark and target object are opposite to the way they were learned. If the target object is above the landmark, then the landmark is below the target object. This requires that *below* is learned, which was not the case, and an inappropriate description was generated. This issue can be worked around by learning the counterpart of each spatial relation, if this exists. Most relations where a counterpart is not obvious, such as relations like *next to* or *around*, can be seen as their own counterpart. Using confidence scores for the spatial relations the system would be able to more easily understand spatial relations it already knows when they are referred to by a human tutor or instructor.

Secondly, when the system mishears the name of an object or relation, it will learn it with this wrong name. A common example encountered during testing is mishearing *boss* for *box*. In a situation where no GUI is present, and where the tutor does not hear the mistake in Kille's confirmation, the object would have to be re-learned. The system can handle re-learning when the tutor notices the mistake immediately, but has no way to correct the names of items or relations that were learned earlier. A feature to add in the future is to be able to deal with clarification from the tutor at a later stage in conversation, where all instances of *boss* could be added to *box*. This also might prove useful for tweaking ontologies, where items may need to be merged.

Thirdly, a different way of finding out how confusable objects are, could be used in post-testing analysis.

For example, when one shows the system an apple, the system could give a top 10 of matching models, with their score and ranking. Ideally, the various models for apples would be the most highly ranked, only being followed by other object models after. A measure could be to count the number of models for apple in the top 10, akin to the precision measure as it is commonly applied in information retrieval. This can be expanded by checking topmost results at different cut-offs, or by checking top ranked models until all models for the apple are found. Precision could also be used to test quality of models, if the confusability is known.

Knowing the name of a spatial relation and the names of the objects involved are not the only thing necessary for proper understanding of the relation. The interpretation of spatial relations and objects changes based on experience. In addition, meaning of linguistic expressions is often underspecified. In these cases, context, or world knowledge, is necessary to grasp the meaning of the expression (Dobnik et al., 2013). To solve these problems, a robot needs to be aware of the frame of reference. This is especially true for non-static robots (Dobnik et al., 2014). To achieve this a robot needs a good model of the world around it, and it needs to observe the objects' interaction over temporal sequences. It would need to create a mapping of objects that are not currently visible (Dobnik, 2009).

The system picks its object of focus based solely on the object being the only visible object in the camera field. The tutor must bring the object into the system's focus. This is not a reliable method any more when the robot is not able to perform proper background removal, such as when the tutor points at an object that is in the background. In the future, a robot needs to look around and single out the object by interpreting the tutor's eye-gaze or pointing. The objects still need to be singled out before running learning algorithms, to prevent noise and bad recognition. Bad recognition occurs when the surroundings are seen as part of the object, and when the same surroundings are visible when a different object is learned or recognised. This is possibly not an issue with a large number of samples and a different learning algorithm, one that is more robust against noise, for example with 3D models of objects.

Studies have shown that eye contact and eye gaze are important indicators in a social learning context (van Gog & Scheiter, 2010; Mayer, 2010). In future solutions of the object and spatial relations learning problem where systems have anything representing eyes, it is important for the robot to hold proper human-like eye contact and use eye gaze to indicate whether something is clearly understood or if there are any uncertainties. Humans are able to pick up on this and help the system faster (Thomaz & Cakmak, 2009).

References

Agrawal, M., Konolige, K., & Blas, M. R. (2008). Censure: Center surround extremas for realtime feature detection and matching. In *Computer Vision–ECCV 2008* (pp. 102–115). Springer.

Altman, N. S. (1992). An introduction to kernel and nearest-neighbor nonparametric regression. *The American Statistician*, 46(3), 175–185.

Artstein, R. & Poesio, M. (2005). *Kappa*^{3} *= Alpha (or Beta)*. Technical report, University of Essex Department of Computer Science. Available at: `http://cswww.essex.ac.uk/technical-reports/2005/csm-437.pdf`.

Atkeson, C. G., Moore, A. W., & Schaal, S. (1997). Locally weighted learning for control. In *Lazy learning* (pp. 75–113). Springer.

Bay, H., Tuytelaars, T., & Van Gool, L. (2006). SURF: Speeded up robust features. In *Computer vision–ECCV 2006* (pp. 404–417). Springer.

Bradski, G. (2000). The OpenCV library. *Doctor Dobbs Journal*, 25(11), 120–126.

Bradski, G. & Kaehler, A. (2008). *Learning OpenCV: Computer vision with the OpenCV library*. " O'Reilly Media, Inc.".

Cakmak, M., Chao, C., & Thomaz, A. L. (2010). Designing interactions for robot active learners. *Autonomous Mental Development, IEEE Transactions on*, 2(2), 108–118.

Chesney, S. & Koelewijn, T. (2015). Flazam: Using ImageNet for hierarchical object identification.

Cooper, R. & Larsson, S. (2009). Compositional and ontological semantics in learning from corrective feedback and explicit definition. In *Proceedings of SemDial 2009 (DiaHolmia): Workshop on the Semantics and Pragmatics of Dialogue* (pp. 59–66).

Cortes, C. & Vapnik, V. (1995). Support-vector networks. *Machine learning*, 20(3), 273–297.

Coventry, K. R. & Garrod, S. C. (2004). *Saying, seeing, and acting: the psychological semantics of spatial prepositions*. Psychology Press, Hove, East Sussex.

Coventry, K. R., Prat-Sala, M., & Richards, L. (2001). The interplay between geometry and function in the comprehension of over, under, above, and below. *Journal of memory and language*, 44(3), 376–398.

De Graaf, E. & Dobnik, S. (2015). KILLE: Learning objects and spatial relations with Kinect. *Proceedings of SemDial 2015, goDIAL*, (pp. 166).

Dobnik, S. (2009). *Teaching mobile robots to use spatial words*. PhD thesis, University of Oxford: Faculty of Linguistics, Philology and Phonetics and The Queen's College, Oxford, United Kingdom.

Dobnik, S., Cooper, R., & Larsson, S. (2013). Modelling language, action, and perception in Type Theory with Records. In D. Duchier & Y. Parmentier (Eds.), *Constraint Solving and Language Processing: 7th International Workshop, CSLP 2012, Orléans, France, September 13–14, 2012, Revised Selected Papers*, volume 8114 of *Lecture Notes in Computer Science* (pp. 70–91). Springer Berlin Heidelberg.

Dobnik, S., Kelleher, J. D., & Koniaris, C. (2014). Priming and alignment of frame of reference in situated conversation. In V. Rieser & P. Muller (Eds.), *Proceedings of DialWatt - Semdial 2014: The 18th Workshop on the Semantics and Pragmatics of Dialogue* (pp. 43–52). Edinburgh.

Fisher, P. F. & Orf, T. M. (1991). An investigation of the meaning of near and close on a university campus. *Computers, Environment and Urban Systems*, 15(1), 23–35.

Ghanimifard, M. & de Graaf, E. (2015). Wall-e. a technical project report for the embodied and situatied language processing course (ESLP) for the masters programme in language technology, university of Gothenburg, autumn 2014.

Harnad, S. (1990). The symbol grounding problem. *Physica D: Nonlinear Phenomena*, 42(1), 335–346.

Harris, C. & Stephens, M. (1988). A combined corner and edge detector. In *Alvey vision conference*: Citeseer.

Kelleher, J. D., Ross, R. J., Sloan, C., & Namee, B. (2011). The effect of occlusion on the semantics of projective spatial terms: a case study in grounding language in perception. *Cognitive Processing*, 12(1), 95–108.

Kennington, C., Dia, L., & Schlangen, D. (2015). A discriminative model for perceptually-grounded incremental reference resolution. In *Proceedings of the International Workshop on Computational Semantics (IWCS) 2015*.

Landau, B. (1996). Multiple geometric representations of objects in languages and language learners. *Language and space*, (pp. 317–363).

Larsson, S. (2013). Formal semantics for perceptual classification. *Journal of Logic and Computation*, online, 1–35.

Lauria, S., Bugmann, G., Kyriacou, T., & Klein, E. (2002). Mobile robot programming using natural language. *Robotics and Autonomous Systems*, 38(3), 171–181.

Lisin, D., Mattar, M., Blaschko, M. B., Learned-Miller, E. G., Benfield, M. C., et al. (2005). Combining local and global image features for object class recognition. In *Computer Vision and Pattern Recognition-Workshops, 2005. CVPR Workshops. IEEE Computer Society Conference on* (pp. 47–47).: IEEE.

Lison, P. (2014). *Structured Probabilistic Modelling for Dialogue Management*. PhD thesis, University of Oslo.

Logan, G. D. & Sadler, D. D. (1996). A computational analysis of the apprehension of spatial relations. *Language and space*, (pp. 493–529).

Lowe, D. G. (1999). Object recognition from local scale-invariant features. In *The proceedings of the seventh IEEE international conference on Computer vision, 1999.*, volume 2 (pp. 1150–1157).: IEEE.

Lowe, D. G. (2004). Distinctive image features from scale-invariant keypoints. *International journal of computer vision*, 60(2), 91–110.

Maillat, D. (2012). Pragmatically disambiguating space: experimental and cross-linguistic evidence. *Space and Time in Languages and Cultures: Linguistic Diversity, John Benjamins, Amsterdam/Philadelphia*, (pp. 35–52).

Mayer, R. E. (2010). Unique contributions of eye-tracking research to the study of learning with graphics. *Learning and instruction*, 20(2), 167–171.

Muja, M. & Lowe, D. G. (2009). Fast approximate nearest neighbors with automatic algorithm configuration. *VISAPP (1)*, 2.

Patel, B. & Patel, N. (2012). Motion detection based on multi frame video under surveillance system. *Int. J. Comput. Sci. Netw. Secur*, 12(3), 100–107.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., & Duchesnay, E. (2011). Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12, 2825–2830.

Philbin, J., Chum, O., Isard, M., Sivic, J., & Zisserman, A. (2008). Lost in quantization: Improving particular object retrieval in large scale image databases. In *Computer Vision and Pattern Recognition, 2008. CVPR 2008. IEEE Conference on* (pp. 1–8).: IEEE.

Quigley, M., Conley, K., Gerkey, B. P., Faust, J., Foote, T., Leibs, J., Wheeler, R., & Ng, A. Y. (2009). ROS: an open-source robot operating system. In *ICRA Workshop on Open Source Software*.

Regier, T. & Carlson, L. A. (2001). Grounding spatial language in perception: an empirical and computational investigation. *Journal of Experimental Psychology: General*, 130(2), 273.

Roy, D. (2005). Semiotic schemas: A framework for grounding language in action and perception. *Artificial Intelligence*, 167(1), 170–205.

Rublee, E., Rabaud, V., Konolige, K., & Bradski, G. (2011). ORB: an efficient alternative to SIFT or SURF. In *Computer Vision (ICCV), 2011 IEEE International Conference on* (pp. 2564–2571).: IEEE.

Serre, T., Kouh, M., Cadieu, C., Knoblich, U., Kreiman, G., & Poggio, T. (2005). *A theory of object recognition: computations and circuits in the feedforward path of the ventral stream in primate visual cortex*. Technical report, DTIC Document.

Skočaj, D., Kristan, M., Vrecko, A., Mahnic, M., Janicek, M., Kruijff, G.-J. M., Hanheide, M., Hawes, N., Keller, T., Zillich, M., et al. (2011). A system for interactive learning in dialogue with a tutor. In *Intelligent Robots and Systems (IROS), 2011 IEEE/RSJ International Conference on* (pp. 3387–3394).: IEEE.

Talmy, L. (1975). Figure and ground in complex sentences. In *Annual Meeting of the Berkeley Linguistics Society*, volume 1 (pp. 419–430).

Thomaz, A. L. & Cakmak, M. (2009). Learning about objects with human teachers. In *Proceedings of the 4th ACM/IEEE international conference on Human robot interaction* (pp. 15–22).: ACM.

Toutanova, K., Klein, D., Manning, C. D., & Singer, Y. (2003). Feature-rich part-of-speech tagging with a cyclic dependency network. In *Proceedings of the 2003 Conference of the North American Chapter of the Association for Computational Linguistics on Human Language Technology-Volume 1* (pp. 173–180).: Association for Computational Linguistics.

van Gog, T. & Scheiter, K. (2010). Eye tracking as a tool to study and enhance multimedia learning. *Learning and Instruction*, 20(2), 95–99.

7 Appendices

Appendix A: An example of a short dialogue with the system.

S	Hello, I want to learn some objects!
Presents	a book.
U	What is this?
S	The object is thought to be book. Could you show me more of book?
Presents	a book in a different location and orientation.
U	This is a book.
S	Ok, I learned book.
Presents	a box.
U	Do you know what this is?
S	I don't know what this is.
U	This is a box.
S	Ok, I learned box.
Presents	a gnome.
U	What is this?
S	I don't know.
U	This is a gnome.
S	Ok, I learned gnome.
Presents	a gnome and a box.
U	What is this?
S	One of the objects is gnome, the other is box. Please tell me where these things are.
U	The box is to the left of the gnome.
S	Ok, this relation is called 'to the left of'.
U	What is this?
S	The box is to the left of the gnome.

Appendix B: Object recognition evaluation data

Baseline

	apple	banana	bear	book	cap	car	cup	paint	shoe	shoebox
apple	**.324**	.200	.053	.034	.037	.037	.092	.082	.029	.108
banana	.133	**.383**	.013	.013	.019	.089	.056	.068	.051	.087
bear	.055	.085	**.180**	.068	.086	.057	.117	.063	.067	.117
book	.132	.217	.065	**.441**	.082	.193	.169	.140	.080	.199
cap	.122	**.120**	.081	.036	.070	.063	.057	.050	.031	.115
car	.027	.171	.031	.040	.042	**.364**	.037	.070	.062	.144
cup	.076	.149	.055	.055	.026	.041	**.237**	.050	.025	.108
paint	.123	.156	.060	.037	.074	.181	.092	**.194**	.052	.114
shoe	.048	.076	.070	.036	.063	.122	.117	.066	**.312**	.127
shoebox	.152	.162	.075	.146	.083	**.319**	.124	.156	.122	.296

Table 11: Raw data for the baseline tests, first learning round.

	apple	banana	bear	book	cap	car	cup	paint	shoe	shoebox
apple	.258	**.300**	.050	.038	.057	.067	.113	.048	.052	.154
banana	.278	**.426**	.047	.025	.093	.083	.152	.056	.036	.174
bear	.053	.141	**.268**	.050	.075	.105	.095	.087	.065	.128
book	.133	.241	.050	**.471**	.077	.156	.315	.109	.063	.259
cap	.103	**.205**	.057	.056	.145	.099	.114	.092	.064	.109
car	.093	.171	.065	.037	.075	**.397**	.198	.051	.078	.142
cup	.063	.144	.054	.063	.058	.037	**.282**	.088	.029	.086
paint	.075	.125	.080	.054	.052	.139	.118	**.266**	.062	.141
shoe	.036	.161	.072	.042	.089	.121	.116	.093	**.329**	.073
shoebox	.141	.325	.108	.149	.098	.265	**.434**	.167	.114	.432

Table 12: Raw data for the baseline tests, second learning round.

	apple	banana	bear	book	cap	car	cup	paint	shoe	shoebox
apple	**.357**	.190	.075	.039	.137	.081	.129	.070	.066	.169
banana	.207	**.348**	.071	.063	.148	.082	.161	.086	.053	.111
bear	.060	.130	**.317**	.075	.096	.091	.125	.125	.085	.139
book	.161	.218	.096	**.596**	.132	.195	.250	.133	.099	.221
cap	.133	**.237**	.062	.036	.218	.102	.113	.096	.073	.121
car	.113	.200	.052	.113	.106	**.390**	.077	.092	.089	.112
cup	.126	.170	.055	.060	.142	.063	**.338**	.120	.080	.120
paint	.145	.119	.085	.099	.070	.125	.136	**.189**	.062	.090
shoe	.069	.118	.066	.083	.082	.088	.128	.073	**.320**	.133
shoebox	.263	**.493**	.101	.221	.198	.307	.336	.176	.113	.344

Table 13: Raw data for the baseline tests, third learning round.

	apple	banana	bear	book	cap	car	cup	paint	shoe	shoebox
apple	**.432**	.217	.128	.071	.167	.048	.169	.094	.065	.232
banana	.188	**.270**	.101	.038	.079	.094	.222	.053	.043	.125
bear	.154	.129	**.275**	.103	.097	.113	.141	.122	.081	.162
book	.142	.258	.085	**.477**	.165	.243	.252	.139	.099	.201
cap	.128	**.271**	.102	.069	.149	.120	.130	.094	.077	.110
car	.182	.190	.065	.078	.085	**.504**	.165	.090	.046	.198
cup	.130	.116	.086	.085	.137	.069	**.463**	.118	.081	.165
paint	.134	.159	.073	.113	.138	.141	.138	**.237**	.072	.097
shoe	.160	.135	.074	.062	.081	.131	.134	.071	**.316**	.079
shoebox	.203	.349	.114	.235	.202	.331	.356	.164	.095	**.433**

Table 14: Raw data for the baseline tests, fourth learning round.

Rotation test

	apple	banana	bear	book	cap	car	cup	paint	shoe	shoebox
apple	**.571**	.368	.197	.179	.280	.205	.182	.205	.214	.427
banana	.160	.161	.203	.161	.112	.112	.112	.160	.107	**.283**
bear	.053	.071	**.261**	.095	.087	.058	.082	.089	.095	.179
book	.095	.055	.109	**.508**	.055	.073	.094	.095	.079	.169
cap	.086	.121	.113	.144	**.259**	.069	.070	.088	.090	.194
car	.133	.110	.124	.157	.094	**.440**	.120	.135	.111	.186
cup	.175	.122	.144	.137	.099	.092	**.297**	.156	.127	.243
paint	.099	.083	.106	.134	.096	.078	.088	**.380**	.080	.177
shoe	.085	.055	.095	.154	.065	.045	.068	.078	**.347**	.095
shoebox	.167	.211	.146	.210	.115	.120	.088	.152	.101	**.420**

Table 15: Raw data for the rotation tests, 0 degrees.

	apple	banana	bear	book	cap	car	cup	paint	shoe	shoebox
apple	**.609**	.414	.281	.267	.212	.169	.246	.230	.176	.430
banana	.235	**.400**	.216	.217	.125	.196	.216	.164	.140	.346
bear	.120	.064	**.137**	.106	.083	.086	.092	.103	.076	.123
book	.192	.127	.208	.261	.217	.182	.125	.179	.102	**.337**
cap	.151	.098	.133	.124	.116	.035	.069	.113	.079	**.168**
car	.246	.125	.201	.174	.111	.211	.164	.166	.136	**.333**
cup	.255	.161	.123	.215	.111	.175	.186	.217	.100	**.298**
paint	.113	.085	.104	.131	.092	.060	.105	.103	.073	**.153**
shoe	.058	.107	.115	.120	.088	.069	.047	.075	.102	**.160**
shoebox	.140	.171	.123	.180	.153	.133	.092	.117	.095	**.262**

Table 16: Raw data for the rotation tests, 45 degrees.

apple	banana	bear	book	cap	car	cup	paint	shoe	shoebox	
apple	**.438**	.267	.199	.219	.169	.146	.200	.154	.127	.338
banana	**.500**	.444	.282	.229	.235	.193	.169	.235	.203	.456
bear	.148	.071	.109	.125	.080	.088	.106	.087	.082	**.145**
book	.122	.333	.125	.222	.173	.132	.154	.201	.121	**.347**
cap	.068	.104	.090	.145	.049	.063	.050	.093	.059	**.168**
car	**.292**	.131	.133	.162	.119	.082	.198	.149	.096	.279
cup	.500	.316	.168	.265	.159	.242	.231	.235	.159	**.518**
paint	.105	.079	.082	**.157**	.072	.061	.086	.091	.092	.157
shoe	.139	.158	.174	.169	.110	.090	.097	.130	.116	**.207**
shoebox	.271	.182	.135	.300	.151	.151	.153	.146	.123	**.395**

Table 17: Raw data for the rotation tests, 90 degrees.

	apple	banana	bear	book	cap	car	cup	paint	shoe	shoebox
apple	**.600**	.313	.207	.155	.147	.146	.123	.154	.168	.234
banana	.200	.314	.161	.147	.253	.212	.200	.181	.170	**.376**
bear	.148	.066	.153	.130	.086	.080	.120	.101	.103	**.171**
book	.118	.150	.104	.133	.136	.065	.121	.131	.089	**.264**
cap	**.173**	.138	.109	.121	.077	.049	.133	.130	.076	.163
car	.123	.077	.170	.172	.088	.134	.111	.147	.105	**.243**
cup	.200	.255	.118	.295	.167	.172	.193	.129	.122	**.298**
paint	.052	.078	.119	.155	.064	.048	.063	.120	.067	**.173**
shoe	.035	.034	**.068**	.050	.054	.035	.037	.052	.066	.061
shoebox	.125	.222	.149	.157	.096	.121	.079	.112	.114	**.374**

Table 18: Raw data for the rotation tests, 135 degrees.

	apple	banana	bear	book	cap	car	cup	paint	shoe	shoebox
apple	**.485**	.205	.120	.160	.118	.107	.136	.089	.090	.304
banana	.146	.226	.162	.190	.185	.119	.101	.136	.133	**.347**
bear	.108	.079	.103	.114	.085	.040	.086	.090	.093	**.135**
book	.100	.140	.098	.254	.099	.094	.125	.143	.069	**.303**
cap	.088	.074	.111	.130	.058	.046	.106	.117	.074	**.155**
car	.165	.104	.130	.184	.085	.128	.078	.088	.109	**.213**
cup	.270	.226	.184	.272	.175	.103	.186	.209	.148	**.286**
paint	.110	.071	.076	**.166**	.065	.065	.049	.107	.056	.113
shoe	.077	.056	.072	.091	.059	.055	.067	.082	.080	**.108**
shoebox	.242	.146	.141	.224	.151	.080	.160	.108	.106	**.346**

Table 19: Raw data for the rotation tests, 180 degrees.

	apple	banana	bear	book	cap	car	cup	paint	shoe	shoebox
apple	**.571**	.429	.178	.234	.257	.175	.200	.152	.157	.526
banana	.381	.489	.217	.217	.256	.129	.197	.224	.231	**.687**
bear	.120	.106	.125	.091	.066	.077	.063	.083	.076	**.138**
book	**.327**	.145	.154	.168	.113	.078	.133	.143	.122	.250
cap	.141	.140	.105	.164	.066	.097	.088	.178	.083	**.213**
car	.273	.167	.205	.176	.159	.180	.149	.184	.165	**.315**
cup	.109	.206	.161	.309	.109	.098	.104	.186	.103	.291
paint	.126	.091	.113	.230	.116	.090	.113	.090	.093	**.229**
shoe	.061	.061	.079	.084	.074	.080	.055	.083	.092	**.126**
shoebox	.100	.140	.133	.168	.120	.113	.052	.106	.084	**.291**

Table 20: Raw data for the rotation tests, 225 degrees.

	apple	banana	bear	book	cap	car	cup	paint	shoe	shoebox
apple	.387	.327	.222	.261	.243	.176	.182	.149	.191	**.530**
banana	.323	.333	.137	.135	.184	.211	.154	.146	.164	.259
bear	.207	.079	.154	.132	.129	.082	.115	.100	.139	**.247**
book	.176	.204	.190	.124	.171	.156	.074	.232	.090	**.289**
cap	**.198**	.114	.160	.191	.078	.083	.092	.120	.120	.163
car	**.259**	.094	.092	.198	.143	.135	.206	.149	.103	.250
cup	.304	.203	.140	.187	.130	.137	.211	.144	.122	**.366**
paint	.155	.094	.110	.166	.073	.110	.139	.135	.074	**.221**
shoe	.195	.089	.177	.095	.110	.102	.064	.125	.097	**.298**
shoebox	.094	.115	.140	.115	.174	.149	.128	.112	.074	**.337**

Table 21: Raw data for the rotation tests, 270 degrees.

	apple	banana	bear	book	cap	car	cup	paint	shoe	shoebox
apple	**.870**	.556	.365	.384	.286	.289	.303	.278	.276	.771
banana	.244	**.311**	.215	.159	.275	.168	.152	.149	.201	.256
bear	.111	.098	.087	.085	.078	.042	.086	.086	.090	**.136**
book	.070	.101	.103	**.418**	.063	.043	.067	.094	.090	.159
cap	.170	.167	.096	.192	.112	.098	.127	.096	.072	**.213**
car	.302	.190	.167	.282	.129	.158	.228	.168	.126	**.303**
cup	.163	.120	.131	.127	.091	.077	.239	.135	.083	**.263**
paint	.087	.086	.091	.139	.060	.062	.079	.088	.086	**.162**
shoe	.027	.036	.083	.068	.028	.031	.031	.070	**.138**	.117
shoebox	.101	.146	.141	.143	.104	.080	.080	.098	.096	**.273**

Table 22: Raw data for the rotation tests, 315 degrees.

Category test

	apple	banana	bear	book	cap	car	cup	paint	shoe	shoebox
apple	**.714**	.545	.229	.248	.259	.214	.233	.222	.220	.523
banana	.220	.357	.214	.217	.198	.176	.144	.168	.148	**.419**
bear	.096	.088	.153	.117	.088	.069	.084	.084	.097	**.176**
book	.112	.104	.119	**.264**	.083	.096	.104	.106	.087	.213
cap	.156	.121	.134	.173	.158	.104	.092	.116	.092	**.231**
car	.302	.202	.181	.192	.145	.240	.169	.164	.138	**.309**
cup	.287	.226	.175	.196	.188	.133	.201	.178	.147	**.390**
paint	.137	.166	.113	.164	.100	.101	.115	**.190**	.091	.183
shoe	.089	.129	.123	.168	.111	.105	.077	.124	.174	**.217**
shoebox	.130	.190	.132	.188	.118	.118	.101	.131	.090	**.324**

Table 23: Average scores for the category tests. Correct object for recognition is underlined, actual recognized object is in bold.

Appendix C: Spatial Relations recognition data

Position	Volunteer 1	Correct	Volunteer 2	Correct	System
1	front	1	front	1	close
16	front	1	front	1	left
15	left	0	left	0	front
3	right	1	near	1	near
9	front	1	front	0	right
14	behind	0	left	0	front
12	right	1	right	1	left (reversed)
7	close	0	close	0	front
2	front	1	near	1	close
5	behind	1	behind	1	behind
8	close	1	front	1	front
4	right	1	close	1	behind
11	right	1	right	1	right
13	behind	1	behind	1	behind
6	close	0	close	0	front
10	right	1	front	1	front
8	front	1	close	1	close
13	behind	0	behind	0	near
11	right	0	right	0	behind
14	left	0	behind	0	front
6	close	0	near	0	front
2	front	1	close	1	front
10	right	1	front	1	right
16	left	1	front	1	left
5	close	0	close	0	front
3	right	0	near	0	behind
12	right	1	behind	1	left (reversed)
15	left	0	left	0	front
9	front	0	front	0	close
4	close	1	close	1	near
7	close	1	near	1	left
1	front	1	front	1	front

Table 24: Raw data of the judgements of the system and the two volunteers. The positions refer to Figure 13.

Relation	x	y	z
behind	-5.899248922	-25.7971297566	25.159424
behind	52.9700973764	-24.9458936331	195.67029
behind	-57.0941310379	-18.9379494358	68.503113
behind	-4.3814409614	-51.240354757	256.68207
behind	67.0083699163	-32.7869058768	125.02954
behind	-1.6550292595	-29.0200090595	52.835754
behind	-55.9691280509	-19.1350315506	42.56366
behind	-15.2305940117	-39.2762856292	178.24469
behind	-37.4383432306	-15.9563714241	40.804688
behind	-38.562405956	-14.053581619	42.562622
behind	43.003867391	-24.3954195658	57.236633
behind	-2.2385462063	-30.3543536631	107.99451
behind	38.7201667751	-15.572121242	58.849121
behind	3.4483884939	-30.6087150945	58.872498
close to	72.7331371642	-11.2203694394	10.517578
close to	54.3019892837	-15.0657236041	80.633301
close to	-21.1237416391	1.1127229295	13.848206
close to	-30.7813648752	-17.5059283565	29.64386
close to	-9.3702505482	6.0984597678	0.52471924
close to	17.5434653754	9.8496034133	-27.456421
in front of	-37.5979425732	11.1356734346	17.597504
in front of	74.3171698838	13.5407262161	-47.650391
in front of	-11.6953726596	23.1944491884	-48.567078
in front of	0.9896407064	37.0970454915	-31.111084
in front of	-48.0301906822	34.2280911177	-90.871063
in front of	26.8227118405	12.3563162023	8.064209
in front of	62.9723123823	35.1807219369	-91.95636
in front of	-33.2209731208	8.4519599375	-100.03784
in front of	14.8578042958	8.9498172336	-42.96521
in front of	-48.6649611618	8.4341749048	-111.38696
in front of	-51.4155054475	18.17266025	-61.303833
near	-19.803331764	1.110220119	-6.265564
near	-22.0670463778	27.3381341133	-22.026855
near	47.7529627723	-16.7268485134	133.88611
near	34.9246418589	13.953116239	-33.744812
to the left of	94.0828955603	0.2178101032	34.702332
to the left of	34.9697452105	-1.7134549314	-41.730896
to the left of	68.4664465039	22.4244404381	-96.07373
to the left of	51.4144018082	-4.1318802992	-31.933655
to the left of	91.1628222604	-8.9677426905	18.767456
to the left of	107.1119922032	-7.4924531064	18.740112
to the left of	57.5061813588	-4.6578077119	77.872559
to the right of	-46.2187566287	-7.5654384444	-37.416016
to the right of	-50.0468429392	-18.0584898837	119.21436
to the right of	-85.0512020825	3.2073099111	-0.59783936
to the right of	-96.7378584092	-20.4079432733	4.9558105
to the right of	-115.0828989861	-6.7493251141	-42.250366
to the right of	-61.7103790912	-8.1859568248	9.7546997

Table 25: Data the system attached to each of the taught spatial relations.

Appendix D: Installation manual

The system has only been tested on Ubuntu 12.04, because of the requirement of drivers for the camera and compatibility for those with ROS. Please use Ubuntu 12.04 if you would like to try this software. It is possible to run the software in a virtual machine, yet this is not recommended because some host systems do not manage to properly forward all three virtual USB devices Kinect offers. Different RGB-d cameras should also work, as long as they provide /camera/depth/image_raw and /camera/RGB/image_color through ROS. This has not been tested.

After installing Ubuntu, update all software with:

```
apt-get update && sudo apt-get upgrade
```

Reboot the machine and run these commands another time. Ubuntu 12.04 is an old version, and running the command once is likely not enough as a reboot is necessary for some updates. If you are running Ubuntu in a virtual machine, install the guest extensions that allow forwarding of USB devices.

Execute the following commands to install ROS:

```
sudo sh -c 'echo "deb http://packages.ros.org/ros/ubuntu precise main" > \
                              /etc/apt/sources.list.d/ros-latest.list'
wget https://raw.githubusercontent.com/ros/rosdistro/master/ros.key -O - \
                                                         | sudo apt-key add -
sudo apt-get update
sudo apt-get install ros-hydro-desktop-full
sudo rosdep init
rosdep update
echo "source /opt/ros/hydro/setup.bash" >> ~/.bashrc
source ~/.bashrc
sudo apt-get install python-rosinstall
```

Install freenect drivers:

```
sudo modprobe -r gspca_kinect
echo 'blacklist gspca_kinect' | sudo tee -a /etc/modprobe.d/blacklist.conf
sudo apt-get install ros-hydro-freenect-stack
```

Install Oracle's Java:

```
sudo add-apt-repository ppa:webupd8team/java
sudo apt-get update
sudo apt-get install oracle-java8-installer
sudo apt-get install oracle-java8-set-default
```

Make sure Java is installed correctly. The software requires Java 1.8 or higher.

```
java -version
```

Install ROSJava:

```
sudo apt-get install ros-hydro-catkin ros-hydro-ros
sudo apt-get install ros-hydro-rosjava python-wstool
```

If a workspace for ROS does not exist yet, create one:

```
mkdir -p ~/catkin_ws/src
cd ~/cat kin_ws/src
```

Download Kille by cloning the github into the src folder:

```
git clone https://github.com/masx/Kille.git Kille
```

Generate the build and devel directories:

```
cd ..
catkin_make
```

ROSDial works best with Opendial 1.3. This can be saved to any folder of choice. Download it from it's offical site at opendial-toolkit.net, and edit the Nuance or Sphinx plugins to exclude the confidence scores they provide. Then download only ROSDial from the github below. Alternatively, download the ready edited version of Opendial and ROSDial:

```
git clone https://github.com/masx/ROSDial
```

Appendix E: User manual

Three terminal windows are required to boot the whole system. First open up the freenect drivers and roscore, with the command:

```
roslaunch freenect_launch freenect.launch
```

Then, run the following command in a new window:

```
rosrun kille kille.py
```

Finally, open up the third terminal and navigate to the folder that contains the ROSDial and Opendial folders. Execute the following command:

```
./ROSDial_start ROSDial.ROSDial
```

This will start ROSDial and Opendial, providing a GUI where it is possible to type or speak your commands.

ROSDial allows for multiple TTS and ASR systems. The default is MaryTTS with Sphinx ASR. In the event those need to be switched to Nuance, edit the domain.xml file in the ROSDial folder. Obtain your Nuance key from the Nuance website.