Academic year 2006–2007

Improving SIFT for 3D Object Recognition using Active Vision and Clustering

- Master's Thesis -

Author
Jelmer Ypma

Internal advisor
Dr. Bart De Boer

Internal advisor
Drs. Gert Kootstra

Artificial Intelligence
Rijks*universiteit* Groningen

Contents

Chapter 1

Introduction

A group of people has gathered on the ground and is looking at the sky. While they're trying to determine which object is flying above them, someone shouts: 'Is it a bird?' Someone else has a different opinion and says: 'Is it a plane?' And then they notice: 'No, it's Superman!' This is the well-known opening scene of the Superman television show and what we see here is prototypical for object recognition systems, the subject of this thesis.

Most object recognition systems receive one image of a scene as input and have to form a hypothesis which object is present. Sometimes a couple of additional images are presented to the system, but like the crowd in the scene above, no action is taken to obtain a better viewpoint to perform the recognition. The people in the crowd do not change position to get a better view of Superman and most object recognition systems are not capable of moving around.

The use of robots in computer vision research has become more widespread, and the benefit of using a robot as a vision platform is that it can move around, gathering information about the scene it is observing. I wanted to use active behaviour to enhance object recognition and originally the goal of my research was to focus on viewpoint planning. Viewpoint planning relates the actions of a vision system to its input. For instance, if the title of a book is hidden behind another book, we can move our head slightly to read the complete title. The action that we take depends on the object we want to see, and which information is missing.

Viewpoint planning has been researched before, usually in a highly controlled environment (e.g. Roy, Chaudhury, & Banerjee, 2004; Borotschnig, Paletta, Prantl, & Pinz, 2000). These methods use a global representation of objects, which can lead to problems. I wanted to use a different object recognition method and also the environment I use is less controlled. Unfortunately, changing all these things at once turned out to be too ambitious. In this thesis I describe and test a method for object recognition. The system is has some components of active vision, but actual planning of actions is not used. However, the methods in this thesis can form a starting point to incorporate viewpoint planning and I will shortly describe one possible way forward in chapter 8.

First, I had to determine which method to use for object recognition. Over time, different techniques have been proposed and tested, some of which I will mention below. Each of these methods has their own advantages and disadvantages, but in general all of them try to determine some distance

between the scene you observe and the objects stored in a database. If the distance between a stored object and the observed scene is small, then it is likely that the object is present in the scene. Object recognition mainly looks at what kind of representation should be used to store objects, and how to define a measure to determine the distance between a scene and the stored objects. Of course, these two problems are related, and multiple solutions have been proposed.

Different levels of object representation exist and, as Keselman and Dickinson (2005) note, in the history of object recognition a shift has taken place from prototypical-based object recognition to exemplar-based object recognition. The key distinction between these two approaches is the way in which the representational gap between object model and the observed scene is overcome. Either the observed scene can be brought closer to the object representation, or we can use an object representation which closely resembles the scene.

The first approach, the prototypical-based or model-based approach, uses very general, abstract representations of objects such as a description using cylinders, cubes and other parametric shapes or, slightly later, CAD models. Since two-dimensional images of objects are different from three-dimensional models, we have to make the image look more like the object representation. A lot of methods that were used during this time focused on extracting lines, corners or ellipses from an image, since these were the building blocks of the object representation. Advantages of this approach are, that we can recognize fully general objects; it doesn't matter which particular table we see, as long as it has four legs it can be recognized as table. A disadvantage is that only simple objects can be represented as a combination of simple shapes. Also, the method only works in simplified cases, where there is no distracting background or occlusion, and the methods are computationally intensive.

In contrast, to recognize objects we can also use an object representation as close as possible to the image. This is called the exemplar-based or appearance-based approach. Objects can for instance be represented by a collection of snapshots taken from different angles. The problem with this approach is that recognizing fully general objects is problematic. If we have seen an image of one table, we can probably recognize this table again in a later stage. However, a slightly different table, e.q. with different texture, can seem to be a truly different object.

Storing multiple images of each object requires a lot of space. Most of the methods that use the appearance of objects are therefore ways to extract information from the image such that not the full image has to be stored. They are different forms of data reduction, trying to keep the most valuable information. I used two of these methods in this thesis and a detailed description is given below.

The first method is principal component analysis (PCA), which is a technique to that can be used to compress different views of objects efficiently. The problem with this approach is that all objects need to fill the same space in an image and that the complete view of an object is used. This method was also used in the active object recognition system by Borotschnig et al. (2000) that I mentioned earlier. In chapter 2, I describe how PCA works and show the results of a small experiment I conducted.

Another method, which is used in the main part of this thesis, is a feature-based method, called SIFT (Lowe, 1999). Although it is also based on the appearance of objects, there is a difference with most systems that use PCA for object recognition. With SIFT multiple keypoints are extracted from an image and the combination of these keypoints form the representation of an object. This in contrast to the PCA technique, where for instance the complete face of a person is represented[1]. The SIFT method is described in chapter 3.

Some of the advantages of using keypoints to represent an object is that not all of the keypoints are necessary for recognition. This makes the method capable of managing occlusion. Furthermore, we can separate keypoints belonging to the object and keypoints belonging to the background using active vision, as I describe in chapter 4.

A disadvantage of SIFT is that a large number of keypoints are generated. Keypoints belonging to the same object, but sometimes also from different objects, can look very similar and I describe a clustering algorithm used to obtain a smaller keypoint database in section 5.

In chapter 6 I show how we can use this representation to actually recognize objects. The next chapter presents the results from several experiments testing the methods and finally, chapter 8 concludes.

[1]There are also systems which detect interest points and describe a small region around these points using PCA. However, most PCA systems use an object representation based on complete objects.

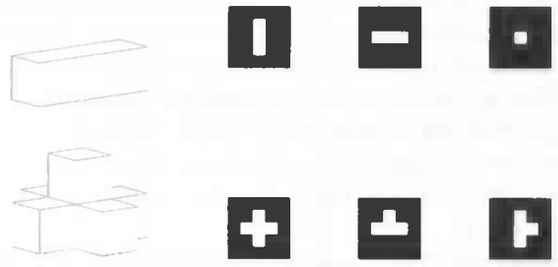
Chapter 2

Principal Component Analysis

Appearance-based methods have a very different way of representing objects than model-based object recognition systems. Instead of representing objects as a relational set of geometrical shapes, objects are represented using the way they look from different viewpoints, their two-dimensional appearance. This can be compared to inserting a photograph of a face in a passport, instead of some abstract description using circles for the eyes and a triangle for the nose.

Figure 2.1(a) shows two simple objects, with some appearances from conveniently chosen viewpoints in figure 2.1(b). The objects are for instance shown from above and from the side. Since each view of the objects consists of an image of 5×5 pixels, a vector of size 25 can be used to represent each appearance. Note, however, that in this example we show an extremely simplified representation. In order to represent objects more accurately, we should of course use more pixels. Furthermore, the appearance of an object depends not only on its shape and pose in the scene, but also on the illumination and the reflection properties of the object (Murase & Nayar, 1995). So, using grayscale or color appearances, which we usually encounter, instead of black and white appearances would be more sensible.

Representing all possible appearances of objects by images from different viewpoints and under different lighting conditions requires a lot of memory. Consequently, searching for matches in the object space takes a long time. Therefore, most appearance based methods are based on finding a compact



(a) Two objects (b) Different 2D appearances

Figure 2.1 – Two objects and some two-dimensional appearances

Figure 2.2 – Some *base* images used to represent objects

representation for the object space.

One way to represent objects in a compact way, is to construct a set of *base* images, such that each object appearance can be written as a linear combination of the base images. Every image in figure 2.1(b) can be represented as a linear combination of the base images in figure 2.2. E.g. the first image, I_1 can be written as

$$I_1 = \begin{pmatrix} 1 & 0 & 1 & 0 & 1 \end{pmatrix} \begin{pmatrix} e_1 \\ e_2 \\ e_3 \\ e_4 \\ e_5 \end{pmatrix},$$

where $e_1, \ldots, e_5$ are the five base images. Instead of 25 values to represent an appearance, we now only need five coefficients and a set of base images. By using the same set of base images for each appearance we can reduce the dimension of our object space. Introducing a new object would only require to store an extra vector of coefficients relating the appearance of the object to the base images. Of course, coefficients different from 1, or negative coefficients can be used to introduce gray values and create an even larger set of appearances.

This example might seem a bit far-fetched, since most of the pixels, for instance the corner pixels, are black for every appearance. But this brings us to the main thought how the data in images can be reduced. Exploiting the lack of variation in some pixel positions, is an easy way of reducing the dimension of the object space. In the example above, the base was constructed by hand, which is a tedious exercise. Luckily, more sophisticated methods exist, such as principal component analysis, which I will explain below.

2.1 Principal Components Analysis

In order to obtain a compact representation of the object space, we want to keep only the most relevant information available in the data. As we saw in the example above, variation in appearance holds the a lot of information; the pixels that are black in every appearance contain no extra information and we could therefore discard these pixels. Therefore, we want a method that encodes the variation in the data in an efficient way. Principal component analysis (PCA) is a statistical method, which does exactly this (Turk & Pentland, 1991).

The idea behind PCA is easiest explained by looking at the two-dimensional data in figure 2.3. In the upper part of the figure we see a scatter plot of some two-dimensional data, with some variation in the direction of the x-axis, and slightly less variation in the direction of the y-axis. Since our

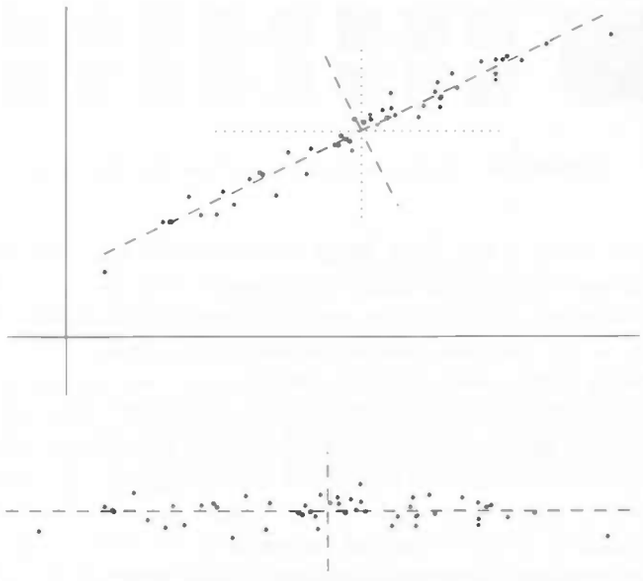


Figure 2.3 – Principal Components Analysis of two-dimensional data. The original data is given in the top figure, with the means of the x and y variable given as dots. The projection after PCA is given by the dashed lines, which is also given in the lower figure.

interest lies in the variation of the data, we subtract the mean from the data. The coordinate system then shifts to the position of the dotted lines.

The next step is a rotation of the coordinate system that still spans the same space. After this rotation we obtain one axis, where the variance is maximised, as can be seen in the lower part of the figure. PCA finds the optimal rotation of the coordinate system, such that the largest variance is encoded in the least number of axes, or principal components.

Data reduction is then accomplished, by discarding the components, which contribute the least to the variation. If we delete the y-axis in the lower part of figure 2.3, there is some information-loss, but it is much less than when we delete the y-axis in the upper part. Using only the new x-axis, the principal component, most of the information in the data is still available, using however one dimension less for storage.

The mathematical procedure to obtain the principal components as described by e.g. Turk and Pentland (1991) or Murase and Nayar (1995) is straightforward. Denote each image in the training set as a vector with fixed size, $\Gamma_1, \ldots, \Gamma_n$, and define the average of the images Ψ as

$$\Psi = \frac{1}{n} \sum_{i=1}^{n} \Gamma_i.$$

Each image differs from the average image by $\Phi_i = \Gamma_i - \Psi$. The principal components are obtained by calculating the eigenvectors of the correlation matrix Σ

$$\Sigma = \Phi\Phi',$$

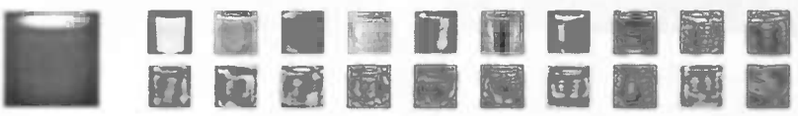


Figure 2.4 – *mean-image* (left) and first 20 *eigenimages* (right)

where $\Phi = [\Phi_1 \cdots \Phi_n]$. Each image can be written as a linear combination of the eigenvectors, usually called *eigenimages*. The points in the scatter plot in the example above can be perfectly represented in the new coordinate system as the weighted sum of the two eigenvectors. When we only store the values of the x-axis after the rotation, i.e. use just one weight and one eigenvector, the variation on the new y-axis is lost. Although this leads to some loss of information, the most important information is still retained.

Since we want a small number of *eigenimages* to represent the objects, we need to select those eigenvectors which span the most variation. The eigenvalues of Σ can be seen as measures of the variation that is explained by the associated eigenvector. If we sort the eigenvalues, and their corresponding eigenvectors, from high to low, then the first couple of eigenvectors are the principal components. Discarding the eigenimages corresponding to a small eigenvalue leads to a large reduction in required memory, while not much information is lost. A small set of eigenimages is enough to recognize objects in most cases, as we will see in the example below.

2.2 Experiment

In order to test the performance of PCA for object recognition, a robot was used to drive around eight objects in circles, capturing an image every 10°. These images are processed such that only the object remains. Since the background of the images contained only carpet, i.e. white noise, separating the object from the background was fairly easy accomplished using edge detection. However, as can be seen in figure 2.7, object segmentation fails in some of the images of the object on the third row. Subsequently, the image was scaled to an image of size 40×40, since the input of PCA should be vectors of equal size. One set of images for each object is used for training and one set, with different lighting conditions, is used for testing.

The recognition process gives a distance between the input image and each of the objects and poses in the database. Each of the different object-pose combinations result in its own distance. I count an object as correctly recognized if the object-pose with the smallest distance is the same object as in the input image. For the recognition rates I don't take the pose estimation into account.

In figure 2.4 the *mean-image* and the first twenty *eigenimages* are shown, which contribute the most to the total variance. The images are highly correlated, since images from the same object taken from a slightly different angle look very similar. Therefore, a low number of eigenimages results in good performance, as can be seen in figure 2.5.

Using only one eigenimage leads to 55% objects that are correctly classified. Increasing the dimension to five eigenimages, improves the recognition

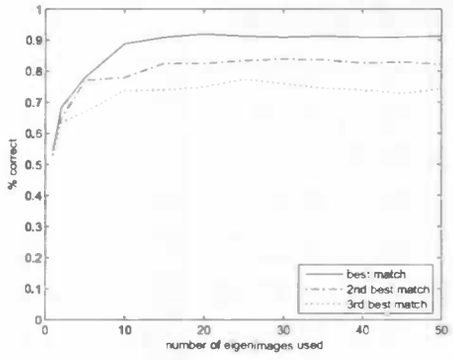


Figure 2.5 – Recognition rate using different numbers of eigenimages. The recognition rates are given for the best, second-best, and third-best match. The recognition rates increase with the number of eigenimages, but stays flat for dimensions above 20.

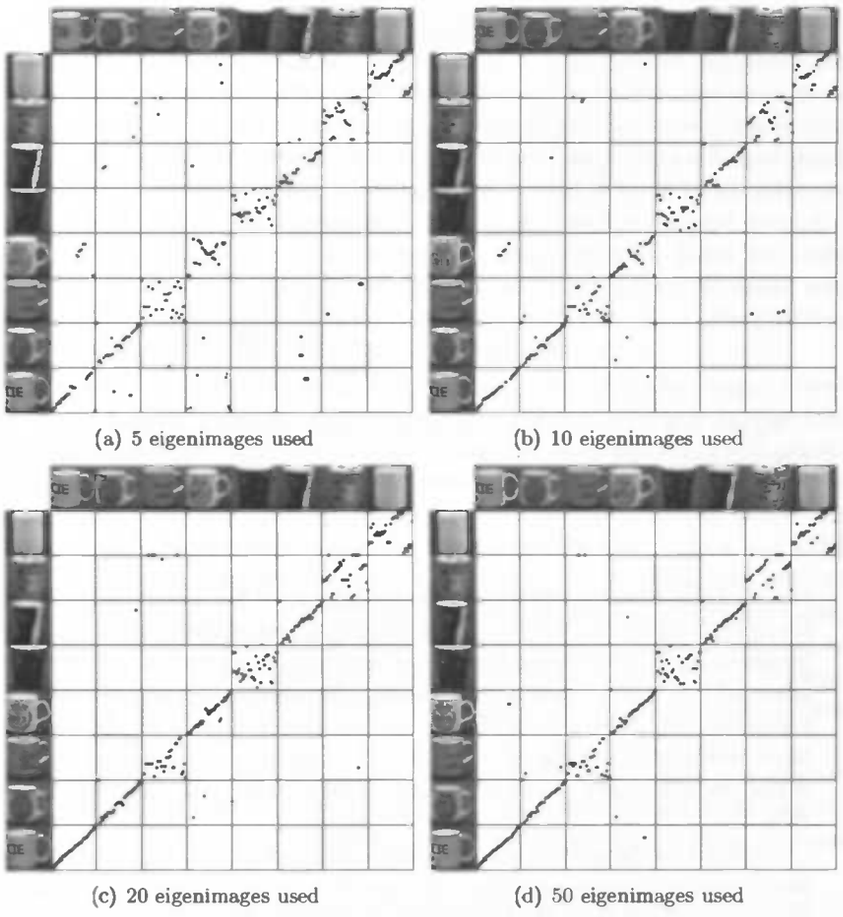
rate to 78%, whereas increasing the dimension to ten eigenimages, improves the recognition rate to 89%. The maximum recognition rate, 92%, is obtained using twenty eigenimages. The performance does not increase further, when adding extra eigenimages, possibly because of the errors in the object segmentation. In the same figure, the recognition rates of the second and third best match are also shown. In 83% of the cases, the second best match is the correct object, while the recognition rate for the third best match is 75%.

Instead of the performance in object recognition, we can also look at the performance in pose estimation. In figure 2.6 the object recognition and pose estimation of the presented training images is shown. On the horizontal axis the presented object and its pose is shown and on the vertical axis we see the estimation of object and pose. Dots within the squares on the diagonal represent correctly recognized objects. Within each square we can see the correspondence between the presented and estimated pose of the object.

Although object recognition is reasonably good when using five dimensions, pose estimation is far from perfect. This can be explained by the fact that *different* objects appear very different and to discriminate between two objects we do not need a lot of information. However, between different poses of *one* object, the differences are a lot smaller and therefore additional information is needed to accurately estimate the pose. Pose estimation is especially difficult for objects 5, 7 and 8. This should come as no surprise, since these objects are symmetric and appear the same from pose to pose. The poor performance in the recognition and pose estimation of the object in the third column is probably due to pre-processing. Object segmentation seems to be especially poor for this object.

2.3 Discussion

Although principal component analysis seems to work well on our example, the method has some drawbacks. In the example, and in most studies (e.g. Murase & Nayar, 1995), good performance is obtained, since the experiment



(a) 5 eigenimages used

(b) 10 eigenimages used

(c) 20 eigenimages used

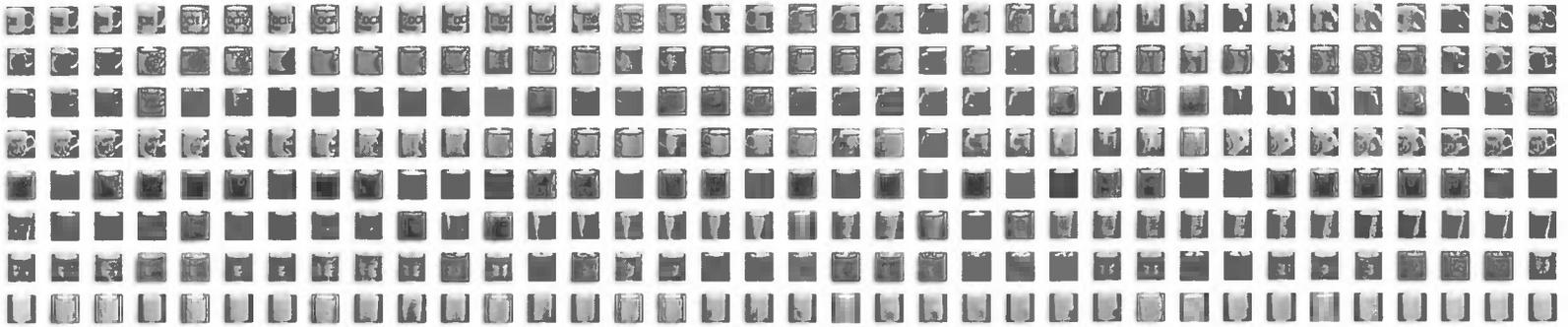
(d) 50 eigenimages used

Figure 2.6 – Object recognition and pose estimation using different dimensions. Dots give the correspondence between the input, on the horizontal axis, and the estimate, on the vertical axis. Pose estimation enhances if we use more eigenimages.

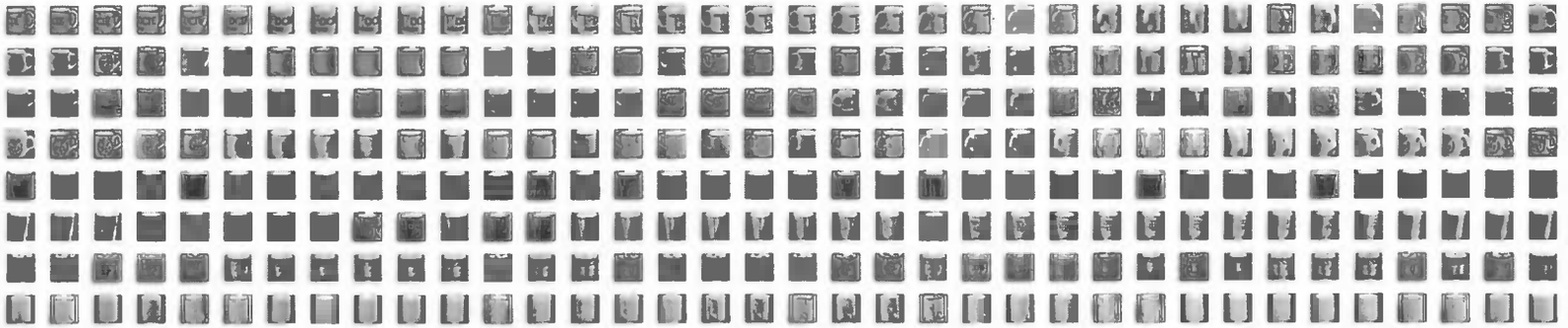
is conducted in an extremely controlled environment. Most studies use a camera observing an object on a turn-table. The object is always in the center of the image, since it is placed on the center of the turn-table, and the background is constant. This greatly simplifies the segmentation process. Lighting is also accurately controlled by a set of lights hanging over the turn-table. The set of objects that are observed, is chosen with care.

In my experiment, we had a less controlled environment, which results in a poorer performance. Lighting was not controlled, the robot is not driving a perfect circle and the background is not black, but noisy. However, this situation is still far from real-world. In the real-world we would not observe one isolated object, but a scene with multiple objects and maybe we would not even observe the object in full. Furthermore, we should be able to recognize all kinds of objects, not only the objects chosen by the experimenter. This brings us to some of the points object recognition should be able to deal with and which are a problem for appearance-based methods.

The presence of a distracting background is called clutter. Clutter is a



(a) Images of 8 objects in 36 orientations used for training, (size 40 x 40 pixels)



(b) Images of 8 objects in 36 orientations used for testing, (size 40 x 40 pixels)

Figure 2.7 – Training and test images used for PCA

problem for appearance-based object recognition since segmenting the object from the background becomes difficult. In the example above, segmentation was relatively easy, since the only edges that are detected, are transitions from background to object, or texture on the object. In more complicated scenes, where the background contains other objects, edges no longer mark the difference between object and background, which leads to an interesting problem. When the segmentation of the object is performed correctly, recognizing the object is straightforward. And similarly, when we know which object we are looking at, we can localize the object in the image and perform the segmentation. However, if we do not know either, the problem is which of these two to start with.

One solution is to look at all the possible locations in the image, where an object can be. There is, however, a large number of locations to take into account. For instance, let the size of the observed image be 320×240 pixels, using a window of 40×40 pixels. Since nothing is know about the location of the object, we start by looking at the 40×40 pixels in the upper-left corner and try to recognize an object. Then, we shift this window one pixel to the right, and perform the same recognition. In the first row, we can look at $320 - 40 + 1 = 281$ different locations in the horizontal direction. Of course, we can also shift the window one pixel down. This would lead to $240 - 40 + 1 = 201$ different possibilities in the vertical direction. It is clear, that it will be unfeasible to visit all the locations on the image and perform recognition in real-time.

Recognition gets even more complicated if the object does not exactly fit exactly within the 40×40 window. In other words, we should also be checking for the object in different scales, which leads to a further increase in possibilities.

Occlusion is another problem frequently encountered in real-world scenes. Occlusion occurs when we observe only part of the object, since some of it is blocked from view. Part of the object could for instance be blocked from view by another object, e.g. a plate can't be completely observed if there is food on it. Another example would be where part of the object, for instance an elbow, is blocked from view by the object itself, another part of the human body. PCA does not work well in the presence of occlusion, since the object representation is based on complete objects. If a part of the object is hidden separating the object from the background becomes complicated. Also, a part of the window contains pixels that do not belong to the object, but the complete window is still used to reconstruct the appearance of the object with eigenimages. If the occluded part covers a large portion of the object, or if the appearance is altered a lot, then the coefficients that we use to sum the different eigenimages can be very different from the coefficients of the object in the database.

Leonardis and Bischof (2000) propose an extension to PCA to more robustly handle images with clutter and occlusion. The basic idea is to estimate the coefficients that project the data on the eigenimages more robustly, by allowing for error. Instead of using all the pixels available in the some 40×40 window, they sample a subset from these pixels. By sampling a number of times the coefficients can be determined more robustly. However, this method is computationally expensive; it takes about 10 minutes to recognize an object in an image. Although the object could be in different

locations in the image with some amount of clutter and occlusion present, the method is not robust to differences in scale.

Another closely related problem concerning PCA is the class of objects it can recognize. The objects in the example all more or less fill the image window. Each pixel in the window is relevant for the object. However, not all objects that we see in the world, have this property. Not all objects are square, and some objects make up for only a small percentage of the image. For example if we use a square window, a garden hose or a ladder contains more background than object.

If you want to successfully recognize objects using PCA, every possible appearance of the object should be learned. Evidently, it is not a problem that we need to obtain different viewpoints from an object to robustly recognize it. However, if we have seen an object before, we are likely to be able to recognize it, if it is turned upside-down. PCA is not able to do this, or in other words, it is not invariant to rotation. If an object in the image is rotated, PCA will most likely not recognize the object, if the rotated appearance of the object has not been learned.

Note that the last two problems are not encountered in face recognition, the domain where PCA has been successful (Turk & Pentland, 1991). Most faces fit within a box and make up most of the area. And in normal circumstances we do not encounter a face that is turned upside-down.

As we have seen above, PCA has three important disadvantages. First of all, the object representation is global, which causes problems when the object is not fully present in the image (due to occlusion), or when the object does not fill the complete window (e.g. due to clutter or non-square objects). Secondly, transformations such as rotation or changes in scale are not easily incorporated in PCA. The last drawback is that the methods that try to overcome these problems require too much computations to make the system run in real-time.

In the next chapter I will focus on feature-based methods, and especially SIFT. This method is expected to overcome most of the difficulties by representing views of an object as a collection of features or keypoints. To correctly recognize an object, only a subset of the available features have to be present in the image, thus being able to cope with occlusion and with clutter. Furthermore, the method is fast and invariant to changes in rotation and scale.

Chapter 3

Features-based methods and SIFT

A different method to recognize objects uses features. Feature-based object recognition methods extract multiple low-level characteristics from an image. A characteristic can be almost anything and examples range from the shape of an object to its texture. A special class of features contains a local description of a neighbourhood around interest points or keypoints, such as corners or edges. For instance, the curvature of a line, a color histogram or the change in light intensity can be used to describe these keypoints.

Extracting keypoints from an image consists of two steps; detecting interest points and describing the region around these points. In principle, any possible descriptor can be used with each of the different detectors. However, some combinations are more successful than others. For instance, it would be unwise to detect straight lines and use the curvature of the line as a feature.

The fact that we represent an object as a set of multiple features has some benefits as opposed to representing the total object, as is the case in appearance-based object recognition.

First of all, feature-based object recognition is capable of dealing with occlusion. Occlusion of the object occurs, when the object is not observed in total. If an object is represented as a combination of multiple independent features, not all features have to be detected. Detection of only a subset of the object features can still lead to a correct recognition of the object, if an adequate amount of features remain.

Performance in cluttered scenes is another advantage of feature-based object recognition over appearance-based object-recognition. Recall that clutter is the presence of a distracting background, which causes problems in the segmentation of the object from the background. A scene with clutter results in the detection of a lot of features, both on the object and in the background. The object features are still present and add to the hypothesis that we see this object. Features in the background can either be generated by other objects in the database or by unknown objects. Features from unknown objects should cause little problems, since these do not provide any evidence for any of the objects. Known objects in the background can alter the object estimation. However, this is not really a problem, since these objects are indeed present in the scene. One could for instance allow for multiple objects to be present, or focus attention on a specific location

in the image.

A third advantage is the fact that feature-based object recognition is not restricted to objects of a certain size. We saw for instance that PCA has problems recognizing rectangular objects, if we use a square window; a lot of information in the window is not used. Feature-based methods can describe a whole range of shapes. However, the performance of different feature detectors still depends on the kinds of objects one uses. Of course, to recognize an object, it must contain features that can be extracted using the chosen feature detector.

Feature detectors and descriptors are only useful if the detected features are robust; i.e. they should still be detected, when circumstances differ slightly. Common characteristics that a feature detector is expected to have, are invariance to small changes in viewpoint and to changes in lighting. However, in order to be useful in a broad range of circumstances additional requirements are frequently made. For instance, the feature detector should also be invariant to rotation and invariant to scale. This means that if we look at the same object turned upside-down from a different distance, we should still be able to recognize the object.

Nelson and Selinger (1998) describe a method using the curvature of edges as features. Curves represent the shape of an object from a certain viewpoint and this method could be successful, when aiming at recognition of object classes. However, complete curves can be quite large and detection of these curves becomes problematic in the presence of occlusion.

An overview of smaller features, typically extracted from a small region around an interest point is given in Mikolajczyk and Schmid (2005). They compare different feature detectors, such as the Harris detector, and their corresponding descriptors, such as PCA. Repeatability after different transformations, such as changes in scale or illumination and image rotation or image blur is used as a measure of performance.

Mikolajczyk and Schmid (2005) conclude that Gradient Location Orientation Histogram (GLOH), which is an extension of Scale Invariant Feature Transform (SIFT), obtains the highest repeatability in most cases, closely followed by SIFT. I chose to use SIFT over GLOH, since the description region used in SIFT is smaller. SIFT uses a 16×16 square region to determine the feature description, whereas GLOH uses a circle with a radius of 15 pixels. So, the description region GLOH uses is about twice as large as the region used for SIFT. Since the images I work with are small, it seems inappropriate to use a large region for the descriptor.

In the following sections I will use the Scale Invariant Feature Transform (SIFT) method introduced by Lowe (1999) to determine which features to use to represent the objects. SIFT consists of two stages. In the first stage, locations of interesting points in the image are detected. These interesting points are detected using Difference-of-Gaussian images, which results in locations with large changes in intensity, such as corners or bright blobs in a dark surrounding. In the second stage, a description of the neighbourhood around these locations of interest is formed using histograms of the image gradient. Both the SIFT detector and SIFT descriptor are explained below.

3.1 SIFT Keypoint detection

The goal of keypoint detection is to find locations in an image, which can be repeatably found under different transformations of the image. For object recognition we are interested in locations on an object, which can be used to robustly represent the object. Since we want to recognize objects under different circumstances, such as different lighting, these changes should not lead to the detection of different locations. Other changes of circumstances that should result in detection of the same locations include rotations of the object in any way, or scaling of the object. The SIFT detector, also called Difference-of-Gaussian (DoG) detector, detects locations on the object which are invariant to the aforementioned type of transformations.

3.1.1 Laplacian filtering

The SIFT detector can be applied to intensity images and it detects points in the image where a large change in intensity occurs. The largest changes occur at the boundary of light and dark areas, such as an edge or corner between to areas. These abrupt changes are of interest, since the location can be determined more precisely than the location of more gradual changes in intensity. Furthermore, large changes can be detected under a broader range of lighting conditions. Although the differences in intensity might be smaller in a dark environment, relative to gradual changes, the differences are still large.

In mathematical terms, large changes in intensity occur at locations in the image where the second-order derivative of the image is large or small. I.e. the locations that are detected by SIFT are the maxima and minima in the second-order derivative of the image. For continuous functions, f, the Laplacian is used as a measure of the second-order spatial derivative of a function, f, given by

$$L_I(x,y) = \frac{\partial^2 I}{\partial x^2} + \frac{\partial^2 I}{\partial y^2}.$$

Images are not continuous however, so we have to approximate the second-order derivative, and thus the amount of change in intensity. This can be done by convolving the image with a discrete Laplacian filter. Three examples of a discrete approximation to the Laplacian filter are

0	1	0
1	-4	1
0	1	0

,

1	1	1
1	-8	1
1	1	1

and

-1	2	-1
2	-4	2
-1	2	-1

.

These three examples have a large negative value in the center and some small values for the other elements, where the sum of all elements equals zero.

Abrupt changes in intensity are strengthened by applying a Laplacian convolution, as can be seen in figure 3.1. In this figure, the intensities of three simple images are shown in the left column, with the resulting image after applying a Laplacian convolution in the right column. For the Laplacian approximation I used the filter in the center above.

The first image consists of an area of uniform intensity. Since there are no changes in values between neighbouring pixels, the first-order derivative contains only zeros. The second-order derivative, approximated by the

1	1	1	1	1	1	1
1	1	1	1	1	1	1
1	1	1	1	1	1	1
1	1	1	1	1	1	1
1	1	1	1	1	1	1
1	1	1	1	1	1	1
1	1	1	1	1	1	1

(a) No intensity change

0	0	0	0	0	0	0
0	0	0	0	0	0	0
0	0	0	0	0	0	0
0	0	0	0	0	0	0
0	0	0	0	0	0	0
0	0	0	0	0	0	0
0	0	0	0	0	0	0

(b) No intensity change

1	2	3	4	5	6	7
1	2	3	4	5	6	7
1	2	3	4	5	6	7
1	2	3	4	5	6	7
1	2	3	4	5	6	7
1	2	3	4	5	6	7
1	2	3	4	5	6	7

(c) Gradual change

0	0	0	0	0	0	0
0	0	0	0	0	0	0
0	0	0	0	0	0	0
0	0	0	0	0	0	0
0	0	0	0	0	0	0
0	0	0	0	0	0	0
0	0	0	0	0	0	0

(d) Gradual change

0	0	0	0	0	0	0
0	0	0	0	0	0	0
0	0	1	1	1	0	0
0	0	1	1	1	0	0
0	0	1	1	1	0	0
0	0	0	0	0	0	0
0	0	0	0	0	0	0

(e) Square

0	0	0	0	0	0	0
0	1	2	3	2	1	0
0	2	-5	-3	-5	2	0
0	3	-3	0	-3	3	0
0	2	-5	-3	-5	2	0
0	1	2	3	2	1	0
0	0	0	0	0	0	0

(f) Square

Figure 3.1 – Laplacian filtering. In the left column the intensity values of three images are shown. The column on the right shows the same image after a Laplacian filter has been applied. In the top two images, without intensity change and with a gradual intensity change, the Laplacian filter does not detect any abrupt changes. In the image with the square, the Laplacian filter does detect abrupt changes, the local extrema -5 and 0.

Laplacian convolution, also contains only zeros, which can be seen in the second image in the first row.

The image on the second row shows a gradual change in intensity in the horizontal direction. In the horizontal direction the change from pixel to pixel, the first-order derivative, is in this example larger than zero. However, since the change in intensity is constant, the second-order derivative, or Laplacian, is zero. This shows that gradual changes in intensity are not noticed by the Laplacian convolution.

The third image is interesting, since the Laplacian does not contain the value zero at all locations. The image contains a square, which exhibits an abrupt change in intensity. Two things are important to note.

First of all, the corners of the square correspond to a local minimum in the Laplacian of the image. If we would observe a square of zeroes on a background with ones, these corner-values would be positive, and we would observe local maxima.

The other important thing is that the center of the square corresponds to a zero in the Laplacian of the image. Although a zero might not seem interesting, the point is a local maximum. However, if we would observe

the same square at a different scale - for instance a 5×5 pixels square - the center of the square would not result in a maximum in the Laplacian of the image. In that case, the center of the square corresponds to a region with no change in intensity and therefore no response in the Laplacian. So, intuitively, the Laplacian operator is not invariant to scale. Imagine that we have an image with a white area on the left side, black area on the right and in the middle a gradual change between the two. If we scale this image to a smaller size, the gradual change might disappear into an abrupt change.

3.1.2 Scale invariance

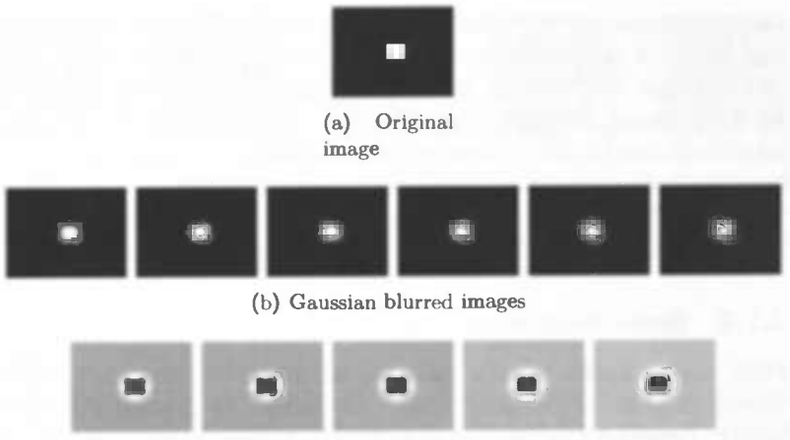
From the examples we saw that Laplacian convolution responds to large abrupt changes in intensity, where no changes or gradual changes are ignored. However, the detection of abrupt changes is not invariant to scale. Furthermore, the Laplacian filter is sensitive to noise in the image. Because of the small size of the filter, only direct neighbours of a pixel are taken into account. This could lead to the detection of local maxima or minima in the Laplacian image due to random errors. Therefore, instead of directly applying a Laplacian convolution to an image, usually the image is first blurred by applying a Gaussian convolution. Blurring the image by Gaussian convolution leads to more gradual changes in the intensities. The benefit is that the error is less observable, which comes at the cost of having a smaller amount of abrupt changes.

There is another benefit of applying a Gaussian convolution. This not only makes the result less sensitive to noise, but also gives us the opportunity to make the detector invariant to scale (Lowe, 2004). Detection of locations is invariant to scale, if the same points are detected in images from an object, taken from different distances. Instead of obtaining multiple images of the same object at a different scale, the difference in scale can also be simulated by blurring the image. Blurring an image can be accomplished by convolving the image two times with the one-dimensional Gaussian function, $g(x)$,

$$g(x) = \frac{1}{\sqrt{2\pi}\sigma} e^{-x^2/2\sigma^2},$$

once in horizontal direction, and once in vertical direction. In the middle row of figure 3.2 the result of Gaussian blur applied to a simple image can be seen. For each of the six images a different value for the smoothing parameter, σ, was chosen and thus these images represent the object from a certain viewpoint at different scales.

In order to determine at which location in the image large changes in intensity occur, we have to obtain the second-order derivative. As mentioned before, this could be accomplished by computing the Laplacian of each of the Gaussian blurred images. However, Lowe (1999) notes that the Laplacian-of-Gaussian can be approximated by a Difference-of-Gaussian (DoG). In figure 3.3, two Gaussian functions are plotted and by calculating the difference between these two functions we get the one-dimensional DoG in the middle. A two-dimensional DoG is shown on the right. We can easily see that these look like the discrete approximations of the Laplacian given above; the DoG has one large negative value in the center, with some small positive values around it.



(a) Original image

(b) Gaussian blurred images

(c) Difference-of-Gaussians

Figure 3.2 – Detecting keypoints using DoG. The six images in figure 3.2(b) are the Gaussian blurred versions of the image in figure 3.2(a). Each of the images use a different values for the smoothing parameter σ. In the lower row, we see the Difference of Gaussians, which are an approximation for the second-order derivative of the image. These images are the result of calculating the difference between two blurred images in the row above.

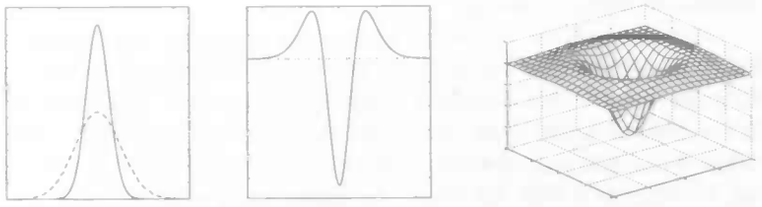
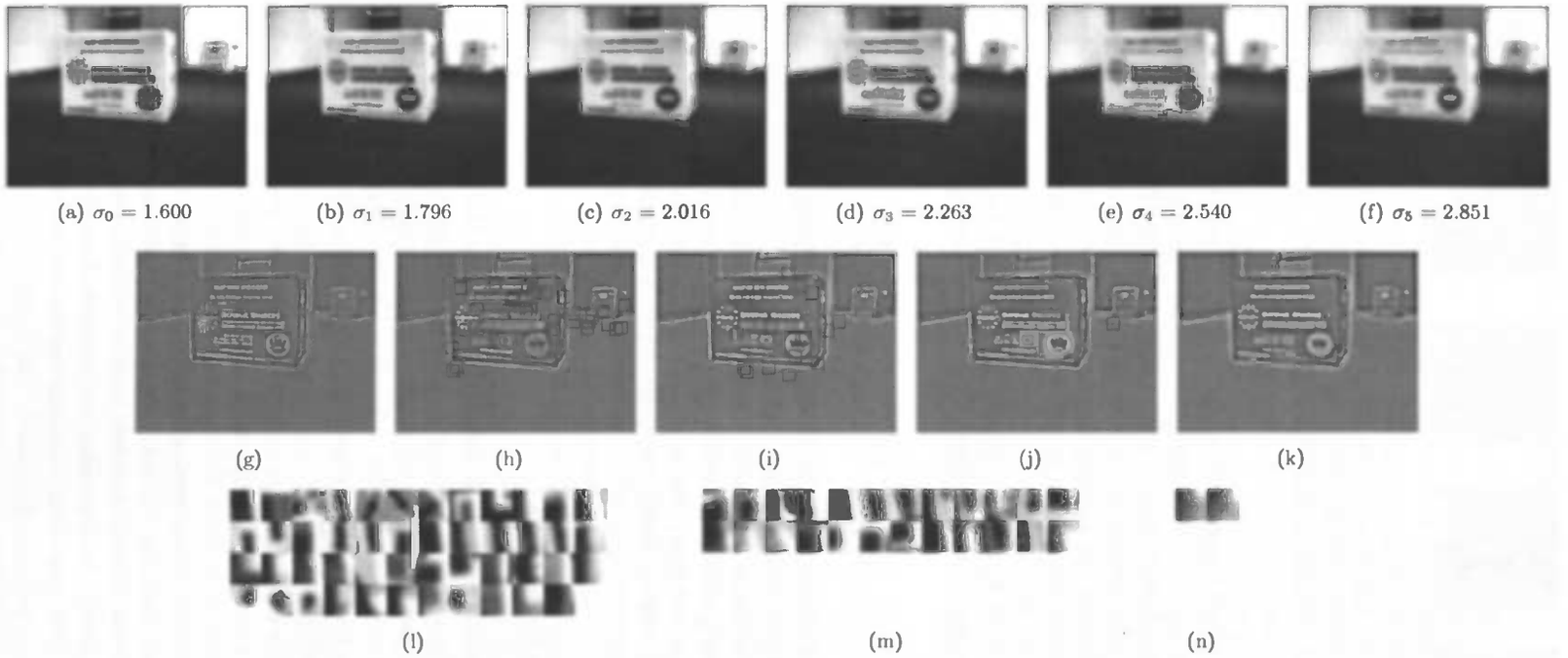


Figure 3.3 – Difference of Gaussians. In the left figure two Gaussian curves with a different value for σ are shown. The figure in the middle show the result if we subtract one of the Gaussians from the other. In the right figure the same result is shown for a two-dimensional difference of Gaussians, the so-called Mexican hat.

If we have an image blurred using two different values of the smoothing parameter, σ, then computing the DoG is equivalent to simply subtracting two Gaussian blurred versions. Evidently, this is a considerably faster operation than performing a convolution using a Laplace filter. This is of course only results in a total improvement in speed if we needed to calculate the Gaussian blurred versions in the first place, since to obtain these we need to calculate a convolution as well. Because we want to detect interesting points at different scales, we need to calculate multiple Gaussian blurred versions of the original image, and therefore using DoG instead of a Laplacian filter is useful in our case.

In order to make the descriptor invariant to large changes in scale, we have to use a wide range of values for the smoothing parameter, σ. Calculating a convolution is a computationally intensive procedure, especially for the large Gaussian filters corresponding to a large value of σ. We start with



(a) $\sigma_0 = 1.600$ (b) $\sigma_1 = 1.796$ (c) $\sigma_2 = 2.016$ (d) $\sigma_3 = 2.263$ (e) $\sigma_4 = 2.540$ (f) $\sigma_5 = 2.851$

(g) (h) (i) (j) (k)

(l) (m) (n)

Figure 3.4 – Detecting keypoints using SIFT. The top six images are Gaussian smoothed images used for detecting keypoints in one octave, with different values for the smoothing parameter σ. In the second row, we see the Difference of Gaussians, which are an approximation for the second-order derivative of the image. The black squares represent local extrema, i.e. points with large changes in contrast. In the lower row the 16×16 rotated regions are given, which are used for calculating the descriptor.

Figure 3.5 – Local extremum in DoG. The five DoG images are enlarged versions of the region around one of the keypoints detected in Figure 3.4. The maximum, marked with the white dot, is the maximum of its 27 neighbours in location and scale space, displayed by white squares.

an initial value of the smoothing parameter σ_0 and increase this value in steps to calculate the subsequent scales. If the value of σ is twice the value of σ_0, instead of calculating a convolution with the filter twice the original size, we can also rescale the image to half its original size. In that case, the convolution on the rescaled image can be calculated using the original smaller Gaussian filter with parameter value σ_0, thus resulting in a considerable speed enhancement. Lowe (2004) uses sequences of six smoothed images calculated from each rescaled image, which he calls an octave. The values of the smoothing parameter in a sequence are given by

$$\sigma_i = \sigma_0 2^{i/6}, \quad i = 0, \ldots, 5.$$

The result of subtracting the Gaussian blurred images in one octave can be seen in figure 3.4(c). Gray points represent points with little change in intensity between the blurred images, whereas black and white points represent large positive or negative changes. In figure 3.4, again six Gaussian blurred images and five Difference-of-Gaussians are shown, derived from one of the images that is used in the final object recognition task.

3.1.3 Local extrema

The points of interest are the local extrema in the DoG images, considering location and scale. I.e. a pixel value should be larger than its 8 neighbouring pixels in the same image, and larger than the neighbouring pixels in the images with a one-step smaller or larger scale (see figure 3.5). The first and last image in the sequence cannot contain extrema, since these images have only one neighbour in scale.

In order for these extrema to be a keypoint, two more conditions have to be satisfied. First, the absolute value of the DoG pixel should be larger than some threshold, θ, in order to exclude extrema with low contrast. Second, the DoG function shows some extrema along edges. Typically, changes in location along the edge, do not alter the descriptor and result in unstable keypoints. Lowe (2004) therefore uses the Harris corner detector (Harris & Stephens, 1988), to determine the amount of cornerness of the keypoint candidates. If the extremum responds too little on the Harris measure of cornerness, the keypoint is eliminated. The locations of the remaining extrema are stored and used when describing the region around it.

3.2 SIFT Keypoint description

In order to retrieve the same keypoint in different images, a description of the region around the locations detected in the previous part is needed. In short, the SIFT descriptor is a histogram of the orientation of the gradient in a region around the point of interest, which will be made more clear below. Calculating the descriptor consists of two stages. In order to make the descriptor rotation invariant, the region used to make the descriptor is rotated in the direction of the overall gradient orientation of the region. Subsequently, the descriptor is calculated from the rotated region.

To make a description of the region around the interest point we could use for instance the intensity values of the pixels. These values might however not be as informative as possible, given the detector that we use. Interest points are detected based on the abrupt changes in intensity. Intuitively, it is therefore more natural to describe the changes in intensity as opposed to the intensity itself. The SIFT descriptor uses the gradient of an image, which is in fact an approximation to the first-order derivative, a measure for the change in intensity.

Two aspects of the gradient are important, the gradient magnitude and the gradient orientation. The gradient magnitude of image, L_i in location (x, y) is given by

$$M_{L_i}(x,y) = \sqrt{(L_i(x+1,y) - L_i(x-1,y))^2 + (L_i(x,y+1) - L_i(x,y-1))^2}.$$

The gradient magnitude is a measure of the strength of the change in intensity. Large changes in intensity give a strong response on the gradient magnitude and this measure is therefore often used to detect edges in an image. A region with large contrast contains large changes in intensity, which can be seen as the edge between two regions.

The gradient orientation is an estimate of the direction of the edge, if the pixel was part of an edge. The direction of the edge is measured in radians. A horizontal line would result in an estimate of $\frac{1}{2}\pi$ rad or $-\frac{1}{2}\pi$ rad and a vertical line would result in 0 rad or π rad. The gradient orientation is

$$G_{L_i}(x,y) = \arctan\left(\frac{L_i(x,y+1) - L_i(x,y-1)}{L_i(x+1,y) - L_i(x-1,y)}\right).$$

Note that both the gradient magnitude and gradient orientation are calculated from L_i. This is the Gaussian blurred image, which has the same scale as the detected keypoint.

To see why this results in an estimate of the direction of an edge, imagine that we have an image with a vertical line. For simplicity, write

$$G_L = \arctan\left(\frac{\Delta y}{\Delta x}\right),$$

where Δx is the change in intensity in the x direction and Δy is the change in the y direction. Along the vertical line in the image the intensity doesn't change, so Δy is zero. In the horizontal direction there is a change when we arrive at the line, which gives a non-zero change, Δx. The resulting estimate of the direction of the line in this case is

$$G_L = \arctan\left(\frac{\Delta y}{\Delta x}\right) = \arctan\left(\frac{0}{\Delta x}\right) = \arctan(0) = 0.$$

So a vertical line is considered as having a direction of 0°. Analogously, if we look at a horizontal line we have

$$G_L = \arctan\left(\frac{\Delta y}{\Delta x}\right) = \arctan\left(\frac{\Delta y}{0}\right) = \arctan(\infty) = \frac{\pi}{2}.$$

This means that a horizontal line is rotated 90° with respect to a vertical line, which seems obvious. Of course, the direction of any other line can be estimated using this method.

3.2.1 Rotation invariance

The regions around the interest points, that are detected in the first step, are used to make the descriptor. If the image of the object we are observing is rotated, the same location is detected as interest point. The region around this point however is rotated. In order to make the descriptor invariant to rotations, the region around the interest point is rotated in the direction of the overall gradient orientation of the region.

The overall orientation of the keypoint is determined from the blurred image, $L_i, i = 0, \ldots, 5$, with the same scale as where the local extremum is detected. In a region around the keypoint location, a histogram with 36 bins covering the 360 degrees is constructed from the gradient orientations. Each sample that is added to the histogram is weighted by its gradient magnitude and by a Gaussian circular window.

Figure 3.6 shows in detail how the overall orientation of a keypoint is determined. In figure 3.6(a) the intensity values of the 16×16 pixels around an interest point are given. From this region, the gradient magnitude and the gradient orientation are calculated, which is shown in figure 3.6(b) and figure 3.6(d) respectively.

The gradient orientation of this region shows the estimated direction of the pixels in the intensity image. A line in the horizontal direction is represented by white pixels, whereas gray pixels correspond with vertical lines. To calculate the overall orientation of the region we want to calculate a weighted average of these orientations.

To calculate weights, we use the magnitude image, given in figure 3.6(b). In this image, white pixels correspond to high values for the magnitude of the gradient. We can see that the changes in intensity are the strongest near the edge of the region. In the lower right corner no changes in intensity takes place, resulting in small values for the gradient magnitude and black pixels. Subsequently, the gradient magnitude is multiplied by a two-dimensional Gaussian window, with σ is 1.5 times that of the scale of the keypoint. The result of this multiplication can be seen in figure 3.6(c), and these values are the weights that are used to construct the histogram.

In figure 3.6(e) we can see the histogram of the overall orientation. The width of each bin of the histogram covers 10 degrees, from $-180°$ to 180°. For each pixel in the gradient orientation, we determine in which bin it fits and we add the weight of this pixel to this bin. For the example in the figure that gives us two strong peaks in the histogram, around 0° and 90°.

The exact location of the peak of the orientation histogram is determined by fitting a parabola through the bin with the maximal value and its two neighbours. This peak corresponds to the dominant direction in the gradient orientation. In order to make the process more robust for noise, other peaks

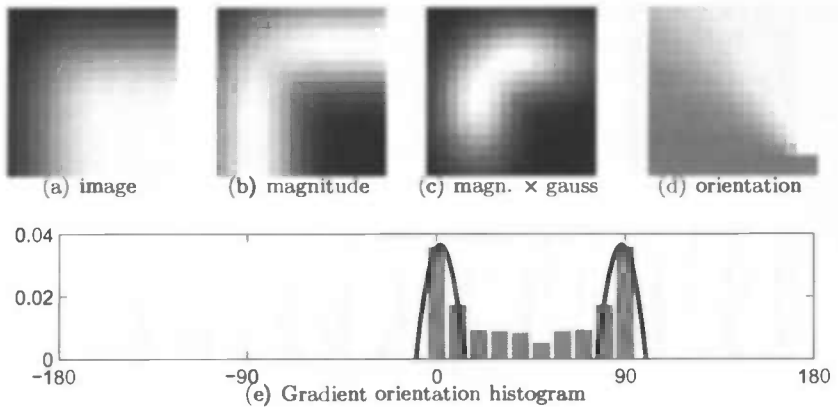


Figure 3.6 – Rotation invariance. In figure 3.6(a) we see the intensity image of a 16 × 16 region around an interest point. The gradient magnitude in figure 3.6(b) and a Gaussian filter are multiplied to obtain the weights in figure 3.6(c). The weights are combined with the gradient orientation in figure 3.6(d) into an orientation histograms with 36 bins. The result is the overall gradient orientation histogram displayed in the lower row.

in the orientation histogram with a similar value, e.g. more than 80% of the maximum, are also considered. In this way, one location can result in multiple keypoints, each with a different orientation. For instance, the keypoint in figure 3.6 has two orientations, one in the horizontal direction and one in the vertical direction.

The dominant direction is then used to rotate the region around keypoint location. A square of 16 × 16 pixels taken from the rotated image around the keypoint location is used for calculating the descriptor. Some examples of rotated keypoints can be seen in figure 3.4(l). For instance, the left three keypoints on the lower row are all rotated versions of the region around the same location.

3.2.2 SIFT Descriptor

Calculating the SIFT descriptor is analogous to calculating the overall orientation. In figure 3.7 the ingredients for calculating the descriptor are shown. Again, we have the gradient magnitude and a Gaussian filter, which after multiplication results in the weights shown in figure 3.7(c). Also, the gradient orientation is given. Note that the example in the figure is a SIFT keypoint detected in one of the images in my dataset. Therefore, it looks somewhat more complicated than earlier examples.

The rotated regions around the detected interest point are divided in 16 segments of 4 × 4 pixels, as can be seen in the figure. For each of these segments an orientation histogram is constructed from the gradient orientation, where the additions to the bins are weighted by the multiplication of the gradient magnitude and the Gaussian filter, similar to calculating the overall gradient histogram. The histogram can be sensitive to noise if the orientation values are close to the upper or lower boundaries of a histogram bin. Therefore, each value is distributed over two adjacent histogram bins with a factor $1 - d$ for each bin. Here, d is defined as the distance between

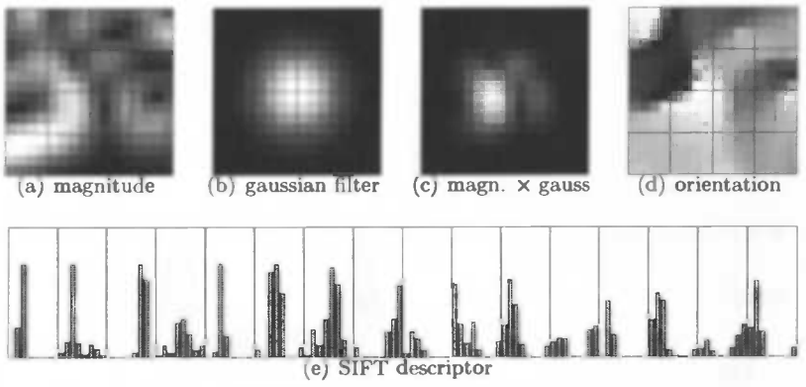


Figure 3.7 – SIFT Descriptor. In the upper row we see the 16 × 16 region around an interest point. The magnitude gradient and the Gaussian filter are multiplied to obtain the weights in Figure 3.7(c). The weights are combined with the orientation gradient into a set of 16 orientation histograms with 8 bins. The result is the SIFT descriptor displayed in the lower row.

the orientation gradient and the central value of the bin.

Lowe (2004) obtains the best results when using 8 bins, so this is also the number of bins that I use. Since each of the 16 segments is summarized in a histogram with 8 bins, we obtain a vector of size 128. Two final operations are applied to obtain the SIFT descriptor, shown in figure 3.7(e). To make the vector more invariant to changes in illumination, the vector is rescaled to a Euclidean length of 1. Also, all values larger than .2 are truncated.

3.3 Generalisation of object classes

By clustering similar keypoints, I also take a first step to generalizing objects to categories. That is, if objects from the same category also generate the same SIFT keypoints.

One of the questions I was interested in at the start of the project, is whether SIFT features can be used for generalisation of object classes. (Dorko & Schmid, 2003) use different clustering techniques to learn object-part classifiers from similar sets of SIFT descriptors. (Dance et al., 2004) try to do the same, while using more object categories. They use faces, buildings, trees, cars, phones, bikes and books. Depending on which class is to be recognized, results differ. For instance faces are correctly classified in most cases, whereas the classification of phones is much worse.

As can be seen in the lower row of figure 3.4, the features that are extracted usually correspond to regions in the image containing texture. However, not all object classes are determined by their correspondence in texture. For instance, the cups in the dataset in chapter 2 correspond in shape, but not in texture. All of the cups are basically a cylinder with an ear on the side. The image on the cup, which defines the texture and which generates the SIFT keypoints, differs from cup to cup. Cars for instance correspond in texture, as well as in shape. Rims on the wheels for instance, or license plates, are textured surfaces present in every car. This would therefore be a good class to use for SIFT classification.

Serre, Wolf, and Poggio (2005) mention that SIFT-based features might perform too good in the re-detection of keypoints under different transformations. Recognition of generic objects requires features which are not too selective, but also are not too invariant. The authors introduce a new biologically inspired feature that clearly outperforms the SIFT descriptor when recognizing object classes.

Chapter 4

Object representation

The SIFT keypoints described in the previous chapter form the building blocks of the object representation. In order to combine the keypoints that we obtain from images into a useful representation of the objects, we need to classify the keypoints we see in an image in object keypoints and keypoints from the background. In this chapter I will describe a method, where active vision is used to do exactly this.

Active vision systems are systems where perception and action are closely linked (Ballard, 1991). Control of camera parameters, such as zoom or viewpoint are important actions to enhance perception. Roy et al. (2004) give an overview of the use of active vision in object recognition. Active vision has been incorporated in appearance-based object recognition methods, where the set-up usually contains a camera looking at a turn-table, in order to simplify object segmentation. The system is allowed to reposition the turn-table leading to an effective change in the viewpoint of the camera with respect to the scene. Borotschnig et al. (2000) use this set-up to recognize objects which are ambiguous from certain viewpoints and conclude that multiple observations lead to a significant increase in recognition rate. This is taken a step further by Paletta and Pinz (2000), who use active vision to autonomously develop strategies in viewpoint planning by reinforcement learning.

Both these examples of active object recognition use appearance-based methods. To my knowledge, similar projects, using feature-based recognition methods, are not available, except for recent research by Brown and Lowe (2005). In their paper the objective is to correctly estimate the pose of an object using active vision. Differences with my project are the fact that I use a robot instead of a turn-table, the environment is complex, with a distracting background, and random objects are chosen. Some of these objects might not be chosen very conveniently in hindsight.

4.1 Active keypoint selection

Our goal is to recognize objects from different viewpoints. To obtain this goal the robot has to observe the object from different viewpoints during learning. Extrapolation beyond the appearances of an object that have not been observed before is not possible. In order to obtain a full representation of the object, the robot circles around this object, storing an image every

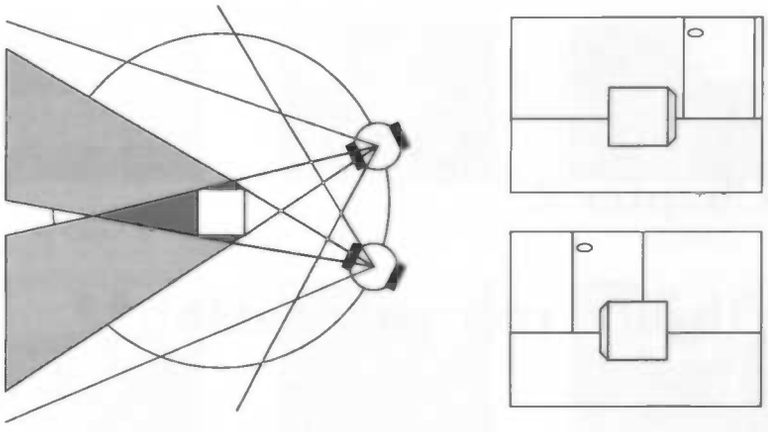


Figure 4.1 – Using active vision for keypoint-classification. In the left part of the figure we see the object learning scene from above. The robot, circling around a box-like object, is shown in two positions, with the corresponding viewing angle. The gray areas are the areas that fall behind the object, from the robots point of view. The two images on the right, show what the robot sees. The background moves fast, whereas the object in the center stays at the same place. The shift in location of keypoints is used to determine, whether the keypoint is on the object, or in the background.

10 degrees. The keypoints in every image are linked to the observed object in the correct pose.

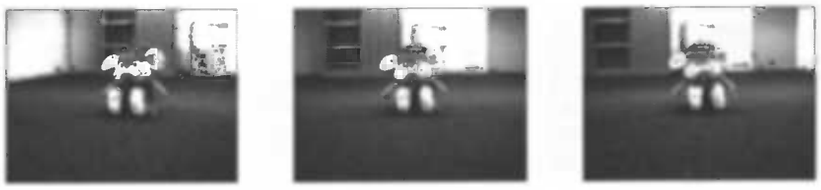
However, not all of the keypoints we observe in the image, are keypoints that belong to the object. Separating the keypoints corresponding to the object of interest and the keypoints corresponding to the background, can be solved using active vision. In the left part of figure 4.1 we see, from above, the robot observing a box. The large circle is the trajectory the robot is following, when completing the full circle around the box. The robot is given in two different position, with the corresponding viewing angle of the camera given. The gray areas mark the area that cannot be seen by the robot, because the box is blocking the view.

In the right part of the figure, two images are shown, corresponding to what the robot sees in the two positions. Since the robot circles around the box, the box stays more ore less in the center of the image. When driving clockwise around the box, objects in the background move from right to left, and objects close to the robot move in the opposite direction. For instance, the door in the background of the image shifts from right of the box, to a position behind the box. Assuming that, during learning, the robot observes one object at the time, we can use this to separate object keypoints from background keypoints.

The algorithm to obtain keypoints belonging to the object is simple. First, for each keypoint i in image 1, $k_{1,i}$, we find the keypoint in image 2, which is closest, measured using the Euclidean distance, $k_{2,j}$. Keypoint $k_{1,i}$ then matches to keypoint $k_{2,j}$, if the distance is smaller than some threshold α

$$|k_{1,i} - k_{2,j}| < \alpha.$$

Subsequently the matching keypoints are classified as keypoints belonging



(a) (b) (c)

Figure 4.2 – Background segmentation. Given are three pictures from the same object taken from slightly different viewpoints. By looking at the distance between two descriptors in subsequent images and the shift in location, the keypoints are classified in three groups. The red points are keypoints with no match in the previous or next image. The blue points represent keypoints from the background, because of a large change in location between the two images. The green points are keypoints generated by the object.

to the object if the locational shift in horizontal and vertical direction is not too large

$$|x_i - x_j| < \delta_x,$$

and

$$|y_i - y_j| < \delta_y.$$

For each image matching keypoints are based on the previous and next image in the sequence.

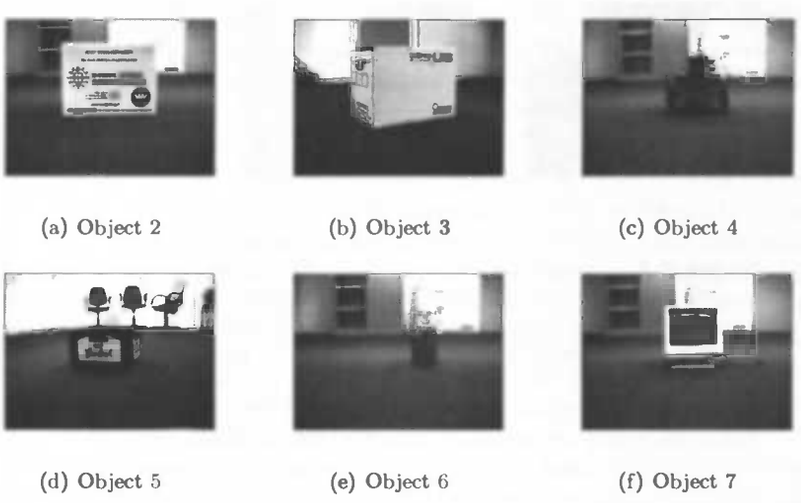
4.2 Classification results

In figure 4.2 we can see the result of the classification process. Red points correspond to keypoints without a matching keypoint with distance smaller than α in the previous or next image. Blue points represent keypoints on the background and green keypoints represent keypoints generated by the object in the center of the circle.

For the parameters I used values $\alpha = .6$, $\delta_x = 12$ and $\delta_y = 4$. Calculations using the viewing angle of the camera and the distance between the robot and the object gave an indication of which values to choose. With these parameter values I obtained results, where only a handful of keypoints in the background are falsely classified as belonging to the object. These keypoints belong to repeating structures in the background. Slightly altering the value of α does not result in substantial changes.

Note that the same principle of classifying keypoints easily translates to different actions. Although the model of movement might be a bit more complex in this case, keypoints obtained from rotating an object in a mechanical hand can be classified analogously. Another measure of movement that could be used is the fact that the overall orientation of keypoints on the background should remain the same under the chosen transformation, whereas keypoints on the object rotate, and thus their overall orientation changes.

In figure 4.3, images are given of the other 6 objects that are used for object recognition. Also, some samples of the background can be seen in these images. The objects that I use, were not chosen with the SIFT descriptor



(a) Object 2 (b) Object 3 (c) Object 4

(d) Object 5 (e) Object 6 (f) Object 7

Figure 4.3 – Objects $2, \ldots, 6$ and object 1 (figure 4.2) are used for object recognition. The background contains real-world objects such as chairs and boxes. Readily available objects were chosen, large enough to fill some of the image. No effort was made to choose objects extra suitable for SIFT recognition.

in mind and some objects might not be very suitable for object recognition using SIFT. For instance, the trash can (object 6) and the computer screen (object 7) contain reflective surfaces.

In table 4.1 and table 4.2 the total number of keypoints detected for each object are given. We see that about 25% of the keypoints that are detected in the images are keypoints generated by the object. This percentage is the same for $\sigma_0 = 1.2$ and $\sigma_0 = 1.6$, although for the former a larger number of keypoints is detected.

Furthermore, the amount of keypoints differs per object. Object 2, one of the boxes, generates more than twice the amount of keypoints than other objects. Object 6 contains the least number of keypoints, 375 and 279. Since each object is observed from 36 different viewpoints, this is around only 10 keypoints per viewpoint in the most positive case. In section 7, I compare the consequences for recognition between using all keypoints and using only the keypoints belonging to an object.

Instead of looking at the total number of keypoints per object, we can also look at the number of keypoints per pose. Figure 4.5 and figure 4.6 give the number of keypoints per pose for each object, for $\sigma_0 = 1.2$ and $\sigma_0 = 1.6$ respectively. We see immediately that the number of keypoints is not constant for each pose. Object 1 for instance, has quite a lot of keypoints when seen from the front (poses 0 to 60 and 320 to 350). Seen from the back, this object does not have a lot of prominent features, and much less keypoints are detected on that side.

Quite remarkable is the fact that object 5 has a low number of keypoints around pose 180. Since the object looks similar from all sides, we would expect the number of keypoints to be roughly constant for all poses. When looking in more detail at the images that were used to construct this particular histogram, I noticed that the object moved substantially within the

Table 4.1 – Numbers of keypoints detected for each object. The total number of detected keypoints are classified in three groups. The keypoint either belongs to the object (2nd column), the background (3rd column) or is not found in two subsequent images (4th column). $\sigma_0 = 1.2$ is used as starting value for the initial smoothing parameter.

Object	Object		Background		Non-matches		Total
1	1000	(27.1 %)	1383	(37.5 %)	1303	(35.4 %)	3686
2	2097	(35.6 %)	2181	(37.0 %)	1614	(27.4 %)	5892
3	1089	(25.9 %)	1753	(41.6 %)	1369	(32.5 %)	4211
4	732	(20.8 %)	1522	(43.2 %)	1273	(36.1 %)	3527
5	821	(21.7 %)	1970	(52.0 %)	997	(26.3 %)	3788
6	375	(13.3 %)	1751	(62.1 %)	695	(24.6 %)	2821
7	692	(21.1 %)	1555	(47.3 %)	1040	(31.6 %)	3287
Total	6806	(25.0 %)	12115	(44.5 %)	8291	(30.5 %)	27212

Table 4.2 – No. of keypoints, $\sigma_0 = 1.6$, see explanation at table 4.1

Object	Object		Background		Non-matches		Total
1	597	(24.2 %)	1109	(44.9 %)	764	(30.9 %)	2470
2	1282	(34.7 %)	1536	(41.6 %)	878	(23.8 %)	3696
3	695	(25.8 %)	1260	(46.9 %)	734	(27.3 %)	2689
4	548	(21.5 %)	1356	(53.3 %)	639	(25.1 %)	2543
5	547	(21.0 %)	1526	(58.5 %)	535	(20.5 %)	2608
6	279	(13.3 %)	1408	(67.0 %)	413	(19.7 %)	2100
7	555	(22.9 %)	1248	(51.5 %)	620	(25.6 %)	2423
Total	4503	(24.3 %)	9443	(51.0 %)	4583	(24.7 %)	18529

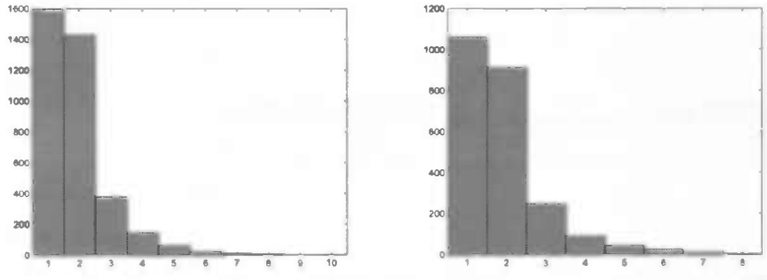


Figure 4.4 – Number of keypoints per object-feature, with $\sigma_0 = 1.2$ (left) and $\sigma_0 = 1.6$ (right). An object-feature can be seen from different viewpoints and thus create multiple keypoints. In this image we see the number of keypoints that generate 2 or more keypoints. The bar at 1 gives the number of keypoints that are rotated versions of other object-features.

image. This could be for instance due to the robot not driving a perfect circle, or due to the object not standing in the center. Probably, when you look at the keypoints generated by images from this object from another run, the number of keypoints per object are more evenly distributed.

Finally, it is interesting to see how persistent features are. In figure 4.4 the number of consecutive poses are shown that the same keypoint is visible. For $\sigma = 1.2$ and $\sigma = 1.6$ the distributions are similar, although the total number of keypoints is larger for the former. We see that most interesting features on the object generate a keypoint in two subsequent images. A good amount of the features remain visible for three or four poses, and some are even visible longer. Note however that this does not mean that the keypoint as seen in the first pose is similar to the keypoint in the fourth pose. They are generated by the same feature, but because of rotation of the object, the resulting keypoints can look different.

All the keypoints that are visible only in one pose, are keypoints that are rotated versions of one of the other keypoints. So, these keypoints are not separate features on the object, but they are associated with one of the features that are visible in two or more poses.

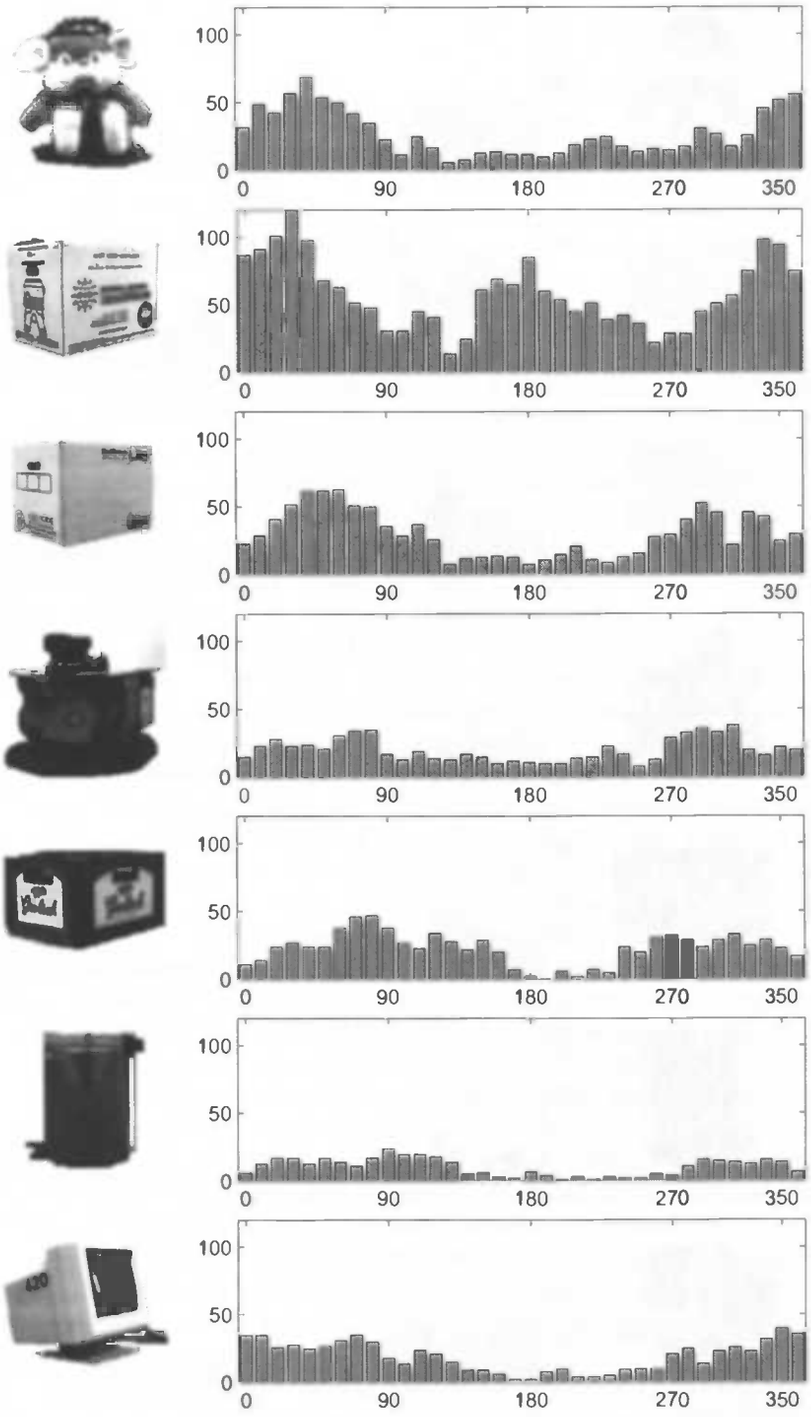


Figure 4.5 – Number of keypoints per viewpoint for each object, with $\sigma_0 = 1.2$.

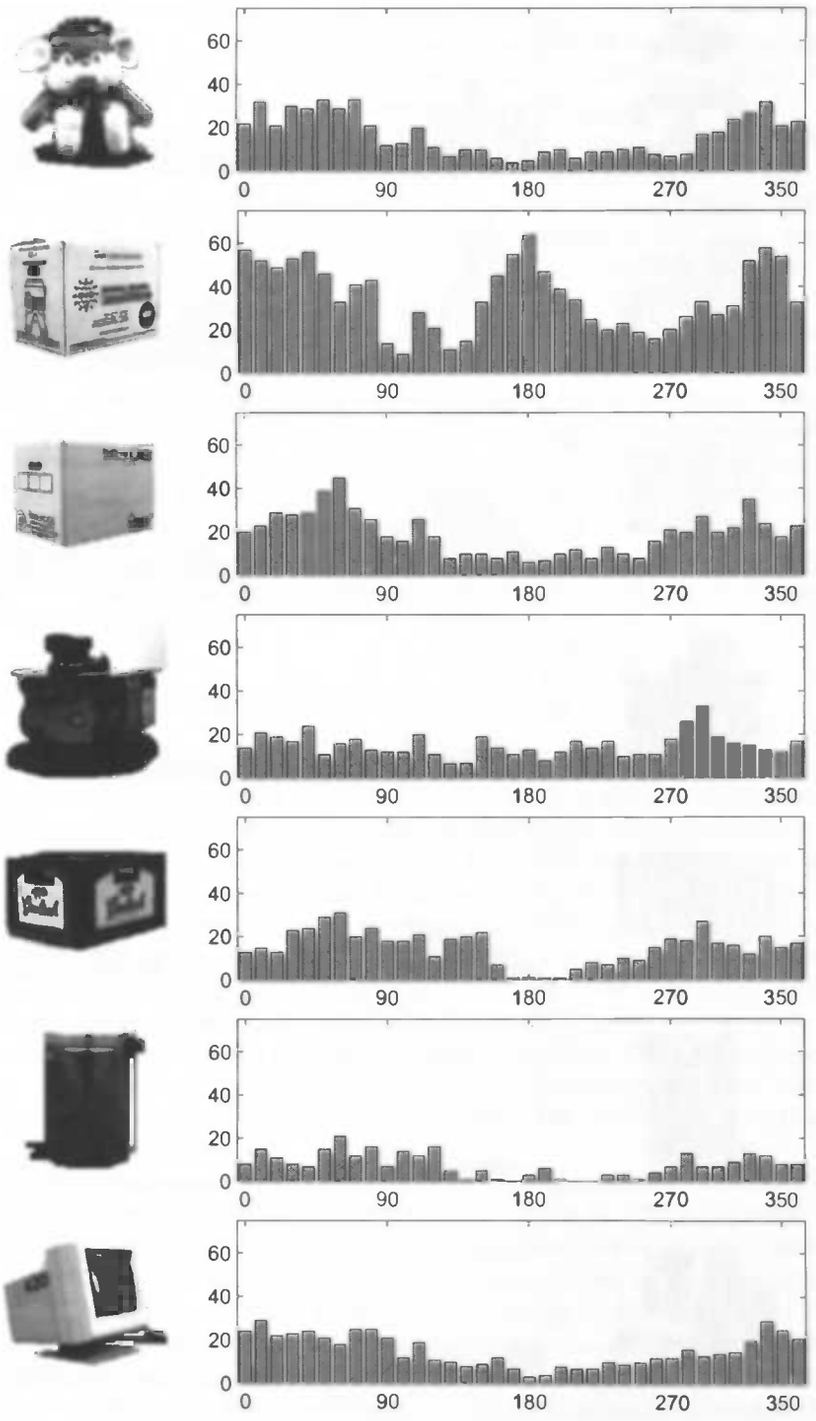


Figure 4.6 – Number of keypoints per viewpoint for each object, with $\sigma_0 = 1.6$.

Chapter 5

GWR Network

In the previous chapters I described the way I retrieve those keypoints from an image that correspond to an object. These keypoints are used to represent each object in certain poses, which results in a large database. Having a large database has some drawbacks, which I explain below. Therefore, I use a neural network, the Growing When Required (GWR) network, to cluster similar keypoints, and thus decrease the size of the database.

The first disadvantage of having a large database is that it takes a lot of time to search through. In the recognition stage the robot will see an image of the object and calculate SIFT keypoints from this image. If an object is present in the image, the SIFT keypoints in the scene will be similar to keypoints in the database associated with this object. Therefore, to estimate which object is present in the scene, we have to calculate the distance between the keypoints in the scene and all of the keypoints in the database. The time it takes to calculate the closest keypoint to the input increases linearly in the number of keypoints in the database. If we just add all keypoints belonging to an object to the database, each time we learn a new object, even after learning a small number of objects, the database will be to large to be able to use for real-time object recognition.

Another problem, related to having many keypoints in the database, is that the probability that two keypoints are similar increases. If we match our input only to the best matching keypoint, mismatches will occur. And, if a lot of keypoints in the database are close to each other, the probability that the input will be matched with the correct keypoint, decreases. One way to solve this problem, is to look at more keypoints than the best matching keypoint. In this way, all the keypoints in the neighbourhood of the input count towards the recognition of the object. However, in that solution there is still an enormous amount of keypoints present in the database.

Lowe uses smart searching, but then the best matching keypoint is only truly the best matching keypoint with prob smaller than 1.

Another way to obtain a fast solution, is to cluster similar keypoints together in one representative node, and associate this node with more than one object or object pose. In that way, the input can be matched only to the closest node, and still be associated with more objects. The other big advantage is that we need less nodes than we would have needed keypoints.

Most neural networks that cluster input are based on self-organizing maps (SOM) (Kohonen, 1990). These networks have the disadvantage that the number of clusters is fixed from the start. Since we do not know the

number of unique keypoints in advance, working with a fixed number of clusters causes problems. If we choose too many clusters, then the network will still be large, and we have to make many useless calculations. If we choose a small amount of clusters, not all the unique keypoints can be represented and valuable information is thrown away. Also, the network wouldn't be able to learn new objects if these contain different keypoints.

The Growing Neural Gas (GNG) is a neural network, based on SOM, where the number of nodes is not fixed in advance. The network is capable of increasing the number of clusters during the learning process (Fritzke, 1995). The expansion of nodes takes place at specified points in time, e.g. after each set of 100 training inputs. This way of expanding the network will lead to a correct representation of the input, if the input is random. Then, if you present representative inputs to the network, every input will be represented by a node in the network. However, in our case the input to the network is not random, but depends on the keypoints we see in an image. Since, we learn objects in sequential order, and since similar keypoints are present in subsequent images, the order in which we add nodes to the network is important. If we can only add a new node after a fixed number of inputs, a lot of keypoints will not be represented by the network. Therefore, we need a slightly different network, which adds new nodes to the network whenever it needs to, the Growing When Required (GWR) network (Marsland, Shapiro, & Nehmzow, 2002).

5.1 GWR implementation

The basic idea behind adding a new node to the GWR-network has to do with the activation of the node that best matches the input. Each input to the network causes activation of the network nodes, where the activation is based on the distance to the input. The node that is closest to the input has the highest activation. If the activation of this node is above a certain threshold, then the input is well represented by this node and we can associate the keypoint to this existing node. A low activation of the best matching node means that the input is not yet represented in the network. If this node is new and has not been the best matching node often, we may want to train the node by adjusting it towards the input. If the node already won before this means that it is representative of some part of the input. We therefore don't want to change the existing node and instead construct a new one from the input.

I have made some simplifications to the original definition of the GWR-network. First, in the implementation by Marsland et al. (2002) the network consists of nodes and edges. Edges are connections between nodes, which link nodes together that represent similar inputs. Edges are used to adjust the best matching node and its neighbouring nodes in the direction of the input. Since in our case the nodes are not only used to span the input space, but each of the nodes is associated with one or more objects and poses, this effect is undesirable. If an input leads to a change in a number of nodes, this results in nodes floating around too much so that they no longer represent the original input and are therefore incorrectly associated with an object. Since in the original implementation edges are used as a means to destroy nodes in our case the network can only grow larger, not smaller.

The second way in which our implementation is different has to do with the way new nodes are created. Marsland et al. (2002) create new nodes as an average of the input and the best matching node. I create a new node exactly on the location of the input. Since the activation of the existing node is too low, we want to add a new cluster to the network, and the best prediction of where the new cluster will be, is given by the current input.

Using the same notation as Marsland et al. (2002), below I give the description of our implementation of the GWR-network. Let A be the set of clusters in the GWR-network, and let K be the set of keypoints generated by an input image of object ID in pose θ. Define $\mathbf{w_n}$ as the weight vector of node n and t_n as the number of times the node has fired, i.e. the number of times node n was the best matching node. Furthermore, each node n holds a record, R_n, of all associated objects and poses. We initialize the network with $A = n_1$, where the weight vector of n_1 is the first keypoint in K and $t_1 = 0$. Then, for each keypoint $\mathbf{k} \in K$ we do:

1. $\mathbf{k}$ and the object ID and pose θ to which the keypoint belongs are input to the network.

2. Select the best matching node $s \in A$, such that

$$s = \text{argmin}_{n \in A} ||\mathbf{k} - \mathbf{w_n}||$$

3. Calculate the activity of the winning node

$$a_s = e^{-||\mathbf{k} - \mathbf{w_s}||}.$$

4. If $(a_s < a_T) \wedge (h_s < h_T)$, add a new node r

   - $A = A \cup \{r\}$,
   - $\mathbf{w_r} = \mathbf{k}$,
   - $R_r = \{\langle ID, \theta \rangle\}$,

   where $a_T = 0.8$ and $h_T = 0.4$.

5. Else, adapt the weights of the winning node

   - $\mathbf{w_s} = \mathbf{w_s} + \eta \cdot h_s \cdot (\mathbf{k} - \mathbf{w_n})$,
   - rescale $\mathbf{w_s}$ to unit length,
   - $R_s = R_s \cup \{\langle ID, \theta \rangle\}$,

   where $\eta = 0.05$.

6. $t_s = t_s + 1$.

7. Update the firing counter of the winning node, h_s

$$h_s = 1 - \frac{(1 - e^{-\alpha_b t_s / \tau})}{\alpha_n},$$

   where $\alpha_b = 1.05$, $\alpha_n = 1.05$ and $\tau = 3.33$.

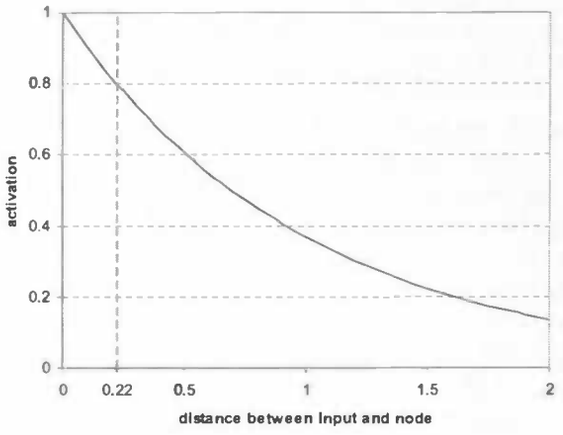


Figure 5.1 – Graph of the relation between activation and the distance between input and node. Note that a Euclidean distance of 0.22 between input and node gives an activation of 0.8, the activation threshold. Inputs with a lower activation than this threshold might be the starting point for a new node.

The first three steps of the algorithm are straightforward. In step 2 and 3, the best matching node and its activation is determined using the exponent of the distance between input and nodes. Then, in step 4, using the activation of the best matching node, a_s, and its value for the firing counter, h_s, we determine whether to create a new node.

Note that the maximum activation a node can admit, is $a_s = 1$. This occurs if the input is exactly equal to an existing node, as can be seen from step 3. If we choose the activation threshold to be the theoretical maximal activation, $a_T = 1$, this means that for every input a new node is created. The resulting network will contain the same number of nodes as the number of inputs that were presented. This 'network' is our baseline model to which the results in chapter 7 are compared.

To achieve some degree of data reduction we have to choose $a_T < 1$, where a smaller a_T leads to a smaller number of clusters in the network. Figure 5.1 shows how the activation of a node depends on its distance to the input. All distances to the right of the vertical line show distances that lead to an activation below the threshold.

For comparison, in chapter 4, I used a maximum distance of 0.6 to determine whether SIFT keypoints refer to the same feature in two subsequent images. In principle we would like to choose these values to be equal. However, using an activation threshold a_T corresponding to a distance of 0.6, leads to too much reduction in the network. Also, since changing the distance parameter in the active vision part does not change the resulting keypoints much, I decided to leave the parameters like this.

Figure 5.2 shows the relation between the firing counter, h_s, and t_s, given by the updating formula in step 7. In the graph, the values for the parameters are used which are given in the description of the algorithm[1].

[1]Note that in our implementation only the value of the fraction α_b/τ is important and not the values of the separate parameters. This notation is used to stay close to the

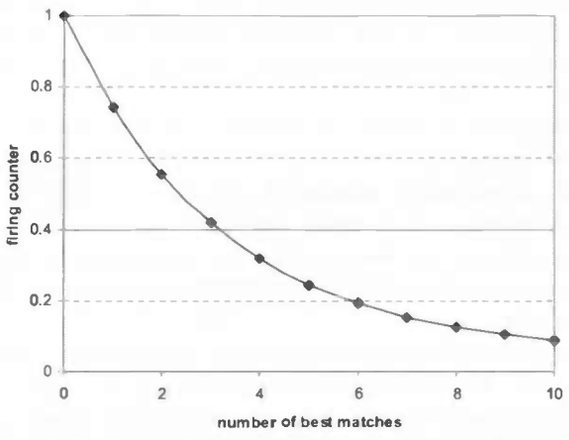


Figure 5.2 – Graph of how the firing counter, h_s, depends on the number of times a node is chosen as best match, t_s. The solid line at 0.4 show the value of the firing threshold h_T.

The choice of $h_T = 0.4$ means that after firing three times, a node cannot move any more. If the activation of this node is not high enough with the current input, a new node is created instead of moving this node towards the input.

If the activation of the best matching node is high enough, in step 5 the weights of this node adjusted towards the input and the current input is associated with this node. The rescaling factor depends on the difference between the input and the weight of the node, $(\mathbf{k} - \mathbf{w_n})$, a learning factor, η, and the firing counter of the node, h_s. If a node is old, the firing counter is low, which we saw from figure 5.2, so the older a node gets, the less it moves around. A small value for η is chosen, since I don't want the nodes to shift location a lot. After moving the node, the weight of the node is rescaled to one, since SIFT keypoints have length one by definition.

5.2 Discussion

An important thing to consider with any neural network is the choice of parameters. Size of the resulting network and learning speed can for instance be adjusted. I played with a number of parameters and chose them such that the network contains approximately a third of the number of keypoints that were used as inputs. Size is an important consideration, since I want a network which is considerably smaller than the original keypoint database. However, if the size is too small, and the clusters are too general, this ultimately has negative consequences for the recognition.

Too make things clearer, let's look at two extreme cases. When $\alpha_T = 1$, the network will contain exactly a copy of all the keypoints as nodes. Obviously, the recognition will be the same as when using all keypoints, but also there is no enhancement in speed. The other extreme is the case where

notation used by Marsland et al. (2002)

the GWR-network contains only one node, and all the keypoints are represented by this node. In this case, the objects will be recognized randomly, and although we have a considerable gain in speed, the system is practically useless.

What happens when we use any number of nodes in between these extremes is of interest and predicting the resulting recognition rate is difficult. When using a reasonable number of nodes the recognition rate could either decrease or increase. A decrease would be the result for instance if some of the keypoints are no longer correctly represented by the network, since the nodes move, or if nodes are associated with too many objects and poses simultaneously.

An increase could be due to beneficial effects of the generalizing function of the network. When using a network, not only the object associated to the best matching *keypoint* in the database is activated. Instead, all objects and poses associated with the *node* with the highest activation is activated. In the former case keypoints could be mismatched due to random fluctuations, whereas in the latter case these keypoints would add towards recognizing to the correct object.

Another point, which has been slightly attended to, is that inputs learned in a later stadium are usually better represented in the network, than inputs which have been learned during the beginning of the process. Since we learn the objects in a linear fashion, the keypoints that are used for inputs to the network, are not totally random. For instance, two subsequent images have a number of keypoints in common, so these are always used as inputs relatively close to each other. Again, this may either have a positive or negative effect on the recognition, depending on the amount of adjustment of the position of the node takes place after it assumes its initial position. In the chapter 7, I will test whether the position of the object in the training set is important for the recognition of the object.

Chapter 6

Object recognition

In the previous sections I described how we can use SIFT, active vision and GWR to detect robust features to obtain an object representation. The goal of this section is to describe how we use this representation to recognize objects in images. First I will give the static way of recognizing objects. In the end I will give some variations how active vision can be used to enhance recognition.

6.1 Static object recognition

In static object recognition the system receives one image of a scene and has to form a hypothesis about which object is present in the scene. Since we don't have any information about which keypoints belong to object or background, or where in the image the object is located, it is probable best to use all available information and extract all SIFT keypoints from the image.

Lowe (1999) uses all available keypoints in the image and finds the best matching keypoint in the database. The Hough transform, using the orientation, scale and location of the SIFT keypoint, is used to obtain a hypothesis about the object and pose. Each of the Hough clusters that contains more than three entries is used to perform an affine projection of the object model on the image. Entries that do not conform to this model are discarded as outliers. Then a new affine projection is estimated without the outliers and possibly additional outliers are removed, until no further changes occur. In principle only three points are necessary to perform the affine projection, so even with considerable amounts of occlusion the method is still valid. A drawback of this methods is the computational intensity of the Hough transform.

In the beginning of my project it seemed that I would not be able to get enough SIFT keypoints per object and pose to implement the method of Lowe (1999). Also, I was worried that the affine transformations would not work well in my case, where a robot circles around an object. Therefore, I looked for a method that does not need a minimum of three keypoints, does not take the locations of keypoints into account and is faster by not using a Hough transform. Of course, using the relations between keypoints can be a valuable way to increase the robustness in object recognition. I will return to this in the discussion in section 8.

Object recognition is performed by a neural network with two layers. The first layer consists of the GWR-network. In the second layer we have one node for each combination of object and position. The connections between the two layers have either strength 0 or 1, where the value is 1 if node n is associated with object ID in pose θ. In that case we write $\langle ID, \theta \rangle \in R_n$. Each keypoint in the image is used as input to the GWR-layer and the node with the highest activation fires. This node passes its activation on to the second layer weighted by the square root of the total number of keypoints associated to the object and pose. The weighting is performed to decrease differences between objects with many or few observations. We expect more keypoints to be present in the image if we look at an object with a lot of keypoints and therefore we want a higher activation. Finally, the activations from different input keypoints are simply summed to obtain the total activation of each object in a certain pose.

The implementation is as follows. We initialize the estimated activation, $e_{\langle ID,\theta \rangle}$, to zero for each object ID and pose θ. For each keypoint k in the image we do:

1. $\mathbf{k}$ is input to the network.
2. Select the best matching node $s \in A$, such that
$$s = \text{argmin}_{n \in A} ||\mathbf{k} - \mathbf{w_n}||$$
3. Calculate the activity of the winning node
$$a_s = e^{-||\mathbf{k}-\mathbf{w_s}||}.$$
4. Add the scaled activation to all object/poses associated with s
$$e_{\langle ID,\theta \rangle} = e_{\langle ID,\theta \rangle} + I\{\langle ID, \theta \rangle \in R_s\} \frac{a_s}{\sqrt{N_{\langle ID,\theta \rangle}}},$$
where $N_{\langle ID,\theta \rangle}$ is the number of keypoints associated to object ID in pose θ.

This procedure results in an activation, $e_{\langle ID,\theta \rangle}$, for each of the objects and poses. The object-pose combination with the highest activation is an estimate for which object in which pose is present in the image.

6.2 Active recognition strategies

Active vision can not only be used during learning, but also during the recognition process. We can compare active recognition to the simple case, where only one image is presented and used for recognition. There are two ways in which active vision can benefit the recognition process. First of all, we can separate object keypoints from the keypoints on the background, analogous to what we did while learning. The other way is to increase the number of keypoints that are used as an input, by driving around the object, and thus obtaining more viewpoints.

Discarding keypoints on the background from the recognition stage, by using two images from subsequent viewpoints, results in robust object keypoints. In principle, background keypoints should not match with any of

the keypoints in the database, since the background does not contain any known objects. However, these keypoints can still be similar to keypoints on objects in the database and thus cause some activation. Therefore, I expect that using only robust keypoints for recognition leads to an increase in the performance, which will be tested in chapter 7. Using only certain keypoints in an image could be especially useful, when looking at a scene containing multiple objects. Active vision can determine which object we want to look at, and thus focusing our attention on the relevant keypoints only.

Another method to use active vision for recognition would be to use an additional image from a different viewpoint. After presenting the initial image, an additional image is presented to the system including the number of degrees that the robot had to drive to arrive at this viewpoint. The estimate derived from the new viewpoint is added to the original estimate to obtain an updated hypothesis about the object that is present in the scene. Combining the information of multiple viewpoints including their relative distance, should enhance the recognition hypothesis. The benefits of this method is tested in chapter 7 as well.

Chapter 7

Experiments and Results

In order to test the methods proposed in the previous sections I ran some experiments for which the results are given below. The criterion that I used for evaluation of the performance of the method is recognition rate of the objects. First, I present the results of using active vision during the learning phase. Then, results are shown for incorporating active vision in the recognition process. Finally, I show what the effects are of using the Growing When Required network to cluster keypoints.

7.1 Dataset

For the construction of object data I used seven objects. Each of these seven objects was placed in the center of a room, while the robot drove around the object, storing an image of the scene every 10 degrees. This resulted in one dataset with 36 images of each of the seven objects. Subsequently, three similar datasets were constructed, but with the object, seen from above, rotated 90 degrees each time. The rotation of the object has two effects; for the same pose the object is seen in different lighting and with a different background. The experiments consist of a learning phase and a testing phase. In each of the experiments one dataset is used for learning, and one for testing. Using different combinations of these four datasets, in total twelve experimental data sets are constructed.

Note that all experiments use the same values for the parameters in order to keep the outcomes comparable. I have experimented with changing some of the parameter values, which give ambiguous results. For specific cases however, the reported recognition rates can easily be increased by tweaking some of the parameters. Although the influence of different parameters on the recognition rate is in itself an interesting question, getting the best possible recognition rate is not the scope of this thesis. One variable that is changed in every experiment is the initial sigma, s_0, in the SIFT algorithm. This value can be either 1.2 and 1.6, but only the results for $s_0 = 1.2$ are reported. The qualitative results using $s_0 = 1.6$ are similar, although the performance is usually slightly lower.

7.2 Recognition rate

As an evaluation criterion for the experiments I use the recognition rate of the objects. The recognition rate of object *ID*, rr_{ID}, is the number of successfully recognized objects divided by the total number of trials

$$rr_{ID} = \frac{1}{36}\sum_{\theta=1}^{36} I(\text{object } ID \text{ in pose } \theta \text{ is correctly recognized as object } ID).$$

Variations on this formula were also used, e.g. using the 12 experimental datasets, or averaging over all objects.

Note that the recognition rate does not incorporate the correctness of the pose estimation. However, a measure of how good the pose estimation is, could also be determined using a variation of the formula above. One choice you have to make then, is how far the pose estimation can be removed from the true pose, to still be counted as correct. Throughout this section I will look at performance in pose estimation using some indicative figures.

7.3 Separation of object and background keypoints

In order to recognize the same object in different backgrounds and different objects in the same background, we need to distinguish between keypoints on the object and keypoints in the background during learning. As we saw earlier, in this set-up driving around an object and looking at the shift in location of keypoints is a good strategy; only a handful of keypoints are misclassified as belonging to the object, due to a repeating background. Below I will show the effects of using only keypoints belonging to the object on the recognition of objects.

When including keypoints from the background in the representation of an object, there will be a lot of keypoints equivalent for each object. Since the background in most of these pictures accounts for more keypoints than the object, we are effectively matching on the background instead of on the object. This would lead to the conclusion that matching is random and the recognition rate should be about 1/7, which is approximately 14%.

In reality, the results are somewhat more subtle, as can be seen from figure 7.1. Here we see the average recognition rates for each of the twelve experiments, when using all keypoints for learning or only the robust keypoints. The datasets are divided in three groups of four based on the difference in rotation between the set used for learning and the set used for testing. For instance, objects in set 3 are, seen from above, rotated 180 degrees with respect to the objects in set 1.

In this figure we see that the recognition rate is constant, about 70%, for all experiments where only the subset of keypoints belonging to the objects are used. The rotation with respect to the background is not important, which we would expect, since background keypoints are not included. Also, the recognition rate increases when there is no distracting background information in the object representation. However, recognition is still correct in around 55% of the cases when using all keypoints for learning. Furthermore, the results in the second column seem especially peculiar. There is only a very small difference between using all keypoints or only robust keypoints in this case.

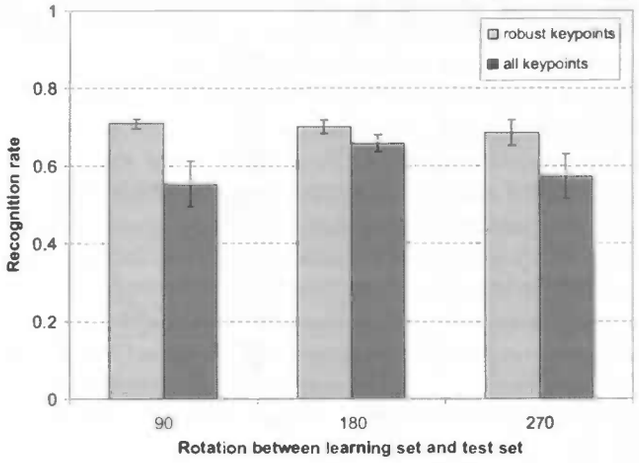
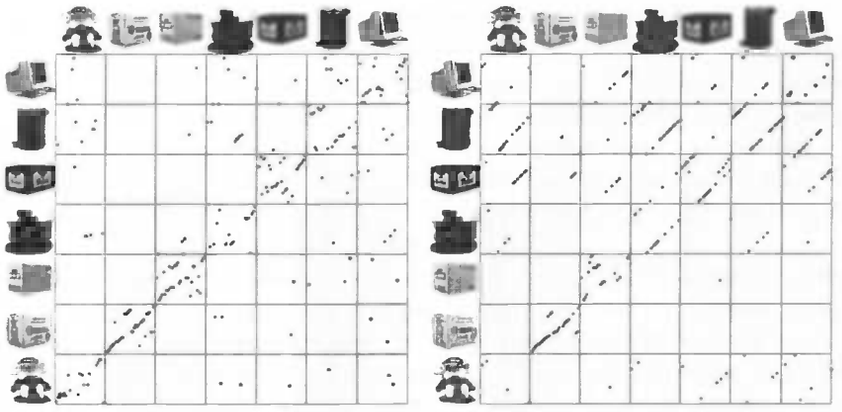


Figure 7.1 – Active keypoint selection. The mean recognition rate is given using all keypoints in the object representation or using only the robust keypoints. In the columns the rotation is given between the dataset that is used for learning and the dataset that is used for testing. The error bars give the 95% confidence interval.

In order to explain the higher than expected recognition when using all keypoints, 55% instead of 14%, we look at the ratio between object keypoints and background keypoints. Most of the activation of an image is caused by the keypoints on the background, since the background contains the most keypoints. Furthermore, the pose of the background does not coincide with the pose of the objects, since the objects are rotated in relation to their original position. Keypoints from the object are in this case adding some noise to the activation, but do not give any structural information.

Since we divide the total activation by $\sqrt{n}$, where n is the number of keypoints one would expect in a certain scene, images with lower n are more likely to have the highest activation. Object 6 has the lowest number of keypoints in most of the images and therefore objects are often falsely classified as object 6. In figure 7.2 we see the correspondence between the object and pose that was presented and the object and pose that were estimated. On the horizontal axis the true object and pose is given, whereas the vertical axis shows the estimate. As can be seen from this figure, the pose that is estimated is not the pose of the object, but the pose of the background, a 90° difference with the original dataset.

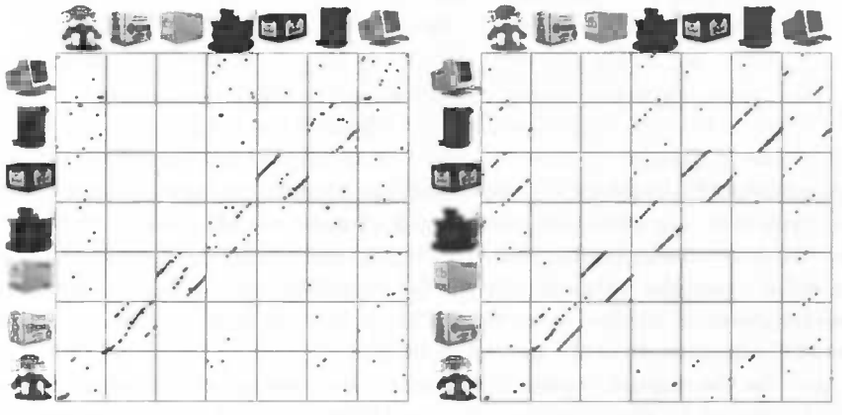
In the case of object 2, however, there are relatively many keypoints belonging to the object, since it is bigger and fills up more space in the image. We see that this object is classified correctly and that the pose is estimated correctly, which explains an increase in the recognition rate. Furthermore, object 5 looks the same from four different angles. So this object can be recognized correctly, although sometimes with a slightly different pose, when background keypoints are present. The keypoints on the object will increase the activation beyond the activation the background keypoints. These combined effects cause the recognition rate in the presence of background keypoints to be higher than expected.



(a) Using object keypoints only (b) Using all keypoints

Figure 7.2 – Object recognition and pose estimation using different learning strategies, $s_0 = 1.2$. In both images dataset 1 is used for learning and dataset 2 is used for testing, which gives a 90° rotation of the objects. The image on the left uses only the keypoints on the object for learning, whereas the image on the right shows the results from using all keypoints.

For the data in the second column of figure 7.1, images are used where the objects are rotated exactly 180° from their original position. For the symmetric objects this results in correct classification, but with a 180° pose difference. This results in an extra increase in the recognition rate, when comparing this number to the datasets where background keypoints were included, but with a 90° or 270° rotation of the object. The result of this can be seen in figure 7.3.



(a) Using object keypoints only (b) Using all keypoints

Figure 7.3 – Object recognition and pose estimation using different learning strategies, $s_0 = 1.2$. In both images dataset 3 is used for learning and dataset 1 is used for testing, which gives a 180° rotation of the objects. The image on the left uses only the keypoints on the object for learning, whereas the image on the right shows the results from using all keypoints.

In these experiments keypoints are classified as robust or non-robust

based on two criteria. First, the keypoints must be recognized in two subsequent images. Second, the keypoints must remain on approximately the same location in these images. The experiments show that the use of active vision for separation of background keypoints and object keypoints works and has a positive effect on the recognition rate. These positive effects could be made much clearer by performing the learning of objects and the recognition of objects in different environments. Then, the confounding effect of recognizing the background and a symmetric object does not occur. Also, using more objects will result in a sharper difference between recognition based on all keypoints and recognition based only on object keypoints. The probability that an object is recognized correctly is random, i.e. $1/n$, when we use all keypoints. Increasing the number of objects, n, results in a smaller fraction of objects being recognized correctly.

Another observation is that separating object keypoints from background keypoints becomes even more important if the space that the object takes in the image is relatively small. We saw that in a lot of cases the objects were classified as object 6, the object with the least number of keypoints. If all objects consist of a small number of keypoints relative to the total number of keypoints present in the image, we must separate object and background keypoints. If most of the keypoints in the image belong to the object, separating object and background is less important.

Although selection of keypoints on the object is important, we do not know if we need to use all the keypoints that are on the object. There might be better ways to select keypoints which not only belong to the object, but are also more interesting from the perspective of recognition. For instance, keypoints on the border of an object belong to the object, but might consist for a large part of information about the background. It would be useful to look at additional ways to select valuable keypoints. On the other hand, maybe we are throwing away valuable information. Some keypoints are present only in one pose, and might therefore be important for accurate pose estimation. These keypoints are discarded in this method.

7.4 Active object recognition

In section 6 we saw that active vision can be used during recognition in two different ways. One way is to filter keypoints in the background from those belonging to the object using the same methodology we used for learning. We can then use only the keypoints generated by the object in the recognition process. In general, the background contains distracting keypoints, maybe even keypoints generated by other objects in the database. Focusing attention only on keypoints on the object of interest might therefore increase performance.

A different active recognition method uses images from multiple viewpoints as inputs to estimate the object and its pose. By driving around the object the robot gathers additional images of the object, thereby using information on how far it has moved. The first image is the usual input seen by the robot, when placed at a random position facing the object. The second image is generated by driving 30° in clockwise direction around the object, the third one by driving 30° further, etc. By combining the new information about the object with knowledge on how far the robot has moved,

the estimate of object and pose can be enhanced.

In figure 7.4 results are given for the two different active recognition strategies. The straight line shows the recognition rate when all keypoints from the input image are used. The mean recognition rate is given, with 95% confidence intervals around them. Clearly, increasing the number of viewpoints increases the recognition rate. When one viewpoint is used as input for the classification process we have a recgonition rate of 70%, which increases to 85% when using four viewpoints.

The dotted line in the same figure shows the results when we use only the object keypoints in the recognition process. We see that this strategy results in a significantly higher recognition rate, when using one viewpoint as input. When adding more viewpoints, the difference between the two strategies stays roughly constant, with the latter strategy resulting in a recognition rate of almost 90% after four viewpoints.

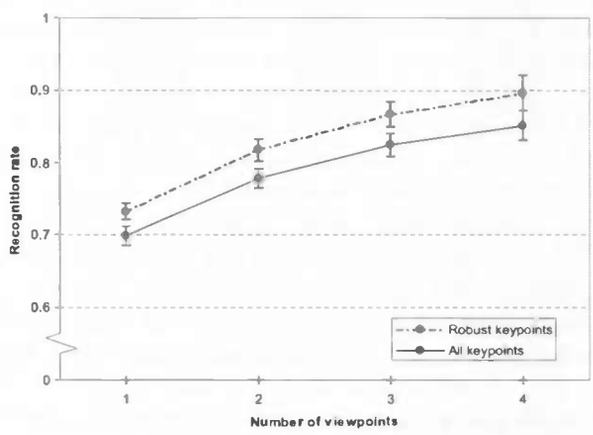
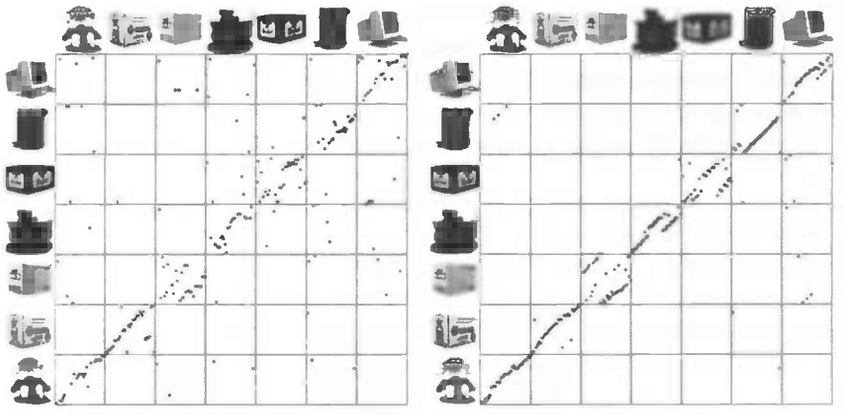


Figure 7.4 – Object recognition using active vision. The mean recognition rates for the active method, using all keypoints in the image for recognition or only the robust keypoints. Both methods are plotted as a function of the number of viewpoints used to explore the object. The error bars give the 95% confidence interval.

In experiments where I used a different number of degrees for the movement of the robot, the results are similar. However, driving only 10 degrees give a much smaller increase in recognition rate. Since a smaller area of the object is covered, the probability that we see the same keypoints twice is larger, so there is not much 'new' information. Whether the robot drives 30°, 40° or more doesn't really make a difference. Since most keypoints are only visible in three subsequent images, new information is obtained every time we drive more than 30°.

In figure 7.5 we see how the pose estimation changes, when we increase the number of viewpoints. Only the object keypoints are used in the recognition process, and the pose estimation is given when using 1 viewpoint or 4 viewpoints. Pose estimation is more accurate when we use 4 viewpoints, which can be seen from the fact that the points are closer to the 45°-line. The pose of object 3 is correctly estimated if we take into account that this object is symmetric. Object 6 is recognized with a 90° difference between the true pose and the estimated pose, which is exactly the rotation between



(a) Using 1 viewpoint (b) Using 4 viewpoints

Figure 7.5 – More accurate pose estimation when using more viewpoints. Both figures show the pose estimation, when using dataset 3 for learning, and dataset 4 for testing. Only the robust keypoints are used for recognition, but figure 7.5(a) gives the pose estimation after using 1 viewpoint, and figure 7.5(b) uses 4 viewpoints.

dataset 3 and dataset 4. This is because the object has a reflective surface, such that the keypoints are in fact reflections of the background. I expect that this object can not be recognized correctly if we see the object in a different environment.

From these experiments I conclude that actively gathering additional information about an object can improve the classification of objects and the estimation of its pose. In my experiments the action the robot could take was fixed. It could only move 30° in clockwise direction. An interesting way forward would be to look at which action the robot should take in order to get the most valuable information. And, related to this, when is the optimal moment to stop gathering extra information. This could be either because you are very sure about which object you are looking at, or because no amount of extra information will ever give you a confident estimate of the object.

7.5 Clustering keypoints with GWR

To make the representation of objects more efficient in terms of storage space, the keypoints are clustered using a Growing When Required (GWR) network. As was explained in section 5, theoretically, using clusters of keypoints can have different effects on recognition rate. A cluster of keypoints can be associated to more than just one object and one pose. The recognition rate could both increase and decrease, because not only the keypoint that best matches the input is considered.

In figure 7.6 we see the results of recognition with or without a network to cluster the learned keypoints. The left column shows the results without clustering, but using four viewpoints as inputs and using only the object keypoints. This is the active recognition strategy which resulted in the highest recognition rate and I'll call this standard SIFT. The second column

shows the results when using the same active recognition strategy, but using the GWR-network to cluster similar keypoints during learning. The GWR-network contains only 37% of total number of keypoints as nodes. The third column shows the results when we use a random fraction of 37% of the original keypoints as object representation.

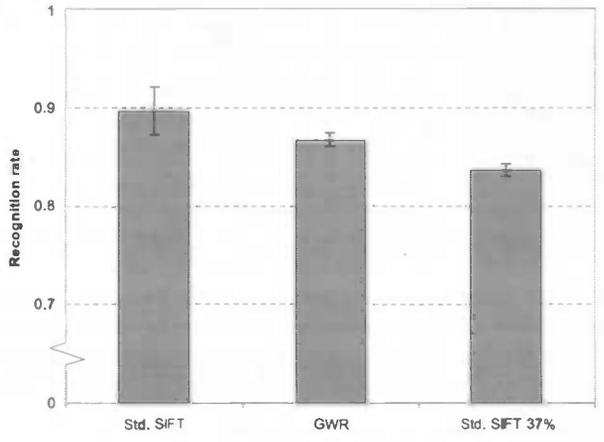


Figure 7.6 – Different learning strategies. The mean recognition rates are given for standard SIFT, the GWR-network and standard SIFT using 37% of the keypoint database. Standard SIFT uses on average 6429 keypoints. The GWR-network has on average 2396 keypoint clusters (37% of standard SIFT). For the GWR-network and SIFT using 37% keypoints, the data is acquired from 10 experimental runs. The error bars give the 95% confidence intervals.

The GWR-network is learned by presenting the objects sequentially. In order to obtain more datasets to use for testing, the objects were learned in 10 different orders. Although the order of the objects is changed, all images of an object are still presented before continuing to the next object. Although randomly presenting keypoints to the GWR-network might result in a better performance, I feel that this method mimics how objects would be learned in the real world. These 10 different orders of learning combined with the 12 different combinations of original datasets, gives a total of 120 datasets that were used for testing.

For standard SIFT only 12 datasets were used, since here the order of presenting the objects during learning doesn't matter; all object keypoints are used anyway. I also use 10 different random subsets of 37% of all object keypoints, so the result in the last column in figure 7.6 is also based on 120 datasets.

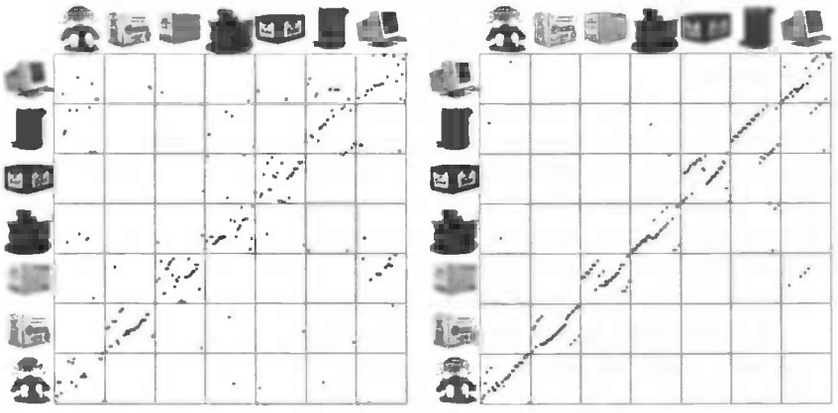
When we use GWR to cluster keypoints, the recognition rate decreases from 89.6% tot 86.7%. The p-value for testing whether the means are equal is .0501. Although formally we cannot reject the hypothesis that the difference is equal on a 5%-level, it is still likely that standard SIFT performs slightly better than the GWR-network.

Beside accuracy we can also compare the two methods in terms of speed. The number of nodes in the GWR-network is equal to 37% of the original number of keypoints. Determining the closest keypoint in the database is computationally the most expensive part of the recognition process. There-

fore, since the GWR-network has less nodes, recognition using the GWR-network is approximately three times faster than using all keypoints. The difference in performance between the two methods is very small, but the gain in computation speed is large when using GWR, so using a GWR-network to cluster keypoints is a useful extension.

Another way to obtain the same enhancement in computation speed is to randomly discard some of the original keypoints. In the third column in figure 7.6 the results are shown when using a random subset of 37% of the keypoints. This method performs as fast as the GWR-network, but the recognition rate is significantly lower. It seems that GWR retains some extra information that is lost when randomly discarding keypoints.

Discarding a random fraction of the keypoints can be seen as some measure of how the method performs in the presence of occlusion. Since the standard SIFT method with only 37% of keypoints still results in a recognition rate above 80%, this method probably performs quite well with occlusion. There seems to be enough redundant information in the keypoints to recognize the objects with less information.



(a) Using 1 viewpoint (b) Using 4 viewpoints

Figure 7.7 – More accurate pose estimation when using more viewpoints. Both figures show the pose estimation, when using dataset 4 for learning, and dataset 3 for testing. The keypoints are classified using GWR and only the robust keypoints are used for recognition. Figure 7.7(a) gives the pose estimation after using 1 viewpoint, and figure 7.7(b) uses 4 viewpoints.

In figure 7.7, we see the correspondence between the input pose and the estimated pose, similar to figure 7.5. The results are shown using 1 viewpoint and 4 viewpoints for recognition, but in this case the keypoints have been clustered using GWR. Again, we can conclude that the pose estimates are more accurate if we use more viewpoints for recognition. The pose of objects 3 and 5 is correctly estimated if we correct for the fact that these objects are symmetric. Again, we see that object 6 is recognized from keypoints that are reflections from the background.

Table 7.1 – Learning order of objects. A linear relation is estimated between recognition rate and position during learning of the GWR-network. A positive β would mean that objects learned at a later stage are better represented by the network. In brackets the standard deviation of the estimates is given.

	1 viewpoint		4 viewpoints	
rr_1	50.94%	(0.94%)	71.65%	(1.12%)
rr_2	95.90%	(0.91%)	100.37%	(1.08%)
rr_3	85.91%	(0.93%)	99.33%	(1.10%)
rr_4	55.07%	(0.93%)	77.51%	(1.10%)
rr_5	84.09%	(0.91%)	99.13%	(1.07%)
rr_6	64.53%	(0.91%)	88.30%	(1.07%)
rr_7	55.14%	(0.92%)	73.53%	(1.09%)
β	-0.26%	(0.15%)	-0.13%	(0.18%)

7.6 Order of objects in learning

Usually, the order in which a neural network is learned is important for its performance. Most networks require that the inputs are presented randomly in the learning stage. In our case the inputs are presented for each object and each pose separately, and this might have consequences for the networks performance.

Because of the way the network is initialized, the first set of nodes move around more than may be satisfactory. That might make us think that objects that are learned at the start, perform worse than objects which have been learned later in the process. In contrast, keypoints that are presented to the network in a later stadium are less likely to create their own node. So, these keypoints might be less accurately represented by the network, which would result in later objects being recognized worse.

As mentioned before, we have 12 combinations of learning and testing datasets, which we denote by the index i. There are seven objects, so each of the objects can be learned at seven different positions. I use 10 different object orders during learning, such that the objects are learned at each of the positions at least once. In total we have $7 \times 10 \times 12 = 840$ observations.

In figure 7.8 we see how the recognition rate of each of the objects depends on the position. Again, the mean recognition rate and a 95% confidence interval is given. From the figure it looks like the recognition rate of object *ID* does not depend linearly on the position, p, in which the object is learned. To test this formally, I write down the following regression

$$rr^p_{ID,i} = rr_{ID} + \beta p + \varepsilon_i,$$

where ε_i is some random error. If the estimated value of β is positive, then objects that are learned at higher positions are better represented by the network.

The results of the estimation are shown in table 7.1 and the predicted regression is shown in figure 7.8 as a solid line. In the table both the results, when one viewpoint is used for recognition, and when four viewpoints are used for recognition, are shown. Similar to what we say in figure 7.8, we see

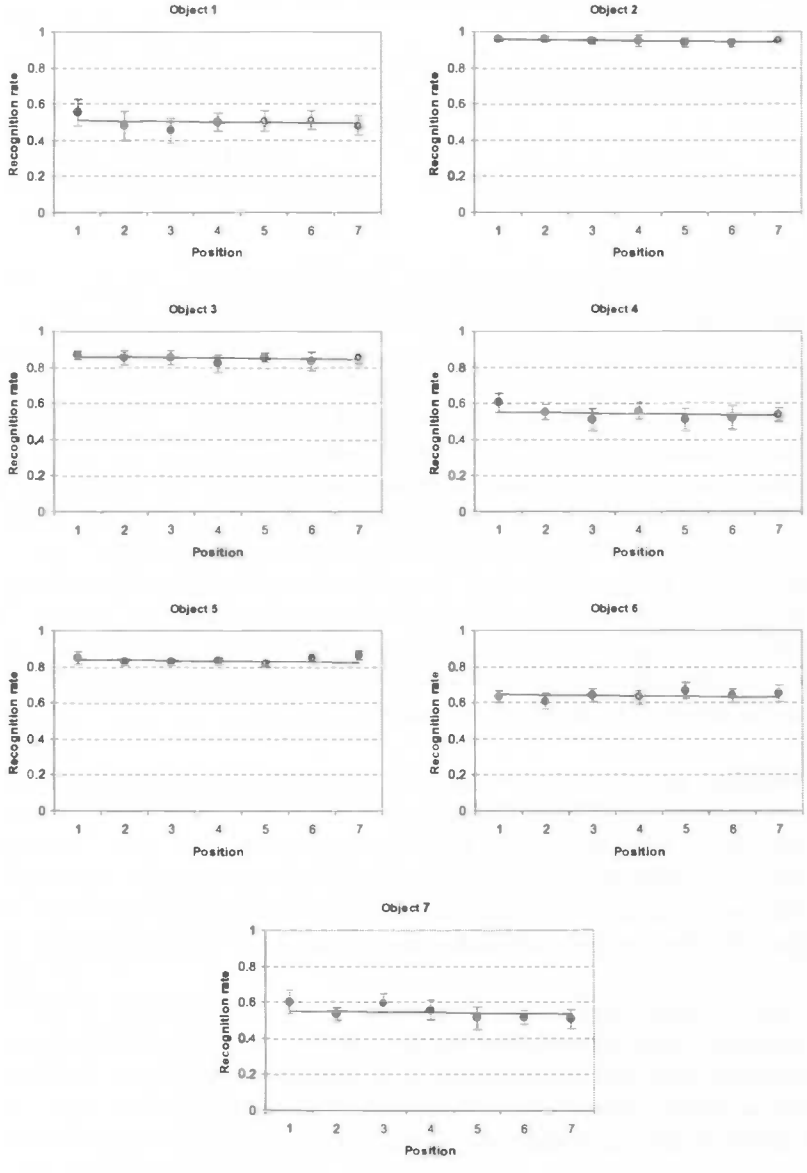


Figure 7.8 – Object recognition and order of learning. These figures show the mean recognition rate for each of the objects as a function of the position it was presented to the GWR-network. One viewpoint is used for recognition. The error bars give the 95% confidence interval.

that the recognition rates vary from object to object. Also, the recognition rate increases when we use four viewpoints instead of one viewpoint. The more important observation is that the value of β does not differ significantly from zero in both cases. This implies that the order of learning is not important for recognition, at least not in a linear way.

Chapter 8

Conclusion

In this thesis, I described the SIFT method and an application to object recognition. In order to learn an object representation in a real-world setting I used active vision. Also, similar keypoints were clustered using a variation of the GWR-network. In this final chapter I will shortly summarize my findings and give some directions in which my research can be extended.

First of all, we saw that using active vision to detect robust keypoints on the object in the image works well. If we have an object in the center of a room, with a robot driving around the object, we can use relative differences in location shift between keypoints in subsequent images to separate keypoints on the object from keypoints in the background. A small number of keypoints were classified as object keypoints, while they were keypoints on the background, due to a repeating background. Therefore, it has to be noted that this simple method will not work in all environments. Also, when the object of interest is not standing by itself, this might result in complications.

Secondly, the use of active vision in the recognition stage significantly increases performance. Especially the recognition of objects, which do not have many keypoints on some sides of the object, increases. Also, active vision results in more accurate pose estimation. Around 90%, the recognition rate might not seem high, but we have to take into account that not all objects are suited for SIFT recognition, since they do not contain enough keypoints.

Thirdly, the use of the GWR-network to cluster keypoints is a valuable improvement. In comparison with using all available keypoints, the recognition rate does not decrease significantly, but computations take only a third of the time. Clustering the keypoints by the GWR-network give a better recognition rate than when we reduce the original keypoints by randomly discarding a similar fraction.

Future research can go in a number of directions. First of all, more experiments can be conducted to better test the findings in this thesis. Some of these experiments can be implemented with relatively small effort. Also, which would be a less straightforward task, the methods described in this thesis can be improved.

An obvious extension is to use more objects in additional experiments. This would shed light on which kind of objects work with SIFT recognition. We already saw that objects which do not generate a lot of keypoints are not recognized correctly in all cases. Also, the objects should be learned

in different environments. Adding different environments should make the benefit of using active vision in the learning stage more explicit. In my case the keypoints of the symmetric objects and the background keypoints suffered from some correlation, which in principle does not exist. This falsely increased the recognition rate considerably, when including the background keypoints in the representation of the objects.

Furthermore, it would be good to test whether the number of keypoints are important for correctly recognizing an object. Maybe the uniqueness of a keypoint in the database is more important. Just looking at the difference in recognition between objects with a lot of keypoints and objects with a low number of keypoints does not work. It could be that objects with a lot of keypoints are also objects which have more unique keypoints that work well for recognition. Therefore we want to look what happens when the same object is represented with a different number of keypoints, keeping the sort of keypoints fixed. We can do this by representing each object by random samples of different sizes and see how this influences recognition. This should give a causal result between changing the number of keypoints and recognition. We could also use more than one dataset for learning to increase the number of keypoints associated with one object.

Other ways to extend the research described in this thesis, is by improving the methods. For instance, the way I classify keypoints as object or background keypoints can be enhanced. Consider the case where a particular appearance of an object contains a number of similar keypoints, for instance because the object contains a texture which generates numerous similar keypoints. These keypoints are presumably useful in order to recognize the object in the current pose. However, my simple method of separating the object keypoints from the background keypoints does not handle this situation well. Currently, I only check whether the best matching keypoints are close to each other in location. A better strategy would probably be to revert the procedure. I.e. look only at keypoints in a region around the current keypoint and check whether any of these keypoints match with the current keypoint. The problem stated above could be overcome in this way.

If we don't change the parameters of the GWR-network, but try to learn more objects, and thus more keypoints, the number of noes in the network will increase. If more objects are available, it is therefore interesting to see if a higher degree of generalization can be obtained with the GWR-network, so that the number of nodes in the network will stay small. Nodes can probably be associated with multiple objects and still result in a good recognition rate if we look at groups of keypoints. Lowe (1999) uses groups of three SIFT keypoints and the relation between the locations of these keypoints in the image. A group of keypoints in a certain relation to each other, gives use more confidence about which object is present in the image, than the presence of multiple unrelated keypoints. The same method as the one Lowe (1999) uses, can be combined with the GWR-network. This could potentially lead to better recognition rates, with more general GWR clusters.

Finally, it would be interesting to add a way to select actions in the recognition process. Recognition rates increase if we use multiple viewpoints of the scene we are looking at. However, driving a fixed number of degrees, which is used in this thesis, is probably not the best way to obtain additional

information about a scene. Some objects do not have many keypoints from some viewpoints, so it would not be beneficial to drive in that direction. Other objects might look the same from most viewpoints, but are different from a specific viewpoint, e.g. if we look at similar boxes with a specific label at one side. To derive the optimal action, a prediction has to be made about the costs of an action, e.g. the driving time, and the benefits of an action, e.g. which action results in the most discriminating viewpoint. The GWR-network can be used in to obtain this, since this network links each of the objects and their poses to a common set of nodes.

We can conclude that SIFT and GWR clustering can form the basis of correct object recognition for some kinds of objects. Although object recognition using SIFT is no panacea, in some restricted domains it works well. Fortunately, there is a vast number of directions one can follow to improve the process, so there are many opportunities for research left.

Bibliography

Ballard, D. H. (1991). Animate vision. *Artificial Intelligence*, *48*(1), 57-86.

Borotschnig, H., Paletta, L., Prantl, M., & Pinz, A. (2000). Appearance-based active object recognition. *Image and Vision Computing*, *18*(9), 715-728.

Brown, M., & Lowe, D. G. (2005). Unsupervised 3d object recognition and reconstruction in unordered datasets. In *3dim '05: Proceedings of the fifth international conference on 3-d digital imaging and modeling* (pp. 56–63). Washington, DC, USA: IEEE Computer Society.

Dance, C., Willamowski, J., Fan, L., Bray, C., , & Csurka, G. (2004). Visual categorization with bags of keypoints. *ECCV International Workshop on Statistical Learning in Computer Vision*.

Dorko, G., & Schmid, C. (2003). Selection of scale-invariant parts for object class recognition. *Ninth IEEE International Conference on Computer Vision (ICCV'03)*, *1*, 634–640.

Fritzke, B. (1995). A growing neural gas network learns topologies. In G. Tesauro, D. S. Touretzky, & T. K. Leen (Eds.), *Advances in neural information processing systems 7* (pp. 625–632). Cambridge MA: MIT Press.

Harris, C., & Stephens, M. (1988). A combined corner and edge detector. *Proceedings of The Fourth Alvey Vision Conference, Manchester*, 147–151.

Keselman, Y., & Dickinson, S. (2005). Generic model abstraction from examples. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *27*(7), 1–16.

Kohonen, T. (1990). The self-organizing map. *Proceedings of the IEEE*, *78*, 1464–1480.

Leonardis, A., & Bischof, H. (2000). Robust recognition using eigenimages. *Computer Vision and Image Understanding*, *87*(1), 99–118.

Lowe, D. G. (1999). Object recognition from local scale-invariant features. *Proceedings of the International Conference on Computer Vision, Corfu*, 1150–1157.

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

Marsland, S., Shapiro, J., & Nehmzow, U. (2002). A self-organising network that grows when required. *Neural Networks*, *15*, 1041–1058.

Mikolajczyk, K., & Schmid, C. (2005). A performance evaluation of local descriptors. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *27*(10), 1615–1630.

Murase, H., & Nayar, S. K. (1995). Visual learning and recognition of 3-d objects from appearance. *International Journal of Computer Vision*,

14(1), 5–24.

Nelson, R. C., & Selinger, A. (1998). Large-scale tests of a keyed, appearance-based 3-d object recognition system. *Vision Research, 38*(15–16), 2469–2488.

Paletta, L., & Pinz, A. (2000). Active object recognition by view integration and reinforcement learning. *Robotics and Autonomous Systems, 31*(1), 71–86.

Roy, S. D., Chaudhury, S., & Banerjee, S. (2004). Active recognition through next view planning: a survey. *Pattern Recognition, 37*(3), 429–446.

Serre, T., Wolf, L., & Poggio, T. (2005). Object recognition with features inspired by visual cortex. *Proceedings of the 2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05), 2*, 994–1000.

Turk, M. A., & Pentland, A. P. (1991). Face recognition using eigenfaces. *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*, 586–591.