Object Recognition Using Digitally Generated Images as Training Data

Anton Ericson
Abstract

Object Recognition Using Digitally Generated Images as Training Data

Anton Ericson

Object recognition is a much studied computer vision problem, where the task is to find a given object in an image. This Master Thesis aims at doing a MATLAB implementation of an object recognition algorithm that finds three kinds of objects in images: electrical outlets, light switches and wall mounted air-conditioning controls. Visually, these three objects are quite similar and the aim is to be able to locate these objects in an image, as well as being able to distinguish them from one another. The object recognition was accomplished using Histogram of Oriented Gradients (HOG). During the training phase, the program was trained with images of the objects to be located, as well as reference images which did not contain the objects. A Support Vector Machine (SVM) was used in the classification phase. The performance was measured for two different setups, one where the training data consisted of photos and one where the training data consisted of digitally generated images created using a 3D modeling software, in addition to the photos. The results show that using digitally generated images as training images didn’t improve the accuracy in this case. The reason for this is probably that there is too little intraclass variability in the gradients in digitally generated images, they’re too synthetic in a sense, which makes them poor at reflecting reality for this specific approach. The result might have been different if a higher number of digitally generated images had been used.

Handledare: Stefan Seipel
Ämnesgranskare: Anders Hast
Examinator: Tomas Nyberg
ISSN: 1401-5757, UPTEC F 13 010
Sammanfattning

Datoriserad objektigenkänning i bilder är ett område vars tillämpningar blir fler hela tiden. Objektigenkänning används i alltifrån teckenigenkänning i skrivna texter och kontroll av tillverkade produkter i fabriker, till den ansiktigenkänning som ofta finns i digitala kameror och på facebook.

Detta arbete syftar till att använda datorgenererade bilder till att träna upp ett objektigenkänningsprogram som skrivits i MATLAB. Objektigenkänningen använder sig av metoden Histogram of Oriented Gradients (HOG) för att tillsammans med en SVM-klassificerare jämföra innehållet i olika testbilder med innehållet i träningsbilderna för att på så sätt kunna lokalisera objekt i testbilderna.


Resultatet visar att programmets precision inte förbättrades av att utöka uppsättningen av träningsbilder genom att lägga till de datorgenererade bilderna. En trolig förklaring till detta är att de datorgenererade bilderna är för syntetiska, det är för lite variation i gradienterna för att de ska återspeglas verkligheten. Eventuellt kan resultatet förbättras om fler datorgenererade bilder används.
## Contents

1 Introduction

2 Theoretical background
   2.1 Structural and decision-theoretic approaches
   2.2 The SIFT descriptor
   2.3 The shape context descriptor
   2.4 The histogram of oriented gradients descriptor
   2.5 Classifiers
      2.5.1 AdaBoost
      2.5.2 Support vector machines (SVM)
   2.6 Related work

3 Methods
   3.1 Sliding window method
   3.2 SVM classification voting strategy
   3.3 Creating the 3D models

4 Implementation and testing
   4.1 Implementation of the object recognition
   4.2 Training images and test images
      4.2.1 Varying the image sizes
      4.2.2 False positives
   4.3 Image scanning algorithm

5 Results
   5.1 Varying size of training images
   5.2 Performance when including digitally generated images

6 Discussion and conclusions
   6.1 Discussions on performance
   6.2 Concluding remarks

5  

10  

15  

18  

20  

23  

24  

26  

28  

30  

31
1 Introduction

Object recognition is a task in computer vision where the aim is to find specific objects in images using a computer. Humans are very good at doing this effortlessly, even for objects that are occluded, and the rotation, scale and size of the object is seldom an issue. For a computer however, this is still a challenge and even the most advanced systems come nowhere near human performance. As noted by Liter and Bülthoff [16] computers do surpass humans in certain areas of object recognition, e.g. manufacturing control where they can detect small manufacturing flaws that would go unnoticed by most humans. At the same time it is difficult to give a computer the visual capacity of a three year old, e.g. distinguishing her own toys from her friend’s.

Object recognition applications are found in many circumstances, such as face and human recognition in images [19], character recognition of written text [2], assisted driving systems [10] and manufacturing quality control [4]. Most object recognition approaches require a training data set, which consist of images containing the object to be recognized. Training data sets need to contain quite a large number of images to be effective, and for the most common objects (e.g. humans, faces and cars) there are good databases of training images freely downloadable from the internet. For less common objects however, it is often required that the training images are gathered manually. Constructing a set of training images based on photos is time consuming, and if digitally generated images could be used instead a great deal of work could be saved.

The three objects that are to be recognized are electrical outlets, light switches, and a wall mounted air-conditioning control system called Lindab Climate (from now on referred to as Lindab). These objects are quite similar to each other, which make them challenging but suitable targets for object recognition.

The question that is to be answered in this paper is the following: can images generated from 3D models be used as training images for computer vision purposes?

To get a feeling for how the size of the images influence the recognition results, an evaluation of performance for different sizes of the training images will be carried out. This will be done without any digitally generated images included in the training image set.
2 Theoretical background

For humans, object recognition is nothing more than visual perception of familiar items. But as put by Gonzalez and Woods [11], for computers, object recognition is perception of familiar patterns. The two principal approaches for pattern recognition for computer vision purposes are decision-theoretic and structural approaches. Decision-theoretic approaches are based on quantitative descriptors (for instance size and texture) and structural approaches are based on qualitative descriptors, e.g. using symbolic features such as knowledge of the interspatial relations between different parts of an object. In other words, for decision-theoretic approaches a classifier is trained with a finite number of classes and when subjected to a new pattern it categorizes it into the class of best fit. Structural approaches represent the patterns as something else (e.g. graphs or trees) and the descriptors and categorizations are based on the representations.

2.1 Structural and decision-theoretic approaches

Structural approaches come in various designs, but they all use an alternative representation of some sort which their categorizations are based on. Part-based models, as mentioned in [8], form one class of structural approaches in which various parts of an object are located separately and the categorization is carried out using knowledge of the relative positions of these parts. This type of approach can be especially useful since different classes contain similar parts, e.g. cars, busses and trucks all have wheels and doors. These type of approaches make it possible to categorize an object without having seen that kind of object before, by merely recognizing it’s parts.

Three common decision-theoretic approaches are the SIFT descriptor [17], the HOG descriptor [6] (the HOG descriptor is the one used in this project), and the Shape Context descriptor, and especially the first two are described thoroughly in [8]. These approaches are invariant to image translation, rotation and scale, which are very important qualities for object recognition techniques.

2.2 The SIFT descriptor

The SIFT descriptor [17] is a method that extracts “interesting points” from an image, points that are invariant to translation, rotation and scale, as well as small changes in illumination, object pose and image noise. The points are usually located on the edges in an image. The first step in the SIFT method
is to calculate the Difference of Gaussians (Figure 1). The Gaussian of an image represents a blurred version of an image, where the amount of blur is decided from the size of the filter. The difference of Gaussians algorithm therefore calculates the Gaussian of an image twice using two different filter sizes, and subtracts these from one another. The SIFT method calculates the maxima and minima in the difference of Gaussians, and these maxima and minima form the SIFT keypoints in an image.


Figure 1: **Figure 1 shows an example of the output from Difference of Gaussians algorithm, which is performed by taking the difference between two different Gaussians of an image.** (Images from http://en.wikipedia.org/wiki/File:Flowers_before_difference_of_gaussians.jpg)

These SIFT keypoints are then modified in numerous ways (e.g. discarding candidates with low contrast), but they can then be used to find matching keypoints in other images. It turns out that SIFT keypoints will be located at similar positions for similar objects, as well as being located at non similar positions for non similar objects. It is therefore an often used and well performing feature descriptor.

### 2.3 The shape context descriptor

The shape context descriptor contains a representation of an object’s shape. The contour of an object is modified to be represented by a number of points, and for each of these points a histogram is created which describes
the point’s relative position to all other contour points. Figure 2 shows an overview of this method.

Figure 2: Figure 2 shows an overview of the shape context method. a) shows a point contour representation of the letter A and b) shows another representation of another A. For each point on the contour a histogram is created by putting all other points into log-polar bins. d) is the shape context for the marked point in a), e) is that for the right marked point in b) and f) is that for the left point in b). d) and e) are similar since they represent related points on the shape, whilst f) has a different appearance. (Image from http://en.wikipedia.org/wiki/File:Shapecontext.jpg)

Shape context descriptors form a powerful method of representing and comparing the shapes of objects in images, but retrieving the object contour in cluttered images isn’t always easy. Problems may occur when object contours are partially occluded, or when the contour definition of an object is vague.

2.4 The histogram of oriented gradients descriptor
The HOG descriptor was presented by Dalal and Triggs in 2005 [6] and has proven to be a well performing feature descriptor [3]. The technique divides an image into regions and counts occurrences of gradient orientations, i.e. gives an approximate of the location and distinction of the edges in the
image. By definition, a *gradient* in an image is a directional change in intensity or color.

The first step in the HOG method is to divide an image into a number of fixed area regions called *cells*, and in each cell the gradients are sorted into histogram bins according to angle. The cells are grouped together in 2x2 groups called *blocks*, and for improved accuracy the cell histograms are normalized using an intensity measure done within the block. This normalization makes the method less sensitive to changes in shadowing or illumination. A schematic representation over the architecture of the HOG method is shown in Figure 3. Figure 4 shows how the different image gradients within a cell are combined into the final descriptor, which is stored as a histogram.

Figure 3: Figure 3 shows the architecture of the HOG method. The image is divided into cells, and for each cell a histogram of the gradient orientations is created. The cells are grouped into blocks and a histogram normalization is carried out in the blocks, which reduces the sensitivity to changes in illumination in the image.
Figure 4: Figure 4 shows how the image gradients inside the cells are combined into descriptors.

I use a cell size of 8x8 pixels, the blocks consist of 2x2 cells, and the size of the angular bins are 8° (out of a total search angle of 360°). Figure 5 shows an example of what the gradients resulting from the HOG method might look like, although the gradients shown in the image are merely a mean value.
Figure 5: Figure 5 shows an example of what the output of the HOG method might look like. a) shows the original image, and b) shows the resulting gradients computed by the HOG method (in this case, b) only shows a mean value of the image gradients, but in reality the HOG method calculates the gradients for many different angles). The gradient information is stored in a vector which is fed into a classifier (in this case the SVM classifier).

The method used in this project is the HOG method. It was chosen because of its versatility; the shape context descriptor was ruled out early on since the different objects to be recognized have quite similar shapes. The HOG and SIFT methods seem to be fairly similar in performance, but HOG
was chosen because it is conceptually simpler. Another reason for choosing HOG is that Heisele et al. use it when training their object recognition system with digitally generated images [13], which is quite similar to what will be done in this project.

2.5 Classifiers

After the HOG descriptors have been extracted from the images, these are fed into a classifier of some sort. Many classifiers are binary, i.e. only two classes are handled at a time. In the training phase the classifier is fed training data from two different classes. When subjected to the descriptors from a test image, the classifier decides which of the two classes the test image belongs.

It is the classifier that analyzes the data and looks for the patterns that enables a classification between different object classes. Two common machine learning algorithms that are often used as classifiers in object recognition approaches are support vector machines (SVMs) and AdaBoost.

2.5.1 AdaBoost

AdaBoost [9] (short for Adaptive Boosting) is a machine learning algorithm that can be used either by itself or together with other learning algorithms to improve their performance. The general idea with AdaBoost is to take many weak classifiers (each of which can be trained to detect a specific feature, with a success rate of only slightly over 50%). The classifier distributes weights over all the classifiers depending on their performance. The AdaBoost classifier is highly suitable for object recognition purposes, and there are good implementations available.

2.5.2 Support vector machines (SVM)

Support Vector Machines (SVM) [5] are linear binary classifiers which when subjected to the descriptors from a test image look for an optimal hyperplane as a decision function, to decide to which class the test image belongs. The hyperplane which forms the decision function is high-dimensional, and this is why SVMs can classify data that is not linearly separable otherwise.

For this project I decided to classify using SVMs, and the main reason for this was that SVMs are easier to operate than AdaBoost, as well as being readily available in Matlab. The SVM implementation used here is the built in Matlab functions svmtrain and svmclassify from the Bioinformatics Toolbox.
2.6 Related work

As mentioned earlier, in this paper I use the HOG method as feature extractor, which is a decision-theoretic approach. I extend the set of images to also include digitally generated images, which in this case are images rendered using a 3D modeling software. This is not often done for decision-theoretic approaches, but for structural approaches it is common to use the spatial information from 3D models (often CAD models) to localize an object as well as determining it’s pose.

This approach is used by Zia et al. [21] who use photorealistic renderings of CAD models rendered from many different viewpoints as training data for shape context descriptors [1]. Their object is to find and determine the pose of cars, and they do this by comparing the shapes in the image to those from their training data.

Liebelt and Schmid [14] use an approach where a CAD model of an object is used to learn the spatial interrelation of a class (how the different parts are connected to each other). In this way, appearance and geometry are separated learning tasks, and provide a way to evaluate the likelihood of detecting groups of object parts in an image. Stark et al. [18] use a similar approach by letting an object class be described by its semantic parts (a car for instance, is made up by the parts right front wheel, left front wheel, windshield, right front door etc). This part-based object class representation contains constraints regarding the spatial layout as well as the relative sizes of the object parts. Another approach is the one used by Liebelt, Schmid and Schertler [15] where they extract a set of pose and class discriminant features from 3D models, called 3D Feature Maps. These synthetic descriptors can then be matched to real ones in a 3D voting scheme.

Another common approach is to construct a 3D model of an object using a number of 2D images, as done by Gordon and Lowe in [12]. This model can then be fitted to the features detected in an image (Gordon and Lowe use SIFT features), and thereby acquiring the location and pose of the object.

The approach I use here is to use photorealistic renderings of 3D models as training images, similar to an approach used by Heisele, Kim and Meyer [13]. They use a vast number of renderings (up to 40,000 images) of 3D models as training data to train a HOG classifier to detect different objects (e.g. a horse, a shoe sole etc). The test images they use are mainly digitally generated images as well, but they also include some photographs of the objects. They compensate for the lack of intraclass variability in the images by using a vast number of images rendered from different angles, as well as changing the light sources intensity and location relative to the object.
The approach chosen is similar but with a training image set consisting of both photos and digitally generated images, with a majority of the test images being photos.
3 Methods

3.1 Sliding window method

The training images all depict the objects close up, filling approximately half the image. In order for the classification (as well as the object localization) to work properly, small sub-images of the test images must be selected and classified one by one.

This is done using a sliding window approach, where a test image is searched a number of times with sliding windows of different sizes. In order to reduce the risk of an area being misclassified the windows move in step sizes equal to 1/5 of the window’s size, which means that each part of the image is classified numerous times.

3.2 SVM classification voting strategy

The classification process is carried out in two steps: in the first step the test image is searched using the sliding window approach, and each sub-image is classified as containing an object or not (without determining to which of the three objects it belongs). In step two each image area that is found to contain an object is evaluated again, to decide to which object class the supposed object belongs.

Since the SVM is a binary classifier and there are three different object classes, the classification process in step two can’t be done in only one step. To decide to which class an object candidate belongs, the one-versus-one voting strategy is used. In this strategy, the SVM decides which of the classes that have the highest similarity to the image; this is done by first comparing classes 1 and 2 to the image, then 1 and 3 and finally 2 and 3. The supposed object is sorted into the class that gets the most votes.

3.3 Creating the 3D models

Today companies often hold 3D representations of their products in the form of CAD designs, and CAD models are also widely available at e.g. Google 3D Warehouse, so if images created from 3D models could facilitate in an object recognition process much effort could be saved. Gathering all of the training images with a camera entails a lot of work, but creating images from 3D models is often simple in comparison.

The digitally generated images were constructed using a 3D modeling software called Blender, in which digital 3D model replicas of the objects to be recognized are built. The electrical outlet and the light switch were
created directly in Blender, and the Lindab model was created primarily in Blender but with a front containing a photographic texture of the real front. The reason for this is that the front of the Lindab device has logos and graphics on it that are captured best in a photo, so it makes the 3D model more similar to the real appearance of the Lindab.

The models were rendered in two different variants, the first with images rendered against a plain light grey background and the other rendered against a dark grey background, both with lighting and shadowing set to resemble an indoor environment. The ones with a light grey background more resembles photos of the objects, but the ones with a dark grey background has a higher contrast and give a more distinct image of the objects. Two screenshots of the Blender environment is shown in Figure 6.

![Figure 6: Figure 6 shows two screenshots of the light switch and the outlet models in Blender.](image-url)
From these models approximately 450 images were rendered, depicting the different objects from different angles. These images were included in the training data set. A number of test images were also rendered, depicting the objects included in some sort of scene, similar to the photographic test images.

The advantage of using images rendered from 3D models is that once a 3D model is created, it is easy to render an arbitrary number of images, from different angles and with different lighting etc. The disadvantage is that all images generated with similar settings (e.g. lighting, shadows and camera settings) will have quite similar features.

If real photos are used as training data, one gets intraclass variability (natural differences among images of objects belonging to the same class) in the data set automatically. Because of the lack of intraclass variability of computer generated images, real photos are often needed in the training data set. I used a training set that contains both photos and digitally generated images.
4 Implementation and testing

4.1 Implementation of the object recognition

The first step was to acquire a good implementation of the HOG method. Here I used a HOG implementation written by Mahmoud Farshbaf Doustar [7]. The gradients were calculated by applying the 1-D centered derivative mask in the horizontal and vertical directions. The normalization was carried out by calculating the L2-Hys norm (a maximum-limited version of the L2 norm). Doustar’s HOG implementation is fed with four parameters: the size of a cell, the size of the angular bins, whether the angle should be signed or unsigned, and whether the output should be a vector or a matrix. I set the cell size to 8 pixels, the angular bin size to 8, the output to vector and the angle to unsigned.

Before calculating the gradients the images were pre-processed using gamma correction. In gamma correction the illumination of the images is made more uniform by shifting the pixel values into a range where the human vision is more capable of differentiating changes. Without gamma correction an image may contain too much information in the very dark or very bright ranges and too little information in the ranges that the human vision is sensitive to, which results in low visual quality.

Here the gamma correction is carried out using the Matlab function \texttt{imadjust} with a gamma value of 0.5, which makes the images generally lighter and is a commonly used value for gamma. Gamma correction is an often used first step in object recognition methods, for HOG however it is slightly unnecessary since the block normalization acquires nearly the same results, but gamma correction was used nonetheless.

The gamma correction of the images is part of the image pre-processing. After this the HOG-descriptors are calculated for each image using Doustar’s HOG implementation. Below is a piece of Matlab code showing how the training phase is initiated by creating the SVMStructs used in the SVM classification.

After the descriptors have been extracted from the training images, they are sorted into matrices where each column corresponds to a descriptor. The matrix \texttt{TrainLiSw} corresponds to the descriptors from the light switch training images, \texttt{TrainLindab} is from the Lindab descriptors and \texttt{TrainOutlet} is from the outlet descriptors. The matrix \texttt{TrainNegTrain} consists of the descriptors from the negative training images, which are images which do not contain any of the three objects. These matrices are used to create SVMStructs, which are used in the classification process. In Matlab this is
done as follows:

\[
\begin{align*}
\text{Training}_1 &= \text{[TrainLiSw;TrainNegTrain];} \\
\text{Training}_2 &= \text{[TrainLindab;TrainNegTrain];} \\
\text{Training}_3 &= \text{[TrainOutlet;TrainNegTrain];} \\
\text{GroupLiSw} &= 1 \cdot \text{ones(numImgsLightSw,1);} \\
\text{GroupLindab} &= 2 \cdot \text{ones(numImgsLindab,1);} \\
\text{GroupOutlet} &= 3 \cdot \text{ones(numImgsOutlet,1);} \\
\text{GroupNegTrain} &= \text{zeros(numImgsNegTrain,1);} \\
\text{Group}_1 &= \text{[GroupLiSw;GroupNegTrain];} \\
\text{Group}_2 &= \text{[GroupLindab;GroupNegTrain];} \\
\text{Group}_3 &= \text{[GroupOutlet;GroupNegTrain];} \\
\text{SVMStruct}_1 &= \text{svmtrain(Training}_1,\text{Group}_1); \\
\text{SVMStruct}_2 &= \text{svmtrain(Training}_2,\text{Group}_2); \\
\text{SVMStruct}_3 &= \text{svmtrain(Training}_3,\text{Group}_3); \\
\end{align*}
\]

The matrices Training\(_1\)–3 contain the image descriptors stored column-wise. The vectors Group\(_1\)–3 are vectors indicating to which object class the columns in the Training-matrices belong (where zero corresponds to the negative training images and 1-3 to the three object classes respectively).

The resulting SVMStruct\(_1\)–3 are used in the searching phase, where a test image is searched using the sliding window approach described in section 3.1. Each sliding window image is rescaled to match the size of the training images, the HOG descriptors are calculated (with Doust\’s implementation findBlocksHOG) and the result is classified using the Matlab function svmclassify.

\[
\begin{align*}
\text{windowImg} &= \text{imresize(windowImg,[100 100]);} \\
\text{blocksTest} &= \text{findBlocksHOG(windowImg,0,8,8,\text{'vector'});} \\
\text{result}_1 &= \text{svmclassify(SVMStruct}_1,\text{blocksTest);} \\
\text{result}_2 &= \text{svmclassify(SVMStruct}_2,\text{blocksTest);} \\
\text{result}_3 &= \text{svmclassify(SVMStruct}_3,\text{blocksTest);} \\
\end{align*}
\]

The above described SVMStructs are used in the searching phase, when the test images are scanned looking for objects. In the classification phase the located objects are to be sorted into the correct object classes, and for this task another set of SVMStructs are used.

\[
\begin{align*}
\text{TrainingLiLin} &= \text{[TrainLiSw;TrainLindab];} \\
\text{TrainingLiOut} &= \text{[TrainLiSw;TrainOutlet];} \\
\end{align*}
\]
TrainingLinOut = [TrainLindab;TrainOutlet];

GroupLiLin = [GroupLiSw;GroupLindab];
GroupLiOut = [GroupLiSw;GroupOutlet];
GroupLinOut = [GroupLindab;GroupOutlet];

SVMStructLiLin = svmtrain(TrainingLiLin,GroupLiLin);
SVMStructLiOut = svmtrain(TrainingLiOut,GroupLiOut);
SVMStructLinOut = svmtrain(TrainingLinOut,GroupLinOut);

Since SVMs are binary classifiers an SVMStruct can only be used to separate between two different object classes, so three different structs were created, one deciding between the light switch class and the Lindab class, another between the light switch and the outlet classes and the last one deciding between the Lindab class and the outlet class.

windowImg = imresize(Isearch,[100 100]);
blocksTest = findBlocksHOG(Isearch,0,8,8,'vector');
resultLiLin = svmclassify(SVMStructLiLin,blocksTest)
resultLiOut = svmclassify(SVMStructLiOut,blocksTest)
resultLinOut = svmclassify(SVMStructLinOut,blocksTest)

The resulting values resultLiLin, resultLiOut and resultLinOut will have the values 1, 2 or 3 respectively, denoting which object class has the highest similarity to the test image. The image will be sorted into the object class with the most votes, as described in section 3.2.

4.2 Training images and test images

A large number of training and test images were gathered, a majority were taken with a camera but some were generated digitally using 3D modeling. The training images depict the objects close up from different angles, and the test images depict the objects at different scales and at different locations. Many test images contain more than one of the objects.

The number of test images and training images belonging to each class is shown in Table 1. The number of test images presented in Table 1b) are written as approximates, and the reason for this is that some of the objects in the test images are partially occluded or presented with poor illumination.
Table 1: Table 1a) shows the number of training images used as well as the distribution between photos and digitally generated images, and test images used, as well as the distribution between photos and digitally generated images. Table 1b) shows the corresponding numbers for the test images.

As seen in Table 1a) the number of negative training images is larger than the numbers for the three object classes. One reason for this is that other articles often use a higher number of negative training images, for example Zhu et al. [20]. Another reason is that the negative images are easier to gather, since one negative training image can be divided into many small images, as described below.

The training images depicting the three object classes are photographed as close-ups of the objects, as seen in Figure 7a). Figure 7a) also shows an example of what the digitally generated images look like. The negative training images are either photos of random motifs, or what might be a better approach, small parts of these photos. The test images are searched using a sliding window approach, so all of the images being tested will depict a small part of a larger image, which is why this approach might be favorable. An example of a negative training image is seen in Figure 7b).

Some examples of what the test images might look like is shown in Figure 7c).
Figure 7: Figure 7a) shows examples of the training images of the objects classes, the images shown in b) are examples of negative training images and c) shows examples of test images.
4.2.1 Varying the image sizes

A part of the evaluation was to investigate how the size of the training images affected the recognition results. One might expect that a low image resolution would result in lower accuracy, since they contain less information. Using small images has the advantage that the execution time is reduced, so if the difference in performance is little, small images are preferable.

The three images sizes that are evaluated are 64x64 pixels, 80x80 pixels and 100x100 pixels, and the size of the test images remains unchanged at 400x240 pixels.

4.2.2 False positives

When doing test runs of the implementation it is common to see false positive detections. A false positive recognition is when an image area is classified as containing one of the objects, even though it doesn’t. Figure 8 shows an example of a false positive recognition, where a fire extinguisher is recognized as an outlet.

![Image of false positive recognition]

Figure 8: Figure 8 shows a false positive recognition of a fire extinguisher that’s recognized as an outlet.

To improve performance false positives can be used as negative training images, to reduce the amount of false positives in the future. The false
positives are then divided into smaller images, and these are used as negative training images. In my training image set approximately 500 of the 1910 images originate from false positive recognitions, identified manually.

4.3 Image scanning algorithm

The test images are searched using a sliding window approach. The size of the test images is 400x240 pixels, and the sliding window searching is performed three times for each image, using the three different window sizes 120x120, 80x80 and 60x60 pixels. The sliding window moves with a step size equal to 1/5 of the window size. This means that every area of the image will be evaluated a number of times, and the threshold for an image area to be considered as containing an object is seven positive evaluations. This threshold was determined using a trial and error approach, by manually looking for a threshold which maximizes the number of found objects and minimizing the number of false positives.

Figure 9 shows an overview of the different steps in the sliding window algorithm. Every positive evaluation (independent of object class) is stored in a reference image (of the same size as the test image) that’s originally black (pixel value 0). Every positive recognition in the test image results in an addition of 10 to the pixel value (where 255 equals to white) in the corresponding area of the reference image (Figure 9b)). When the searching is completed, all values over 60 (seven recognitions or more) are set to 255, and all others are set to zero (Figure 9c)). After this the shape of the object is transformed into a rectangle, and the corresponding area of the test image is now assumed to contain an object, so it is singled out and classified (Figure 9 d) and e)).
Figure 9: Figure 9 shows the different steps behind the sliding window algorithm used in the classification process.
5 Results

The performance of the implementation was evaluated by varying two parameters, the size of the training images as well as testing with or without the digitally generated images in the set of training images.

The performance was measured by manually looking at the output images and evaluating them using a scoring system. The scoring system works accordingly: a correct recognition is awarded with 1 point and a correct localization but with wrong object classification gives 0.5 points. The points are measured against the total number of points available. A false positive recognition results in an addition of 1 point to the total number of points available in that image.

The results are presented as number of acquired points divided by the total number of points, in percentages.

5.1 Varying size of training images

The three training image sizes that were evaluated were 64x64 pixels, 80x80 pixels and 100x100 pixels. Table 2 shows the accuracy of the performance of the three image sizes, i.e. what proportion of the total number of points they earned in each of the test images categories.

<table>
<thead>
<tr>
<th></th>
<th>64x64</th>
<th>80x80</th>
<th>100x100</th>
<th>No. of Imgs</th>
</tr>
</thead>
<tbody>
<tr>
<td>Empty</td>
<td>12.5%</td>
<td>20.0%</td>
<td>50%</td>
<td>40</td>
</tr>
<tr>
<td>Outlet</td>
<td>53.5%</td>
<td>53.4%</td>
<td>53.7%</td>
<td>60</td>
</tr>
<tr>
<td>LightSw</td>
<td>34.8%</td>
<td>30.1%</td>
<td>31.6%</td>
<td>60</td>
</tr>
<tr>
<td>Lindab</td>
<td>48.8%</td>
<td>45.1%</td>
<td>41.1%</td>
<td>65</td>
</tr>
<tr>
<td>3D</td>
<td>35.6%</td>
<td>31.6%</td>
<td>31.1%</td>
<td>75</td>
</tr>
<tr>
<td>Total:</td>
<td>39.5%</td>
<td>37.9%</td>
<td>40.7%</td>
<td></td>
</tr>
</tbody>
</table>

Table 2: Table 2 shows the accuracy for three different sizes of the training images, 64x64 pixels, 80x80 pixels and 100x100 pixels, in each of the test image categories. The best overall performance is found in the 100x100 pixel implementation, though the difference is small. The number of test images in each category is shown in the column to the right.

The 100x100 pixel implementation is overall better than the two others, with a higher performance especially for the empty class. For the other classes however, the 64x64 and 80x80 pixel implementations generally perform equally good or better.
More specifically one can look at how many of all the objects in the test images that were localized and correctly classified. The results are shown in Table 3.

<table>
<thead>
<tr>
<th></th>
<th>64x64</th>
<th>80x80</th>
<th>100x100</th>
</tr>
</thead>
<tbody>
<tr>
<td>Outlet</td>
<td>46.1%</td>
<td>54.6%</td>
<td>52.0%</td>
</tr>
<tr>
<td>LightSw</td>
<td>46.1%</td>
<td>35.7%</td>
<td>31.2%</td>
</tr>
<tr>
<td>Lindab</td>
<td>43.2%</td>
<td>42.0%</td>
<td>26.0%</td>
</tr>
<tr>
<td>Total:</td>
<td>45.1%</td>
<td>44.0%</td>
<td>36.2%</td>
</tr>
</tbody>
</table>

Table 3: Table 3 shows the proportion of all objects in the test images that were correctly localized and classified for the different sizes of the training images.

Table 4 shows the number of false positive recognitions from the three image sizes. We see that the smaller image sizes have a higher return of false positives.

<table>
<thead>
<tr>
<th></th>
<th>64x64</th>
<th>80x80</th>
<th>100x100</th>
</tr>
</thead>
<tbody>
<tr>
<td>No. of false positives</td>
<td>137</td>
<td>154</td>
<td>99</td>
</tr>
</tbody>
</table>

Table 4: Table 4 shows the number of false positive recognitions returned from the three different training image sizes.

It is clear that the implementations with the smaller image sizes have a higher success rate when it comes to maximizing the number of found objects. The reason why their overall performance (Table 2) is slightly worse than that of the 100x100 pixel implementation, is because they return more false positives. An example of this is shown in Figure 10.
Figure 10: Figure 10 shows the differences between the images resulting from HOG 100x100 and HOG 64x64. HOG 64x64 finds more of the objects than it’s counterpart, but has also a higher return of false positives.

When including the digitally rendered images in the set of training images, the size of 100x100 pixels was used. Mainly because it performed slightly better overall than the other two sizes, but also because of it’s lower return of false positives.

5.2 Performance when including digitally generated images

The purpose of this project was to evaluate whether it is possible to use digitally generated images as training data for computer vision purposes. At an early stage, the possibility of using only digitally generated images in a set of training images was evaluated. It proved to be difficult to get positive classification results using this approach, most likely because the lack of intraclass variability in the digitally generated images.

The main focus was then to evaluate whether or not one can add digitally rendered images to an existing set of training images (consisting of photos) to get a higher performance. Onto the existing training set consisting of roughly 200 images per class, an additional 150 digitally generated images per class were added. All of the images were rendered from different angles, to give the as much information as possible about the objects. Two different digitally
rendered image sets were created, one containing the objects against a dark grey background, and one with the objects against light gray background, and these different sets were tested separately.

The classification result were negative, in the sense that the resulting classifications with digitally generated images included were identical to the classifications of the original implementation (without digitally generated images).

The testing strategy was to evaluate the results from using the two new training sets (with digitally generated images included, light gray or dark gray background). The evaluation would be carried out in the same way as the evaluation for the image sizes, measuring accuracy, correct classification and false positives.

The results when including digitally generated images into the training data set, however, were identical to the previous results. Every reference image (as described in Figure 9b)) was compared pixel-wise against the corresponding image classified by the original implementation, and they did not differ in a single pixel.
6 Discussion and conclusions

Some of the results of this project are unexpected, for example that the smaller image sizes have a higher performance when it comes to correctly finding and classifying objects. The total performance is however slightly worse than that for the larger images, since using the smaller image sizes leads to a higher return of false positives.

The reason for this might be because the smaller training images contain less information, hence they give a less clear image of the object classes. With their less distinct image of the objects, they have a lower threshold and will more easily give a positive output. This might be why they manage to find and correctly classify more of the objects, but also return more false positive recognitions.

Another unexpected result was that including digitally generated images in the training set didn’t change the resulting classifications at all, the images displaying the positive classifications didn’t differ in a single pixel.

The reason for this might lie in the synthetic nature of the gradients in digitally generated images. Not only do they differ from the gradients in photos, but they’re also very similar to each other. Therefore it doesn’t seem to matter if one adds the digitally generated images to the training data set or not, since they carry little information compared with the original data set, in which the variability was large.

6.1 Discussions on performance

The overall performance of the implementation is quite poor. More specifically, it can be said that the program has a lower hit rate near the image borders, since the sliding window algorithm will evaluate the areas in the middle of the image more times than the areas near the border. These effects might be reduced by adding a black border (~50 pixels wide) around the test images.

The runtime of the program is around a minute, which is high. The program probably could be made more efficient in many ways. Except for more efficient Matlab functions (the function eval which is known to be slow, is used for its lucidity) the implementation could probably gain from choosing another size of the angular bins. The angle that was used was 8°, and without loosing much accuracy a larger bin angle probably could be used (e.g. 30° or 45°), and this would probably reduce execution time substantially.

Regarding the image sizes, a better classifier than the one used here
might be constructed by using the 64x64 or the 80x80 pixels classifier in the sliding window phase, since they have a higher return of localized objects. The 100x100 could be used in the classifying phase to decide whether the images from the first phase are objects (as well as classifying objects) or if they’re empty (since the 64x64 and 80x80 classifier have a higher return of false positives).

Regarding the possibility of adding digitally generated images to a set of training images, it is possible that digitally generated images are more useful when other feature descriptor methods are used, rather than HOG. The HOG descriptor takes only the gradients into account, and the gradients in digitally generated images differ from gradients in photos. There are descriptors that are based on other image properties than gradients, for example the shape context feature descriptors, that are based on an object’s shape [1].

It is also possible that a better result is obtained if a higher number of digitally generated images is used. Heisele, Kim and Meyer [12] used 40,000 digitally generated images in their training data set, which consisted of digitally generated images depicting a shoe sole, an elephant, and other objects with distinct shapes. Here I only used 150 per class, so maybe a higher number of these images are needed to get the same variance as when photos are used.

### 6.2 Concluding remarks

The purpose of this project was to investigate if inclusion of digitally generated images as training data for an object recognition implementation could result in a higher performance. My findings were that including digitally generated images did not result in a higher performance. It is possible that better result can be obtained if other descriptors than HOG are used, or if a larger number of digitally generated training images is used.

The possibility of using digitally generated images as training images for object recognition purposes is worth studying further, since the benefits would be large. There exist many other methods of including information from 3D models in object recognition schemes, but training images are still used in many approaches and whenever this is the case, the usage of digitally generated images would greatly facilitate the gathering of training images compared to the alternative of collecting photos.
References


