1

System overview

1.1 Borcode detection overview

The system for detection of barcodes is basically divided in two parts. The data consist of a big amount of images which contain 1D- and 2D-codes. Each image has a corresponding ground truth where the parts of the image containing codes has been labeled as "true" and the rest as "false". The first part of the system involves the training using machine learning which produces a classifier. It consists of the following steps.

- Preprocessing of images.
- Divide images and ground truth into tiles.
- Calculate features for each tile.
- Use machine learning to train the dataset.

The objective of the second part of the system is to predict unlabeled data in real time. This is done with the trained classifiers and a cascade. It involves the following steps:

- Preprocessing of image.
- In each step of the cascade calculate some specific features.
- Use the corresponding classifier for each feature to reduce the amount of data between each step in the cascade.
- Postprocessing of the output.

An overview of the system can be seen in 1.1.

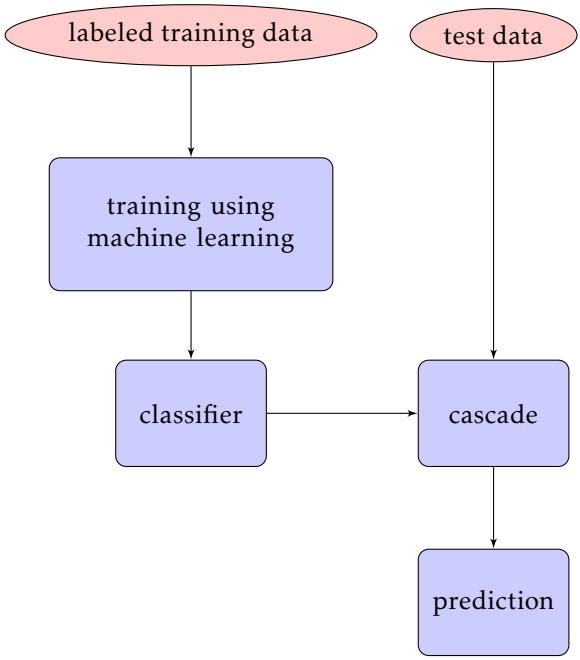

Figure 1.1: *Overview of the system*

1.2 OCR overview

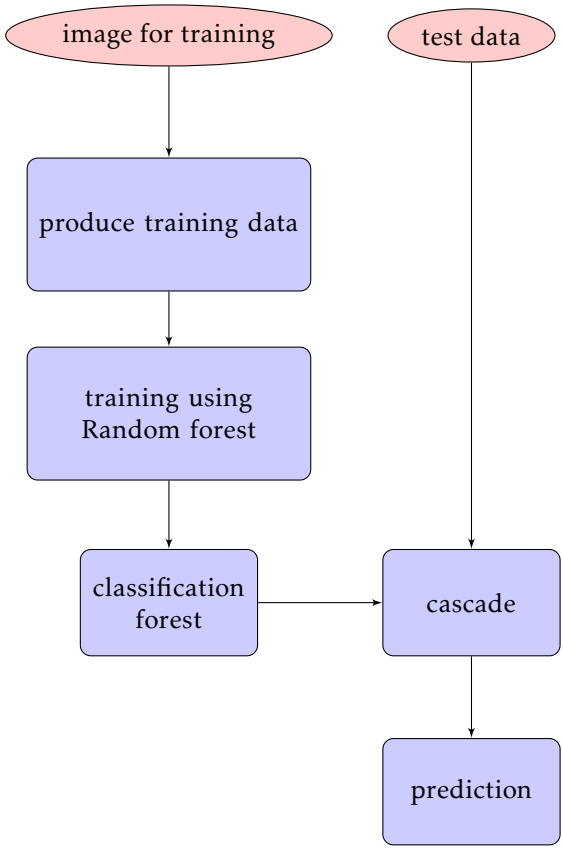

Figure 1.2: *Overview of the system for OCR*

2

Data processing

2.1 The dataset

The dataset consists of a big amount of gray scale images containing 1D- and 2D-codes of different sizes and orientation. For each image the corresponding ground truth are also available. The images are first of all divided into one training dataset and one testing dataset. Each image are then divided into tiles of the same size.

$$x = [x_1, ..., x_N]$$

With corresponding ground truth:

$$y = [y_1, ..., y_N]$$

The tiles can either overlap each other or just lay next to each other. For each tile a number of different features are calculated, i.e. each tile consists of a feature vector:

$$x_i = \begin{pmatrix} f_1 \\ \vdots \\ f_M \end{pmatrix}$$

The features are in some way based on the pixel values in the tile.

During testing each data will get a prediction. These can have four different states based on the relation between the prediction and the ground truth:

- true positive
- false positive

- true negative
- false negative

2.2 Preprocessing

Some of the features used in the system gives better result with some preprocessing. The reason for this is that the codes in some of the images have less contrast than others. Here a Laplace filter is applied to the images to make them a bit sharper. However some of the features gives better result without any preprocessing, therefore the filtered images are only used for calculation of some of the features.

2.3 Postprocessing

At the end of the cascade when all the test data have been predicted there are some postprocessing before obtaining the final result. The reason for this is to remove some possible true false classifications. The false classifications are often distinctive since the true true classifications usually lay in clusters. Here a filter is applied which removes isolated true classifications. After that some morphological operations are done to fill some possible holes in the areas where the codes are.

3

Machine learning

3.1 Classification

3.2 Discrete AdaBoost

For detection of barecodes the machine learning method discrete AdaBoost has been used which is described in [J. Friedman, 2000]. The algorithm is described in 3.1. The basic idea is to train a number of weak classifiers which during the testing will be combined to a strong classifier. To each data in the training dataset there are corresponding weights which are equal for all data at the beginning. The weak classifiers are trained sequentially and after each step the weights are adjusted depending if they were correctly or incorrectly classified. In each iteration all the training data are used.

Given: $(x_1, y_1), ..., (x_m, y_m)$, where $x_i \in X, y_i \in [-1, +1]$.
Initialize: $D_1(i) = 1/m$for $i = 1, ..., m$.

For $t = 1, ..., T$:

- Train weak learner using distribution D_t.
- Get weak classifier $h_t : X \rightarrow -1, +1$.
- Aim: select with low weighted error:

$$\epsilon_t = \sum_{i=1}^{m} D_t(i) I(y_i ! h_t(x_i))$$

where I is the split function

- Chose $\alpha_t = \frac{1}{2}(\frac{1-\epsilon_t}{\epsilon_t})$
- Update, for $t = 1, ..., m$:

$$D_{t+1}(i) = \frac{D_t(i) exp(-\alpha_t y_i h_t(x_i)}{Z_t}$$

Where Z_t is a normalization factor.

Final strong classifier

$$H(x) = sign(\sum_{t=1}^{T} \alpha_t h_t(x) - \varphi)$$

Figure 3.1: *Algorithm for discrete AdaBoost*

The split functions may consist of a number of different parameters. The most basic function, which has been used here, only has one parameter, a threshold. The function search for a threshold in one dimension at a time and choose the one which best separates the data.

3.3 Discrete Random forest

Random forest is described in [A. Criminisi and Konukoglu, 2011]. The algorithm creates decision trees which in the end are combined to a strong classifier. For each tree only a randomly chosen subset of the training data is used. This is in contrast to the AdaBoost algorithm where all training data are used in each iteration. The Ranom forest can be used for classification with more than two classes which is one of the advantage to AdaBoost. This makes it a lot more suitable for character recognition.

During training the trees are created one at a time with a random subset of training data. In each split node in a tree it is possible to use the whole feature vector or just a part of it. This depends on the amount of features. For a problem with a large amount of features it is common to chose a specific number of features randomly for every split.

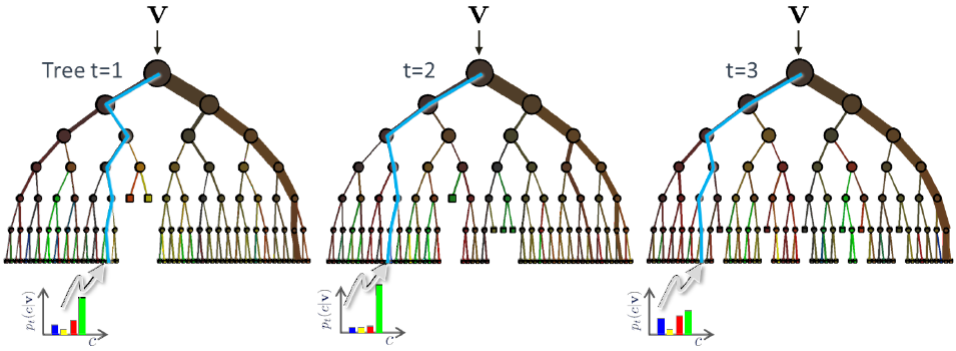

Figure 3.2: *Image illustrates testing using Random forest. Source: [**?**]*

4

Features

4.1 Features used to detect barcodes

Several different features has been tried out but many of them have not given any satisfying result. Here are the features that have shown best result.

4.1.1 Standard deviation

One good method to exclude tiles that don't contain any code is to compute the standard deviation of each tile. Tiles which contain code will have a high standard deviation; hence all data with standard deviation under a certain threshold can be discarded. This fast method to reduce the amount of data before applying other more complicated features.

4.1.2 Structure tensor

The structure tensor is an effective way to detect 1D-codes and to distinguish between 1D- and 2D-codes. The structure tensor for a tile is produced by first computing the gradients, this can be done by convolving the tile with a sobel filter. The structure tensor is then calculated in the following way:

$$T = \begin{pmatrix} (I_x)^2 & I_x I_y \\ I_x I_y & (I_y)^2 \end{pmatrix}$$

The structure tensor can be used in several ways to estimate the structure inside the tile. Areas that contain 1D-codes will have a one dimensional structure, i.e. the gradient in this area will only vary in one direction. On the other hand, areas that contain 2D-codes the variation will be fairly equal in both directions. The amount of variation of the gradient is measured by studying the eigenvalues of

the structure tensor. The following feature is calculated for each tile:

$$f = \frac{(\lambda_1 - \lambda_2)^2}{\lambda_1^2 + \lambda_2^2}$$

This feature is a measure of how likely the tile contains 1D-codes.

4.1.3 FAST corner detection

FAST (Features from Accelerated Segment Test) is described in [Rosten and Drummond, 2006] and [Rosten and Drummond, 2005]. This is a method to detect corners in images, which is a good way to distinguish between 1D- and 2D-codes. For a pixel p with intensity I, consider a circle of 16 pixel and a threshold t. The pixel is considered to be a corner if there exist a set of n contiguous pixels in the circle which are all brighter than I+t or darker than I-t. The algorithm is described in 4.1.

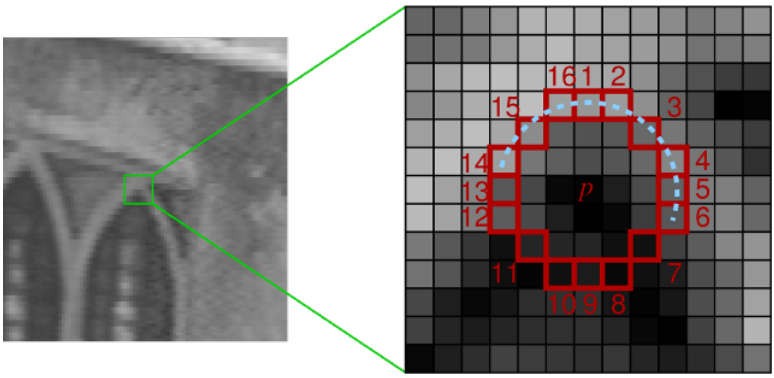

Figure 4.1: *Image describing FAST corner detection*

4.1.4 Distance map

A method for barcode detections using a distance map is described in [P. Bodnar]. The objective of this method is to first calculate the edges in the image by using the canny edge detection algorithm. After that a distance transform is used which in each pixel calculates the closest distance to an edge. There after the mean and standard deviation for the distance map are calculated which are then used as features. The distance map has in this system been used only for detection of 1D-codes.

4.1.5 Local binary pattern

Local binary pattern, which is described in [Pietikainen, 2010], is a method that can be used for detection of 2D-codes. The basic idea is to compute a binary code for every pixel, based on the difference of the intensity between the pixel and the surrounding pixels, illustrated in 4.2. The binary code will then be transformed

to a decimal scalar value. If a 3x3 neighbourhood is used there will be 256 different possible values. For each block a histogram will be calculated for all these values. Every bin in the histogram will then be used as a feature. If there is a bin for every possible value, there will be 256 features.

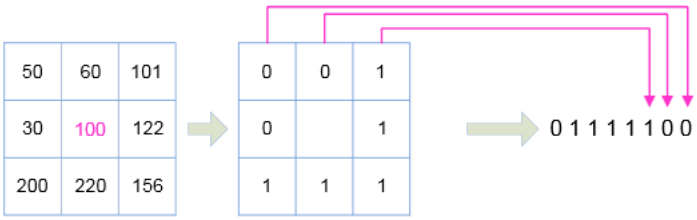

Figure 4.2: *Image describing local binary pattern*

4.2 Features used for OCR

Here are the features that have been used for character detection.

4.2.1 Two-rectangle features

The two-rectangle features are based on the same idea as the Haar-features presented in [P. Viola, 2001]. This method was used in an earlier project regarding OCR at SICKIVP and showed a rather good result. For that reason it has been tried out in this project as well. The main concept is to use a large number of filters in different sizes consisting of two areas. The result from the filters will then be the difference of the sum of the pixel values between the two areas. There are 17 different filters illustrated in 4.5 each in 64 different sizes. The sizes are in the range:

12x12, 12x16, 16x12, 16x16, 16x20, 20x16, 20x20.....40x40

The filters can also have different positions which depends on the size of the tile that is used. For example if the tile has the size 120x120 a filter with the size 12x12 can have totally 100 positions without overlapping each other. For a tile with the size 120x120 there will in total be 28578 different features.

Figure 4.3: *Image describing different types of two rectangle features*

To make the calculation of the rectangle features faster a s.k. integral image is used. This is a method presented in [P. Viola, 2001] for calculation of Haar-features. An integral image has the same size as the original image. The value for a pixel in the integral image is the sum of all the pixels above and to the left in

the original image. If the $i(x, y)$ is the original image the integral image $ii(x, y)$ will be the following.

$$ii(x, y) = \sum_{x'<=x, y'<=y} i(x', y')$$

4.2.2 Random point pairs features

One type of feature that has been tried out for OCR is a method presented in [M. Nenad]. The idea is to compare a large amount of point pairs in a tile. The point pairs are randomly chosen but with some constraint. The first point in each pair is chosen from the central part of the tile and the the second point is chosen from the whole tile. There is also a constraint of the distance between the points in each pair. The reason for this is that the character should be somewhat centralized in the tile. If both points are at the edge of the image or if they are too close to each other there will likely be less information to gain. The same point pairs are used for every tile both during training and testing. For each tile the corresponding pixel values are compared in the following way:

$$f_i = \begin{cases} 1 & I(p_1) < I(p_2) \\ -1 & I(p_1) > I(p_2) \\ 0 & \text{otherwise} \end{cases}$$

Each point pair will then correspond to a feature which can have one of three different values.

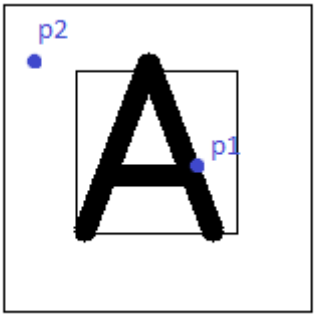

Figure 4.4: *Image describing two point pairs in a tile*

4.2.3 Random line features

The is a method similar to the random point pairs features. A large amount of point pairs are randomly chosen on the tile. Between the each point pair a line is defined.

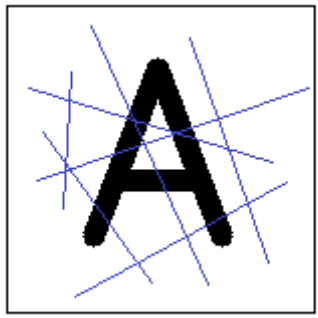

Figure 4.5: *Image describing some random lines in a tile*

5
Cascade

Since the system needs to work in real time it needs to be fast. One effective way to speed it up is to use a cascade when predicting the data. In the cascade one or several features are calculated at each step. In each step the corresponding classifier of the feature is used to classify the data. If the data is classified as true it will continue to the next step otherwise it will not be used any more. In the figure below the principle of the cascade is illustrated.

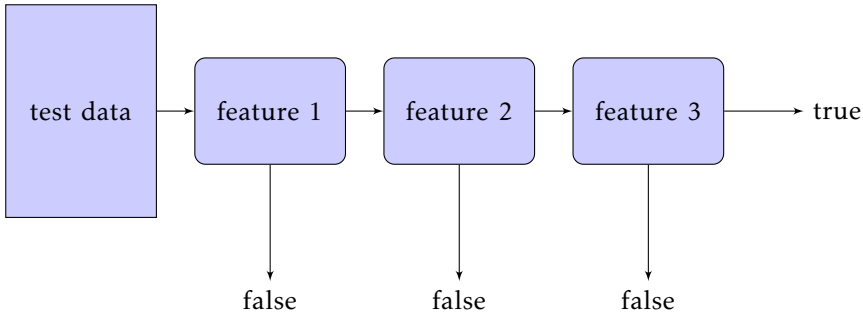

Figure 5.1: *Cascade model*

By testing different setups the cascade in 5.2 has given good results. Here the standard deviation is calculated first. This feature is not very heavy computationally and it also reduces the amount of data quite a lot. In the second step the structure tensor has been used since it is a good feature for separating 1D-codes from 2D-codes. In the last step the most computationally heavy methods are used.

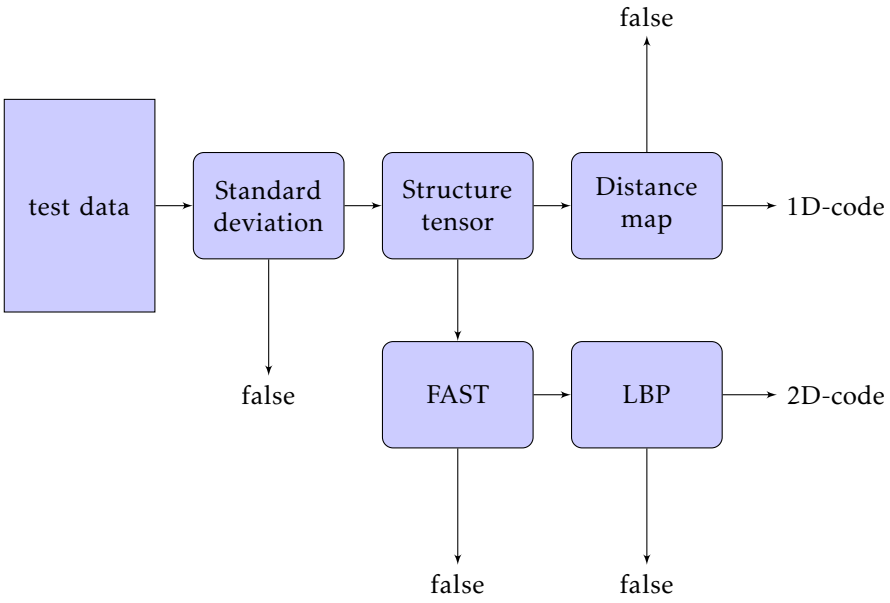

Figure 5.2: *Cascade model 1*

One alternative is to use the FAST corner detection in the second step instead of the structure tensor. This is an other good way to separates 1D- and 2D-codes. This cascade model is illustrated in 5.3

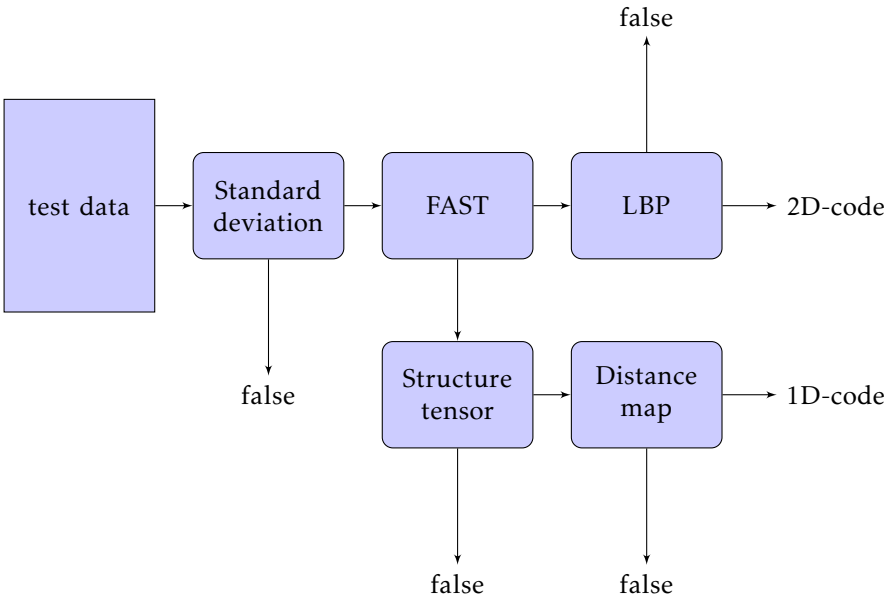

Figure 5.3: *Cascade model 2*

6 Evaluation

Here is a compilation of the evaluation that has been made for different methods for detections of barcodes. First some results from the different features are evaluated individually regarding their accuracy. Then the accuracy and the speed of the whole system is evaluated.

To get a measure of the accuracy of the system the number of true detections and the number of false detections are calculated and plotted. There are a lot of parameters and variation of the preprocessing of the tiles that will affect the result:

- Size of the tiles
- Use of overlapping tiles or not
- Use of down sampling
- φ from the algorithm in figure 3.1
- Use of Laplace filtering

In the evaluation 265 images has been used containing 1D-codes and 2D-codes. Of them 100 has been used for training and 165 for testing.

6.1 Calculation of ground truth

One of the problems that can occur with too big tiles and when if the down sampling is too high is that the ground truth is not calculated correctly. Since 70% of the tile has to be inside the code area there is a chance some codes will be missed out entirely. This is primarily the case for the 1D-codes since some of them are

very thin. There are not really any good way to evaluate this except pick out some difficult images and look at the result. If the the tile size is too big to detect some of the codes one solution is to make them overlap with each outer.

In the table below some tile sizes have been tried out with different down sampling. The values written in the table is how much overlap of the tiles which is necessary to achieve an acceptable result. The places in the table which is empty are cases when it's not possible to get a good result.

tile size	no down sample	down sample 2	down sample 3	down sample 4
24x24	1	1-2	3	
32x32	1	2-3		
48x48	2			
64x64	3			

Table 6.1: *Necessary overlap of the tiles to calculate ground truth*

6.2 Evaluation of features

Here is an evaluation of each feature that are used in the cascade. The objective is to keep as many true tiles as possible in each step of the cascade and in the same time reduce the amount of date as much as possible. In this evaluation the objective has been to preserve at least 98% of the true tiles in each step. Here different tile sizes and different amount of down sampling have been tried and best value of φ has been calculated to preserve 98% of the true tiles and as few false tiles as possible. Here only cases which gave a good result in the evaluation of the ground truth above are tested.

6.2.1 Standard deviation

When using standard deviation the images have been preprocessed with Laplace filtering and no overlapping tiles. It has been trained on both 1D- and 2D-codes.

tile size	no down sample	down sample 2	down sample 3
24x24	4	4	4
32x32	4	4	
48x48	3		
64x64	3		

Table 6.2: *Lowest value of φ that keeps at least 98% of the true tiles when using standard deviation*

tile size	no down sample	down sample 2	down sample 3
24x24	280	12.23	33.6
32x32	96.27	163	
48x48	77		
64x64	69		

Table 6.3: *The average number of false detection per image using standard deviation*

6.2.2 Structure tensor

When using structure tensor the images have been preprocessed with Laplace filtering and no overlapping tiles. It has been trained only on 1D-codes.

tile size	no down sample	down sample 2	down sample 3
24x24	1.5-2	2.5	bad result
32x32	1	2	
48x48	1.5		
64x64	1		

Table 6.4: *Lowest value of φ that keeps at least 98% of the true tiles when using structure tensor*

tile size	no down sample	down sample 2	down sample 3
24x24	226	121	bad result
32x32	92	228	
48x48	23		
64x64	28		

Table 6.5: *The average number of false detection per image using structure tensor*

6.2.3 Distance map

The distance map is trained only with 1D-codes and without Laplace filtering.

tile size	no down sample	down sample 2	down sample 3
24x24	1	1.5	2
32x32	1	1.5	
48x48	1		
64x64	1.2		

Table 6.6: *Lowest value of φ that keeps at least 98% of the true tiles when using distance map*

tile size	no down sample	down sample 2	down sample 3
24x24	90	15	71
32x32	35	23	
48x48	45		
64x64	52		

Table 6.7: *The average number of false detection per image using distance map*

6.2.4 FAST corner detection

The FAST corner detection feature is trained only with 2D-codes and without Laplace filtering. When down sampling with 3 the result seems to drop significantly, then it seems like it detects a lot of corners in the 1D-code.

tile size	no down sample	down sample 2	down sample 3
24x24	1.5	4	3
32x32	3	4	
48x48	2.2		
64x64	3		

Table 6.8: *Lowest value of φ that keeps at least 98% of the true tiles when using FAST corner detection*

tile size	no down sample	down sample 2	down sample 3
24x24	75	17	138
32x32	59	21	
48x48	80		
64x64	44		

Table 6.9: *The average number of false detection per image using FAST corner detection*

6.2.5 Local binary pattern

The LBP feature is trained only with 2D-codes and without Laplace filtering. The evaluation was in this case done together with the cascade. The reason for this is that it takes too long to test this features for all data. When using the cascade the data is reduced a lot before the step where the LBP is used. Also the training was done with 50 weak classifiers instead of 100.

tile size	no down sample	down sample 2	down sample 3
24x24		3	
32x32		2	
48x48	2		
64x64	2		

Table 6.10: *Lowest value of φ that keeps at least 98% of the true tiles when using LBP*

6.3 Evaluation of cascade

Here are some conclusions that can be drawn from the evaluation of the features before evaluating the cascade.

- There is no reason to use tiles with sizes 24x24 and 32x32 without down sampling with 2, since all feature seems to work in this case.
- To down sample with more than 2 seems to give really bad results with structure tensor.
- The cascade will be a lot faster with down sampling with 2 and tile sizes 24x24 and 32x32. This is because the amount of false tiles when using standard deviation are very low.

From these conclusions the evaluation of the cascade will be done for four different cases regarding the tile size, the down sampling and the overlap of the tiles. Also the cascades has been evaluated for the two different variants which was described in 5. Here is the result from cascade 1.

	ms/image	true detections in percent	false tiles per image
tile size 24x24 down sampling 2 overlap 1	75	1D: 98 2D: 95	1D: 0.7 2D: 0.14
tile size 32x32 down sampling 2 overlap 2	190	1D: 98 2D: 93	1D: 0.26 2D: 0
tile size 48x48 down sampling 1 overlap 2	420	1D: 96 2D: 94.5	1D: 0.3 2D: 0.097
tile size 64x64 down sampling 1 overlap 3	631	1D: 0.96 2D: 0.93	1D: 0.35 2D: 0.93

Table 6.11: *Result from evaluation of cascade 1*

Here is the evaluation of cascade 2.

	ms/image	true detections in percent	false tiles per image
tile size 24x24 down sampling 2 overlap 1	74	1D: 96 2D: 95	1D: 0.5 2D: 0.12
tile size 32x32 down sampling 2 overlap 2	210	1D: 97 2D: 94	1D: 0.18 2D: 0
tile size 48x48 down sampling 1 overlap 2	390	1D: 96 2D: 94.5	1D: 0.02 2D: 0.097
tile size 64x64 down sampling 1 overlap 3	630	1D: 0.96 2D: 0.93	1D: 0.18 2D: 0

Table 6.12: *Result from evaluation of cascade 2*

Bellow is the result from the first two tests of cascade 1 (first two row in table 7.11).

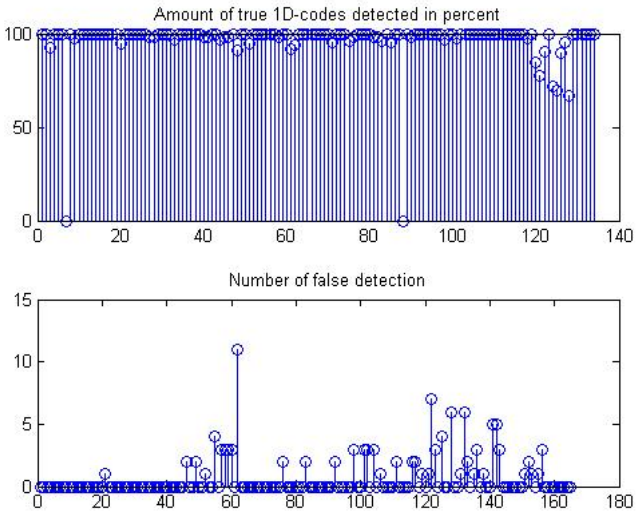

Figure 6.1: *Result for 1D-codes from evaluation of cascade using tile size 24x24, down sampling 2, and no overlap*

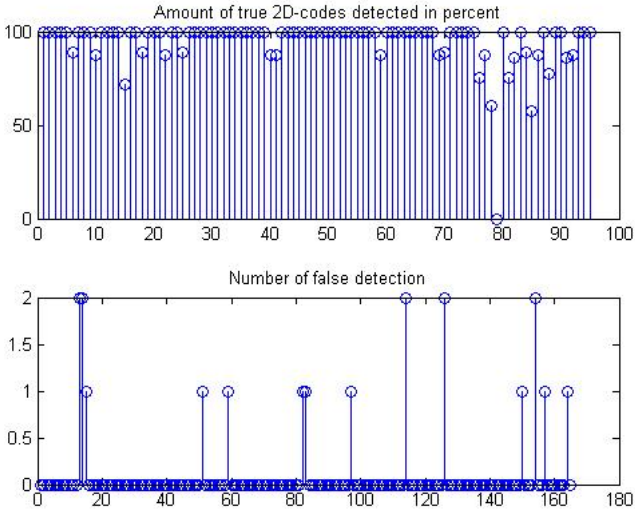

Figure 6.2: *Result for 2D-codes from evaluation of cascade using tile size 24x24, down sampling 2, and no overlap*

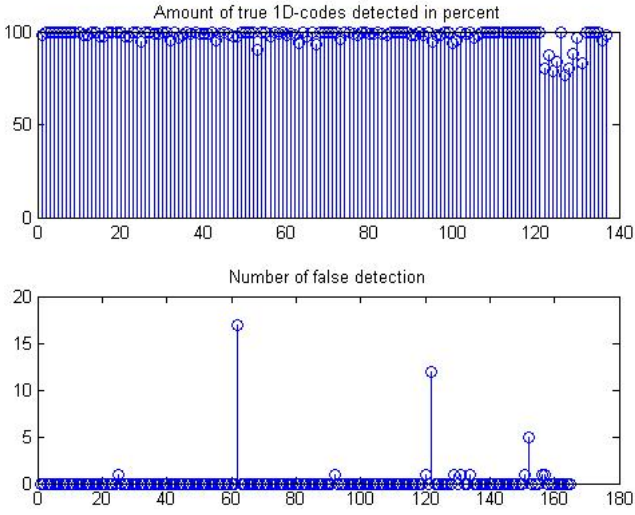

Figure 6.3: *Result for 1D-codes from evaluation of cascade using tile size 32x32, down sampling 2, and overlap 2*

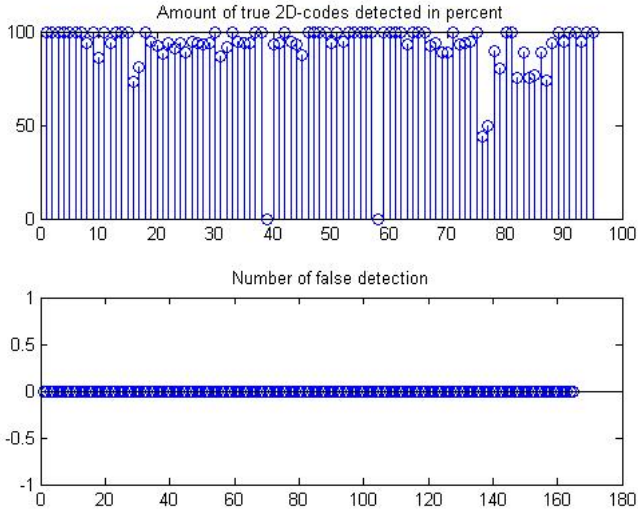

Figure 6.4: *Result for 2D-codes from evaluation of cascade using tile size 24x24, down sampling 2, and no overlap*

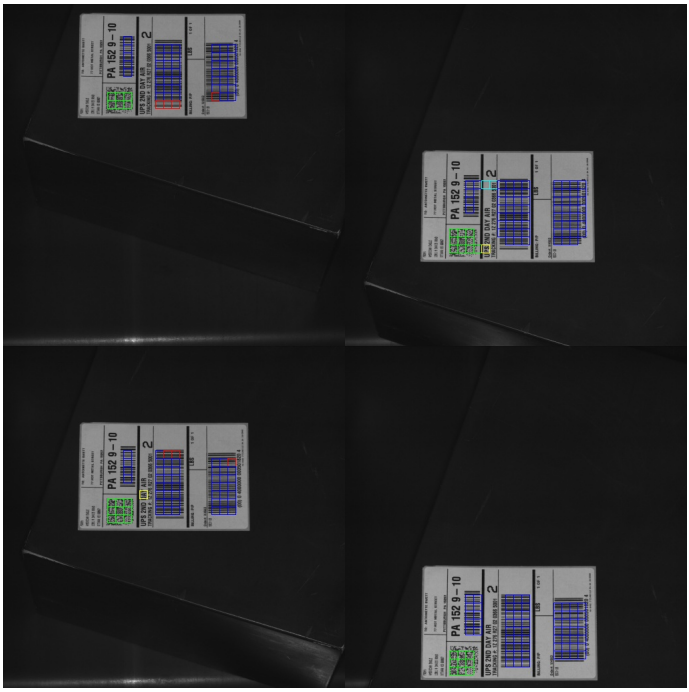

Figure 6.5: *Result from evaluation of cascade using tile size 32x32, down sampling 2, and overlap 2*

6.4 Conclusions

The two variants of the cascade gave rather similar results. Both the structure tensor and the FAST corner detection are rather good at distinguish between 1D- and 2D-codes.

The first two cases (the two first rows in table 7.11 and table 7.12) gives overall better result, especially regarding the speed. Which one of them that is best depend depend of what is required of the system, regarding speed and accuracy.

One alternative is to omit the distance map feature in the cascade and only use standard deviation and structure tensor for detection of 1D-codes. The result is still rather good, the amount of true detections is even higher but the amount of false tiles will increase.

The amount of true tiles detected can in some cases seem a bit weak, especially for 2D-codes. In the upper graph in 6.4 one can see that in one image there are no detections at all. This is the case when the codes are at the border of the image and only a small part are inside. Since in this case only a few tiles will cover the codes there is a big chance that these will be lost in the post-processing. The alternative is to reduce the amount of post-processing but this will instead lead

to more false tiles.

7

Further investigations

There are a lot of different 2D-codes that are currently used. The system has so far only been tested on Maxicodes. Since other kinds of 2D-codes have similar structure they will presumably give a very similar result. Maxicodes should be more difficult to detect then most other types of 2D-codes since it consists of small circular areas. Most other types of 2D-codes, e.g. QR-codes consists instead of small squares, this should make them even more distinguishable then Maxicodes.

The system could also be tested on a more complicated data set. If one would use a data set that contains more details in the background the system would probably detect a lot more false tiles. However many of the data sets that are available look very similar to the one that has been used.

Since there are not many details in the background in the current dataset the amount of data is reduced a lot when using standard deviation in the first step of the cascade. A dataset with more details in the background would probably lead to a much slower system. It would also probably detect more false tiles. The amount of details in the background is therefore very critical both for the accuracy and the speed.

To speed up the system even further one possibility could be to train a cascade for the local binary pattern alone. Since the LBP has 256 features it is likely that some of these features are more common than others for 2D-codes. If the system is trained to use these features in a certain order it would probably speed up the system. However since the LBP features have been used in the last step of the cascade and the amount of data already has been reduced a lot it is uncertain if this will make much impact.

Bibliography

J. Shotton A. Criminisi and E. Konukoglu. Decision forests for classification, regression, density estimation, manifold learning and semi-supervised learning. *Microsoft Research Cambridge*, 2011. Cited on page 8.

R. Tibshiani J. Friedman, T. Hastie. Additive logistic regression: A statistical view of boosting. *Stanford University*, 2000. Cited on page 7.

S. Igor J. Ahlberg R. Forchheimer M. Nenad, F. Miroslav. Localizing facial landmark points with pixel intensity comparisons organized in decision trees. *University of Zagreb, Linköping University*. Cited on page 14.

L. Nyul P. Bodnar. Barcode detection with uniform prtitioning and distance tranformation. *Univerity of Szeged*. Cited on page 12.

M. Jones P. Viola. Rapid object detection using a boosted cascade of simple features. *Cambridge*, 2001. Cited on page 13.

M. Pietikainen. Local binary pattern. *Scholarpedia*, 2010. Cited on page 12.

E. Rosten and T. Drummond. Fusing points and lines for high performance tracking. *IEEE International Conference on Computer Vision*, 2005. Cited on page 12.

E. Rosten and T. Drummond. Machine learning for high-speed corner detection. *European Conference on Computer Vision*, 2006. Cited on page 12.