Technische Universität Hamburg-Harburg
Vision Systems

Prof. Dr.-Ing. R.-R. Grigat

Machine Learning in Context of Robot Soccer on the Humanoid Nao Robotic System

Project Thesis

Malte Erik Schröder

January 29, 2016

Contents

List of Figures

List of Tables

Author’s Declaration

I solemnly declare that I have written this thesis independently, and that i have not made use of any aid other than those acknowledged in this thesis. Neither this research paper, nor any other similar work, has been previously submitted to any examination board.

Hamburg, January 29, 2016 Erik Schröder

Chapter 1

Introduction

This thesis is written within the robot soccer team of the TUHH, the *"Hamburg Ultra Legendary Kickers"* or *"HULKS"*. The used robot platform and details about the RoboCup, the event that the HULKs compete in, are introduced in the following section.

1.1 The NAO

The NAO robotic platform is manufactured by the french corporation Aldebaran. It is a humanoid robot with a height of 57.3 cm and a weight of 5.2 kg. As a CPU an Intel ATOM Z530 with 1.6 GHz clock speed is used. It can make use of 1GB RAM and has two cameras. One that looks straight ahead and the other one is used to observe the ground in front of the feet of the NAO. Both cameras have a resolution of 1280x960 pixel and can deliver a maximum frame rate of 30 fps [Aldb]. Figure 1.1 gives an overview about the positions of the two cameras.

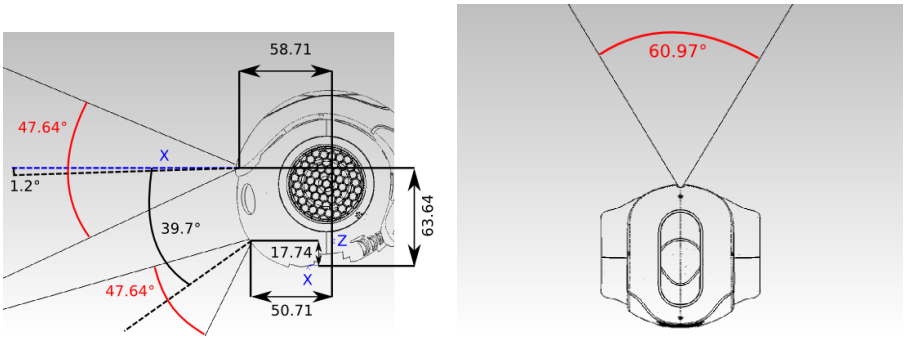


Figure 1.1: Overview about the cameras of the NAO [Alda]

1.2 RoboCup

The RoboCup is a competition for robots and artificial intelligence. The objective of the RoboCup is, to provide a motiviation for engineers and scientists in their work. The first RoboCup took part in 1997. Since then a lot of different competitions have been added to the RoboCup:

RoboCup Soccer
Soccer was the original competition in the RoboCup. The teams compete in different leagues. There are leagues for humanoid robots, robots on wheels or simulation. The HULKs take part in the Standard Platform League (SPL). In this league it is not allowed to modify the hardware and all teams have to use the NAO robot. The main goal is to beat the current human FIFA world champion in 2050.

RoboCup Rescue
The rescue league simulates catastrophic scenarios. The teams build robots that have to find casualties and provide first aid or rescue them. The robots are either controlled or they are working autonomously.

RoboCup@home
The RoboCup@home aims to develop robots that are able to execute various tasks in a home environment. For example they have to be able to get something out of the fridge or assisting people with getting their pills . A key challenge is the human-robot interaction.

RoboCupJunior
This competition is for children. The children compete in three different challenges. The Soccer Challenge, Dance Challenge and Rescue Challenge. It is meant to introduce children to robotics and technology.

Logistics League
The Logistics League tries to automate industrial processes. The robots have to provide resources at the right time at the right place to perform an efficient production chain.

1.3 Motivation

The current approach, that is used for image processing, is completely analytically. There exist different modules which are dedicated to a certain task.

These modules mostly depend on the results of their predecessors, which reduces the computational complexity of some tasks (e.g. the ball detection module doesn't have to search the whole image and iterates only through the field, if the field border detection provides this information). The disadvantage of this vision pipeline is, that the errors of all modules add up and therefore lead to a very high number of false detections. This approach produces very unreliable output and a lot of calibration and parameter adjustment has to be done to apply it to different environmental conditions.
It is hard to face these problems only with analytical methods. Another approach would be useful. Therefore a machine learning approach is evaluated in this thesis.

1.4 Aim of this Thesis

As mentioned before in section 1.3, the analytical approach has it's limits regarding the generalization and applicability. It is expected, that a machine learning approach promises better outcomes. Therefore different algorithms and methods should be evaluated in this thesis. The main issue about this topic is the high number of different methods and variations. It would exceed the limits of this thesis to consider all of them. To reduce the number of possibilities, this thesis will focus on a pixel based approach. A pixel is the smallest possible unit in an image and is therefore a reasonable start to begin with evaluating machine learning, because the training as well as the execution time will be small in comparison to more sophisticated approaches. The first part of this thesis will consider how a database has to be constructed and how the data has to be prepared, to make it usable for machine learning. In the second part an analysis of the data will be provided and finally in the third part different algorithms will be implemented and tested.

Chapter 2

The Data

For machine learning tasks it is mandatory to gather, understand and prepare the data [WFH11]. Therefore, this chapter discusses what is important to create a database of images, that can be used either for the evaluation as well as for the training of different machine learning methods.

2.1 Requirements

The topic, that is discussed here, is image processing. Therefore it is obvious that the initial data input for all operations are images. The important point is what these images shall contain. To specify details for the images, it might be convenient to start at a higher level. What requirements should the machine learning method meet in the end ? The most important points are:

1. Noise resistance
2. Lighting independence
3. As few false positives as possible
4. Model transfer
5. Object recognition

The first point is obvious and is required in nearly all machine learning tasks. Point *2.* aims on the different lighting conditions, which appear during the games. Even during one game the lighting conditions might change. This results in very unreliable results with an analytical approach. Therefore, a

high robustness and independence from the environment is required.
A special focus lies upon the points *3.* and *4.* The modules, that depend on the information from the image processing, are currently very sensitive to false positives. For example, if the goal keeper sees a ball in front of him, he immediately jumps and tries to save the ball. After the robot fell down, he likely looses orientation and it needs time until he is back in the game. This can result in a major disadvantage for the team. Therefore the false positive rate should be as low as possible or at least a confidence measure should be introduced. The fourth point, model transfer, means that it should be possible to copy the classifiers or algorithms after the learning phase to all robots. This would reduce the time and effort for deployment drastically, because the training of certain classifiers may consume a lot of time. The fifth point is also obvious. Of course the images shall include all the objects, that the algorithm shall detect.
Based on those basic requirements, it is now possible to break them down to more specific ones:

1. Noise Resistance
   (a) Images shall be taken with different camera parameters (gain, saturation, etc.)
2. Lighting Independence
   (a) Images shall be taken under different lighting conditions
      i. Different intensity levels
      ii. Different light sources with different spectrum and frequency
3. As few false positives as possible
   (a) Include images that can lead to false positives
      i. Other robots (with and without jerseys)
      ii. Items besides the field
      iii. Include images that are taken during movement
4. Model transfer
   (a) Include images from different robots
5. Object recognition

(a) Include images of the following objects with variations in position and orientation

i. Ball
ii. Goal
iii. Field
iv. Line
v. Robot

For the training set is is important, that it contains only one object of interest at the time. Even though there are training methods, that can teach an algorithm more than one object at the time, a disjoint training set can be applied for a wider range of training methods.

2.2 The Set Size

Another important point in machine learning is the size of the training set. There are several different public data sets, which are frequently used in papers and publications to evaluate algorithms and make the results comparable. Some data sets, that contain images, are quickly described below:

CIFAR-10/100 [Kri]
The CIFAR-10 dataset consists of 60000 32x32 color images divided in 10 classes with 6000 images each.
The CIFAR-100 dataset is very similar to the CIFAR-10 dataset. The difference is that it contains 20 superclasses, where each of them is divided in another 5 subclasses. Which makes 100 classese in total.

MNIST Database [Mni]
The MNIST Database of handwritten digits is a collection of 60000 training sets. The database is structured in 2 files. The first file contains all the labels, which indicate the ground truth, and the second file contains all the images.

It can be seen that a high number of images for a machine learning task is desirable. Unfortunately, there is no standard image database for robot soccer images yet. However, some of the SPL Teams have published their image datasets, so that these sets can be used for this thesis. But even with these images the size of the dataset can not reach the quantity of those listed

```
{
        "ball": [180,366,65,67]
}
```

Figure 2.1: An example JSON file, which is used to store the position of the objects, that can be seen in the image

above. This should be kept in mind for the further evaluation
To ensure that the learning algorithms work properly in a real environment, most images will be taken from real in-game data. Even if the in-game images provide realistic data, it might be convenient to add some more images which show the objects in some defined poses and angles. Therefore images which are made in the lab will be added to the database.

2.3 Ground Truth

To be able to use the dataset for supervised learning, labeled images are required. The positions of the objects of interest will be indicated by a bounding box. This additional information will be stored in an extra file besides the image. As a file format the java script object notation (json) [For14] will be used, because a parser is already available in the codebase. This file contains the type of the object, the upper left corner, the height and the width of the bounding box. Figure 2.1 shows an example of such a json file.

The bounding box method is not sufficient for all object types. For example the goal and the lines require a more precise labeling technique. For those object types a binary pixel mask is created. Figure 2.2 shows a pair of a line image and the pixel mask.

2.4 Database

The basis for the training set in this thesis is a collection of images, which was provided by the Nao-Team HTWK Leipzig [Htw]. A few more images were added from the HULKs Laboratory. This whole collection holds 13789 images of the upper camera, which were taken on four different occasions:

1. RoboCup Brazil 2014
2. Robot Hamburg Open Workshop (RoHOW) TUHH 2014

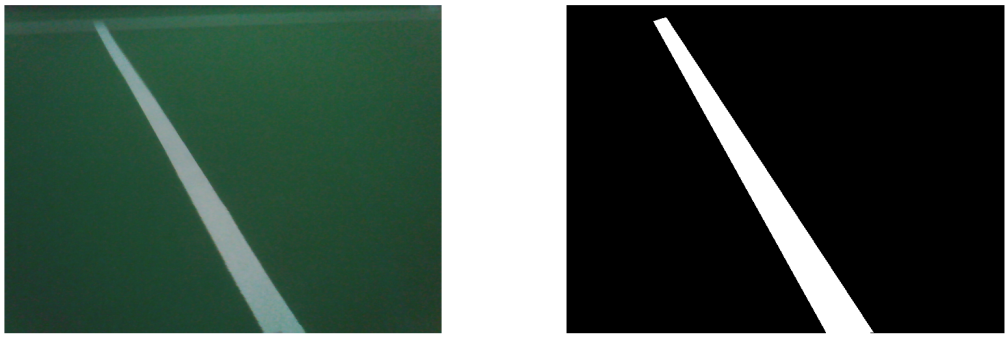


Figure 2.2: An image of a line with the belonging pixel mask

3. Laboratory HTWK

4. Laboratory HULKs

To gather a disjoint training set, these images were categorized in a way that only one object type can be seen. This results in a massive reduction in the number of valid images, but might be required for some machine learning methods. Table 2.1 gives an overview about the numbers of images and the number of extracted pixels for each object type.

Object Type	Ball	Field	Robot	Line	Goal
Number of Images	336	16	256	81	14
Number of extracted pixel	1478306	4915200	7871712	1052023	21035

Table 2.1: Overview over the images in the database

Table 2.1 shows that the number of images and therefore also the number of extracted pixels are varying a lot between the different classes. This might lead to a problem, because the prior probabilities of the classes are not consistent with the detection error costs [Mal03]. For example is the goal class much less represented than the field class. But the detection error for the field is not higher than the error for the goal class. Two different methods will be used in this thesis to solve this problem. The first method is over-sampling the data from the minority classes [LL98]. This can easily implemented by using examples from the minority class multiple times. The second method is under-sampling also called one-sided selection [KM97]. In this method the minority class stays untouched and examples in the majority classes will be removed until the classes are balanced.

	Ball	**Field**	**Robot**	**Line**	**Goal**	**Total**	**Partition**
un-sampled	1.478	4.915	7.871	1.052	0.021	15.338	3.067
over-sampled	4.915	4.915	4.915	4.915	4.915	24.576	4.915
under-sampled	0.021	0.021	0.021	0.021	0.021	0.105	0.021

Table 2.2: The number of mega-pixels in each set for different sample mechanisms

2.5 K-fold Cross-Validation

To make an efficient use of the existing data set, the k-fold cross-validation [RN02] method shall be applied for all further machine learning tasks. The idea of k-fold cross-validation is, that the whole data set is partitioned in k equally sized parts. For the training $\frac{k-1}{k}$ parts will be used and $\frac{1}{k}$ parts will be used for validation. This will be repeated k times. The classifier can than be assessed by the average error on the validation set. The disadvantage of this procedure is, that the already high computational complex learning process must be executed multiple times. Therefore the k will be set to five in this thesis. Table 2.2 shows the resulting set sizes for 5-fold cross-validation applied on the proposed sampling methods introduced in section 2.4.

Chapter 3

Image Analysis

In this chapter images from the robot shall be analyzed, to determine features, that could be used to classify objects, but first the color space, in which the robot perceives its environment, shall be introduced.

3.1 Colorspace

Even though the colorspace, that is used on the nao, is YUV422 [Aldb], the images are taken in the RGB colorspace. The images are converted per software afterwards. The YCbCr [IR94] color space consists of three channels. The following shows how they can be calculated from RGB:

Y-Channel

The Y-Channel represents the luminance. To calculate YCbCr from RGB, the values have to be gamma adusted and normalized. The new RGB triple will be denoted by R', G' and B'. The resulting luminance value Y' can be calculated as follows:

$$Y' = 0.299 \cdot R' + 0.587 \cdot G' + 0.114 \cdot B' \tag{3.1}$$

Cb-Channel

The Cb channel describes the deviation of the chroma from blue to yellow. Equation (3.2) shows how to obtain the Cb value from $R'G'B'$:

$$Cb = 128 - 0.168736 \cdot R' - 0.331264G' + 0.5B' \tag{3.2}$$

Cr-Channel

The Cr channel describes the deviation of the chroma from red to

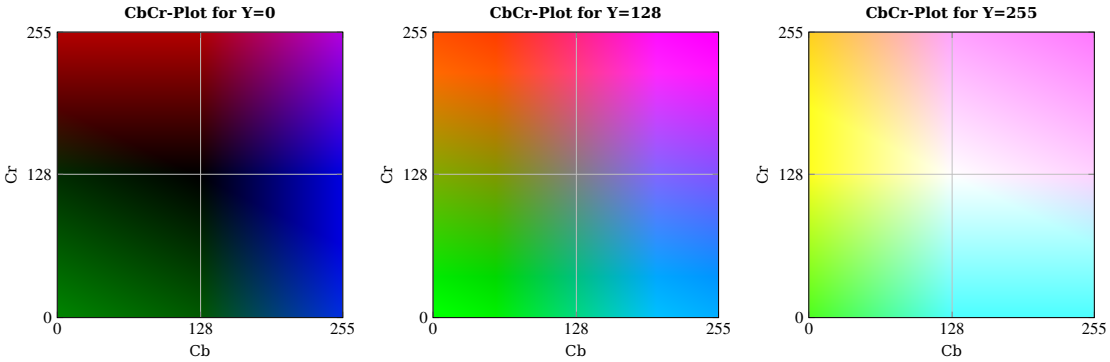


Figure 3.1: Comparison of the CbCr plane for different Y values

green. Equation (3.3) shows how the Cr value can be calculated from $R'G'B'$:

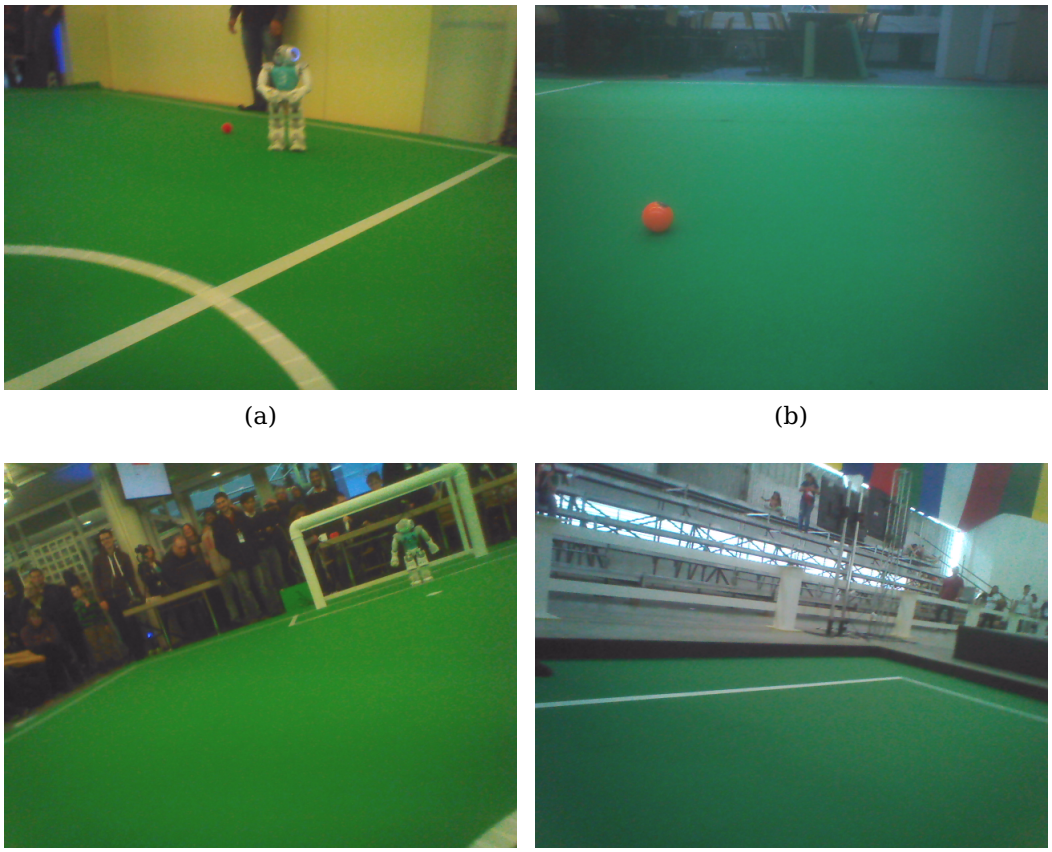
$$Cr = 128 + 0.5 \cdot R' - 0.418688G' - 0.081312B' \tag{3.3}$$

3.2 Color Segmentation

Figure 3.2 shows four images, that were taken by robots on different occasions. Because of the color coded environment, it is possible to classify each object based on the color. The challenge is, that every color can be perceived very differently by a robot. This is caused by different lighting conditions or e.g. different carpets. This section shall evaluate the five object classes and the representing pixels.

3.2.1 Field Color

Figure 3.2 shows, that the most significant feature on almost all images is the green field. Basically all objects, that should be detected, could be extracted by using all "non-green" regions. The following section shall show how the green of the field could be separated from all other colors.
To determine how different the green values can get, the figure 3.3 shows all the green pixels in one picture. Figure 3.4 shows the histogram of all these pixels. A more detailed look an different carpets under different lighting conditions can be seen in Appendix 7.



(a) (b)

(c) (d)

Figure 3.2: (a) The field is framed by a yellow wall. (b) The green as a blueish tone. (c) A lot of people watching the game from behind the goal. (d) A white fence separates the audience and the field.

One notable thing in the histogram 3.4 is, that the highest peaks in the Cb and the Cr channel are both at 128. This results from pixels, that are very low illuminated. Due to (3.2) and (3.3) the Cb and Cr values converge against 128 for low RGB values. When the Cb and Cr values are near 128 the color becomes quite indistinguishable (as can be seen in Figure 3.1).

To get a better idea of the distribution and how the green color could be distinguished from other colors, a three dimensional histogram plot in the CbCr-plane and also the YCb- and YCr-plane is shown in figure 3.5.

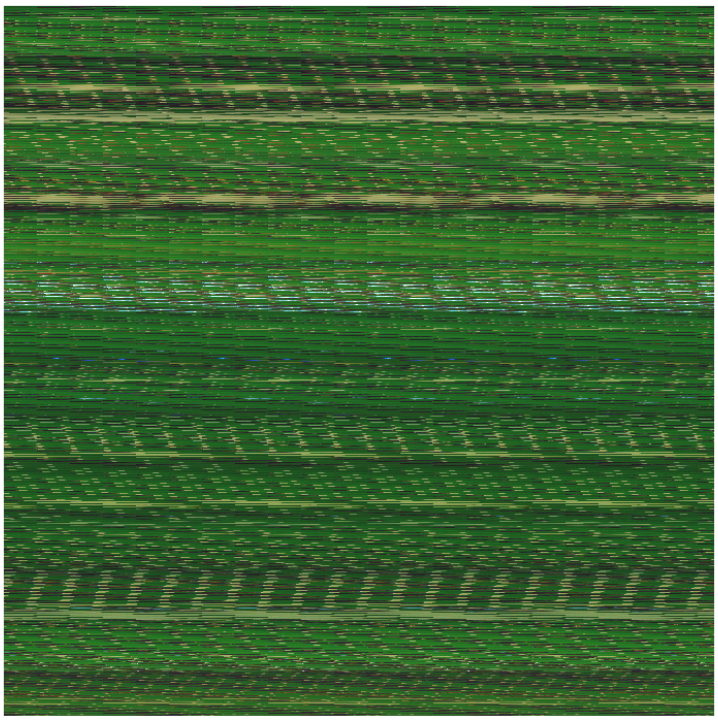


Figure 3.3: All green pixels

3.2.2 Robot Color

For the analysis of the field color, it was possible to select those images, that contain only green field. To obtain the pixels belonging to the robots, it is necessary to crop robot images. In order to do so, the existing bounding boxes are used. With them it is possible to cut the part with the robot out of the image. This method has the disadvantage, that always a few field pixels are classified as robot pixels. This label noise could lead to a lower prediction performance of some classifiers [FK]. Figure 3.6 shows all the pixels, that are classified as robot-pixels.

Figure 3.7 shows the three-channel histogram of 256 robot images and figure 3.8 shows the pixel density plot of those images.

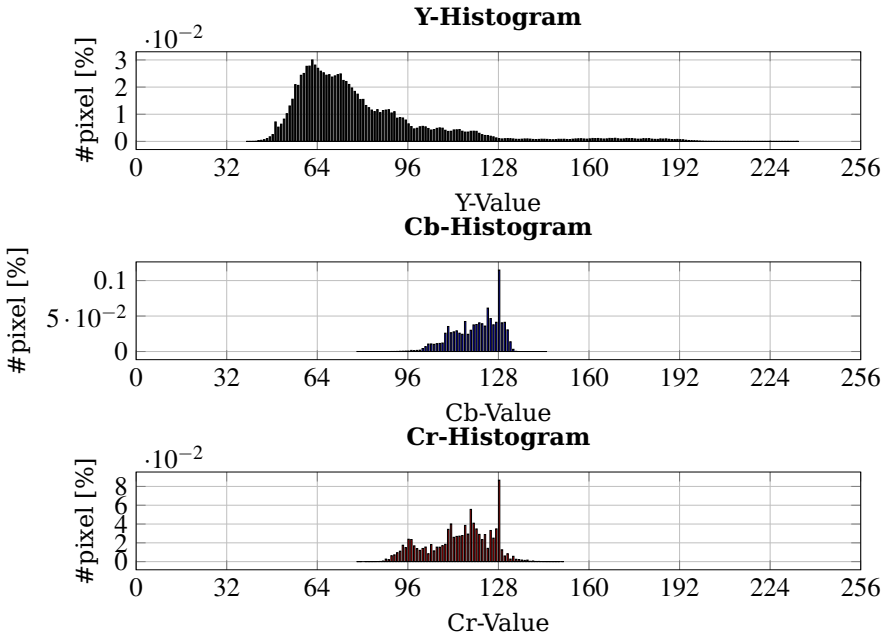
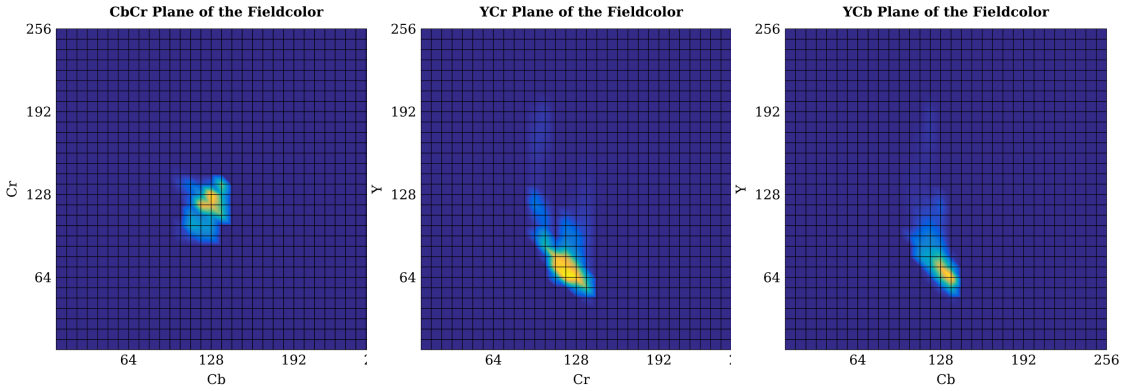


Figure 3.4: YCbCr Histogram over 16 field images



(a) (b) (c)

Figure 3.5: Pixel density plots of the YCbCr colorspace. Figure (a) shows the CbCr plane. The majority of the pixels are in the bottom left quarter. The maximum is in the middle at the point (128,128). The reason for that can be seen in the figures (b) and (c), which show the pixel density in the YCr- and YCb-plane of the colorspace. If the Y-channel becomes low, the Cb and Cr values shift through 128.

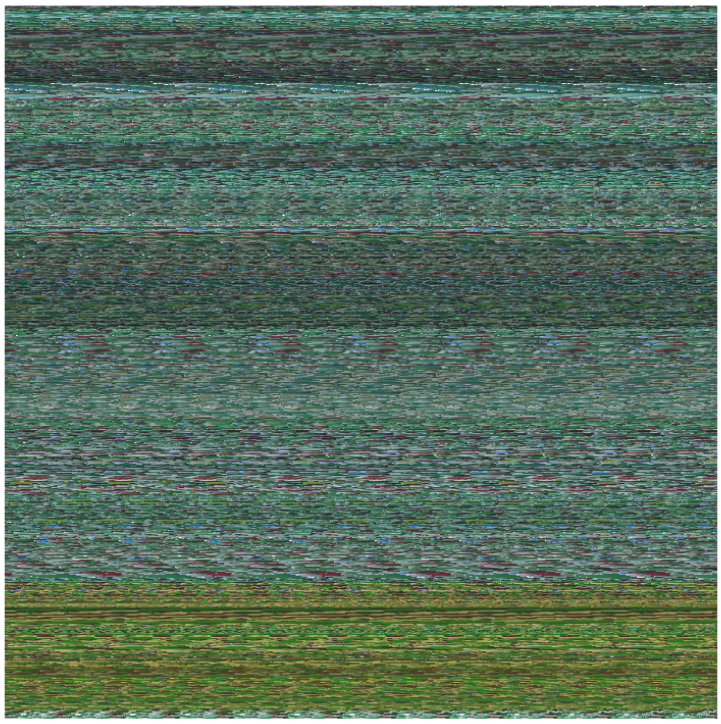


Figure 3.6: All robot pixels

3.2.3 Ball Color

To obtain the pixels from the ball, the same method as in section 3.2.2 is used. All the ball pixels are shown in figure 3.9, the associated histogram in figure 3.10 and the pixel density plot in figure 3.11.

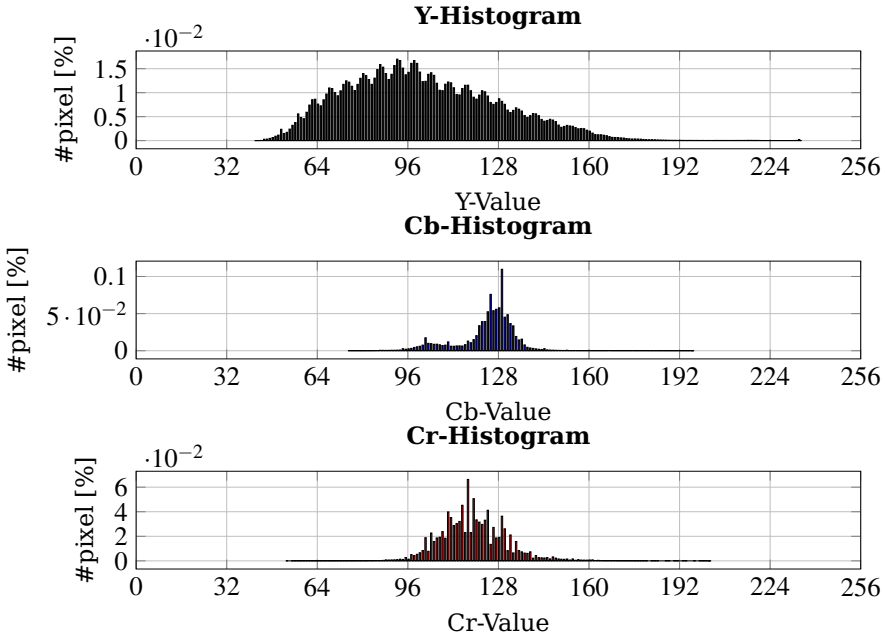
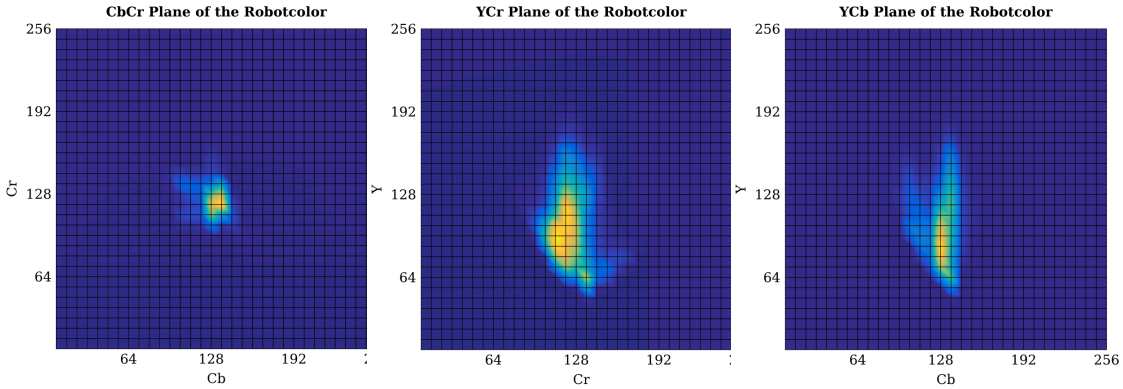


Figure 3.7: YCbCr Histogram over 256 robot images



(a) (b) (c)

Figure 3.8: The pixel density shown in figure (a) is very centralized. Only a few pixels are arranged around the center. Figure (b) shows that the robotcolor is mostly below 128. The Cb Values are distributed around 128 along the Y axis.

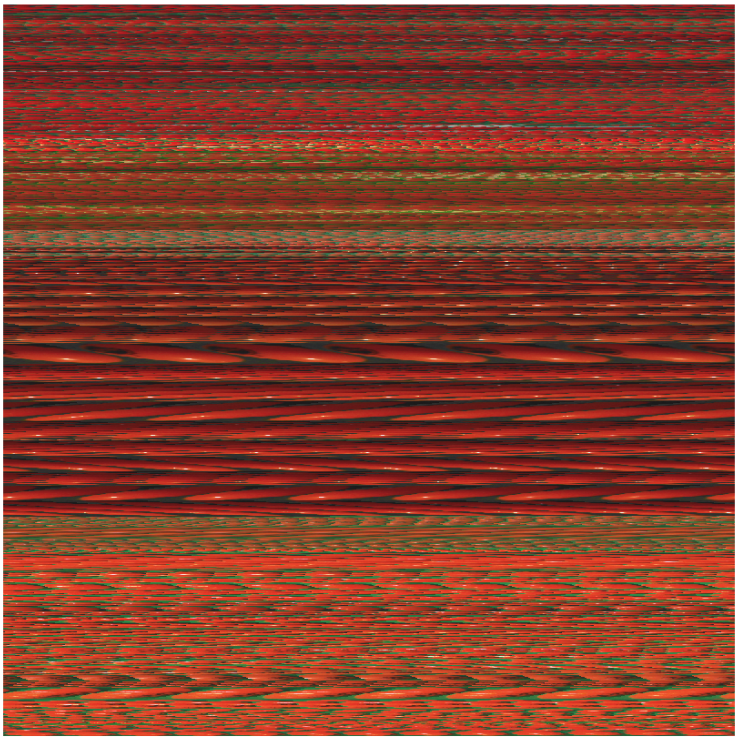


Figure 3.9: All ball pixels

3.2.4 Line Color

All the used line color pixels can be seen in figure 3.12 and the resulting histogram and pixel density plots can be seen in figure 3.13 and 3.14.

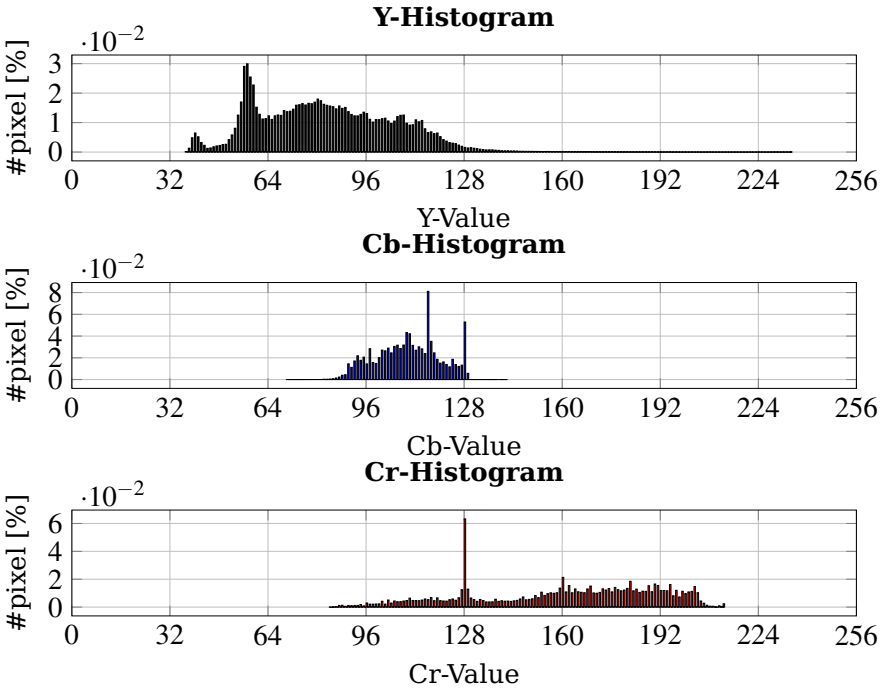
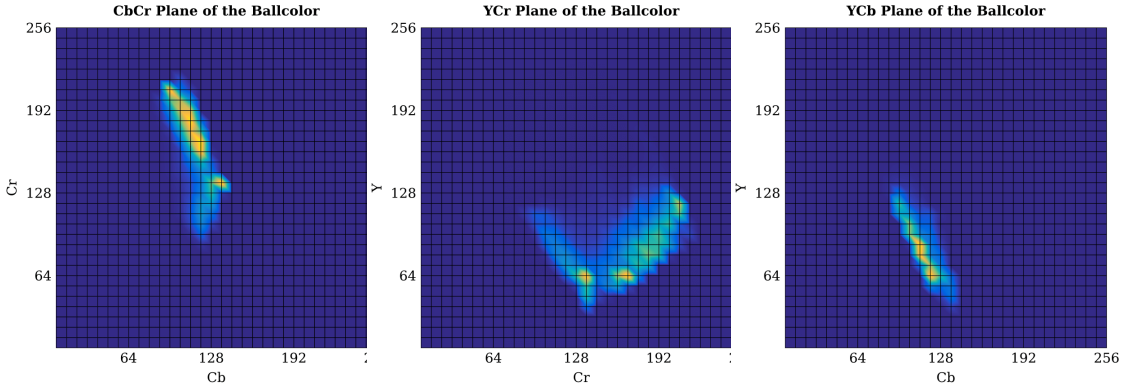


Figure 3.10: YCbCr Histogram over 336 ball images



(a) (b) (c)

Figure 3.11: The distribution of the red ball pixels in figure (a) shows a very individual pattern in comparison to the other and are almost all in the upper left quadrant. Figure (b) shows two peaks at the same Y Value. The rest of the pixels are arranged in some sort of "v-shape". The Cb value in figure (c) seems to be proportional to the Y-channel.

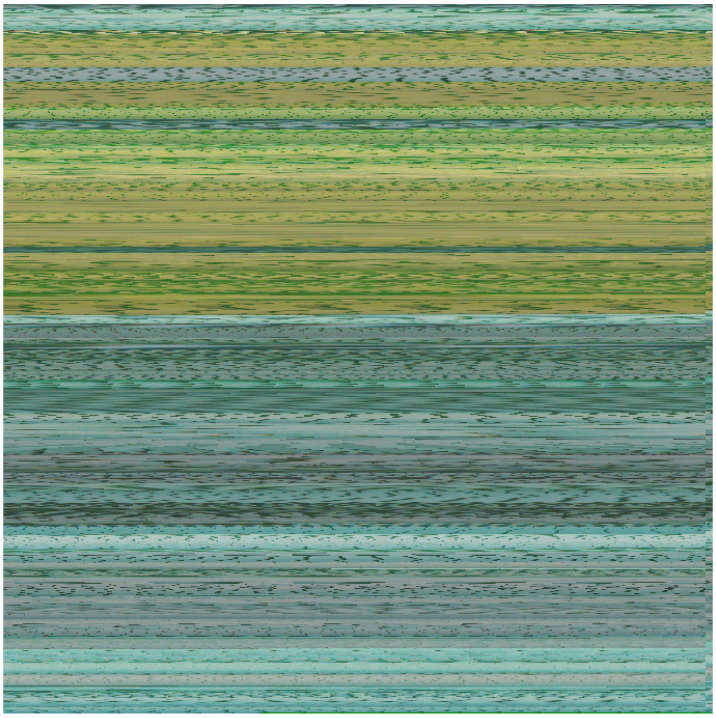


Figure 3.12: All line pixels

3.2.5 Goal Color

The resulting histogram and pixel density plots can be seen in figure 3.16 and 3.17.

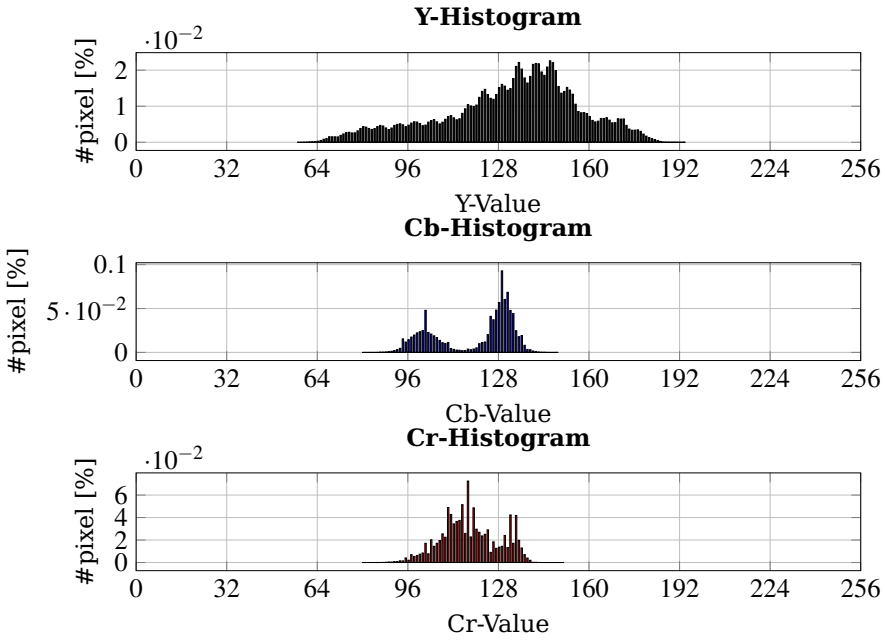
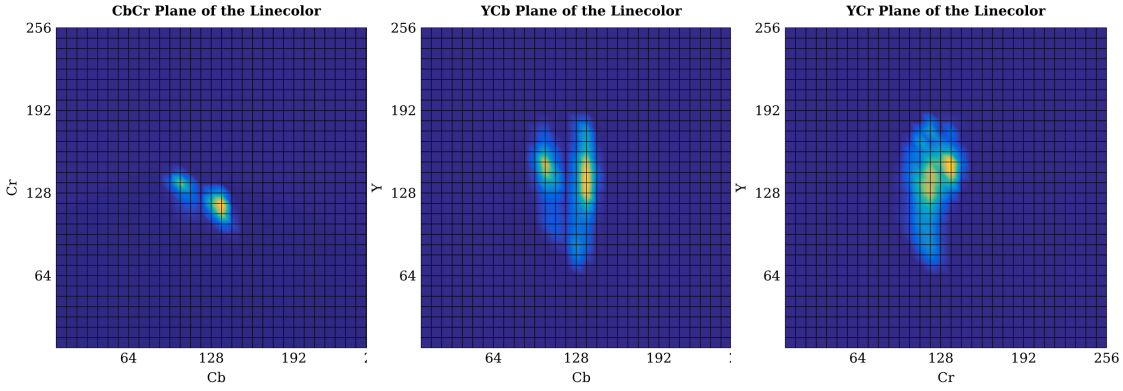


Figure 3.13: YCbCr Histogram over 81 line images



(a) (b) (c)

Figure 3.14: The linecolor is mainly white, therefore the majority of the pixels have an Cb and Cr values of 128 and the Y value is higher than in the other classes. One thing, that is noticeable here, is ,that there are two separate accumulations. Therefore the images must be taken from at least two fields with very different conditions. Figure 3.12 shows also two different color classes. It appears to be, that the surrounding field color has a big influence on how the linecolor is perceived.

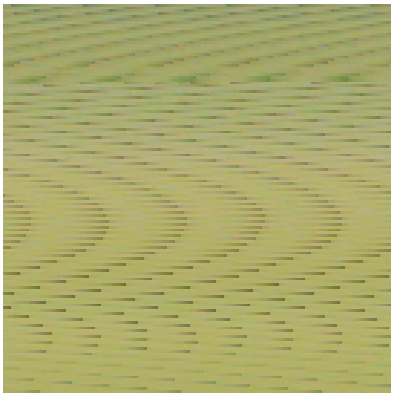


Figure 3.15: All goal-pixels

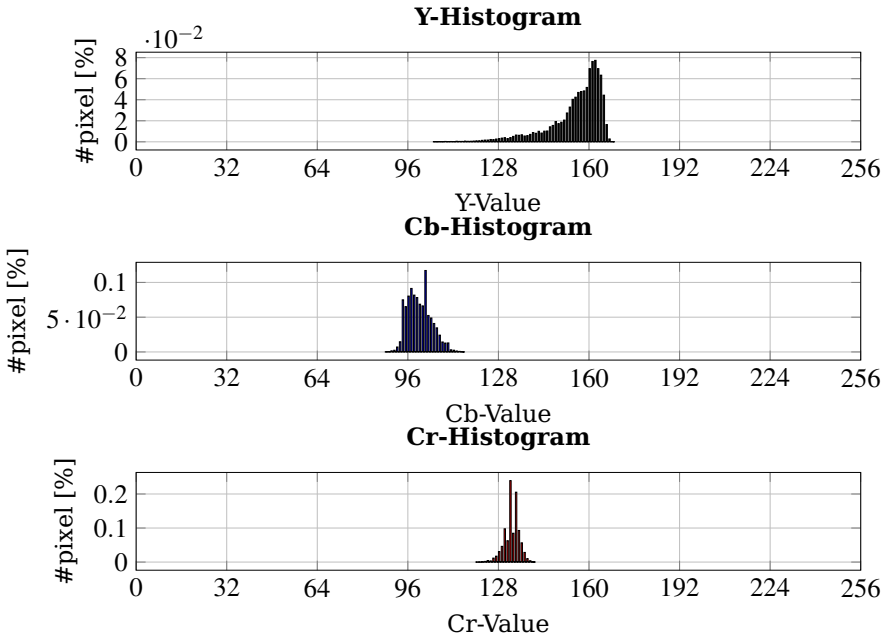
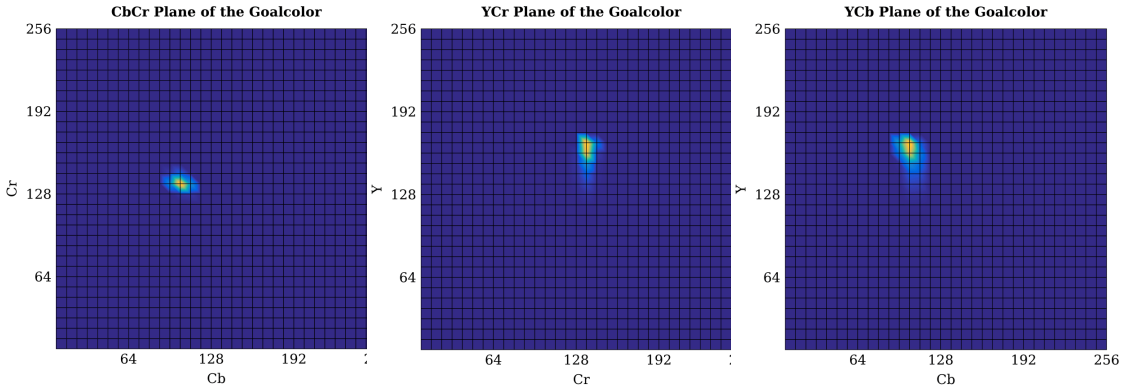


Figure 3.16: YCbCr Histogram over 14 goal images

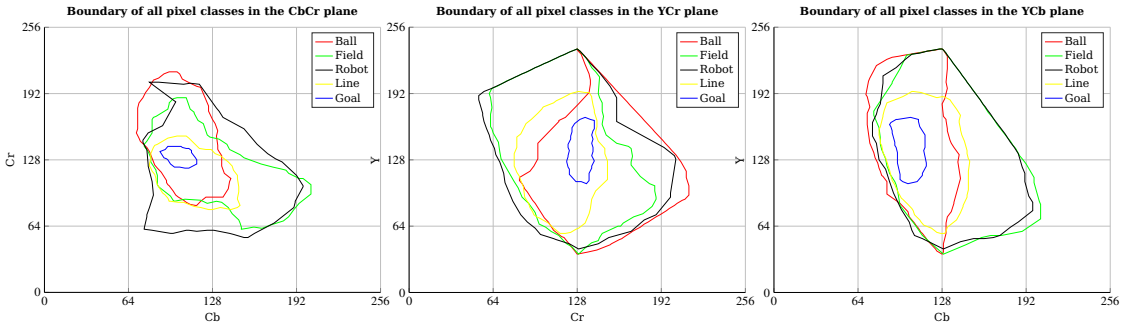


(a) (b) (c)

Figure 3.17: The distribution of the goal-pixels is more concentrated as the linecolor. One reason for that might be the small amount of images, that was available. But nonetheless the center of the distribution of the goalcolor has a lower Cb-value and the Cr-value is above 128. The figures (b) and (c) show also a more concentrated distribution, which also might be the result of the small data set

3.2.6 Conclusion

The sections above have shown that the five different pixel classes overlap a lot. Figure 3.18 shows the boundaries of the five classes projected on the three planes of the YCbCr space. It is obvious that those classes are not linear separable. A classifier, that separates these classes, should therefore be able to separate in a three dimensional space in non-linear way.



(a) (b) (c)

Figure 3.18: The Boundaries of each pixel class plotted in the three different YCbCr planes. The two biggest classes are the the field and the robot class. They are also very similar. The reason for that is probably the high label noise in the robot training data. It should be noted that these plots take all pixels into account and don't consider the density of the given classes.

Chapter 4

Machine Learning

As stated above a non-linear classifier is needed. There exists a vast variety of different machine learning methods, that are able to operate in a non-linear manner. But to achieve a feasible performance on the robot, there are a few more requirements, that the algorithm should meet:

Execution Time
The whole processing time is fixed by the frame rate, which is set to 30 FPS. Hence, there are only $\frac{1}{30}$ seconds available for a whole cycle. In this cycle the calculations for vision, motion and the AI have to be performed. Therefore the execution time for the pixel classifier is a very hard constraint and should be as low as possible.

Memory efficiency
The NAO robot provides 1 GB of RAM and a 8 GB SD-Card. This might be enough for a lot of applications. But might be a limiting factor when it comes to some learning methods, where the storing of a whole database is necessary (e.g. instance based approaches like KNN).

The different machine learning approaches can be categorized by the following two categories [RN02]:

Parameterized models
These models usually have a certain parameter set, which is determined in a learning phase. Once these parameters are fixed, the model changes no more and can be deployed. Such classifiers are for example:

1. Artificial Neural Networks

2. Polynomial Classifiers
3. Linear Regression

Parameter free models

Models without parameters are often also called instance based models. The main difference is, that they perform the classification based on the examples, that they have stored in some way. One advantage is, that some of those approaches are able to adapt to the current setting, because they are always updating the database and are therefore able to perform online-learning. Some examples for paramter free models are:

1. K-Nearest Neighbor
2. K-Means Clustering
3. Support Vector Machines

The following section introduces different machine learning methods and evaluates their applicability to the defined classification problem.

4.1 K-means Clustering

The following section is based on [Bis06].
The K-means clustering algorithm is capable of processing multidimensional data and assigns each data point to a certain cluster. The total number of clusters is denoted by K. The cluster centroids are denoted by $\boldsymbol{\mu}_k$, where $k = 1, ..., K$. The D-dimensional vector $\boldsymbol{\mu}_k$, where D is the dimension of the input, represents the cluster middle-point for the k^{th} cluster. Each data point is then assigned to the cluster with the smallest euclidean distance to the corresponding middle-point.
In order to determine $\boldsymbol{\mu}_k$ an objective function is set as:

$$J = \sum_{n=1}^{N} \sum_{k=1}^{K} r_{nk} \parallel \mathbf{x}_n - \boldsymbol{\mu}_k \parallel^2 \tag{4.1}$$

Where N is the number of data points, K is the number of clusters, $\mathbf{x}_n$ is the n^{th} data point and r_{nk} is a N-by-K binary indicator, that indicates to which of the clusters the n^{th} sample is assigned. To find r_{nk} and $\boldsymbol{\mu}_k$ that minimizes J, the EM-Algorithm can be used. For the first iteration of the algorithm the $\boldsymbol{\mu}_k$ can be chosen arbitrarily. The EM-Algorithm then consists of two steps:

Expectation (E-Step)
In the E-step the r_{nk} will be determined. Each data point will be assigned to it's nearest cluster, in order to minimize the sum of squared distances. Equation (4.2) expresses this assignment for every $n = 1, ..., N$. During this step the $\boldsymbol{\mu}_k$ is constant.

$$r_{nk} = \begin{cases} 1, & \text{if } k = \arg\min_j \| x_n - \boldsymbol{\mu}_j \|^2 \\ 0, & \text{otherwise} \end{cases} \tag{4.2}$$

Maximization (M-Step) In this step the r_{nk} will be constant and the the expression (4.1) will be minimized with respect to $\boldsymbol{\mu}_k$. This can be done by setting the partial derivative of J equal to zero:

$$\frac{dJ}{d\boldsymbol{\mu}_k} = 2\sum_{n=1}^{N} r_{nk}(x_n - \boldsymbol{\mu}_k) = 0 \tag{4.3}$$

Which can be solved to:

$$\boldsymbol{\mu}_k = \frac{\sum_{n=1}^{N} r_{nk} x_n}{\sum_{n=1}^{N} r_{nk}} \tag{4.4}$$

The denominator in equation (4.4) can be interpreted as the number of data points, that are assigned to the cluster k and the nominator is the addition of all the data points assigned to cluster k. Therefore it expresses the average value for all points assigned to the cluster.

4.1.1 Identifying the Cluster

K-means clustering is an unsupervised learning algorithm. During the training process no expected output is fed into the system, therefore the cluster middle points are adjusted only based on the minimization process. This results in the problem ,that after training, there are five different clusters, but it is not known which cluster is for example representing the ball. Hence, it is necessary to find a mapping from the training classes to the cluster.
At first each class will be encoded by an integer number: ball = 1, field = 2, goal = 3, line = 4 and robot = 5. The given training set $\mathbf{x}$, comes along with a result vector $\hat{\mathbf{y}}$, that is encoded in the described way. After the training process, the K-means algorithm returns a set of five middle points and the vector $\mathbf{y}$, that describes how the training examples were classified. The elements in the vector $\mathbf{y}$ only describe to which cluster the example belongs. To

decide which cluster belongs to which pixel class, all $M = 5! = 120$ permutations of possible combinations must be evaluated. The variable α_{nm} describes an element wise "not-equal" function for each element of $\mathbf{y}$ and $\hat{\mathbf{y}}$.

$$\alpha_{nm} \begin{cases} 1, & \text{if } y_n - \hat{y}_n = 0, \text{ for permutation } m \\ 0, & \text{otherwise} \end{cases} \tag{4.5}$$

Where $m = 1, ..., M$ indicates which combination is currently used. The combination that minimizes then equation (4.6) describes the best possible classifier.

$$\arg\min_m \sum_{n=1}^{N} \alpha_{nm} \tag{4.6}$$

4.1.2 Loss Function

In order to evaluate the results of the k-means algorithm, a cost function, that weighs missclassifications, has to be defined. For this, it is possible to use equation (4.6) in a slightly modified way:

$$L = \sum_{n=1}^{N} \beta_n \tag{4.7}$$

with β_n defined as:

$$\beta_n \begin{cases} 1, & \text{if } y_n - \hat{y}_n = 0 \\ 0, & \text{otherwise} \end{cases} \tag{4.8}$$

Hence all missclassification are rated as equally weighted. The error rate can then be calculated with:

$$e_T = \frac{L_T}{N_T}; \quad e_V = \frac{L_V}{N_V} \tag{4.9}$$

Where L_T and L_V are the loss values from equation (4.7) and N_T and N_V are the number of samples in the training and validation set.

4.1.3 Results

4.1.3.1 Unsampled Data Set

Table 4.1 shows the results for the unsampled data set. It shows the error on the training set e_T as well as the error on the validation set e_V for all five training iterations, that occur because of the 5-fold cross-validation.

	k=1	**k=2**	**k=3**	**k=4**	**k=5**	**Average**
e_T	0.560	0.453	0.560	0.468	0.560	0.520
e_V	0.560	0.628	0.791	0.773	0.848	0.720

Table 4.1: The error rate for the K-means clustering algorithm for the unsampled training set

The cluster middle point coordinates for the best classifier are shown in table 4.2.

	Y	**Cb**	**Cr**
Ball	90.723	104.540	179.105
Field	63.999	125.563	123.706
Goal	116.629	123.478	115.868
Line	151.906	119.077	118.481
Robot	89.807	119.063	112.116

Table 4.2: The YCbCr coordinates for the best classifier for unsampled data

4.1.3.2 Under-Sampled Data Set

Table 4.3 shows the results for the under-sampled data set.

	k=1	**k=2**	**k=3**	**k=4**	**k=5**	**Average**
e_T	0.466	0.470	0.470	0.371	0.371	0.430
e_V	0.938	0.478	0.893	0.667	0.774	0.750

Table 4.3: The error rate for the K-means clustering algorithm for the under-sampled training set

The cluster middle point coordinates for the best classifier are shown in table 4.4.

	Y	**Cb**	**Cr**
Ball	97,839	118,967	112,169
Field	141,888	128,815	112,420
Goal	152,490	100,107	132,213
Line	64,984	123,588	124,808
Robot	91,879	103,195	181,536

Table 4.4: The YCbCr coordinates for the best classifier for under-sampled data

4.1.3.3 Over-Sampled Data Set

Table 4.5 shows the results for the over-sampled data set.

	k=1	**k=2**	**k=3**	**k=4**	**k=5**	**Average**
e_T	0.370	0.371	0.372	0.372	0.470	0.391
e_V	0.370	0.370	0.892	0.801	0.6844	0.624

Table 4.5: The error rate for the K-means clustering algorithm for the over-sampled training set

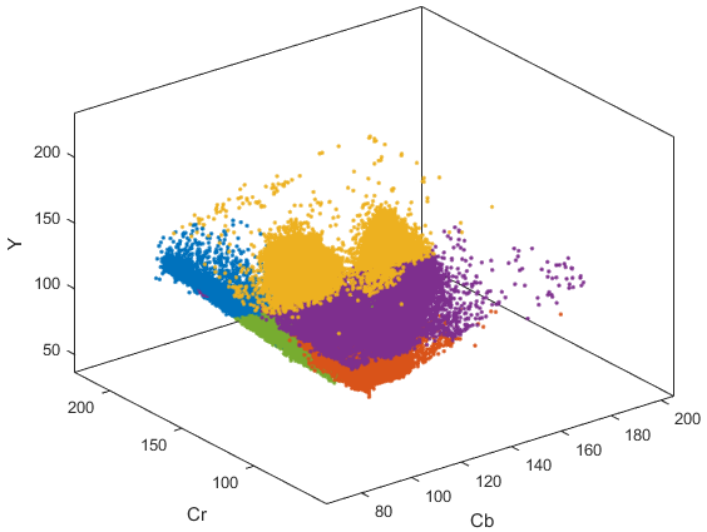
The cluster middle point coordinates for the best classifier are shown in table 4.6.

	Y	**Cb**	**Cr**
Ball	91,432	103,371	181,308
Field	65,015	123,644	124,489
Goal	152,550	100,080	132,260
Line	142,361	128,700	112,454
Robot	98,233	119,098	112,219

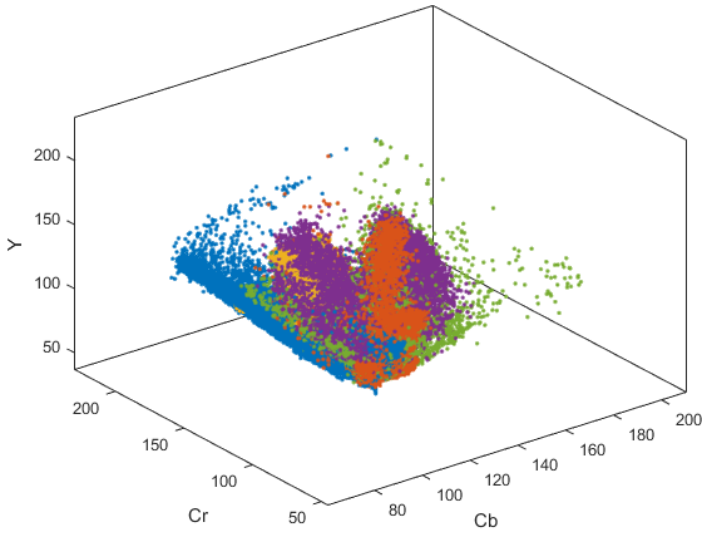
Table 4.6: The YCbCr coordinates for the best classifier for over-sampled data

4.1.4 Discussion

The best result was achieved by using the over-sampled data set. But with an error rate of 62.4 % even this result is not usable for a reliable detection. In addition to that, the resulting cluster middle points for the different classes are varying a lot between these five classifiers. What means, that it is not possible for the algorithm to differentiate between the five classes and it was only capable of minimizing the error regarding the squared sum of distances. Figure 4.1 compares one of the resulting classifications with the true classification.



(a) K-mean result



(b) Correct classification

Figure 4.1: Figure (a) shows the result of the K-means clustering algorithm for the under-sampled data set. Figure (b) shows the correct classification for this data set.

4.2 Artifical Neural Network

This section is based on [RN02].
Section 4.1 has shown, that a more sophisticated approach is needed, to classify the data correctly. A solution to this problem might be artificial

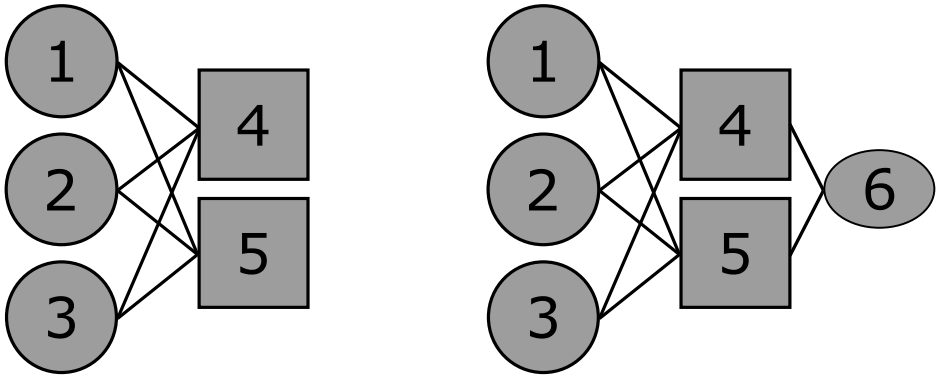


Figure 4.2: Figure (a) shows a neural network with three input (1, 2 and 3) and two output neurons (4 and 5). The network in figure (b) has an additional hidden layer (neuron 4 and 5) and only one output neuron (6)

neural networks. Neural networks are inspired by the human brain, because they are also built from small connected units, that built a network. These units are called neurons. Figure 4.2 shows two different neural networks.

Within a neural network the neurons in a layer l are always connected with all the neurons in layer $l+1$. The current value, that is assigned to the neuron i, is denoted by a_i. The connections between the neurons are weighted edges. The weight for the connection from neuron i to neuron j is denoted by $w_{i,j}$. Each node processes all the incoming values by summing up the output values of the predecessor neurons multiplied with the weight of the connection to that neuron (eq. (4.10)) . The model of a neuron can be seen in figure 4.4.

$$in_j = \sum_{i=1}^{n} w_{i,j} a_i \tag{4.10}$$

The output value a_j can then be calculated by applying the activation function $g(x)$.

$$a_j = g(in_j) = g(\sum_{i=1}^{n} w_{i,j} a_i) \tag{4.11}$$

As activation function often the sigmoid function is used.

$$g(x) = \frac{1}{1+e^{-x}} \tag{4.12}$$

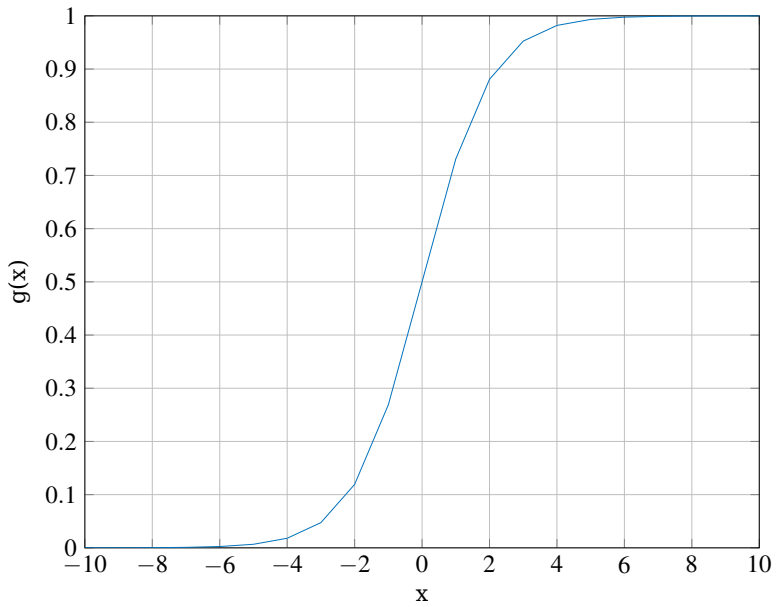


Figure 4.3: A plot of the sigmoid function, which is used as the activation function $g(x)$

The advantage of the sigmoid function over e.g. a simple threshold function is, that it is differentiable. Figure 4.3 shows a plot of a sigmoid function.

To calculate the output for a neural network, is basically done by setting the input nodes to the input vector $\mathbf{a}_{in} = \mathbf{x}$ and then propagating the values through all the nodes. By putting the weights between two layers in a matrix, the whole net can be evaluated by simple matrix vector multiplications. To

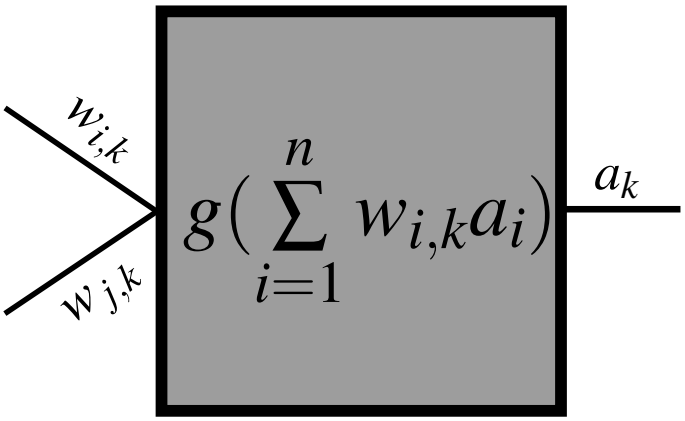


Figure 4.4: A mathematical representation of a neuron.

determine the output values for the netork in figure 4.2 (a), the weights can be written as follows.

$$W_{in,out} = \begin{pmatrix} w_{1,4} & w_{2,4} & w_{3,4} \\ w_{1,5} & w_{2,5} & w_{3,5} \end{pmatrix} \tag{4.13}$$

The output then can be calculated by.

$$\mathbf{a}_{out} = g(W_{in,out}\mathbf{x}) \tag{4.14}$$

For the network 4.2 (b) the calculations are the same, except that the output layer from (a) now becomes the hidden layer and an additional calculation must be done.

$$W_{in,hidden} = \begin{pmatrix} w_{1,4} & w_{2,4} & w_{3,4}; \\ w_{1,5} & w_{2,5} & w_{3,5} \end{pmatrix}; W_{hidden,out} = \begin{pmatrix} w_{4,6} & w_{5,6} \end{pmatrix} \tag{4.15}$$

$$\mathbf{a}_{hidden} = g(W_{in,hidden}\mathbf{x}) \tag{4.16}$$

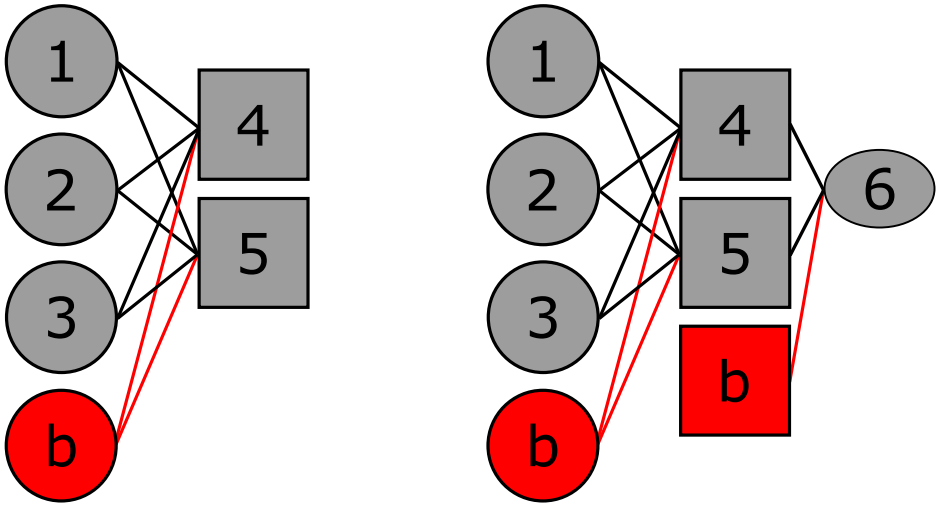
$$\mathbf{a}_{out} = g(W_{hidden,out}\mathbf{a}_{hidden}) \tag{4.17}$$

4.2.1 Bias Neurons

All neural networks have an additional neuron in each layer, except for the output layer. This neuron is called bias neuron. The difference to normal neurons is, that bias neurons have no inputs and they always have a constant output of $b = 1$. So they are only connected to each neuron in the subsequent layer. These connections are like the regular connections and are weighted in the same way. These bias neurons are capable of shifting the threshold boundary of the activation function to the left or to the right according to the sign of the weights. Figure 4.5 shows two neural networks with the bias neurons.

4.2.2 Training

To determine all the weights for a neural network, the backpropagation algorithm is used. At first a loss function has to be defined. The loss of a neural network is the squared sum of distances between the output of the network a_{out} and the training data y.



(a) (b)

Figure 4.5: The neural networks from figure 4.2 with bias neurons in the input and hidden layer

$$L = |\mathbf{y} - \mathbf{a}_{out}|^2 = \sum_{k=1}^{n_{out}} (y_k - a_k)^2 \tag{4.18}$$

Now that the error in the output layer is known it is possible to propagate the error back through the previous layer, hence the name backpropagation. Equation (4.19) shows how the new values for the weights to the output layer have to be calculated.

$$w_{j,k} = w_{j,k} + \alpha \cdot a_j \cdot \Delta_k \tag{4.19}$$

where

$$\Delta_k = Err_k \cdot g'(in_k) \tag{4.20}$$

and Err_k is the k^{th} entry in the error vector $\mathbf{y} - \mathbf{a}_{out}$,in_k is the input for the k^{th} neuron in the output layer and α is the learning rate, that defines how big the step size is.

To update the weights in the hidden layer, the error in the output neurons will be backpropagated through the node, but it is multiplied by the same weight as in the forwardpropagation. The idea behind this, is that the influence of one neuron to the overall error is proportional to the weight, that is between the neuron and the output. The partial error for a hidden neuron is defined by Δ_j.

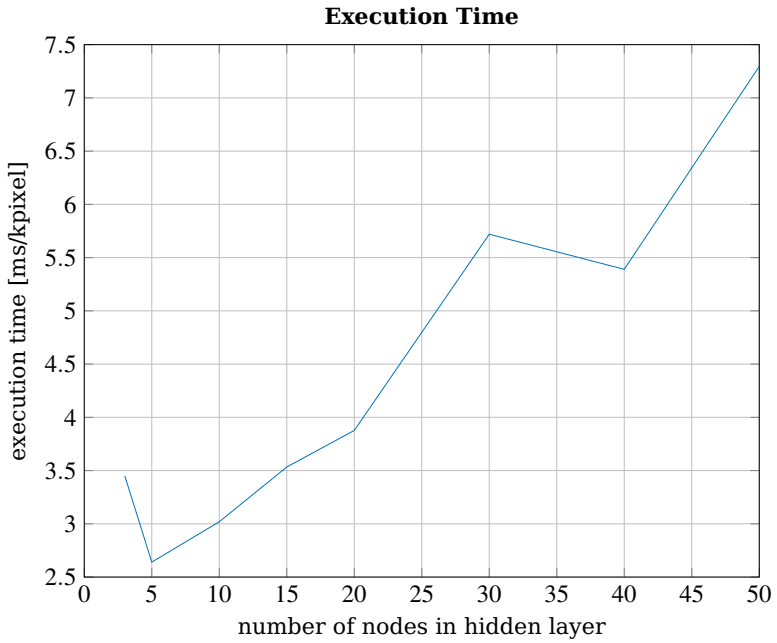


Figure 4.6: Runtime of a neural network with one hidden layer

$$\Delta_j = g'(in_j) \sum_k w_{j,k} \Delta_k \tag{4.21}$$

Now the weights in the hidden layer can be updated with equation (4.22).

$$w_{i,j} = w_{i,j} + \alpha \cdot a_i \cdot \Delta_j \tag{4.22}$$

4.2.3 Determining the Topology

There are two basic requirements, that the network has to meet. A short execution time and a low error rate. The data has to be separated in a non-linear way, therefore at least one hidden layer is necessary. Figure 4.6 shows the execution time of a single layer network in dependency of the number of neurons in the hidden layer.

The execution time for some networks with two hidden layer can be seen in table 4.7

A very small network with two hidden layers would be feasible, although a single layer network would be much more preferable.

First Layer	Second Layer	Average Runtime [$\frac{ms}{kpixel}$]	First Layer	Second Layer	Average Runtime [$\frac{ms}{kpixel}$]
3	3	5.445	20	15	9.475
5	3	3.822	20	20	11.062
10	3	6.181	30	20	10.927
10	5	6.735	30	30	16.013
10	10	6.109	40	40	11.477
15	10	6.109	50	50	25.22
15	15	5.978			

Table 4.7: Execution times for a network with two hidden layer

4.2.3.1 Under-Sampled Data Set

Figure 4.7 shows the error rate of neural networks trained with the under-sampled data set. The error rate converges slightly above 20%.

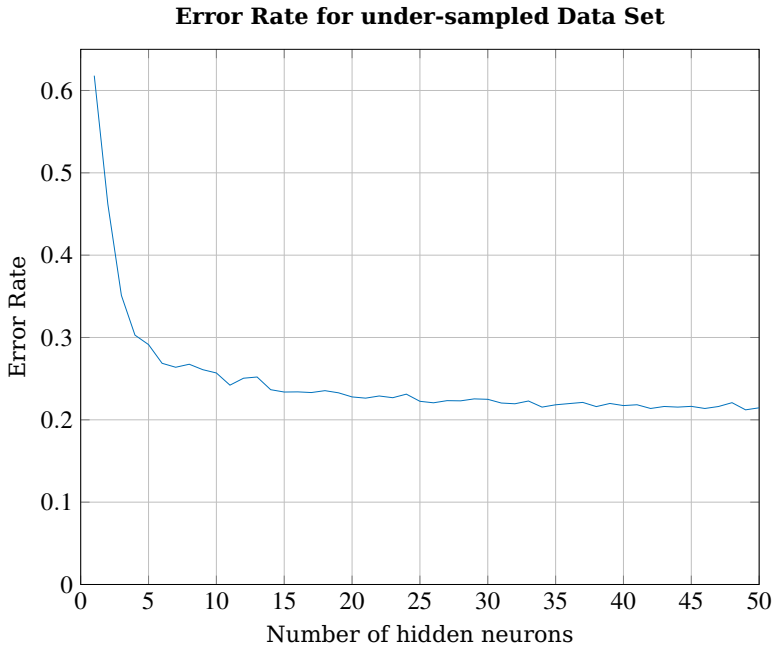


Figure 4.7: The Mean squared Error for a neural network with one hidden layer dependent on the number of neurons for the under-sampled data set.

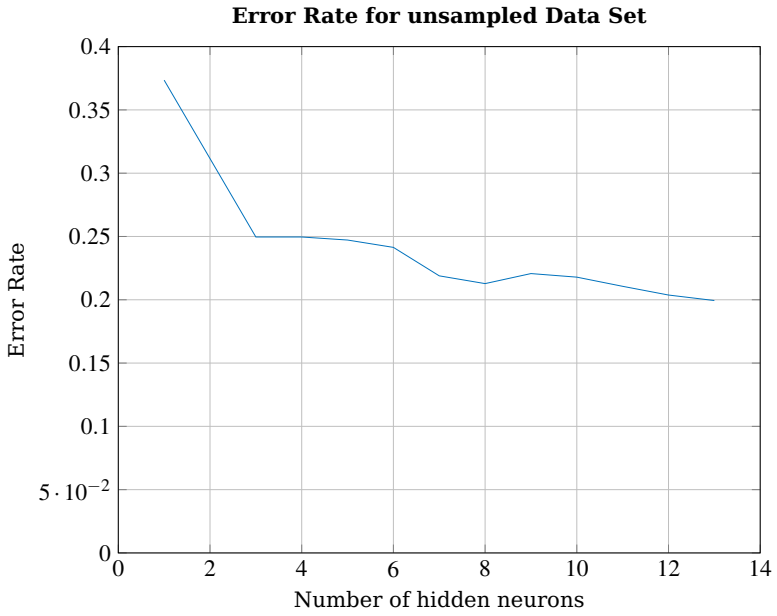


Figure 4.8: The Mean squared Error for a neural network with one hidden layer dependent on the number of neurons for the under-sampled data set.

4.2.3.2 Unsampled Data Set

The result for the unsampled data set can be seen in figure 4.8. The measurement was only done for 14 hidden neurons, because the training effort was getting too high. But it can be estimated, that the error rate will converge slightly under 20%. This is, compared to the much higher training time, just a small improvement.

4.2.3.3 Over-sampled Data Set

The available hardware was not able to handle the over-sampled data set. But based on the results of the two other sets, it is not expected that this data set achieves a lower error rate than 15%.

4.2.4 Discussion

The comparison of neural networks trained with the different data sets has shown, that the error rate is nearly equal for the unsampled and under-sampled set. Although the over-sampled data set could not be evaluated, it is expected, that the error rate is similar too. The slightly better error

rates for the bigger data sets come along with much higher hardware and time requirements, which is not comparative. Therefore a smaller data set with precisely picked examples should be preferred. A compromise between the error rate and the execution time has been made. This might always be dependent on the use case, but based on the runtime from figure 4.6 and the error estimation, a neural network with one hidden layer and eight to ten neurons seems like a good solution.
An interesting fact is, that even the network with 50 hidden neurons shows no signs of overfitting. This might be the case, because the complexity with only one hidden layer might not be enough to fit the complex data set.

4.3 Support Vector Machines

Another approach that shall be introduced in this thesis are support vector machines. The following section is based upon [AL08].
The main idea of the support vector machine is, to find a maximized margin between the two classes C_1 and C_2, that shall be separated. The margin is described by a hyperplane, that lies in its middle. The margin is defined by assigning the nearest examples of the training set to the plane the distance 1 or -1 respectively. The plane can be expressed with:

$$\mathbf{w}^T\mathbf{x}^t + w_0 \geq +1, \text{for } r^t = +1 \tag{4.23}$$

$$\mathbf{w}^T\mathbf{x}^t + w_0 \leq -1, \text{for } r^t = -1 \tag{4.24}$$

Where $\mathbf{w}^T$ is the normal vector of the plane,$\mathbf{x}^t$ and r^t are from the training set $\chi = [\mathbf{x}^t, r^t]$. $\mathbf{x}^t$ contains the examples and r^t is either $+1$, if $\mathbf{x}^t \in C_1$, or -1, if $\mathbf{x}^t \in C_2$. (4.23) and (4.24) can be combined to.

$$r^t(\mathbf{w}^T\mathbf{x}^t + w_0) \geq +1 \tag{4.25}$$

The figure 4.9 shows an example with two classes and the margin.

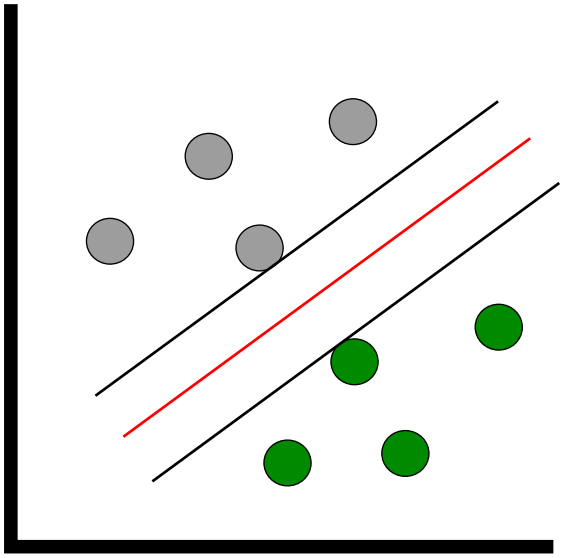


Figure 4.9: The margin between the two classes is defined by the red hyperplane

The distance, that should be maximized, between $\mathbf{x}^t$ and the hyperplane, can be determined by

$$\frac{r^t(\mathbf{w}^T\mathbf{x}^t + w_0)}{\| \mathbf{w} \|}. \tag{4.26}$$

It can be seen that this term will be maximized when $\| \mathbf{w} \|$ is minimized. Therefore the following minimization problem can be formulated:

$$\min \frac{1}{2} \| \mathbf{w} \|^2, \text{subject to: } r^t(\mathbf{w}^T\mathbf{x}^t + w_0) \geq +1, \forall t \tag{4.27}$$

This Problem can be rewritten by using the lagrange multiplier α^t.

$$L_p = \frac{1}{2} \| \mathbf{w} \|^2 - \sum_{t=1}^{N} \alpha^t [r^t(\mathbf{w}^T\mathbf{x}^t + w_0) - 1] \tag{4.28}$$

Which is equal to

$$\frac{1}{2} \| \mathbf{w} \|^2 - \sum_{t=1}^{N} \alpha^t r^t(\mathbf{w}^T\mathbf{x}^t + w_0) + \sum_{t} \alpha^t. \tag{4.29}$$

This is a convex quadratic optimization problem. It is possible to solve the equivalent dual problem by using the Karush-Kuhn-Tucker conditions:

$$\frac{\partial L_p}{\partial \mathbf{w}} = 0 \Rightarrow \mathbf{w} = \sum_t \alpha^t r^t \mathbf{x}^t \tag{4.30}$$

and

$$\frac{\partial L_p}{\partial w_0} = 0 \Rightarrow \sum_t \alpha^t r^t = 0 \tag{4.31}$$

The dual problem can be determined by combining equations (4.30) and (4.31) with (4.29).

$$L_d = -\frac{1}{2} \sum_t \sum_s \alpha^t \alpha^s r^t r^s (\mathbf{x}^t)^T \mathbf{x}^s + \sum_t \alpha^t \tag{4.32}$$

Which shall be maximized with subject to

$$\sum_t \alpha^t r^t = 0, \text{and } \alpha \geq 0, \forall t \tag{4.33}$$

The main point with equation (4.33) is, that it only depends on the example set and it can be solved by common optimization methods for quadratic problems. After solving this problem, most of the α^t will be zero. All examples with a non-zero α^t are the support vectors. With equation (4.30) it is possible to determine the normal vector of the hyperplane $\mathbf{w}$ only using the support vectors. The bias w_0 can be calculated by

$$w_0 = r^t - \mathbf{w}^T \mathbf{x}^t \tag{4.34}$$

To classify a new example, the decision function $g(\mathbf{x})$ can be defined.

$$g(\mathbf{x}) = \mathbf{w}^T \mathbf{x}^t + w_0 \tag{4.35}$$

The classification then works as follows:

$$\mathbf{x} \in \begin{cases} C_1, & \text{if } g(\mathbf{x}) > 0 \\ C_2, & \text{otherwise} \end{cases} \tag{4.36}$$

4.3.1 The Kernel Trick

One problem of support vector machines is, that the above described classifier can only be applied to linear separable data. Often it is required to classify non-linear data. To achieve that, it is possible to project the examples in a feature space, where the data becomes linear separable. This feature space has often a higher dimension than the original space. For this

projection it is convenient, that the whole classification only depends on the scalar product $\mathbf{w}^T\mathbf{x}^t$. The projection in the feature space is described by $\phi(\mathbf{x})$. The classifier can then be modified.

$$g(\mathbf{x}) = \mathbf{w}^T\phi(\mathbf{x}) \tag{4.37}$$

Using the dual problem leads to

$$\mathbf{w} = \sum_t \alpha^t r^t \phi(\mathbf{x}). \tag{4.38}$$

Substituting w in (4.37) with (4.38) leads to the new $g(\mathbf{x})$.

$$g(\mathbf{x}) = \sum_t \alpha^t r^t \phi(\mathbf{x}^t)^T \phi(\mathbf{x}). \tag{4.39}$$

In [CV95] the idea of using kernel functions is introduced. A kernel function $K(\mathbf{x}^t, \mathbf{x})$ is meant to substitute the scalar product $\phi(\mathbf{x}^t)^T\phi(\mathbf{x})$. The advantage of such a kernel is, that it is no longer necessary to project into the feature space, because the kernel function represents the scalar product in the feature space, while it only requires the input vectors from the original space. This feature is called the "kernel trick". Some common used kernel functions are:

Linear :

$$K(\mathbf{x}^t, \mathbf{x}) = \mathbf{x}^T\mathbf{x}^t \tag{4.40}$$

Polynomial :

$$K(\mathbf{x}^t, \mathbf{x}) = (\mathbf{x}^T\mathbf{x}^t + 1)^q \tag{4.41}$$

Radial Basis Function:

$$K(\mathbf{x}^t, \mathbf{x}) = \exp[-\frac{\| \mathbf{x}^t - \mathbf{x} \|^2}{\sigma^2}] \tag{4.42}$$

The decision function with the added kernel trick can then be written like.

$$g(\mathbf{x}) = \sum_t \alpha^t r^t K(\mathbf{x}^t, \mathbf{x}). \tag{4.43}$$

4.3.2 Multiple Classes

A support vector machine by itself is only capable of distinguishing between two classes. Often the data sets have a higher number of classes $K > 2$. For this case two simple methods exist to add multi-class support to support vector machines [Bis06].

One-versus-rest

The idea of the one-versus-rest method is, to train K different support vector machines. Each of them is trained with one class as positive examples and the remaining $K-1$ classes as the negative examples. Each $\mathbf{x}$, that shall be classified, is then fed into each of the K support vector machines. Therefore it is possible that a sample is assigned to multiple classes. The disadvantage of this method is, that the training data gets very imbalanced because of the splitting.

One-versus-one

The one-versus-one approach uses $\frac{K(K-1)}{2}$ different support vector machines of all combinations of the K classes. Each sample is classified by all the SVMs and the end result is determined by a voting process. This method doesn't suffer from the imbalanced data set, but it causes much higher computational effort for training and in execution.

4.3.3 Determining the parameter

In order to find a configuration, that meets the requirements best, at first the runtimes for the different kernel functions are evaluated. The runtime also depends on the number of support vectors, because for every classification the decision function $g(\mathbf{x})$ has to iterate through all examples with a $\alpha^t > 0$. Table 4.8 shows the execution times for different configurations per 1000 pixel.

Table 4.9 shows the results for a support vector machine with a linear kernel function trained with the under-sampled data set.

These results show, that the error rate is lower than the error rate of the neural networks, but the number of support vectors is very high. This would cause a very high runtime, especially because all of the five classifier has to be executed after another. An estimate of the overall runtime with the data from table 4.8 leads to a execution time of ca. 9604.37 ms. This violates the real time constraint and is therefore not feasible for practice. Albeit that a comparison of the performance of the different kernels shall be provided.

Runtime [ms/kpixel]	Linear	Polynomial	Radial Basis Function
10 SVs	1.098	2.831	15.154
50 SVs	5.326	13.615	76.169
100 SVs	10.663	26.812	149.738

Table 4.8: The runtime of support vector machines with three different kernels for 10/50/100 support vectors

	Ball	Field	Goal	Line	Robot
Error Rate	0.062	0.236	0.194	0.223	0.347
#SV	13707	30912	12614	31179	32845

Table 4.9: The results for a support vector machine with linear kernel function

To achieve a reduced number of support vectors the training will be further sub-sampled to a total size of 1000 with 200 samples per class. The results can be seen in table 4.10.

The best result achieved the linear kernel. Which is convenient, because it causes the lowest runtime. The polynomial kernel produced very bad results. With $q = 3$ it wasn't even possible to determine a hyperplane for the field classifier. The result of the radial basis function seems not too bad, but the number of support vectors show, that it needs almost all examples to define the hyperplane. This means the classifier suffers from very high overfitting.

4.3.4 Discussion

Figure 4.10 shows an image wich was classified by the support vector machines trained with the under-sampled data set from table 4.9. For this example the missclassifications are tremendous. All classifiers, except for the field detector, have nearly detected the same regions as true. The field detector performed best and was able to distinguish the field from all other objects. This is only one example and therefore not representative, but it shows the one issue of the one-versus-rest method. The white pixels in the combined image represent the case, in that a pixel is classified with multiple classes. For this example this might be an extreme case, but also for better working classifiers this still might occur. One way to solve this could be to

Linear	**Ball**	**Field**	**Goal**	**Line**	**Robot**
Error Rate	0,059	0,173	0,096	0,165	0,2
#SV	151	374	215	390	441

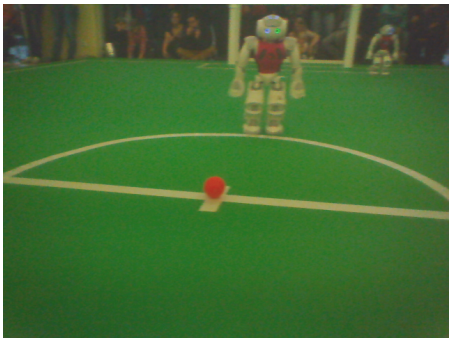
Polynomial (q=2)	**Ball**	**Field**	**Goal**	**Line**	**Robot**
Error Rate	0,44	0,392	0,701	0,607	0,549
#SV	163	313	154	396	450

Polynomial (q=3)	**Ball**	**Field**	**Goal**	**Line**	**Robot**
Error Rate	0,412	NaN	0,215	0,371	0,532
#SV	8	0	2	3	3

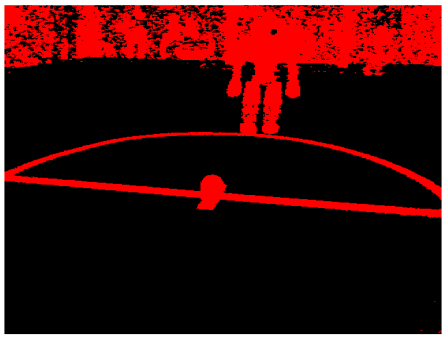
Radial Basis Function	**Ball**	**Field**	**Goal**	**Line**	**Robot**
Error Rate	0,184	0,198	0,166	0,2	0,199
#SV	978	975	970	969	972

Table 4.10: Results for support vector machines with different kernel functions

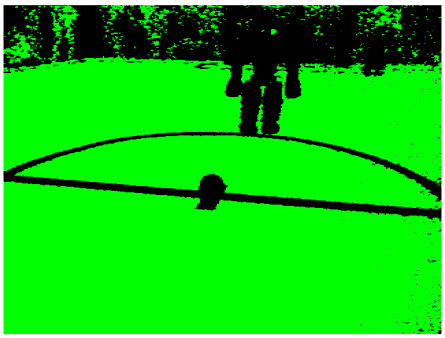
produce integral images [VJ01] for each object class from the binary images provided by the support vector machines. Those integral images can be used to perfom an efficient calculation of the number of pixels, that appear in a certain area. With those numbers a simple majority vote could be used to decide which class the pixel belongs to.



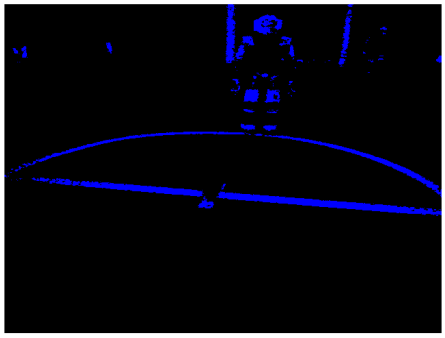
(a) Original image



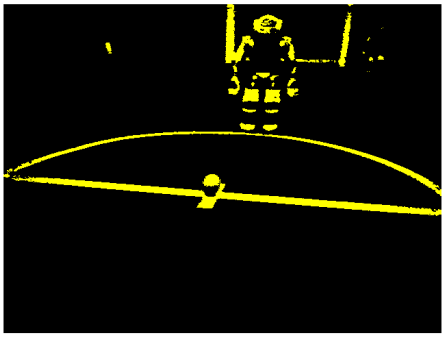
(b) Ball detection



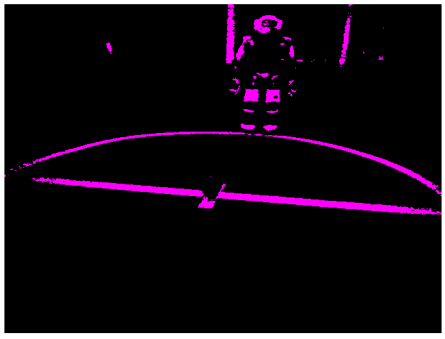
(c) Field detection



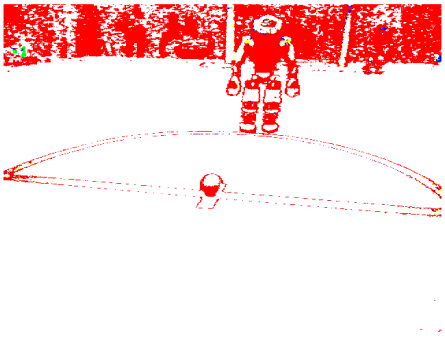
(d) Goal detection



(e) Line detection



(f) Robot detection



(g) Combined image

Figure 4.10: The results of the different SVMs

Chapter 5

Conclusion

Although the results of the actual classifiers were not satisfying, some things could be achieved that might help the team in the future with further developing with machine learning. The first and most essential thing for machine learning is creating a database with fully labeled training data. This database contains ca. 700 images at the moment. Furthermore an infrastructure to use machine learning was established. This includes a code base for the nao, that provides configurable code for support vector machines and neural networks and a json converter that can be used to transfer trained models from matlab to the robot. With this framework it should be possible to further refine machine learning approaches for image processing.
The comparison of the three implemented algorithms has shown that k-means is not applicable to the problem, because it is not able to separate the given data set. The support vector machines had the lowest error rate, although those were still very high, but due to the high execution times it is not feasible. Additionally, due to the multiple classifications, the decision problem must also be solved.
The most promising approach are artificial neural networks. Those had feasible runtimes and the error rates were comparable to the support vector machines.

Chapter 6

Outlook

This thesis has shown that the pixel based approach doesn't provide satisfying results. An extension of that approach, that can easily be implemented, could be to combine the color information with the pixel coordinates. This increases the input dimension by two, but it shouldn't have a big impact either on the training or the execution time. The advantage would be, that for example, lines and goals could be separated much better, because the goal pixels will usually occur in higher regions of the image than the line pixels. Also some missclassification due to objects outside of the field could be avoided. An even better approach than combining pixels and positions would be, to take also the patterns of the objects into account. This could be done by convolutional neural networks [LB95] . Convolutional neural networks are modified artificial neural networks that are able to learn filter operations. The major drawback of convolutional neural networks is that it has very high computational costs for training as well as for execution. Additionally the input for those networks must have a constant size. This means that every object somehow has to be up- or down-scaled before it can be fed into the network. This requires, that regions of interest are evaluated beforehand. These two things alone would probably exceed the real-time constraint already. Therefore a lot of work and effort has to be made to make these approaches work on the robot.

Bibliography

[AL08] E. Alpaydın and S. Linke. *Maschinelles Lernen*. Oldenbourg, 2008. ISBN: 9783486581140. URL: https://books.google.de/books?id=zsShZ68qJ2AC.

[Alda] Aldebaran. *NAO - Video Camera*. URL: http://doc.aldebaran.com/2-1/family/robots/video_robot.html.

[Aldb] Aldebaran. *Nao Datasheet*. URL: https://www.aldebaran.com/sites/aldebaran/files/nao_datasheet.pdf.

[Bis06] Christopher M. Bishop. *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Secaucus, NJ, USA: Springer-Verlag New York, Inc., 2006. ISBN: 0387310738.

[CV95] Corinna Cortes and Vladimir Vapnik. "Support-Vector Networks". In: *Machine Learning*. 1995, pp. 273–297.

[FK] Benoît Frénay and Ata Kabán. *A Comprehensive Introduction to Label Noise*.

[For14] Internet Engineering Task Force. *The JavaScript Object Notation (JSON) Data Interchange Format*. RFC 7159. IETF, 2014.

[Htw] *HTWK NAO SPL Team*. URL: http://robocup.imn.htwk-leipzig.de/.

[IR94] ITU-R. *Encoding Parameters Of Digital Television For Studios*. Tech. rep. BT.601-4. ITU-R, 1994.

[KM97] Miroslav Kubat and Stan Matwin. "Addressing the Curse of Imbalanced Training Sets: One-Sided Selection". In: *In Proceedings of the Fourteenth International Conference on Machine Learning*. Morgan Kaufmann, 1997, pp. 179–186.

[Kri] Hinton Krizhevsky Vinod. *CIFAR 10/100 Dataset*. URL: http://www.cs.toronto.edu/~kriz/cifar.html.

[LB95] Yann LeCun and Yoshua Bengio. “Convolutional networks for images, speech, and time series”. In: *The handbook of brain theory and neural networks* 3361.10 (1995).

[LL98] Charles X Ling and Chenghui Li. “Data Mining for Direct Marketing: Problems and Solutions.” In: *KDD*. Vol. 98. 1998, pp. 73–79.

[Mal03] Marcus A Maloof. “Learning when data sets are imbalanced and when costs are unequal and unknown”. In: *ICML-2003 workshop on learning from imbalanced data sets II*. Vol. 2. 2003, pp. 2–1.

[Mni] *The MNIST Database of handwritten digits*. URL: `http://yann.lecun.com/exdb/mnist/`.

[RN02] Stuart J. Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach (2nd Edition)*. Prentice Hall, 2002. ISBN: 0137903952. URL: `http://www.amazon.ca/exec/obidos/redirect?tag=citeulike09-20\&path=ASIN/0137903952`.

[VJ01] Paul Viola and Michael Jones. “Rapid object detection using a boosted cascade of simple features”. In: *Computer Vision and Pattern Recognition, 2001. CVPR 2001. Proceedings of the 2001 IEEE Computer Society Conference on*. Vol. 1. IEEE. 2001, pp. I–511.

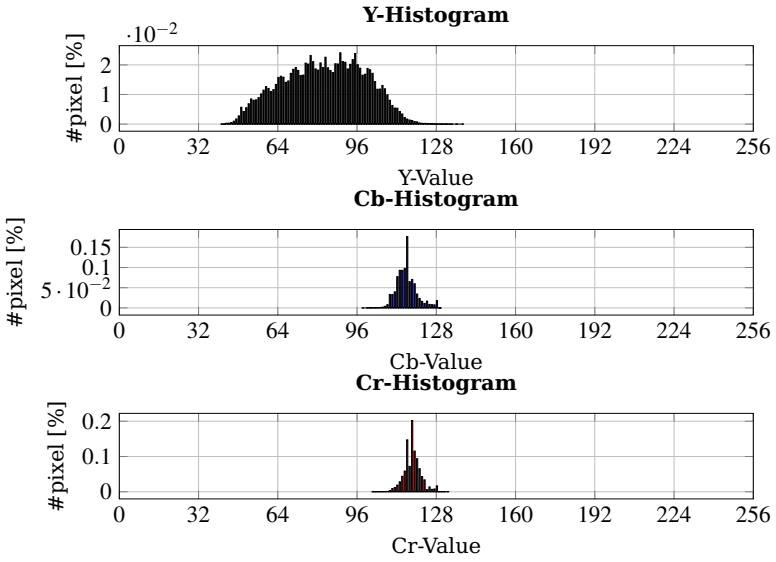
[WFH11] Ian H. Witten, Eibe Frank, and Mark A. Hall. “Chapter 1 - What’s It All About?” In: *Data Mining: Practical Machine Learning Tools and Techniques (Third Edition)*. Ed. by Ian H. WittenEibe FrankMark A. Hall. Third Edition. The Morgan Kaufmann Series in Data Management Systems. Boston: Morgan Kaufmann, 2011, pp. 3 –38. ISBN: 978-0-12-374856-0. DOI: `http://dx.doi.org/10.1016/B978-0-12-374856-0.00001-8`. URL: `http://www.sciencedirect.com/science/article/pii/B9780123748560000018`.

Chapter 7

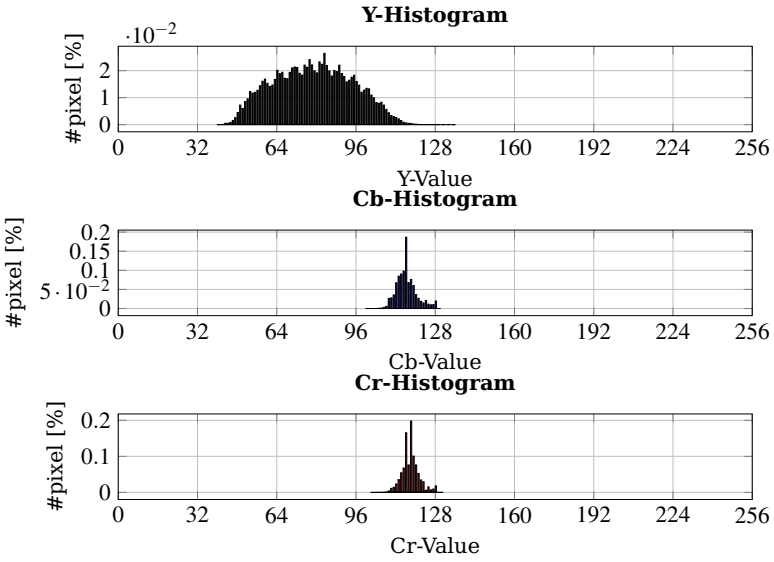
Appendix A

Figure 7.1: Different Carpets with high illumination

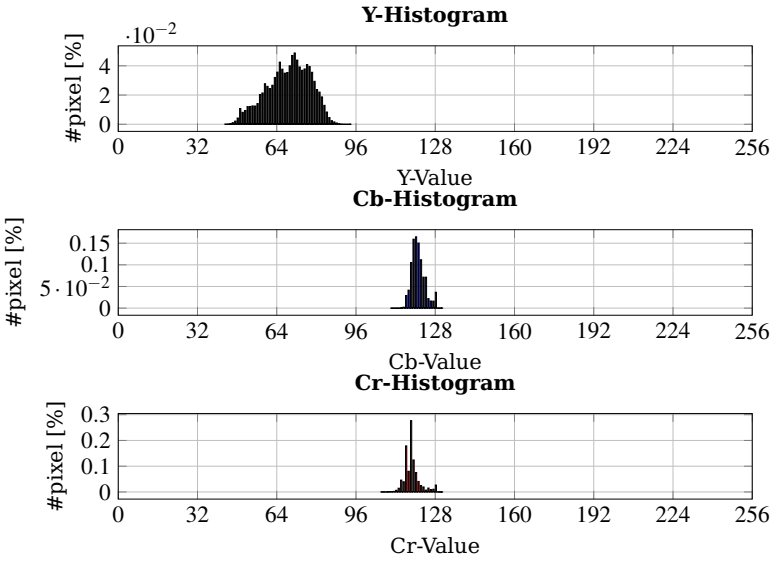
(a)



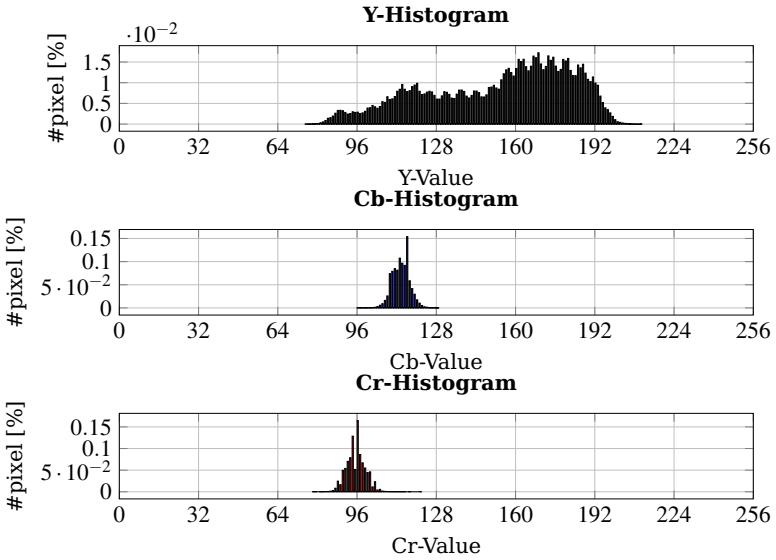
(b)



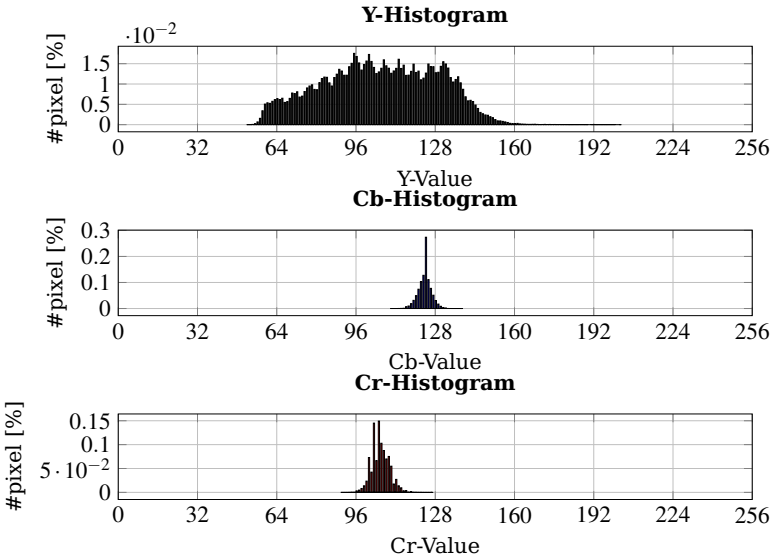
(c)



(d)


Y-Histogram
$\cdot 10^{-2}$
#pixel [%]
1.5
1
0.5
0
0 32 64 96 128 160 192 224 256
Y-Value
Cb-Histogram
#pixel [%]
0.15
0.1
$5 \cdot 10^{-2}$
0
0 32 64 96 128 160 192 224 256
Cb-Value
Cr-Histogram
#pixel [%]
0.15
0.1
$5 \cdot 10^{-2}$
0
0 32 64 96 128 160 192 224 256
Cr-Value

(e)


Y-Histogram
$\cdot 10^{-2}$
#pixel [%]
1.5
1
0.5
0
0 32 64 96 128 160 192 224 256
Y-Value
Cb-Histogram
#pixel [%]
0.3
0.2
0.1
0
0 32 64 96 128 160 192 224 256
Cb-Value
Cr-Histogram
#pixel [%]
0.15
0.1
$5 \cdot 10^{-2}$
0
0 32 64 96 128 160 192 224 256
Cr-Value

(f)

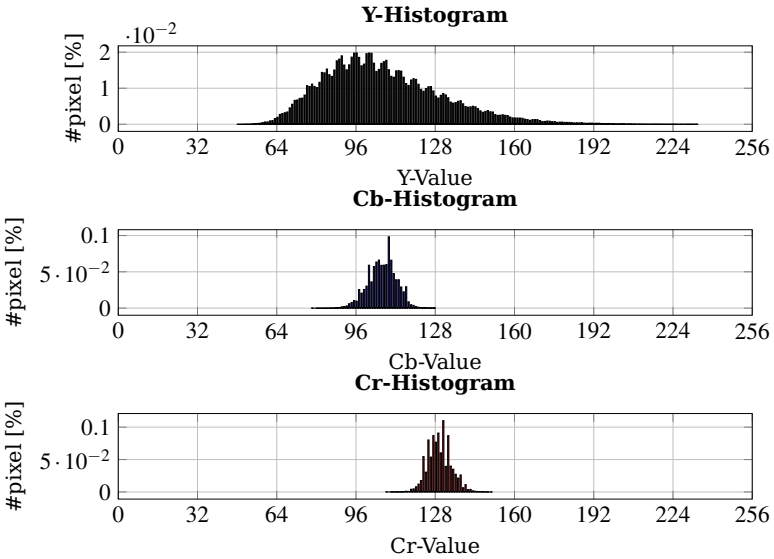
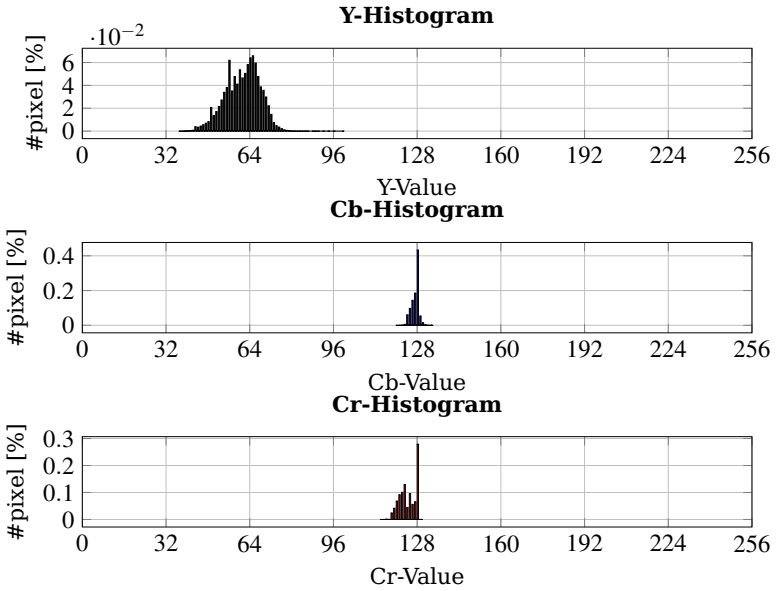
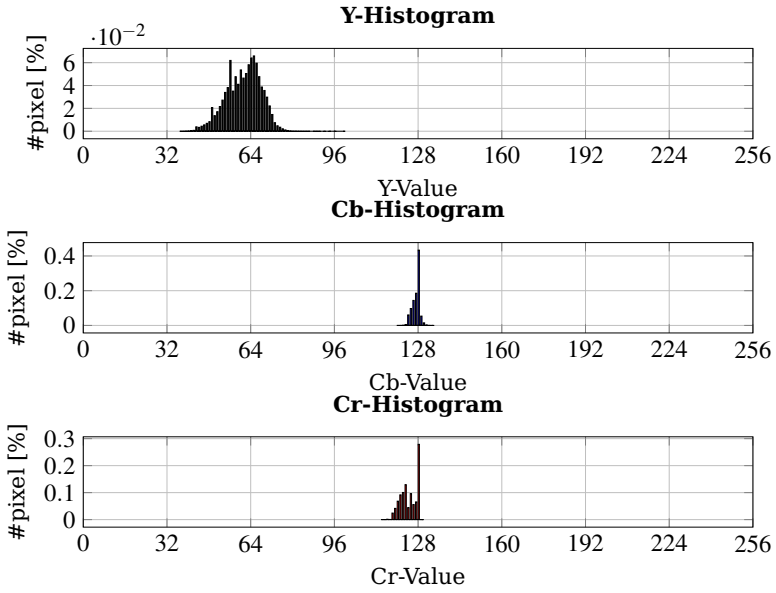

Y-Histogram
$\cdot 10^{-2}$
#pixel [%]
2
1
0
0 32 64 96 128 160 192 224 256
Y-Value
Cb-Histogram
#pixel [%]
0.1
$5 \cdot 10^{-2}$
0
0 32 64 96 128 160 192 224 256
Cb-Value
Cr-Histogram
#pixel [%]
0.1
$5 \cdot 10^{-2}$
0
0 32 64 96 128 160 192 224 256
Cr-Value

Figure 7.2: Different Carpets with normal illumination

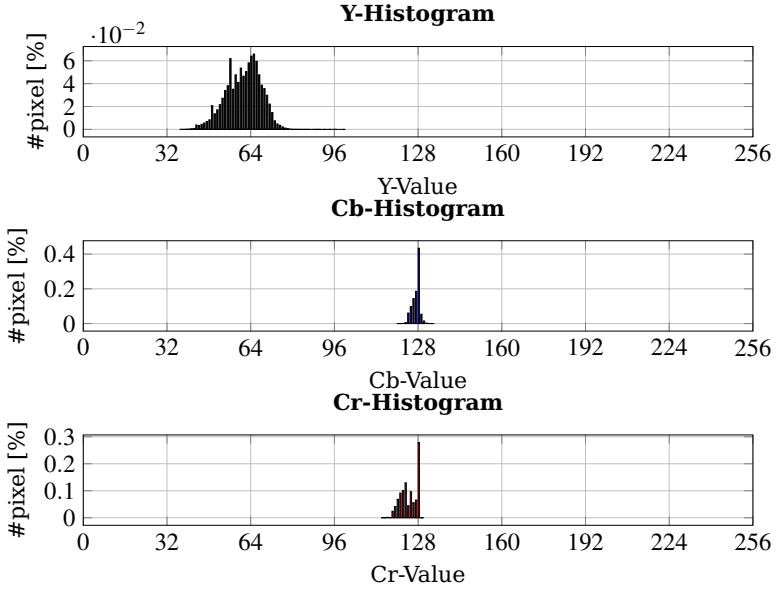
(a)



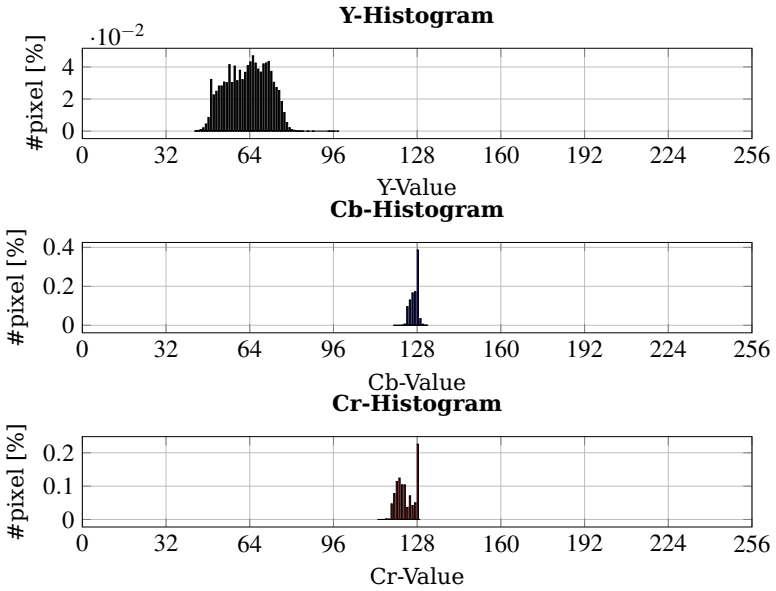
(b)



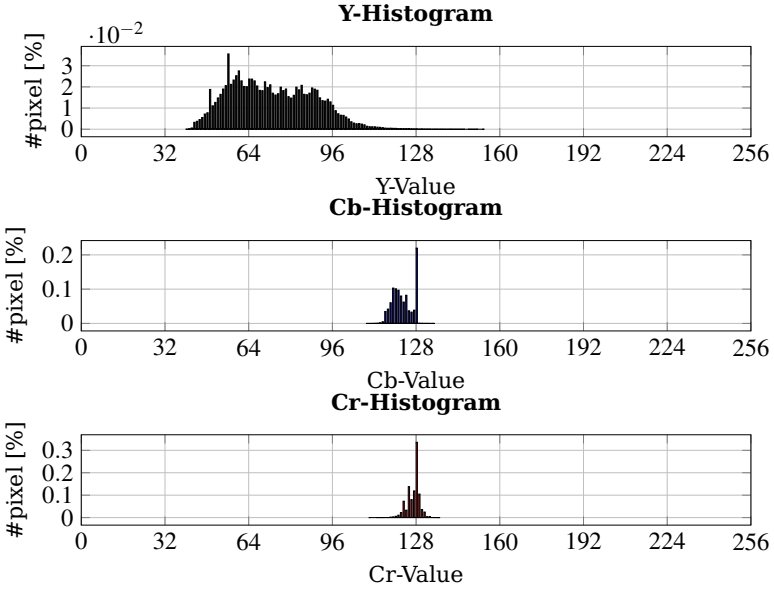
(c)



(d)


Y-Histogram
$\cdot 10^{-2}$
#pixel [%]
Y-Value
Cb-Histogram
#pixel [%]
Cb-Value
Cr-Histogram
#pixel [%]
Cr-Value

(e)


Y-Histogram
$\cdot 10^{-2}$
#pixel [%]
Y-Value
Cb-Histogram
#pixel [%]
Cb-Value
Cr-Histogram
#pixel [%]
Cr-Value

(f)

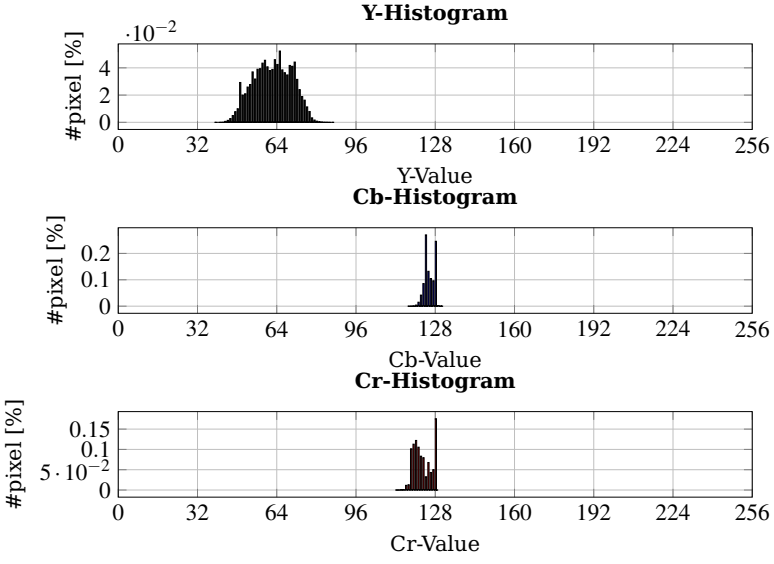

Y-Histogram
$\cdot 10^{-2}$
#pixel [%]
Y-Value
Cb-Histogram
#pixel [%]
Cb-Value
Cr-Histogram
#pixel [%]
Cr-Value

Figure 7.3: Carpet with low illumination