Pontificia Universidad Católica del Perú
Escuela de Posgrado

Tesis de Maestría

Autor:	Kathrin Lang
Fecha de Nacimiento:	07.06.1989
Lugar de Nacimiento:	Sulzbach-Rosenberg
Título a obtar:	Maestría en Ingeniería Mecatrónica

Título de la Tesis:
"Software for calibrating a digital image processing"

Asesor de la PUCP:	Ericka Patricia Madrid Ruiz
Asesor de la TU Ilmenau:	Florian Schale
Entrega de la Tesis:	03.03.2014
Lugar:	Ilmenau (Alemania)

Technische Universität Ilmenau
Fakultät für Maschinenbau

Masterarbeit

für Frau	Kathrin Lang
geboren	am 07.06.1989 in Sulzbach-Rosenberg
Studiengang	Mechatronik

THEMA

„Software for calibrating a digital image processing"

verantw. Hochschullehrer: Prof. Dr.-Ing. habil. Mathias Weiß

Ausgabedatum: 01.09.2013 Abgabedatum: 03.03.2014

Ilmenau, den 03.09.2013

[illegible]

Univ.-Prof. Dr.-Ing. Thomas Sattel
Vorsitzender des Prüfungsausschusses

TECHNISCHE UNIVERSITÄT
ILMENAU

Fakultät für Maschinenbau

Fachgebiet Rechneranwendung im Maschinenbau

Task for the Master's thesis

of Ms. Kathrin Lang

Topic: Software for calibrating a digital image processing

A digital image processing software was developed for the RoboCup project at the Faculty of Mechanical Engineering at the Technical University of Ilmenau, which determines the position of multiple mobile robots on a playing field.

The image information provides a digital network camera (Prosilica GC1600H). For the successful position detection of the robot it is necessary to determine various parameters and settings of the camera. For this purpose, a software should be developed that semi-automatically determines the necessary data.

Programming will done with Visual Studio in C # and an provided camera API.

Subtasks:

- Analysis of image processing software for required parameters and camera settings
- Definition or adaptation of the interface between image processing software and the calibration software
- Programming of an User interface, algorithms and data interfaces
- Creation of documentation and manual

Date of issue:	01.09.2013
Responsible proffessor:	Univ.-Prof. Dr.-Ing. habil. Mathias Weiß
Carer at TU Ilmenau:	Florian Schale M. Sc.
Carer at Pontificia Universidad Católica del Perú:	Ericka Madrid Ruiz M.Sc.

Ilmenau, 14.8.13
City, Date

[illegible]
Signature of the responsible proffessor

Lima, 28.8.2013
City, Date

[illegible]
Signature of the carer at Pontificia Universidad Católica del Perú

Ilmenau, 03.09.13
City, Date

K. Lang
Signature of the student

Statement of Authorship

I declare on oath that I completed this work on my own and that information which has been directly or indirectly taken from other sources has been noted as such. Neither this, nor a similar work, has been published or presented to an examination committee.

Ilmenau, February 28th, 2014

Kathrin Lang

Abstract

This work is about a learning tool which provides the necessary parameters for a program controlling robots of type LUKAS at the Faculty of Mechanical Engineering.

The robot controlling program needs various parameters depending on its environment, like the light intensity distribution, and camera settings as exposure time and gain raw. These values have to be transmitted from the learning tool to the robot controlling software.

Chapter one introduces the robots of type LUKAS which are created for the RoboCup Small Size League. Furthermore, it introduces the camera used for image processing.

The second chapter explains the learning process according to Christoph Ußfeller and deduces the requirements for this work.

In the third chapter theoretical basics concerning image processing, which are fundamental for this work, are explained.

Chapter 4 describes the developed learning tool which is used for the learning process and generates the required parameters for the robot controlling software.

In chapter five practical tests with two test persons are represented.

The sixth and last chapter summarizes the results.

Abstract (German)

Ziel der vorliegenden Arbeit war es, ein Anlernprogram für das Steuerungsprogram der Roboter Typ LUKAS im Fachgebiet Rechneranwendung im Maschinenbau zu entwickeln.

Für dieses Steuerungsprogram müssen verschiedene Umgebungseinflüsse wie z.B. die Helligkeitsverteilung über dem Spielfeld oder Belichtungszeit und Helligkeitsverstärkung der Kamera angelernt werden. Anschließend müssen die im Anlernprozess ermittelten Parameter an das Steuerungsprogram übergeben werden.

Kapitel 1 geht auf den Roboter Typ LUKAS ein und stellt den Zusammenhang zur RoboCup Small Size League her. Außerdem wird kurz die für die Bildverarbeitung verwendete Kamera vorgestellt.

Das zweite Kapitel stellt den Anlernprozess nach Christoph Ußfeller vor und leitet davon die Anforderungen für diese Arbeit ab.

Im dritten Kapitel werden die benötigten theoretischen Grundlagen aus dem Bereich der digitalen Bildverarbeitung erläutert.

Kapitel 4 schließlich beschreibt das entwickelte Learning Tool, das den Anlernprozess übernimmt und die nötigen Daten für das Steuerungsprogram bereitstellt.

Im fünften Kapitel wird kurz auf Versuche mit zwei Probanden eingegangen.

Das sechste und letzte Kapitel fasst die Ergebnisse der Arbeit noch einmal zusammen.

Table of Contents

1 Introduction

The first chapter gives an overview of the RoboCup project and explains which part the Ilmenau University of Technology plays in it. Furthermore it introduces the camera used for determining the position of the robots.

1.1 RoboCup

After 40 years of research, IBM Deep Blue, a chess playing robot, defeated the human world champion in May 1997. Until then the game of chess was the standard problem for artificial intelligence. After it was solved successfully, a new and more complex problem was required. In that year the first RoboCup games were held. [Rob14]

The robot world cup initiative started in Japan. At a Workshop on Grand Challenges in Artificial Intelligence in Tokyo in 1992, a group of Japanese researchers decided the game of soccer could be a new challenge for artificial intelligence. In September 1993 RoboCup was announced publically for the first time. [Rob14]

The goal of the robot world cup initiative is stated as follows:

"By the middle of the 21st century, a team of fully autonomous humanoid robot soccer players shall win a soccer game, complying with the official rules of FIFA, against the winner of the most recent World Cup." [Rob14]

In difference to chess, soccer has a dynamic environment and a continuous game flow – that means real time robot controlling – while in chess the environment is static and the game flow turn taking. In chess, the control is central and all information is accessible. On the contrary, in soccer the control is distributed on the robots and the information is incomplete. [Rob14]

The RoboCup Soccer domain is divided in Humanoid, Standard Platform, Middle Size, Small Size and Simulation. In the Humanoid League there are up to three robots pro Team. Main problem is to keep the balance while moving and taking a shot at the goal. In the Standard Platform League every team uses the same robot hardware. So the main issue is the controlling software. [Beh13]

In the Middle Size League there are up to six robots in a team. These robots are completely autonomous. In the Small Size League there are five robots in a team. Their position is monitored by a camera above the playing field and they are controlled by an external computer. The Simulation League is based on software; the robots are only simulated and the game itself is most important. [Beh13]

Nowadays, there are more domains like RoboCup Rescue, RoboCup@Home and RoboCup Junior. RoboCup Rescue treats disaster scenarios. It includes simulations as well as actual robotic systems. RoboCup @Home includes autonomous robots which can help in household. RoboCup Junior is built up of teams of students. [Beh13] [Rob14]

1.2 Robot LUKAS

A team from the Ilmenau University of Technology is building a robot team in the dimensions of the RoboCupSoccer Small Size League. The name of the robot version is LUKAS.

This robot has a diameter of 180 mm and a weight of 1.7 kg (see Figure 1.1). It can reach a top speed of 4 m/s. The robots contain a position controller; so they can move to a position on the playing field on their own. The robots can be controlled by a robot controlling program which uses a camera above the playing field to detect their positions. The robots can be distinguished by their tricots. [Uss13]

The robots have tricots in different colors. The team marker in the center is yellow or blue. The front of the robot is marked by a white part. And the robot number is coded in a combination of cyan, magenta and green.

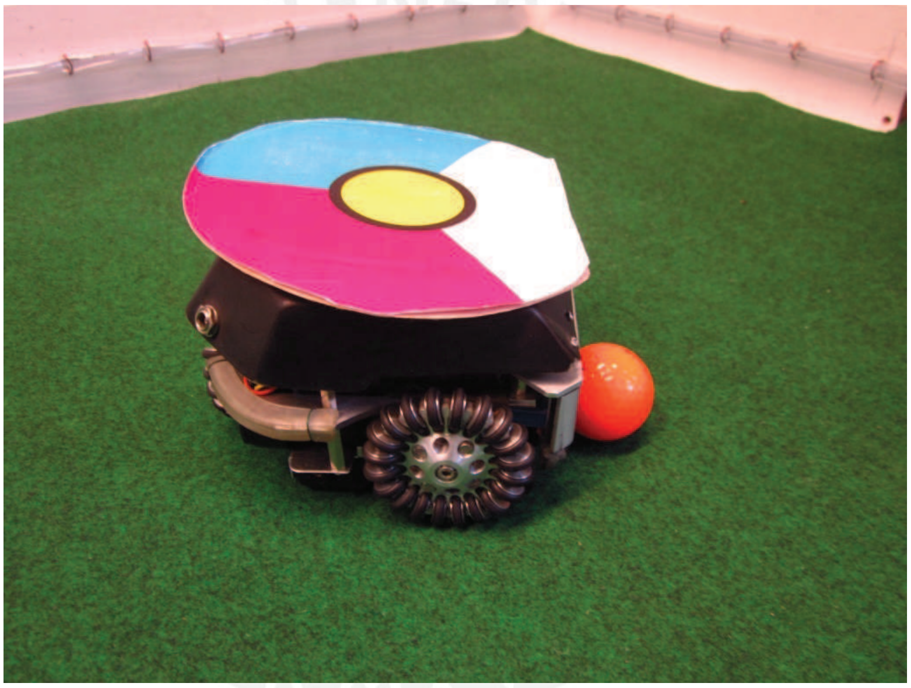


Figure 1.1: Robot LUKAS

At the faculty of Mechanics the software for controlling the robots was written by Christoph Ußfeller [Uss13]. This software needs special parameters (like the light intensity distribution for example) as input values. Because of this a second program (the so-called *learning tool*) should be written which determines these parameters semiautomatic and helps user to adjust the camera. The aim of this learning tool is to make the learning process easily repeatable and useable for everybody by using a self-explanatory user interface.

1.3 Prosilica GC1600H

The Prosilica GC1600H is part of the GigE series of Allied Vision Technologies. Its dimensions are 59 x 46 x 33 mm and its weight is 105 g. The camera contains a Sonic ICX274AL sensor with a resolution of 1620 x 1220 pixels.

The camera offers the possibility to select a region of interest; that is the camera streams only part of the image, but with a higher frame rate.

The frame rate of the camera can be determined by formula (1-1). It can be increased by reducing the height of the image; that is by reducing the region of interest. A reduction of the width of the image does not increase the frame rate. The maximum frame rate of the camera at full resolution is 25 fps.

$$F_r[fps] = \frac{1}{29.24\ \mu s * height\ [pixels] + 3082.14\ \mu s} \qquad (1\text{-}1)$$

Gain control, exposure control and white balance of the camera can be adjusted manually and automatic. Furthermore, the camera is capable of multicast, which means, that the camera image can be streamed on more than one computer at the same time.

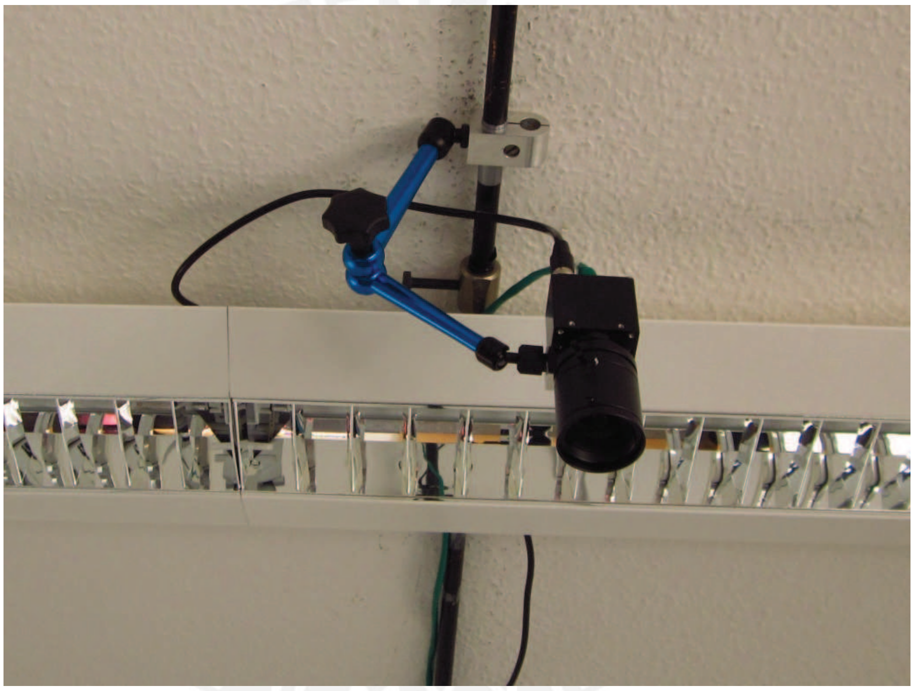


Figure 1.2: Prosilica GC 1600H

The camera uses IEEE 802.3 1000BASE-T as hardware interface standard and GigE Vision Standard 1.2 as software interface standard.[AVT13]

2 Establishment of the Thesis' Task

The second chapter introduces the programs written by Christoph Ußfeller. The task of this work is derived from these programs.

2.1 Software by Ußfeller

The software written by Christoph Ußfeller consists of two programs. The first one is used for guiding the robots during a game. This program requires different parameters like the size of the robots or the recent light intensity distribution. Those parameters are contributed by the second program. The process of providing these parameters is called *learning process*.

Robot Controlling Software

The robot controlling software has to fulfill different tasks. At first, the raw data has to be preprocessed. Following, parts of the image which have the same color are put together into segments. In the last step the position of the robots is determined via Helmert transformation.

The preprocessing takes place in two steps. In the first step the raw data is Bayer interpolated. This is necessary because the used camera contains only one color chip. So the color filters are arranged according to a special pattern. The missing color information has to be reconstructed afterwards. To reconstruct the color information there are different processes. The one used by Ußfeller is called linear interpolation. The information is reconstructed according to formula (2-1) (see Figure 2.1):

$$\begin{pmatrix} v_R \\ v_G \\ v_B \end{pmatrix}_{i,j} = \frac{1}{4}\begin{cases} \begin{pmatrix} 3v_{i+1,j+1} + v_{i-1,j-1} \\ 2v_{i+1,j} + 2v_{i,j+1} \\ 3v_{i,j} + v_{i+2,j+2} \end{pmatrix}, (i \bmod 2 = 0) \wedge (j \bmod 2 = 0) \\ \begin{pmatrix} 3v_{i+1,j} + v_{i-1,j+2} \\ 2v_{i,j} + 2v_{i+1,j+1} \\ 3v_{i,j+1} + v_{i+2,j-1} \end{pmatrix}, (i \bmod 2 = 0) \wedge (j \bmod 2 = 1) \\ \begin{pmatrix} 3v_{i,j+1} + v_{i+2,j-1} \\ 2v_{i,j} + 2v_{i+1,j+1} \\ 3v_{i+1,j} + v_{i-1,j+2} \end{pmatrix}, (i \bmod 2 = 1) \wedge (j \bmod 2 = 0) \\ \begin{pmatrix} 3v_{i,j} + v_{i+2,j+2} \\ 2v_{i+1,j} + 2v_{i,j+1} \\ 3v_{i+1,j+1} + v_{i-1,j-1} \end{pmatrix}, (i \bmod 2 = 1) \wedge (j \bmod 2 = 1) \end{cases} \qquad (2\text{-}1)$$

The second step consists of a light intensity regulation. A static local regulation is combined with a dynamical global one. The local regulation divides the area of the playing field in 37 x 20 parts. For each part exists a coefficient with which the light intensity in this part is multiplied. These coefficients are determined in the learning process and consist of the reciprocal values of the matrix of the average light intensity.

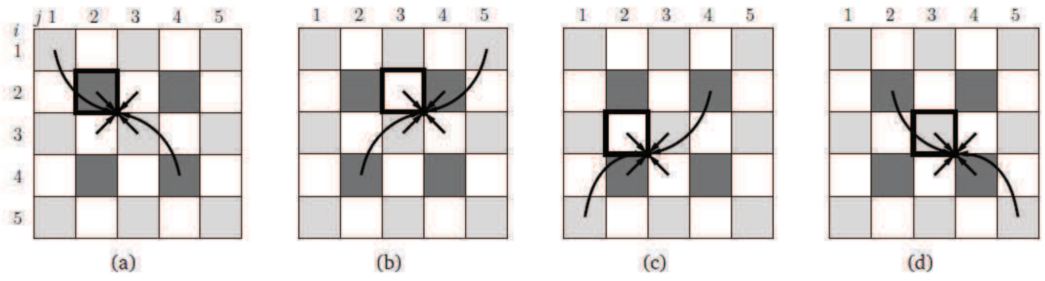


Figure 2.1: Linear Interpolation [Uss13]

The global regulation depends of two parameters of the camera. One is the exposure time of the camera and the other one the gain raw. The gain raw is between 0 and 30 and consists of integers.

The next task of the robot controlling software is the segmentation. In this step pixels of the same color group in each line of the picture are united to groups and afterwards these groups are put together into segments of one color group. Each segment has a center of gravity and an area.

In the last step the direction of the robots is determined. This step uses the Helmert transformation. Input data for the transformation are center of gravity and area of the segments. This data is compared with the positions of the centers of gravity of an ideal tricot.

For the positioning of the robots only the centers of gravity are used. But the segments provide areas as well. These areas can be used to improve the results by reducing the influence of segments which are not detected very well.

[Uss13]

Learning Process

The learning process by Ußfeller consists of three steps: In the first step the camera is adjusted. In the second step the necessary light intensity compensation is determined. And in the third step the color classification takes place.

At the camera there are three joints to adjust its direction. The joints have to be adjusted at the same time. To determine how good the camera has been adjusted there is an image in the program which contains the image of the camera as well as a grid. The grid tells where the playing field should be. When the borders of the playing field are parallel to the grid the camera can be focused via zoom and diaphragm at the camera. These adjustments of the camera are hardware based. The software is only used to display the camera image for reference purpose.

For the first step there have to be some robots on the playing field to decide if the camera is well-focused, but the second step does not work properly if there are any objects on the field. So, between step one and two, the user has to empty the playing field.

The second step treats the light intensity compensation. At first the user has to mark the playing field in the camera picture. Afterwards the program creates a so called gain grid. It consists of the coefficients, with which the light intensity value of each part of the field has to be multiplied to get the same intensity in the whole field. These coefficients are written in a text document.

After the second step the program has to be closed and reopened. Besides, some robots (or at least their tricots) have to be put on the playing field.

For the color classification a color has to be selected (that is, the exact name of the color has to be written in a field) and after that, parts of the picture, which are in this color, have to be marked. This is done with all necessary colors. After collecting this data a second program has to be used to generate a lut-file out of this data. The lut-file provides the information which RGB value belongs to which color.

2.2 Derivation of Task

The learning process by Ußfeller has different advantages and disadvantages. It should be modified and joint to a learning tool. Basing on the learning process the learning tool has to fulfill different requirements.

Both programs are written in C++. While the robot controlling program can be used on any computer using Windows as well, the learning program by Ußfeller runs on exactly one computer using Linux and can't be transferred to another computer. So the first requirement is *mobility*. That is, the learning tool can be transferred to different computers and does not depend on a special one.

One big disadvantage of the learning program by Ußfeller is that the user has to know in which order the program has to be used. For example there is no reference that the program has to be closed and reopened or that a second program is needed to create the lut-file. Furthermore, for selecting a color the user needs to know the exact name of the color as it is used for generating the lut-file. If it is spelled another way the program will not recognize it. So the next requirement is *user-independency*. For this the new learning tool has to be self-explanatory or at least well-documented.

On the other hand, the learning process by Ußfeller does work. At least, if one has experience in using the process. So the third requirement is an *improvement of the three basic steps*.

At first the camera has to be adjusted hardware based. While doing this the user needs to be able to watch the screen. The camera is adjusted correctly when the grid displayed in the image is parallel to the borders of the playing field. It is very difficult to adjust the camera while watching the screen so some sound-signal for notifying when the orientation of the camera is all right would be an improvement.

Secondly, for the light intensity compensation the user has to mark the area of the playing field by hand. In the new learning tool the program should find the borders of the playing field automatically, if possible.

At last the color classification should be performed more automatically. In the learning process by Ußfeller the user has to know the exact name of the color. Furthermore, he has to mark the area of this color by hand. The new learning tool should recognize the colors and areas by itself. In ideal case the user just has to press one button to start the process.

Another requirement is to *collect the necessary parameters and transmit them* to the robot controlling program. In the learning process by Ußfeller this happens via three text files with numbers but without explanations in them. In the new learning tool the number of text files should be reduced and the parameters in the text files named.

Last but not least, the learning process should take place using only *one single program*, the so called *learning tool*.

3 Digital Image Processing

The third chapter describes the theoretical foundations for this work. It contains information about digital image processing topics like color models or edge detection and explains tools for digital image processing written in C#, for example the BlobCounter algorithm by Aforge .Net.

3.1 Colors and Color Models

The soccer playing process is based on different colors (see Figure 3.1): The ball is orange. Small circles in yellow or blue in the jerseys of the robots tell which team the robot belongs to (team yellow or team blue). In addition the colors green, cyan and magenta are used to encode the player number of the robot while the white segment always shows the front of the robot. Other colors are defined as "rest" which includes for example the dark green playing field. [Uss13]

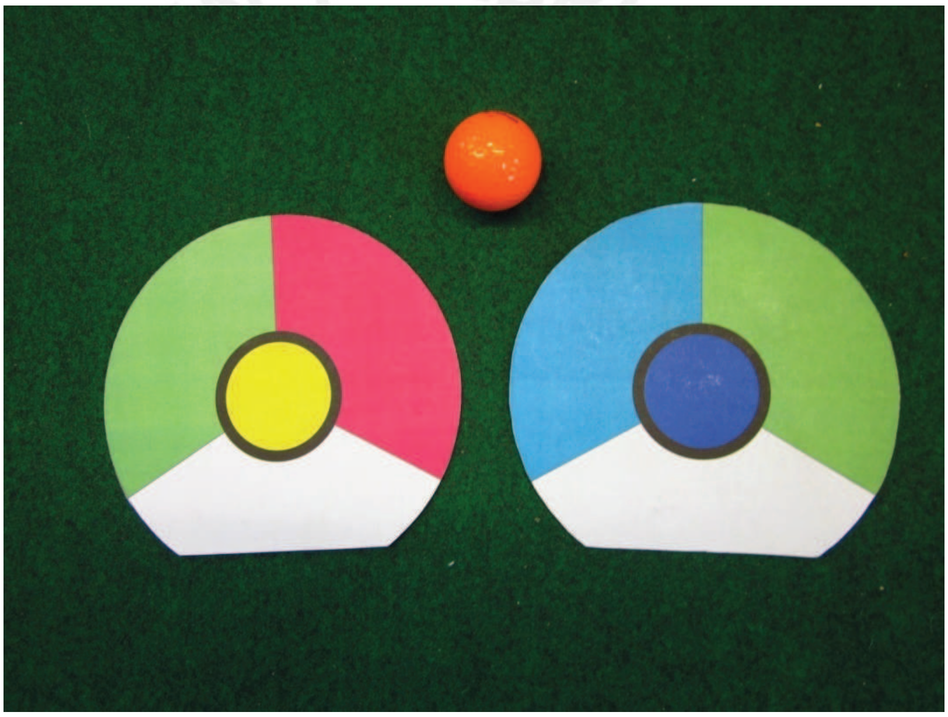


Figure 3.1: Robot Tricots and Ball

Usually a human can tell which part of the picture belongs to which of the color groups named above. That is because humans have learned as children how each color looks like. But the camera as well as the software analyzing the pictures doesn't know how these colors look like. Because of this the program has to learn the colors, too.

But how do you tell a program how a color looks like? For this the color has to be somehow classified. Different color models yet exist. For example the most common system, the RGB (red, green, blue) system, is used for monitors and cameras while the CMY (cyan, magenta, yellow) system is used for printers. The one most similar to the human visual perception is the HSI(Hue, Saturation, Intensity) system.[Gon08]

RGB

This system uses red, green and blue as elementary colors. The combination is called *additive colors* because the combination of these colors produces the color white. The RGB system is used for describing things which emit light like monitors. [Ric09]

CMY

The CMY system is similar to the RGB system. The difference is it uses the complementary colors of the RGB system. These are cyan, magenta and yellow and they are called *subtractive colors*. Their combination produces the color black. [Ric09]

For changing from the RGB system into the CMY system formula (3-1) is used.

$$\begin{bmatrix} C \\ M \\ Y \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} - \begin{bmatrix} R \\ G \\ B \end{bmatrix} \tag{3-1}$$

HSI

It is the system most similar to the human view. The abbreviation does not stand for colors this time but for the words Hue, Saturation and Intensity. The hue value defines a color at the chromatic circle in the unit degree (for example red is equal 0°). Saturation stands for the distance to the center of the chromatic circle; the border of the circle corresponds to 100%. The intensity or brightness is found on the vertical axis; 0% means a very dark color. [Ric09]

The conversion from RGB system to HSI system is described by formula (3-2) to (3-4). [Ric09]

$$H = \begin{cases} \theta \\ 360° - \theta \end{cases} if \begin{matrix} B \leq G \\ B > G \end{matrix}$$

$$with \quad \theta = \cos^{-1}\left\{ \frac{\frac{1}{2}[(R-G)+(R-B)]}{[(R-G)^2+(R-B)(G-B)]^{\frac{1}{2}}} \right\} \tag{3-2}$$

$$S = 1 - \frac{3}{R+G+B}\left[\min(R,G,B)\right] \tag{3-3}$$

$$I = \frac{1}{3}(R+G+B) \tag{3-4}$$

Other color systems exist as well but to mention all of them would be too much for this work.

3.2 Sobel Operator

The Sobel operator is a high pass filter which is used for edge detection. Usually it consists of a 3x3 filter matrix.

The Sobel filter matrix is the product of a gradient filter and a mean filter (see formula (3-5)and (3-6)).

$$F_{Sobel}(x) = \begin{pmatrix} 0 & 1 & 0 \\ 0 & 2 & 0 \\ 0 & 1 & 0 \end{pmatrix} * \begin{pmatrix} 0 & 0 & 0 \\ 0 & -1 & 1 \\ 0 & 0 & 0 \end{pmatrix} = \begin{pmatrix} 0 & -1 & 1 \\ 0 & -2 & 2 \\ 0 & -1 & 1 \end{pmatrix} \tag{3-5}$$

where: $\begin{pmatrix} 0 & 0 & 0 \\ 0 & -1 & 1 \\ 0 & 0 & 0 \end{pmatrix}$ - gradient filter for lines

$\begin{pmatrix} 0 & 1 & 0 \\ 0 & 2 & 0 \\ 0 & 1 & 0 \end{pmatrix}$ - mean filter for lines

$$F_{Sobel}(y) = \begin{pmatrix} 0 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 1 & 0 \end{pmatrix} * \begin{pmatrix} 0 & 0 & 0 \\ 1 & 2 & 1 \\ 0 & 0 & 0 \end{pmatrix} = \begin{pmatrix} 0 & 0 & 0 \\ -1 & -2 & -1 \\ 1 & 2 & 1 \end{pmatrix} \tag{3-6}$$

where: $\begin{pmatrix} 0 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 1 & 0 \end{pmatrix}$ - gradient filter for rows

$\begin{pmatrix} 0 & 0 & 0 \\ 1 & 2 & 1 \\ 0 & 0 & 0 \end{pmatrix}$ - mean filter for rows

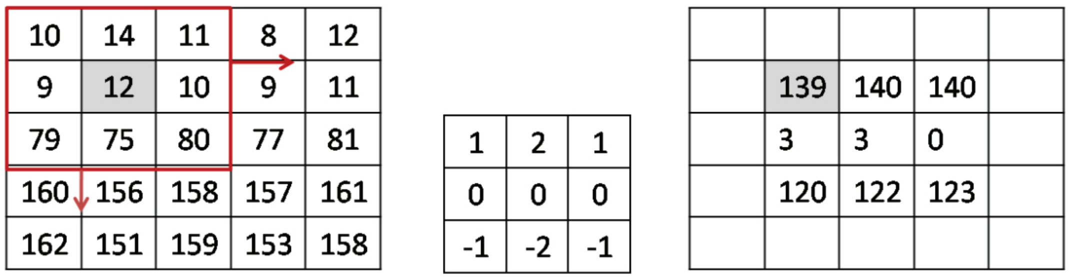
The gradient filter is of first order and detects edges in an image while the mean filter for neighbored lines / rows makes the image smoother and eliminates noise. For the Sobel filter, the central line / row of the mean filter is twice of a neighbored line / row. [Ric09]

According to Bässmann, the Sobel operator can be written as in formula (3-7) as well. [Bäs04]

$$F_{Sobel}(x) = \begin{pmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{pmatrix}, F_{Sobel}(y) = \begin{pmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{pmatrix} \tag{3-7}$$

Figure 3.2 shows an example for a Sobel filtering process. The original image is represented by its grey level values. The filter matrix is put in the left upper edge and each value of the original image is multiplied with the corresponding value of the filter matrix. The results of these nine multiplications are summed up and written in the central position of the filter matrix (marked with grey background).

Thereafter the filter matrix is moved step by step to the right and down and new values are determined. As it is not possible to calculate values for the borders of the image, these are filled with default values.



a) Original Image

10	14	11	8	12
9	12	10	9	11
79	75	80	77	81
160	156	158	157	161
162	151	159	153	158

b) Filter Matrix

1	2	1
0	0	0
-1	-2	-1

c) Scaled and Filtered Image

	139	140	140	
	3	3	0	
	120	122	123	

Figure 3.2: Example for Sobel Filter

Some of these new values can be negative, so in the next step the filtered image has to be scaled to real grey level values. Figure 3.2 c shows the scaled and filtered image.[Ric09]

3.3 Hough Transform

The Hough Transform detects lines, circles, ellipses and other objects with known shape in images. One big advantage of the Hough Transform is that it is able to find those objects even if their borders are not complete. The Hough transform was introduced in 1962. [Nix08][Sch95][Tre10]

The Hough transform for lines starts with the definition for lines in Cartesian parameterization. As one can see in formula (3-8), the line can be defined by a pair of coordinates (x, y) as well as by a pair of parameters, the slope m and the intercept c.[Nix08]

$$y = m\,x + c \tag{3-8}$$

In normal form the line is written as in formula (3-9) [Tre10]. In this case the line is characterized by the parameters r and α.

$$r = x\cos\alpha + y\,sin\,\alpha \tag{3-9}$$

The values of r and α can be inserted in a 2D accumulator space. This is a table which at the beginning is filled with zeroes. Each pair of parameters increases the corresponding cell by one. At the end, the local maximum value in the table marks the values for r and α of the line. [Sch95][Tre10]

The Hough transform can only find lines of infinite length. That means it can determine the position of the line but not its actual starting and end points. [Seu00]

3.4 AForge .Net Framework: BlobCounter

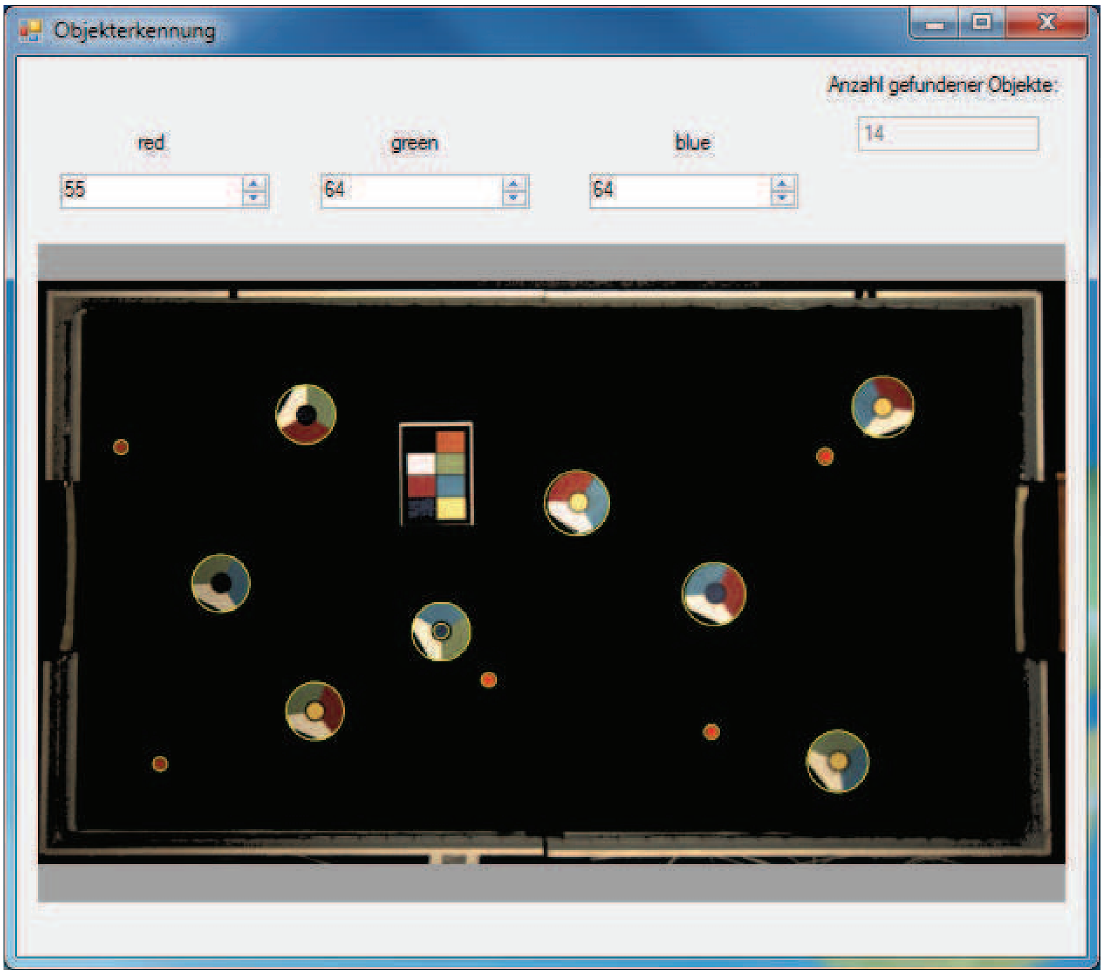
AForge .Net Framework is a C# framework for Computer Vision and Artificial Intelligence. Its first version was released on December 21st 2006 by Andrew Kirillov. The current version is version 2.2.5. The AForge .Net Framework is released under LGPL license. [Kir09][Kir11]

The BlobCounter tool by AForge .Net Framework recognizes objects which are separated by background. The default background color is black but can be changed in the code. The background color is assigned in RGB format.

```
void ObjectDetection (Bitmap image)
//Turn background of the image to black
ColorFilter(ref image, red, green, blue);
//Look for objects
blobCounter = new BlobCounter();
blobCounter.minWidth = widthMin;
blobCounter.maxWidth = widthMax;
blobCounter.ProcessImage(image);
Blob[] blobs = blobCounter.GetObjectsInformation();
```

Algorithm 3.1: Implementation of BlobCounter

Parameters which can be changed are the minimum and maximum size of the found objects. Furthermore, the threshold values for the color filters can be altered. Algorithm 3.1 shows how the BlobCounter can be used.


Objekterkennung

Anzahl gefundener Objekte: 14

red: 55

green: 64

blue: 64

Figure 3.3: BlobCounter algorithm, looking for circles

Figure 3.3 shows an example program using the BlobCounter algorithm. The color filter values can be altered until every circle is found. Other geometrical objects than circles are ignored. Circles found by the algorithm are marked with yellow circles.

3.5 DBSCAN Algorithm

The abbreviation DBSCAN signifies *Density Based Spatial Clustering of Applications with Noise*. The DBSCAN algorithm was introduces in 1996. [Clu09][Dun03]

One Advantage of the DBSCAN algorithm is that it can detect clusters of various shapes and sizes. Furthermore, it is capable of filtering noise in images. On the other hand, the DBSCAN algorithm has difficulties to find the correct clusters when they have different densities. In this case it is possible that the algorithm detects noise as clusters. [Clu09]

The DBSCAN algorithm divides a quantity D of elements into clusters. Each cluster has a minimum size and density. Dunham defines density as "a minimum number of points within a certain distance [Eps] of each other." [Dun03] The minimum size of a cluster is characterized by the minimum number of points MinPts in the cluster. How many clusters the algorithm creates is determined by the algorithm itself. [Dun03]

The sequence of the DBSCAN algorithm shows Algorithm 3.2. The phrase *density-reachable* signifies that the distance between element i and element j is less than Eps. The cluster is valid if each element in it has at least MinPts neighbors within a radius Eps. [San99]Elements, which can't be assigned to a cluster, are defined as noise. [Dun03]

List<Cluster>dbscan(List<Elements> D, int MinPts, double Eps)

```
List<Cluster> K;
int k = 0 ;   //no clusters at the beginning
for (i = 0; i< n; i ++)
  if (D[i] not in a cluster)
    Cluster X = Cluster (D[j] | D[j] is density-reachable from D[i]);
    if (X is valid cluster)
      k ++;
      K[k] = X;
    end
  end
end
return K;
```

Algorithm 3.2: DBSCAN algorithm[Dun03]

The expected time complexity of this algorithm is O(n*t), where n is the number of elements and t the time for counting the elements within a distance of Eps. In worst case, the time complexity can be O(n^2). In most cases it is possible to reach a time complexity of O(n log n). [Clu09]

4 Learning Tool

The forth chapter introduces the new *learning tool* as the program for determining the required parameters is called now. It starts with the general design of the tool, explains the main steps and shows how the results are transmitted to the robot controlling software.

4.1 General Design

The learning tool should fulfill different specifications. It has to be user-friendly and self-explanatory. One can see these tasks are reflected in the design of the tool.

4.1.1 Required Parameters

The robots are controlled by special software. This software requires various parameters for example the size of one robot or the light intensity distribution. All of these parameters have to be generated by the learning tool.

The first group of parameters depends on the camera settings. It includes name and ID of the camera as well as settings like gain raw, exposure time and white balance values.

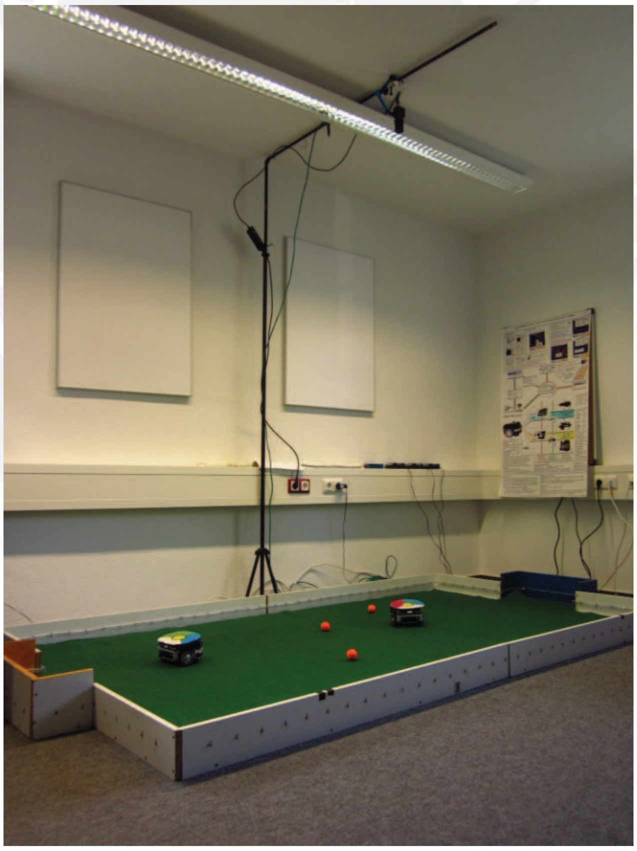


Figure 4.1: Playing Field with robots

Another group of parameters depends on the robots. Each robot has a diameter and a height. For the learning tool, I assume that each robot has the same diameter and height. The diameter of the ball is of interest as well.

Furthermore, each robot has a tricot for distinguishing them. The assignment of the tricots has to be defined in the learning tool. Another parameter of the tricot is the diameter of the circle which marks the team the robot belongs to.

The next group of parameters depends on the playing field. These include length and width of the field as well as the height of the camera above the field. The position of the playing field in the camera image is another value for this group.

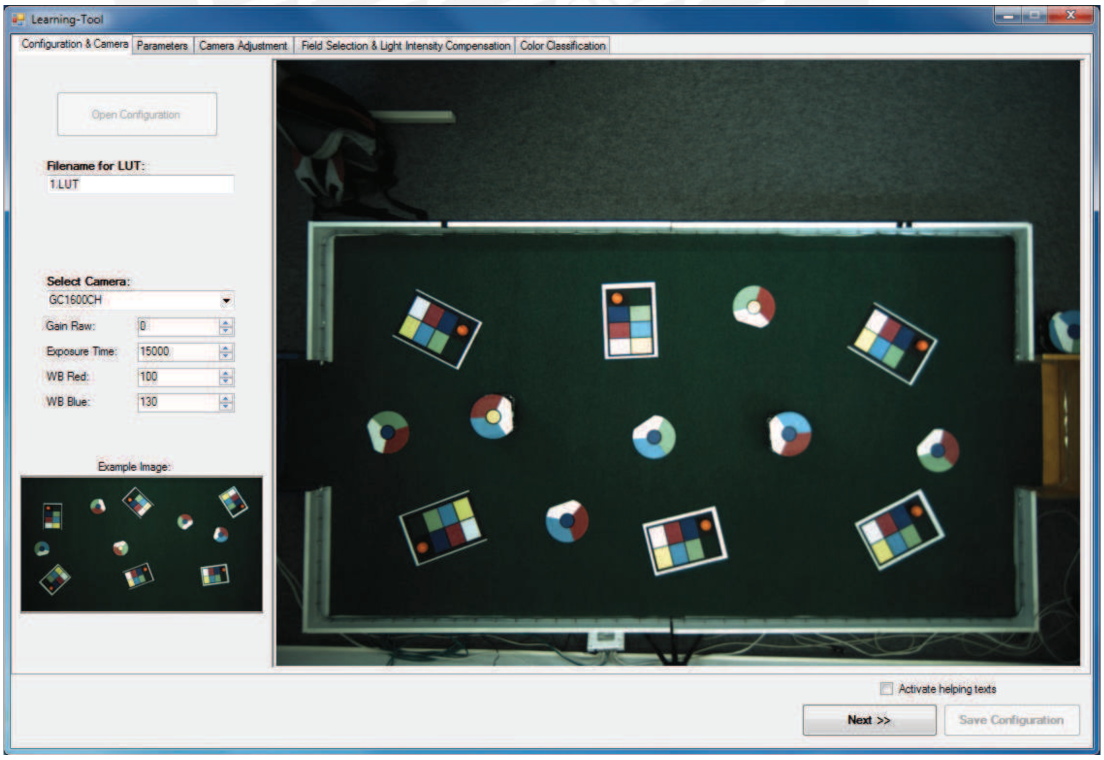
Of great importance is the light intensity distribution. It differs above the field and depends on the light intensity, position of the lamps etc. The configuration file should include in how many sections the field is divided in x- and y-direction as well as the light intensity compensation matrix.

The last information which has to be generated for the robot controlling program is called lut-file. This file is used for assigning a RGB value to a color name.

4.1.2 Program Flow

After starting the learning tool there are two possibilities. The first one is to create a new configuration and the second to alter an already existing one.

Creating a new configuration starts with *Tab One, Configuration and Camera* (see Figure 4.2*)*. Here the file name for the lut file can be defined. Furthermore, a camera can be selected. After selecting a camera the camera image is displayed. Some basic settings like gain raw, exposure time and white balance can be made. An example image shows how the camera image should more or less look like.



Filename for LUT: 1.LUT

Select Camera: GC1600CH

Gain Raw: 0

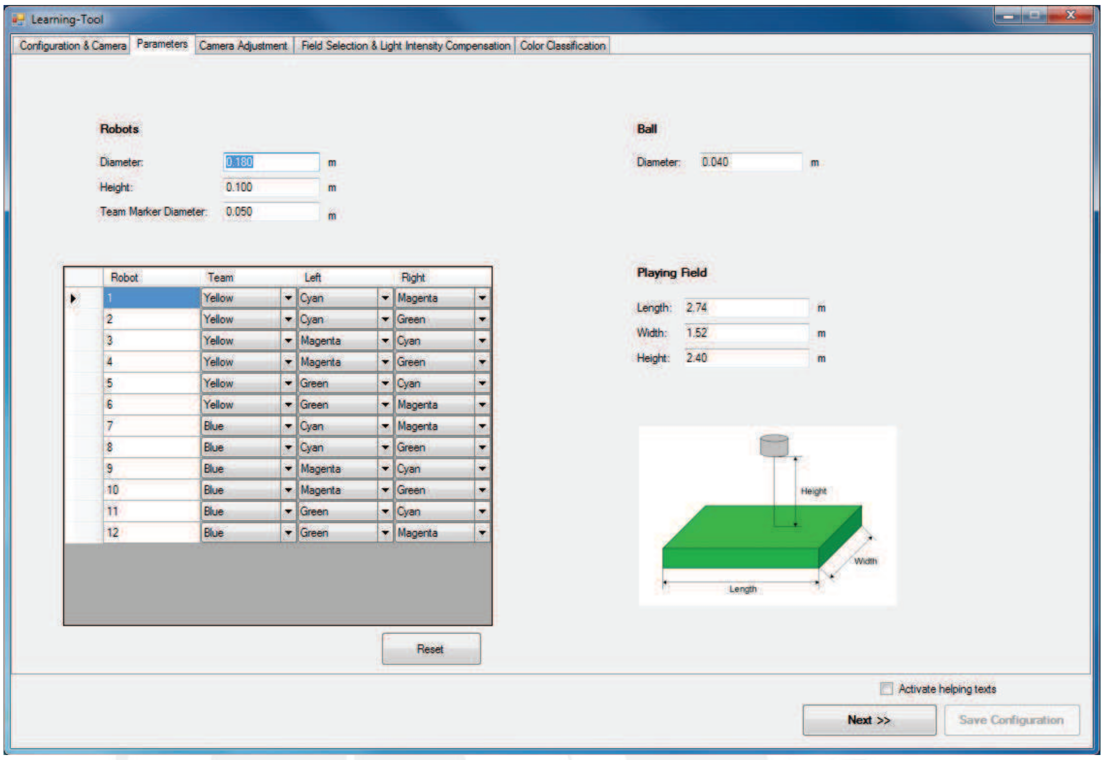
Exposure Time: 15000

WB Red: 100

WB Blue: 130

Figure 4.2: Learning Tool Tab One: Configuration and Camera

In *Tab Two, Parameters*, the dimensions of the robots and the playing field can be altered (see Figure 4.3). In addition the tricot colors of each robot are presented.



Robots

Diameter: 0.180 m
Height: 0.100 m
Team Marker Diameter: 0.050 m

Robot	Team	Left	Right
1	Yellow	Cyan	Magenta
2	Yellow	Cyan	Green
3	Yellow	Magenta	Cyan
4	Yellow	Magenta	Green
5	Yellow	Green	Cyan
6	Yellow	Green	Magenta
7	Blue	Cyan	Magenta
8	Blue	Cyan	Green
9	Blue	Magenta	Cyan
10	Blue	Magenta	Green
11	Blue	Green	Cyan
12	Blue	Green	Magenta

Ball

Diameter: 0.040 m

Playing Field

Length: 2.74 m
Width: 1.52 m
Height: 2.40 m

Figure 4.3: Learning Tool Tab Two: Parameters

In *Tab Three, Camera Adjustment*, a grid is drawn in the camera image. In this tab, the camera should be adjusted hardware based. This step requires robots on the playing field.

In *Tab Four, Field Selection and Light Intensity Compensation*, the position of the playing field can be marked and the coefficients for the light intensity compensation are calculated. For this step the playing field has to be empty.

After the forth tab the configuration can be saved.

In *Tab Five, Color Classification*, the lut file is created. After this step is done, the learning process is finished and the learning tool can be closed. While closing the learning tool, there is another possibility to save the configuration file. The lut file does not need to be saved; it is saved automatically in *Tab Five, Color Classification*.

In the second possibility in *Tab One, Configuration and Camera,* an already existing configuration can be load. This includes all parameters except the lut matrix.

After loading a configuration single parameters or the position of the playing field can be altered. A new light intensity compensation can be generated as well. For the robot controlling program, a new lut file can be created in *Tab Five, Color Classification*, or an old already existing one can be used.

4.1.3 Design

The general design of the learning tool is shown in Figure 4.2.

The main part of the learning tool is a *TabControl Object* with five tabs. The tabs and their output parameters are listed in Table 4.1.

Tab Name	Parameters
Configuration and Camera	- Camera name and ID - Gain raw, exposure time, white balance
Parameters	- Robot dimensions - Ball dimensions - Dimensions of playing field - Robot-tricot assignment
Camera Adjustment	/
Field Selection and Light Intensity Compensation	- Position of playing field - Number of elements in x-/y- direction - Gain grid
Color Classification	- Lut matrix

Table 4.1: Tabs and distributed parameters

Moreover, there are two buttons on the form. With the *Next Button* the user can shift from one tab to the next one. With the *Save Configuration Button* the recent configuration can be saved as configuration file (Note: This does NOT include the lut file!). After *Tab Four, Field Selection and Light Intensity Compensation*, the necessary parameters for the configuration file are known.

If the check mark at *Activate helping texts* is set, there are texts shown on each tab which explain what to do. Furthermore, a *ToolTip* is activated. If the mouse hovers above one object, a little explaining text is faded in.

Last but not least, one design rule for the learning tool consists in deactivating the controls which are not needed currently. For example, as long as no camera is selected, gain raw and exposure time can't be changed. The progress bar in *Tab Five, Color Classification*, is another example. It's invisible until the color classification is started.

4.2 Connection to Camera

To get a connection to the camera an instance of the camera program *vimba* is used. The function *init()* creates a new instance, and the function *shutdown()* closes the connection to the camera program.

Feature	Value
Pixel Format	BGR8Packed
Offset X	0
Offset Y	0
Height of Image	1220
Width of Image	1620

Table 4.2: Camera Settings

After an instance of *vimba* is created, a camera in the network can be selected and connected. Basic settings for the Prosilica GC1600H are listed in Table 4.2.

At first the function *CameraList()* is called. It returns a list of in the network accessible cameras. It contains name and id of each camera.

Second, a camera is selected and a connection to this camera is opened. The function *connect(cameraId)* is used. After a connection to a camera is opened, single images or a stream of images can be acquired. The connection is closed using the function *disconnect()*.

The function *getImage(cameraId)* returns a single image. It fetches the recent frame from the camera and converts it to an image using *ConvertToBitmap(frame, ref bitmap)* (see Algorithm 4.1).

void ConvertToBitmap(Frame frame, ref Bitmap bitmap)

```
if (frame == null) throw new ArgumentNullException(“frame”); end
if (bitmap == null) bitmap = new Bitmap (frame.Width, frame.Height,
                                         PixelFormat.Format24bppRgb); end
bitmapData = bitmap.LockBits(new Rectangle(0, 0, frame.Width, frame.Height),
WriteOnly, PixelFormat.Format24bppRgb);
try
  for (y = 0; y <frame.Height; y ++)
InteropServices.Marshal.Copy(frame.Buffer, 3*y*frame.Height,
                new IntPtr(bitmapData.Scan0 + y*bitmapData.Stride), 3*frame.Width);
  end
end
finally
bitmap.UnlockBits(bitmapData);
end
```

Algorithm 4.1: Converts a Frame to a Bitmap

The streaming process is started and stopped with the functions *start()* and *stop()* as shown in Algorithm 4.2 and Algorithm 4.3.

void start()

```
try
isWorking = false; isStartet = true;
if (m_isopen) return; end
m_isopen = true;
m_camera.OnFrameReceived += m_camera_OnFrameReceived;
m_camera.StartContinuousImageAcquisition(frameBufferSize);
end
catch
m_isopen = false; isStarted = false;
m_camera.OnFrameReceived -= m_camera_OnFrameReceived;
end
```

Algorithm 4.2: Starts the streaming process

void stop()

```
try
  if (!= m_isopen) return; end
m_isopen = false;
m_camera.OnFrameReceived -= m_camera_OnFrameReceived;
  try
m_camera.StopContinuousImageAcquisition();
```

```
  end
  catch
  end
catch
m_isopen = true;
m_camera.OnFrameReceived += m_camera_OnFrameReceived;
end
```

Algorithm 4.3: Stops the streaming process

Algorithm 4.2 and Algorithm 4.3 both call the function *OnFrameReceived(frame)* (see Algorithm 4.4).

```
void m_camera_OnFrameReceived (Frame frame)
if (isWorking == false)
  if (frameBuffer != null)
m_camera.QueueFrame(frameBuffer);
  end
frameBuffer = frame;
isWorking = true;
end
else
m_camera.QueueFrame(frame);
end
```

Algorithm 4.4: Every time a new frame is received

4.3 Main Steps

The learning process consists of three main steps. At first the camera has to be adjusted hardware based. Secondly, the light intensity compensation takes place. And the last step is the color classification where the lut file is created.

4.3.1 Camera Adjustment

The camera has to be adjusted hardware based. At first the camera has to be directed on the playing field and adjusted parallel to its borders. Secondly, zoom and diaphragm have to be adapted. For the camera adjustment the user has to spread some robots on the playing field.

The first idea was to use the BlobCounter algorithm (see chapter3.4) for rectangles to find the borders of the playing field. This wasn't functional because the BlobCounter algorithm only detects areas and no lines. Looking for the playing field itself did not work because the dark green of the playing field and the grey of the bottom around the field couldn't be separated by a color filter. For recall: The BlobCounter algorithm requires black background around the objects.

The second idea was to use edge detection for finding the borders. At first filters were used to intensify the edges in the camera image. Afterwards the borders can be found via Hough transformation (see chapter 3.3). The advantage of the Hough transformation is that it doesn't require continuous lines, so the empty space for the goals wouldn't matter.

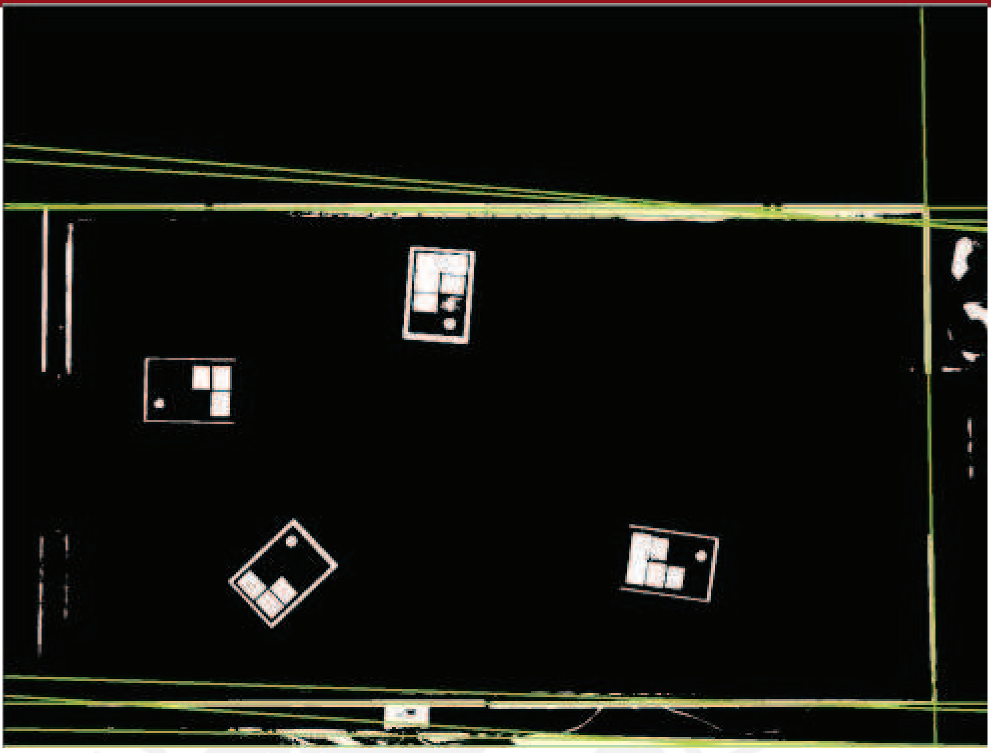


Figure 4.4: Hough lines, detected in the color filtered camera image

In ideal case the result of the Hough transformation would be four lines representing the borders of the playing field. The camera is adjusted parallel to the playing field if the Hough lines are parallel to the image borders (via the slope of the lines). A sound signal with increasing frequency would give audible feedback. Zoom and diaphragm are adjusted correctly if the BlobCounter algorithm detects every robot on the field.

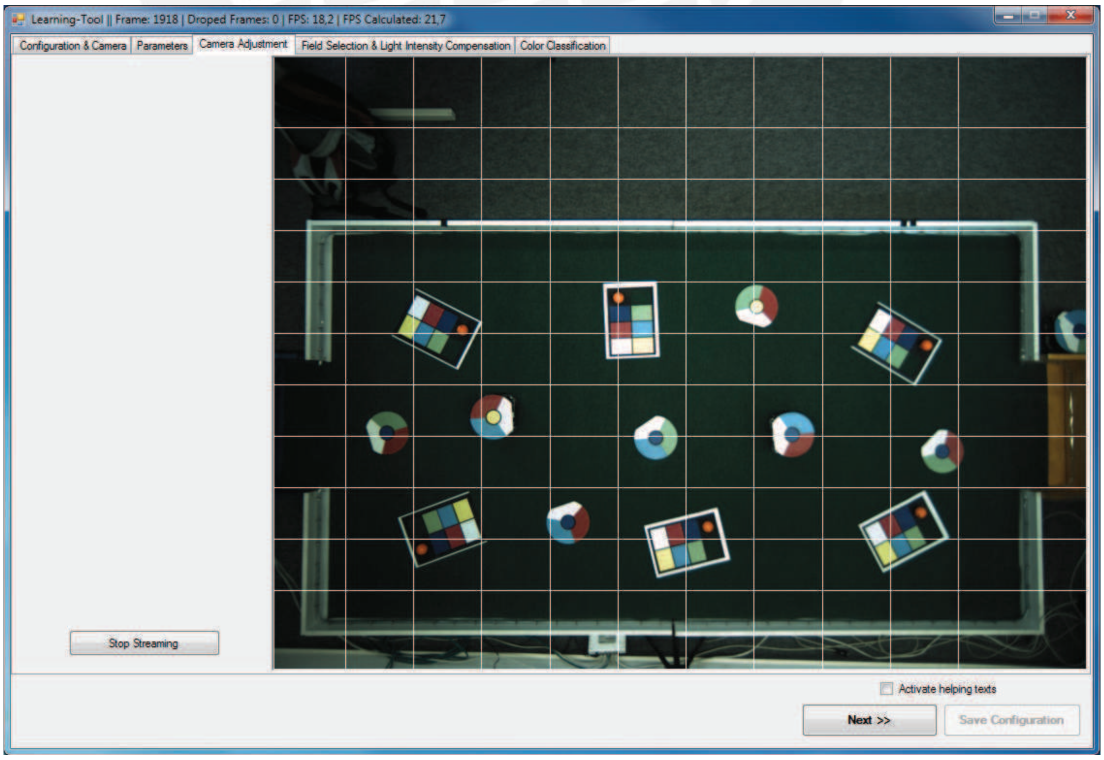


Figure 4.5: Learning Tool Tab Three: Camera Adjustment

The idea with the Hough transformation did not work properly as well. It did not find all four borders of the playing field, independent of the number of Hough lines. Furthermore, the borders the algorithm did find couldn’t be detected in every frame. So the algorithm detected neither reliably the lines nor all of them.

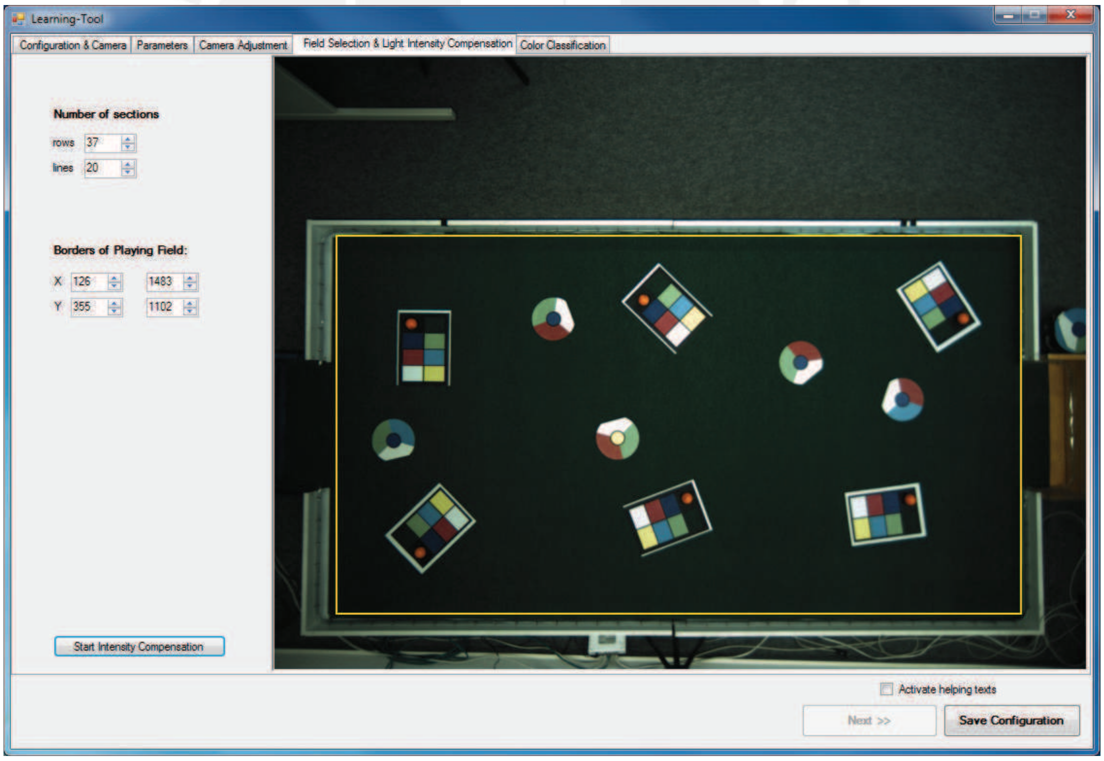
The problem is the white wall next to the playing field. It generates many Hough lines which are more intensive than the desired ones (see Figure 4.4).

The current solution consists in drawing a grid on the camera image (see Figure 4.5). The grid contains of horizontal and vertical lines and can be moved using the mouse. Moreover, the distance between the lines can be altered using the mouse wheel.

For adjusting the camera a second person would be utile. One person could watch the screen while the other one is moving the camera.

4.3.2 Light Intensity Compensation

The light intensity compensation is necessary because of the inhomogeneous distribution of the light intensity on the playing field. One reason is that the playing field is erratic illuminated because of the distribution of the light sources like lamps or windows. Another reason is the fall off in brightness towards the edge of the objective. Furthermore, there are random effects caused by electronic parts. [Uss13]



Number of sections
rows 37
lines 20

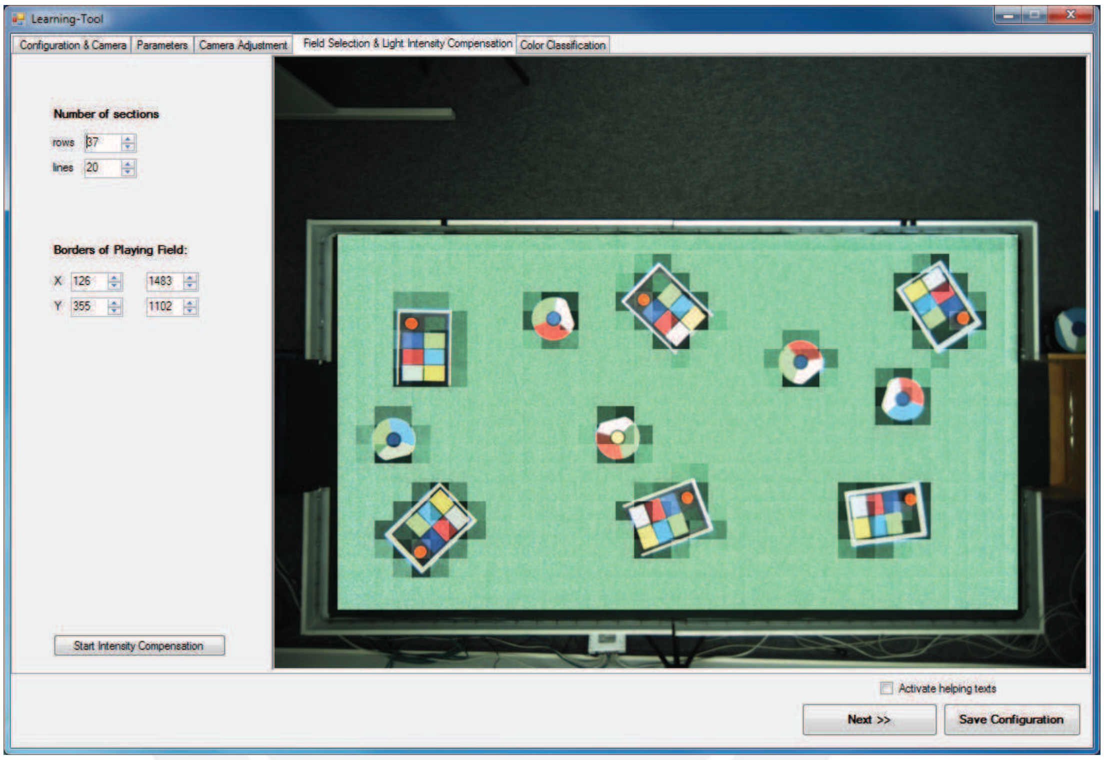
Borders of Playing Field:
X 126 1483
Y 355 1102

Figure 4.6: Learning Tool Tab Four: Field Selection

When tab four is entered, a single frame is requested from the camera. If an old configuration was opened, this frame is displayed as image with light intensity compensation; otherwise the original image is displayed.

In the next step, the user can mark the area of the playing field (see Figure 4.6). For this, he clicks with the mouse in the left upper edge of the field and drags the mouse arrow to the right lower edge. The outline of the marked area is drawn as a yellow rectangle. For little changes of the position the *numericUpDown* controls on the left side of the image can be used.

Besides, the user can chose in how many parts the playing field is divided in x – and y – direction. 37 parts in x – direction and 22 in y – direction is set as default.



Number of sections
rows 37
lines 20

Borders of Playing Field:
X 126 1483
Y 355 1102

Activate helping texts

Figure 4.7: Learning Tool Tab Four: Light Intensity Compensation

One click at the button starts the process. The *btn_Click* event calls the function *LightIntensityDistribution*(see Algorithm 3.2Algorithm 4.5). Input values for this function are the camera image, start and stop position of the playing field in x and y direction and the number of areas in x and y direction. The function returns length and width of one area and the light intensity distribution matrix.

double[,] LightIntensityDistribution(Bitmap image, intw_count, inth_count, intw_min, intw_max, inth_min, inth_max, out int e, out int f)

```
1   intensity = new double[w_count, h_count];
2   count = new int[w_count, h_count];
3   intf = ((h_max – h_min) / h_count) + 1; int e = ((w_max – w_min) / w_count) + 1;
    //sum up pixel intensities for each area
4   for (a = w_min; a <= w_max; a ++)
      row = (a – w_min) / e;
6     for (c = h_min; c <= h_max; c ++)
        line = (c – h_min) / f;
        Color pixel = image.GetPixel(a, c);
        intensity[row, line]+ = Intensity(pixel); count[row, line] ++;
      end
```

```
end
//determine average intensity in each area
for (k = 0; k <w_count; k ++)
  for (l = 0; l <h_count; l ++)
    intensity[k, l] /= count[k, l];
  end
end
//scale intensity
for (k = 0; k <w_count; k ++)
  for (l = 0; l <h_count; l ++)
    if (intensity[k, l] >i_max)
i_max = intensity[k, l];
    end
  end
end
for (k = 0; k <w_count; k ++)
  for (l = 0; l <h_count; l ++)
    intensity[k, l] = i_max / intensity[k, l];
  end
end
return intensity;
```

Algorithm 4.5: Determine Light Intensity Distribution

After determining the values of the light intensity compensation matrix, the image is redrawn. The intensity of every pixel in the playing field is multiplied with its gain, using the HSI area.

Figure 4.7 shows the effect of the light intensity compensation. To increase the effect for better demonstration, the robot tricots and color classification sheets are already on the field. While generating the configuration file this step has to be done with an empty playing field.

4.3.3 Color Classification

Because humans have learned as children how each color looks like, they can tell which part of the picture belongs to which of the color groups. But the camera as well as the software analyzing the pictures doesn't know how these colors look like. So the program has to learn the colors, too.

The result of this learning process will be the lut file. This file assigns which color value belongs to which color group.

The first idea was to use the BlobCounter algorithm (see chapter 3.4) to find the robots and balls in the camera image. Afterwards the program could collect the pixels in rectangles around the robots and balls and save them in a new file.

The pixel collection could be classified into color clusters using the DBSCAN (see chapter 3.5) algorithm. The user would have to tell the program which color cluster belongs to which color. In the last step, these clusters would be expanded and saved as lut file.

This idea had some disadvantages. First of all, the DBSCAN algorithm needed a lot of time. For example, a test with two million pixels was aborted after 23.5 hours. These two million pixels contained a lot of identical color values. So the number of pixels can be reduced by using each color value only one time.

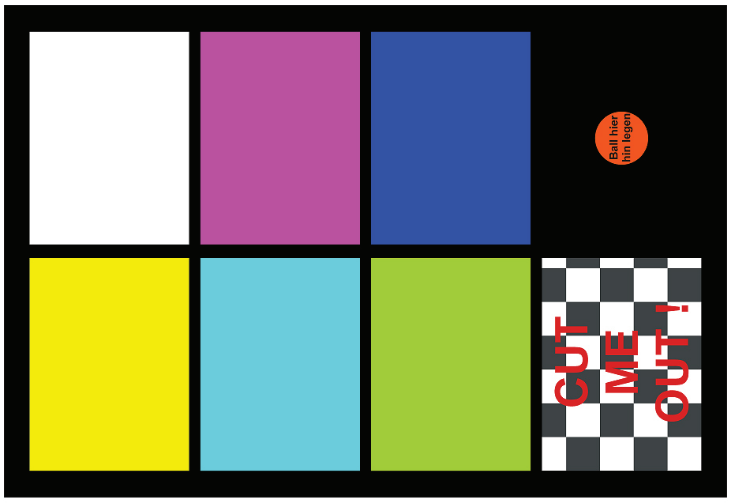
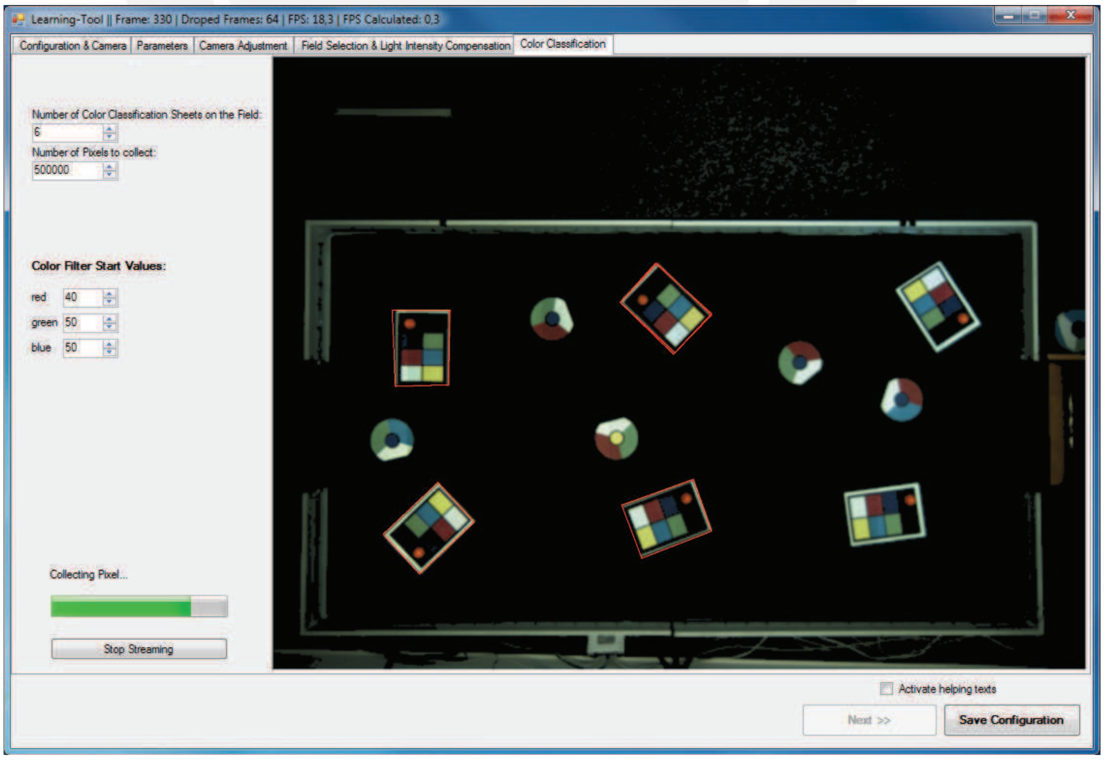


Figure 4.8: Color Classification Sheet

The next disadvantage of the DBSCAN algorithm is that it did not find all of the color groups which are used for the robot tricots and the ball. Even if the algorithm got more color clusters than color groups existing, it did not contain each color group. Instead there were same clusters which belonged to the same group while other groups were missing.



Learning-Tool || Frame: 330 | Droped Frames: 64 | FPS: 18,3 | FPS Calculated: 0,3

Number of Color Classification Sheets on the Field: 6
Number of Pixels to collect: 500000

Color Filter Start Values:
red 40
green 50
blue 50

Collecting Pixel...

Figure 4.9: Learning Tool Tab Five: Searching for Color Classification Sheets

Because the first idea did not work properly, another approach was tried. Here the pixels are collected in so called *color classification sheets*. These sheets consist of rectangles in each color and a circle where a ball should be placed. One rectangle has to be cut out so the color of the playing field can be recorded by the camera.

List<Bitmap>GetColorSheets(List<Point[]> positions, Bitmap bitmap)

```
colorSheets = new List<Bitmap>();
foreach(Point[] position in positions)
  //create bitmap with area of the color classification sheet
max = Max(colorsheet.Width, colorsheet.Height);
  bmp = new Bitmap(max, max); rect = new rectangle(startX, startY, max, max);
  using (grD  = Graphics.FromImage(bmp))
grD.DrawImage(bitmap, new Rectangle(0, 0, max, max), rect, Pixel);
  end
  points = new Vector2D[5];
  for(i = 0; i< 5; i ++)
    points[i] = new Vector2D(position[i].X – startX – bmp.Width / 2,
                                      position[i].Y – startY – bmp.Height / 2);
  end
  //find angle and rotate bitmap
bool positive = false;
 dx = position[3].X – position[0].X; dy = position[3].Y – position[0].Y;
 alpha = Math.Atan2(dx, dy); //angle in rad
 if(alpha > 0)
positive = true;
 end
 dx = position[0].X – position[1].X; dy = position[0].Y – position[1].Y;
   alpha += Math.Atan2(dx, dy) + Math.PI / 2;
   dx = position[1].X – position[2].X; dy = position[1].Y – position[2].Y;
   if (positive)
     alpha += Math.Atan2(dx, dy) + Math.PI;
   end
   else
    alpha += Math.Atan2(dx, dy) - Math.PI;
  end
  dx = position[2].X – position[3].X; dy = position[2].Y – position[3].Y;
  alpha += Math.Atan2(dx, dy)-Math.PI / 2;
  alpha /= 4; //average alpha
  bmp = RotateBitmap(bmp, alpha);
  points_ = points.Rotate(alpha);
  //cut bitmap to new size
  x = points_[0].X + bmp.Width / 2; y = points_[0].Y + bmp.Height / 2;
  width = points_[2].X – points_[0].X; height = points_[2].Y – points_[0].Y;
rect = new Rectangle(x, y, width, height); bmp2 = new Bitmap(width, height);
  using (Graphics grD = Graphics.FromImage(bmp2))
grD.DrawImage(bmp, new Rectangle(0, 0, width, height), rect, Pixel);
  end
colorSheets.Add(bmp2);
end
return colorSheets;
```

Algorithm 4.6: Rotate Color Classification Sheets

In these color classification sheets the position of the colors is known. The color classification sheets can be found using the BlobCounter algorithm. The BlobCounter returns the four edge points of the color classification sheet and diameter and position of the ball on the color classification sheet.

With this information each color classification sheet can be cut out of the camera image and rotated in horizontal position (see Algorithm 4.6). The ball is in the right lower edge of the color classification sheet after this step.

```
void CollectPixel(List<Bitmap>colorSheets, ref double colPixels, ref List<RGBPoint>
                           points)
found = new bool[256, 256, 256];
foreach(Bitmap colorSheet in colorSheets)
fac = colorSheet.Height / 297.0;  //DIN A 4 Sheet: 297 mm length
ObjectDetection(colorSheet, out circles, out points2);
  for (each color group)
startX = startPosition(color group).X; endX = endPosition(color group).X;
startY = startPosition(colorgroup).Y; endY = endPosition(color group).Y;
symbol = symbol(color group)
    for (y = startY; y <= endY; y ++)
      for (x = startX; x <= endX; x ++)
        Color pixel = colorSheet.GetPixel(x, y);
        if (found[pixel.R, pixel.G, pixel.B] == false)
found[pixel.R, pixel.G, pixel.B] = true;
points.Add(newRGBPoint(pixel.R, pixel.G, pixel.B, symbol));
        end
colPixels ++;
      end
    end
  end
end
```
Algorithm 4.7: Collect color pixels

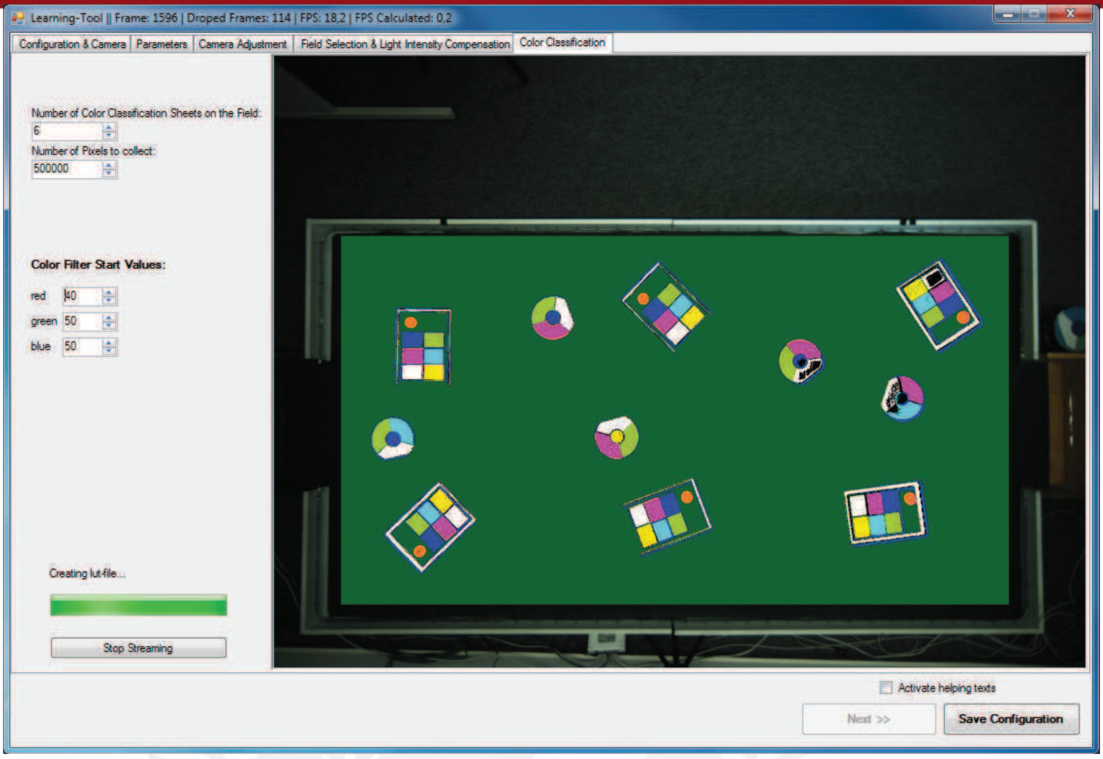
In the next step the pixels can be collected as shown in Algorithm 4.7. In contrast to the first idea the color value of each pixel is already assigned to a color group. So the DBSCAN algorithm is not needed in this case.

After collecting pixels the pixel cloud is expanded and saved as text file.

For expanding the pixel cloud Christoph Ußfeller took one color value and looked at its neighbors. The color value was assigned to the color group which most of its neighbors belonged to.

In contrary in this work a color value which already belongs to a color group is taken and each of its neighbors, which does not already belong to a color group, is assigned to this color group.

This alternative was chosen because it is easier for writing a program in C#. As the results show, it does work as well.


Learning-Tool || Frame: 1596 | Droped Frames: 114 | FPS: 18,2 | FPS Calculated: 0,2

Number of Color Classification Sheets on the Field: 6

Number of Pixels to collect: 500000

Color Filter Start Values:

red 40

green 50

blue 50

Creating lut-file...

Figure 4.10: Learning Tool Tab Five: Color Classification (The black spots on the robot tricots are color values which are not assigned to a color group.)

4.4 Interface to Robot Controlling Software

The interface to the robot controlling software consists of two files: The first one is a text file and contains the look up table for the color classification. The second one includes all necessary parameters and the light intensity distribution matrix.

4.4.1 Lut-File

The so called *lut file* is a look up table for the color groups which belong to the color values. It consists of the values of the lut matrix which are saved as text file. The name of this text file can be changed in *Tab One, Configuration and Camera*. It is saved during the button click event in *Tab Four, Light Intensity Distribution*.

Which symbol belongs to which color shows Table 4.3.

At first pixels in the color classification sheets are collected and saved as a list of points. Each point consists of a red, a green and a blue value as well as its color symbol.

Symbol	Color	R - G - B	Meaning
Y	Yellow	255 – 255 – 0	Team Marker
B	Blue	0 – 0 – 255	Team Marker
M	Magenta	255 – 0 – 255	Robot Number
C	Cyan	0 – 255 – 255	Robot Number
G	Green	149 – 255 – 0	Robot Number
W	White	255 – 255 – 255	Front of the robots
O	Orange	255 – 127 – 39	Ball
F	“Field”	0 – 100 – 0	Playing Field
	Black	0 – 0 – 0	All other pixels

Table 4.3: Color Groups

In the second step, the points are inserted in a matrix and this matrix is expanded as shown in Algorithm 4.8.

char[,,] build(List<RGBPoints>points, maxIterations)

```
lut = new char[256, 256, 256];
for (i = 0; i<points.count; i ++)
RGBPoint p = points[i];
lut[p.r, p.g, p.b] = p.symbol;
  for (all neighbours p_)
    if ( (lut(p_) == null) && (p.iterationlevel<maxIterations) )
points.Add(p_);
    end
  end
end
return lut;
```

Algorithm 4.8: Expand the lut matrix

A three dimensional matrix of char values is created. The red value of a color is encoded in the position in the first dimension. The second dimension stands for the green value and the third for the blue value. The symbol for the color group is written at the position encoded by the color values.

void CreateLut(char[,,] lut)

```
s = new Streamwriter(lutname.txt, encoding.ASCII);
for (r = 0; r < 256; r ++)
  for (r = 0; r < 256; r ++)
    for (r = 0; r < 256; r ++)
s.Write(lut[r, g, b]);
    end
  end
end
s.Close();
```

Algorithm 4.9: Create the lut file

The points in the list are registered in the lut matrix. If a point already exists, it is skipped. If it doesn’t exist, its symbol is written at its color position. Then its 26 neighbors are examined. Each neighbor which doesn’t has a color symbol yet gets the color symbol of the point and is added to the list of points.

Besides each color point possesses an iteration level. The iteration level of a new created point is by one greater than the iteration level of the original point. If the iteration level gets bigger than a maximum value, this point is skipped.

In the third step the lut matrix is saved as text file as shown in Algorithm 4.9.

4.4.2 Configuration File

The rest of the parameters are saved in INI file format. The configuration file can be created by clicking on the *Save Configuration* button or when the learning tool is closing.

On the other hand, an old configuration file can be load in *Tab one, Configuration and Camera*. In this case only the color classification is missing; the rest of the parameters already exists and can be altered.

Property	**Meaning**	**Default**	**Data Type**
Section: Camera			
Id	Id of the camera	02-2153A-06018	String
Name	Name of the camera	GC1600CH	String
GainRaw	Gain for camera image	0	Decimal
ExposureTime	Exposure Time	10000	Decimal
WBRed	White Balance: Red Content	100	Decimal
WBBlue	White Balance: Blue Content	130	Decimal
PacketSize	Data packet size	1500	Decimal
MeanDesired	Correction factor	22	Decimal
Section: Robots			
Diameter	Diameter of a robot	0.18	String
Height	Height of a robot	0.12	String
TeamMarkerDiameter	Diameter of the Team Marker	0.05	String
Tricot n	Tricot colors of the n-th robot	*e.g.Yellow,Cyan,Magenta*	String
Section: Ball			
Diameter	Diameter of a ball	0.04	String
Section: Playing Field			
Length	Length of the playing field	2.74	String
Width	Width of the playing field	1.52	String
Height	Height of the camera above the field	2.4	String
Section: PositionOfPlayingField			
StartX	Left upper point of the field, x-value	*Depends on recent selection*	Int
EndX	Right lower point of the field, x-value	*Depends on recent selection*	Int
StartY	Left upper point of the field, y-value	*Depends on recent selection*	Int
EndY	Right lower point of the field, y-value	*Depends on recent selection*	Int
Section: LightIntensityDistribution			
NumX	Parts in x direction	37	Int
NumY	Parts in y direction	20	Int
GainField	Light intensity compensa-tion matrix	*Depends on recent selection*	String
Section: ColorClassification			
LUT	Name of the lut file	1.lut	String

Section: DataOutput			
IP	IP of data output	239.255.0.1	String
Port	Port of data output	30002	String
Section: VideoOutput			
IP	IP of video output	127.0.0.1	String
Port	Port of video output	30003	String

Table 4.4: Parameters in the configuration file

Generating the configuration file starts with collecting the necessary parameters in an instance of the class *parameters*. Next, the function *WriteConfig* is called. This function creates a configuration file with the parameters listed in Table 4.4.

For loading an old configuration the functions *ReadConfig* and *LoadConfig* are used. *ReadConfig* loads the parameters from the configuration file in an instance of the class *parameters* while *LoadConfig* loads the parameters from an instance of the class *parameters* into the learning tool form and opens a connection to the camera.

5 Practical Tests

The fifth chapter summarizes practical tests. It shows if other people could use the learning tool without more explanations.

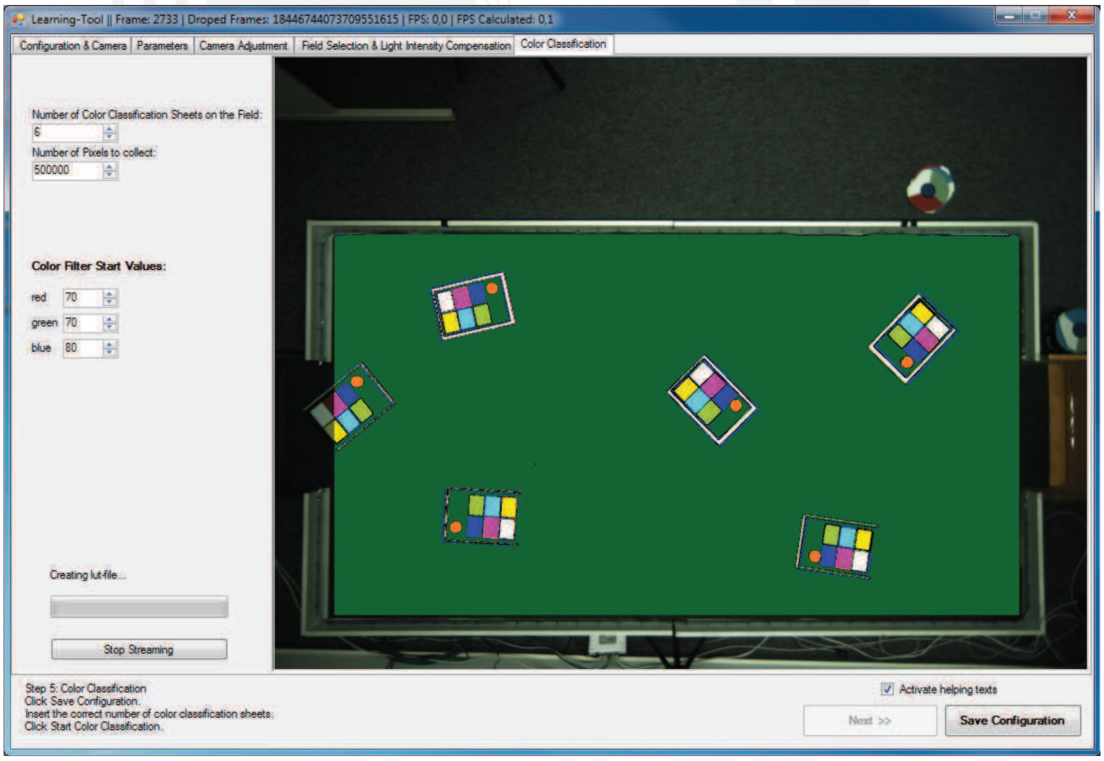
Test Person 1

My first test person was a student. He was told to generate two files with the learning tool, one being called lut file and the other one, the configuration file, should contain the parameters.

He had no problems with selecting the camera and changing to the second tab, *Parameters*. Here he recognized he could change the parameters, and assumed that the correct parameters already are preset. Therefore he didn't change them.

The camera in *Tab Three, Camera Adjustment*, already was adjusted correctly and he went on to the next tab.

In *Tab Four, Field Selection and Light Intensity Compensation*, it wasn't clear for him, that he has to empty the playing field of robots manually. He had the same problem with *Tab Five, Color Classification*. For him, the helping text did not state clearly that he has to spread color classification sheets on the playing field. Furthermore, he suggested writing explicitly that the balls have to be put on the color classification sheets.


Learning-Tool || Frame: 2733 | Droped Frames: 18446744073709551615 | FPS: 0,0 | FPS Calculated: 0,1

Number of Color Classification Sheets on the Field: 6
Number of Pixels to collect: 500000

Color Filter Start Values:
red 70
green 70
blue 80

Creating lut-file...

Step 5: Color Classification
Click Save Configuration.
Insert the correct number of color classification sheets.
Click Start Color Classification.

Figure 5.1: Result of the Color Classification by test person 1

Another point he suggested was to show a message when the light intensity compensation is finished, because this is difficult to diagnose. Besides, a hint that the lut file will be generated automatically with the light intensity compensation would be useful.

In *Tab Five, Color Classification*, he put one color classification sheet in the area of the goal. This color classification sheet wasn't detected by the learning tool, because it was not completely in the area marked as playing field. While placing the color classification sheet he didn't recognize that it would not be in the playing field. A possible solution for this could be to mark the playing field borders in the region of the goal with tape.

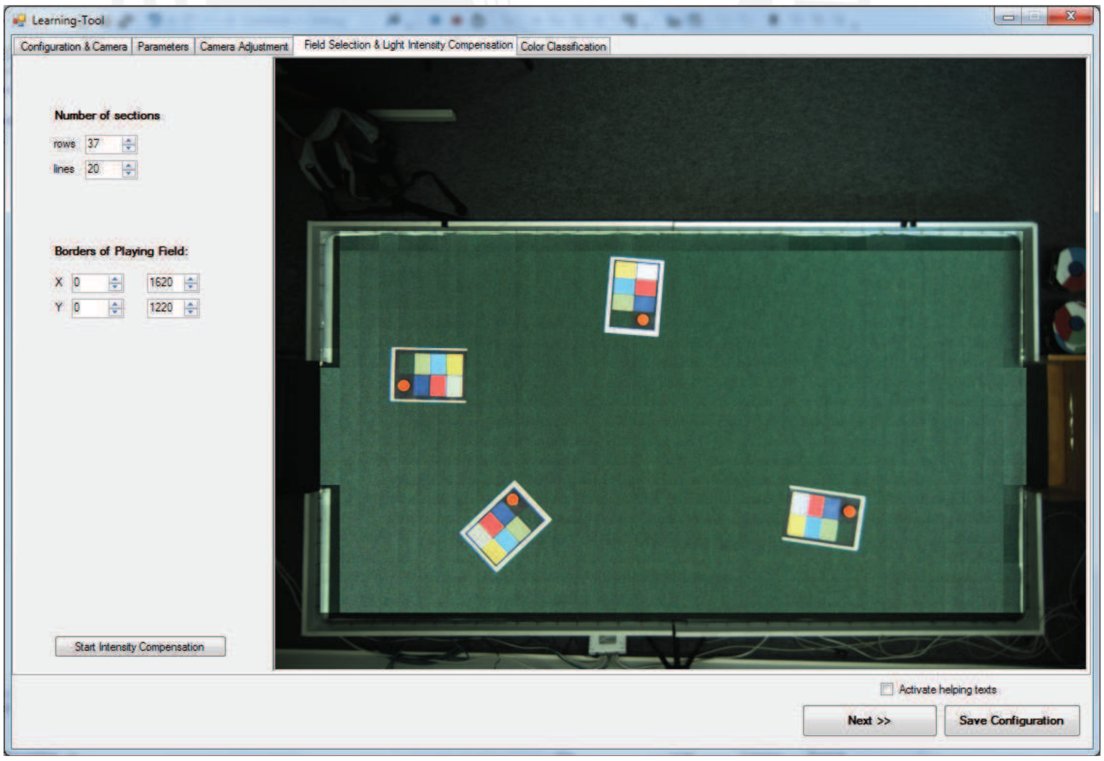
In summary, my first test person could generate the two required files without major difficulties. His result for the color classification is shown in Figure 5.1. After this first test I revised my explanatory texts.

Test Person 2

My second test person was another student. She got the same instructions as my first test person, but I had revised my explanatory texts before we met.

At first she activated the helping texts. Afterwards she followed the instructions step by step. She didn't change any default values either.

In Tab Four, Field Selection and Light Intensity Compensation, she not only marked the playing field but the borders as well (as shown in Figure 5.2).

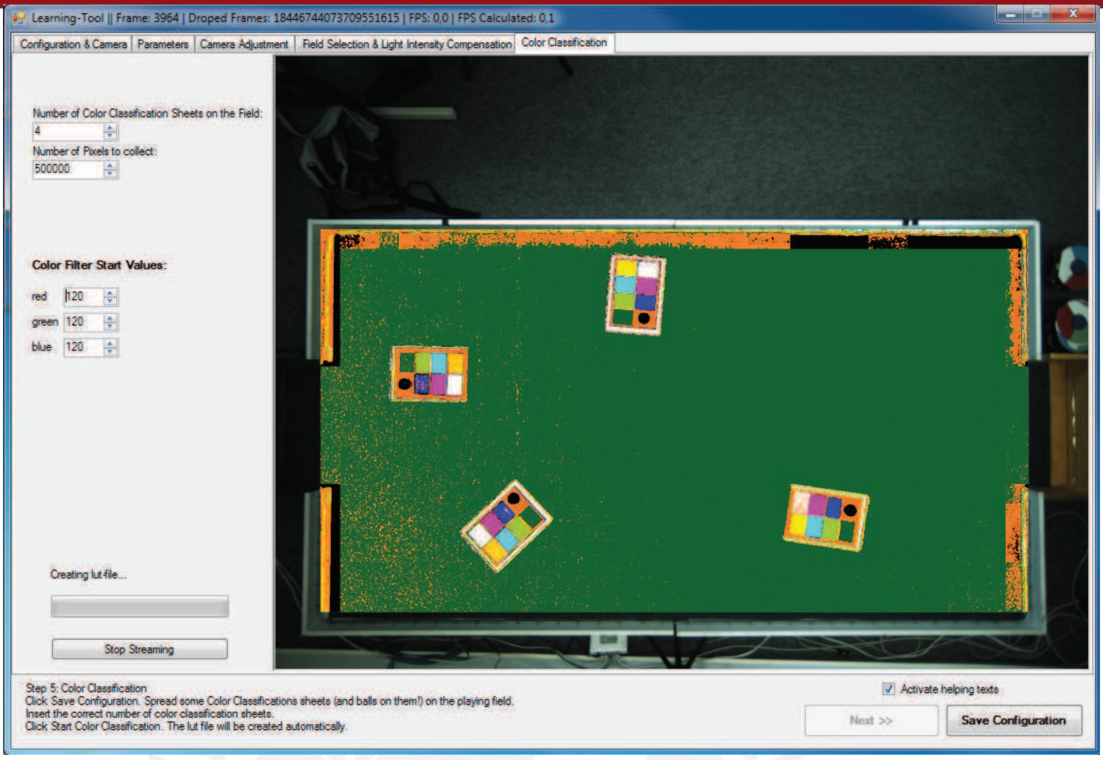


Number of sections
rows 37
lines 20

Borders of Playing Field:
X 0 1620
Y 0 1220

Figure 5.2: Tab Four, Field selection by test person 2

This led to the result shown in Figure 5.3.



Learning-Tool || Frame: 3964 | Droped Frames: 18446744073709551615 | FPS: 0,0 | FPS Calculated: 0,1

Number of Color Classification Sheets on the Field: 4
Number of Pixels to collect: 500000

Color Filter Start Values:

red 120
green 120
blue 120

Creating lut-file...

Step 5: Color Classification
Click Save Configuration. Spread some Color Classifications sheets (and balls on them!) on the playing field.
Insert the correct number of color classification sheets.
Click Start Color Classification. The lut file will be created automatically.

☑ Activate helping texts

Figure 5.3: Result of the Color Classification by test person 2

Her result was not as exact as the one by test person 1 because she marked the borders of the playing field. This led to a stronger light intensity compensation und therefore the color group *orange* was not detected correctly.

Conclusion

Both test persons could follow the instructions in the learning tool and use it to generate the configuration and the lut file at first try. After test person 1 I improved the helping texts and test person 2 had no problems with them.

So the learning tool is user independent. Further instructions are not necessary for generating the required files.

6 Conclusion

The last chapter starts with a summary of the work done and explains how the learning tool fulfills the requirements of chapter 2.2 Derivation of Task. Furthermore, it gives an overview about what else could be tried to improve the learning tool.

The requirements which the learning tool should fulfill were described in chapter 2.2 Derivation of Task. The current learning tool is introduced in chapter 4 Learning Tool. It does fulfill all of the requirements.

The learning tool is *one single program* written in C# using Visual Studio 2012. It can be transferred to other computers which are in connected to the same network as a camera, so the aspect of *mobility* is given.

As chapter 5 Practical Tests shows, it can be used by different people, who try to configure the robot controlling software for the first time, as well. The learning tool is *user-independent*.

How the *necessary parameters are collected and transmitted* to the robot controlling software is described in chapter 4.4 Interface to Robot Controlling Software. The learning tool produces two files. The first one is a text file and contains a look-up table for the classification which color value belongs to which color group.

The second one is in INI file format and consists of the parameters and the gain grid. Each parameter has a property name and belongs to a section, so the user can see where which parameter is saved and change them manually in the configuration file if necessary.

The last requirement was an *improvement of the three basic steps*. These steps are camera adjustment, light intensity compensation and color classification.

The camera adjustment is hardware based. The learning tool helps the user by showing a grid in the camera image. The lines of this grid should by parallel to the playing field borders. The grid can be moved and the distance between the lines of the grid can be altered. In the learning process by Christoph Ußfeller there was only one rectangle painted in the image, which could not be moved.

The first idea was to find the borders automatically. In chapter 4.3.1 Camera Adjustment different approaches are explained how to find the borders, but none of them worked dependable by now. In a future work different filters could be added to the Hough transform. Perhaps this could improve the detection of the borders of the playing field.

For the light intensity compensation the user has to mark the playing field in the camera image. It can be marked using the mouse arrow or by changing the values of four *numericUpDown* controls. Furthermore, the user can select, of how many parts the gain grid should consist. The light intensity compensation is started with a button. In the learning process by Christoph Ußfeller the user had to mark the borders of the playing field using the mouse arrow, but could not fine adjust the borders by using *numericUpDown* controls or something similar.

The light intensity compensation could work more automatically if the borders of the playing field are already known in *Tab Four, Camera Adjustment*.

For the color classification the learning tool uses special color classification sheets. While in the learning process by Christoph Ußfeller the user had to mark areas of a color, the color classification sheets can be found automatically and the colors are extracted using a mask of an ideal color classification sheet. In my version the user does not need to know the exact names of the colors or mark them. On single button starts the color classification.

As one can see, the learning tool does fulfill the requirements and can be used to configure the robot controlling software.

List of Figures

List of Figures

List of Algorithms

List of Tables

List of Abbreviations and Symbols

API	application programming interface
CMY	Cyan, Magenta, Yellow
DBSCAN	density-based spatial clustering of applications with noise
fps	frames per second
HSI	Hue, Saturation, Intensity
LGPL	Lesser General Public License
lut	look-up table
RGB	Red, Green, Blue
ROI	region of interest

Bibliographic References

[AVT13] Allies Vision Technologies GmbH: Technical Manual. AVT GigE Vision Cameras.Version 2.0.8.Stadtroda (Germany): 2013.

[Bäs04] Bässmann, Henning; Kreyss, Jutta: Bildverarbeitung Ad Oculos. 4. Edition. Berlin (Germany): Springer, 2004.

[Beh13] Behnke, Sven: RoboCup in Deutschland. http://www.ais.uni-bonn.de/robocup.de/. – Last Visit: 02.03.2014

[Clu09] Team Clusteranalyse: DichteverbundenesClustern. http://www.m9.ma.tum.de/material/felix-klein/clustering/Methoden/Dichteverbundenes_Clustern.php. - Last Visit: 02.03.2014

[Dun03] Dunham, Margaret H.: Data mining. Introductory and advanced topics. New Jersey (United States of America): Pearson, 2003.

[Est00] Ester, Martin; Sander, Jörg: Knowledge Discovery in Databases. Berlin (Germany): Springer, 2000.

[Gon08] Gonzalez, Rafael C.; Woods, Richard E.: Digital Image Processing. 3. Edition. New Jersey (United States of America): Pearson, 2008.

[Kir09] Kirillov, Andrew: AForge .NET Framework changes its license. http://www.aforgenet.com/news/2009.03.20.framework_license.html. - Last Visit: 02.03.2014

[Kir11] Kirillov, Andrew: AForge .NET Framework celebrates its 5 years birthday. http://www.aforgenet.com/news/2011.12.21.five_years_framework.html. - Last Visit: 02.03.2014

[Nix08] Nixon, Mark; Aguado, Alberto: Feature Extraction & Image Processing. 2. Edition. Oxford (UK): Elsevier, 2008.

[Ric09] Richter, Christiane; Teichert, Bernd: Einführung in die Digitale Bildverarbeitung. 1. Edition. Dresden (Germany): Diskurs, 2009.

[Rob14] The Robocup Federation: A Brief History of Robocup. http://www.robocup.org/about-robocup/. – Last Visit: 02.03.2014

[San99] Sander, Jörg: Generalized Density-Based Clusteringfor Spatial Data Mining. München (Germany): Herbert Utz, 1999.

[Sch10] Schale, Florian: Implementation von Kommutierungs- und Regelungsalgorithmen für elektronisch kommutierte Gleichstrommotoren auf Mikrocontrollern. Ilmenau (Germany): 2010.

[Sch95] Schmid, Reiner: Industrielle Bildverarbeitung. Vom visuellen Empfinden zur Problemlösung. Braunschweig (Germany): Vieweg, 1995.

[Seu00] Seul, Michael; O'Gorman, Lawrence; Sammon, Michael J.: Practical Algorithms for Image Analysis. Description, Examples, and Code. New York (United States of America): Cambridge, 2000.

[Tre10] Treiber, Marco: An Introduction to Object Recognition. Selected Algorithms for a Wide Variety of Applications.London (UK): Springer, 2010

[Uss13] Ußfeller, Christoph: Beiträge zur Lokalisation und zur modellbasierten Lageregelung mobiler Roboter. Ilmenau (Germany): Universitätsverlag, 2013.