Opponent Detection By Humanoid Robots For The RoboCup SPL Using Multi-scale Color Local Binary Patterns

J.I. van Wely

University of Amsterdam

Opponent Detection By Humanoid Robots For The RoboCup SPL Using Multi-scale Color Local Binary Patterns

by

J.I. van Wely

to obtain the degree of Bachelor of Science
at the University of Amsterdam,

Student number:	11289988
Project duration:	April 1, 2019 – June 28, 2019
Thesis supervisor:	dhr. dr. A. Visser
Course coordinator:	dhr. dr. S. van Splunter

Abstract

In the RoboCup Standard Platform League, fully autonomous humanoid robotic soccer players compete against each other. During a match, it is critical that the robot is capable of perceiving and comprehending its surroundings. This study is focused on the opponent detection by the robots. An opponent detector based on Multi-block Local Binary Patterns has been proven effective by a recent study from the University of Miami. However, many improvements to speed and accuracy are still achievable. Therefor, this study expanded on this by implementing six Multi-scale Color Local Binary Patterns: RGB-, nRGB, Transformed-, Opponent-, nOpponent- and Hue-LBP. All these LBP′s are trained by using functions from the OpenCV library. This study concludes that the Multi-scale Color LBP′s outperform the Multi-block LBP based on the argument that their mean average percentage is higher.

Contents

1 Introduction

In 1997 the first ever Robot Soccer World Cup was held [1]. It was the first step towards reaching a goal set by the the RoboCup Federation: the development of fully autonomous humanoid robotic soccer players that could beat the champion of the World Cup following the official FIFA rules. The RoboCup Federation's prediction was that this goal would be reached around the middle of the 21st century [2]. The RoboCup is a way of promoting AI and robotics, so it provides a common task for evaluation of various theories, algorithms, and agent architectures [8].

The competition motivates researchers to reach for breakthroughs in the wide range of technologies surrounding the RoboCup. These breakthroughs might not have a significant social impact, directly. However, they may still be considered substantial accomplishments within the field of AI and robotics. On that account, the RoboCup can be considered a landmark project. The direct social impact of a landmark project is slim, but to reach the goal set by the project, certain technical breakthroughs have to be accomplished, which could have a more noticeable social impact. The end goal set by the RoboCup federation might still be distant, but reaching smaller subgoals in the process will undoubtedly produce technologies that influence a broad range of industries [7].

RoboCup has multiple leagues in which competitions are held. This study will focus merely on the Standard Platform League (SPL). This league focuses on the development of the robots. The hardware remains the same for all teams ball[2]. Each year new goals

[1] http://www.cs.utexas.edu/ pstone/Papers/99aij/node29.html
[2] http://spl.robocup.org/wp-content/uploads/downloads/Rules2019.pdf

are set within the competition and in this way advances the progression of the software developed by the programmers. An example would be the color of the soccer ball that the game is played with. The bright orange ball, which is relatively easy to detect when played on a green surface, has been replaced by a black-and-white ball[2].

During a match, it is critical that the robot is capable of perceiving and comprehending its surroundings. Objects that are typically present in a player's environment include: the ball, teammates, opponents, the goals and field lines. In previous years, the RoboCup competition used color coding. For example, the ball was colored orange, and the goalposts yellow and blue. This way, algorithms employing visual inputs could examine these inputs, concentrating on colors to identify what was observed by the robot. However, recently the color coding has been removed from the competition. Currently, among other modifications, the ball is colored black and white in a dotted pattern and the goalposts have been painted white [2].

This elimination of color coding has made it difficult for the robots to locate their opponents and so a new challenge has emerged. A new strategy shall have to be introduced in which the robots can locate their opponents without the color coding used in the previous years. An opponent detector based on Multi-block Local Binary Patterns has been proven effective by a recent study from the University of Miami [3]. However, many improvements to speed and accuracy are still achievable. Many implementations of multiscale local binary patterns exist, and so the goal of this study is to attain a variant strategy from the specific one used in aforementioned study, though this strategy must still be within the domain of Multivariant Local Binary Patterns.

Multi-scale color LBP seems promising for reasons which will become clear in the upcoming sections of this thesis. The strategy that was used in the study from Miami is Multi-block LBP . Therefore, the goal of this study is to answer the following question: Does the performance shown by the NAO robots within the RoboCup Standard Platform League increase when using Multi-scale color LBP relative to that of using multi-block LBP?

Does the performance shown by the NAO robots within the RoboCup Standard Platform League increase when using Multi-scale Color Local Binary Patterns relative to that of using Multi-Block Local Binary Patterns?

2

Theoretical Framework

2.1. Jerseys to be Detected

This first section sets out on clarifying what the classifier is supposed to detect. In the RoboCup SPL, NAO robots play a match of soccer against each other. The players wear colored jerseys as team markers (see figure 2.1)). These jerseys are what the classifier is trained on detecting. The following rules have been set for the design of the jerseys:

- Jerseys should be tanktop style
- Jerseys must have a primary color which covers at least 70% of the jersey
- Once the match has started the jersey must be worn for the rest of the game
- The jersey number must be printed on the front and the back of the jersey. The size should be big enough so that it is easily recognizable for humans
- All players of a team must wear identical jerseys except for the number
- Jerseys should not contain distracters like big pictures of the soccer-ball printed on them
- Jersey material must be non-reflecting, non-shiny, and non-textured
- Every team should have two jersey designs, which are notably different from each other

Further information on the RoboCup SPL and the NAO robots can be found in Appendix

A.

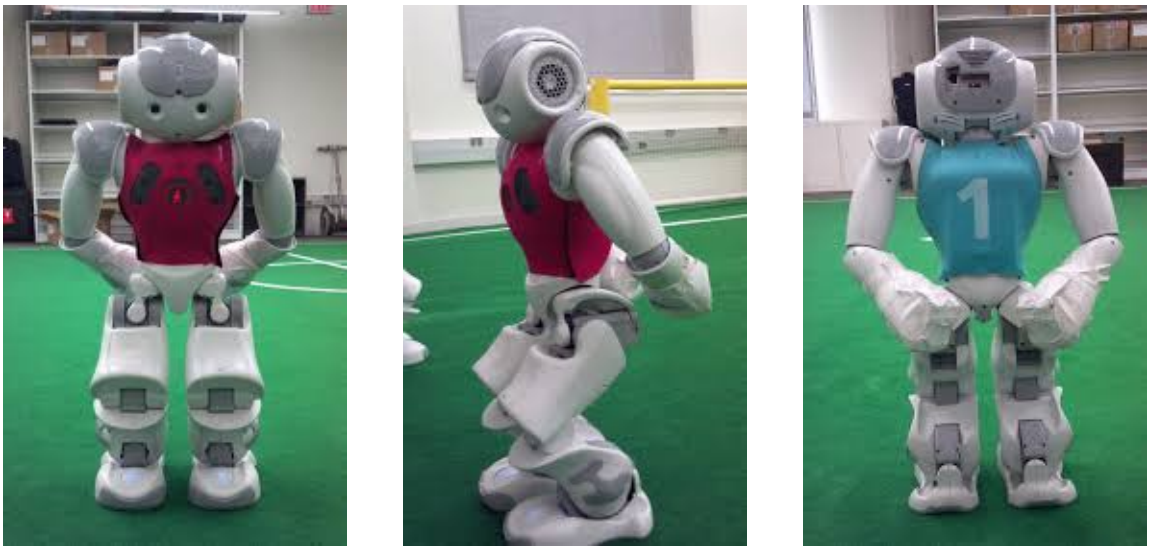

Figure 2.1: NAO robot jersey.

2.2. Haar-like Features

'Haar-like features have scalar values that represent differences in average intensities between two rectangular regions. They capture the intensity gradient at different locations, spatial frequencies, and directions by changing the position, size, shape, and arrangement of rectangular regions exhaustively according to the base resolution of the detector.' [9]. Three categories of features are edge, line, and center-surround features (see figure 2.2). Edge features look for where in the image does the intensity of the pixels shift from high intensity to low intensity. Line features are features where the pixel-intensity shift from low to high and then to low again. Or of course the other way around from high to low and then to high again.

Even after greyscale conversion of an image, it hardly occurs that an image contains areas where the pixel intensity differences are as extreme as the features displayed in figure 2.2. It does not have to be. When an area within an image is selected to be analyzed for a specific feature, all the pixel-value of what is expected to be the high intensity (white) part are summed together and averaged. The same is done for the high intensity (black) part. The high-intensity average is subtracted from the low-intensity average. The closer

this gets to one, the more likely it is a Haar-feature has been found. Because ideal haar features will usually not be found, a threshold is set. If the subtraction of the averages is greater or equal than this threshold, the feature has been found within the image[4].

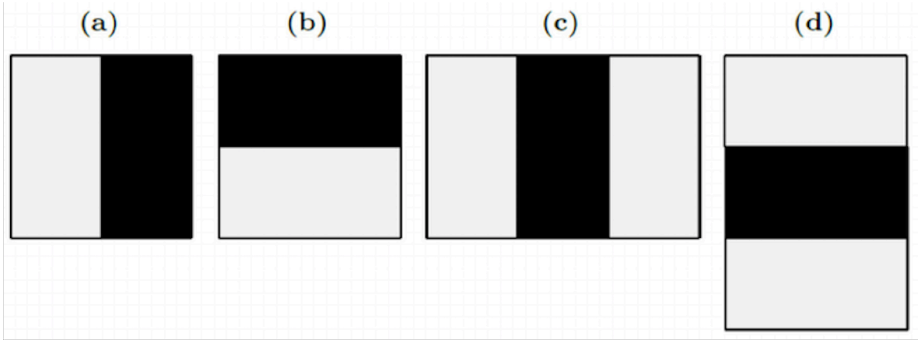

Figure 2.2: Haar-like Features.

2.3. Local Binary Patterns

Local Binary Patterns is a strategy that has been proven to be effective for opponent detection by the NAO-robots in the RoboCup Standard Platform League (SPL) [3]. In this section of the theoretical framework, the theory around LBP is explained.

Original Local Binary Patterns

The original LBP strategy was introduced by Ojala in 1996 [11]. It was first used for texture recognition, and now it is being used as one of the main strategies in face recognition [1].

It looks at each pixel and takes its surrounding neighbors. One could represent this as a 3 x 3 matrix of pixel values with the pixel that is being evaluated in the middle of this matrix. (see figure 2.3a). This central pixel-value is used as a threshold which all the surrounding pixel values will be compared to [11]. When the pixel value of the neighbor that is being compared to the threshold is lower than the threshold, the pixel is represented with a '0'. If the pixel value is higher or equal than the threshold, the pixel is represented with a '1' (See equation 2.1).

$$f(x) = \begin{cases} 1, & \text{if } x \geq 0; \\ 0, & \text{if } x < 0, \end{cases} \tag{2.1}$$

Because there are eight neighbors, we will get a string of 8 elements from the set $\{0, 1\}$. It does not matter in which order one alines the ones and the zeros as long as it is done in the same order each time. The string can also be represented as a matrix, as seen in figure 2.3 [6].

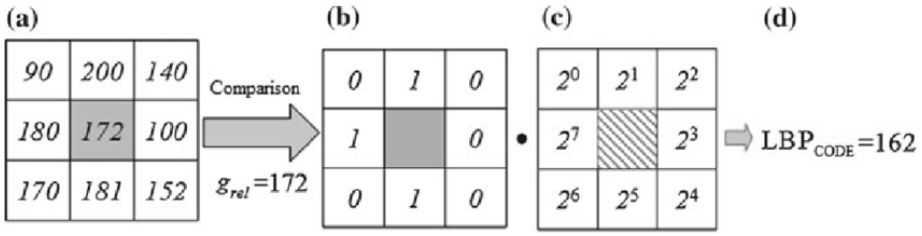

Figure 2.3: LBP-code calculation.

This binary string is converted into a decimal number, which can be done in different ways. One is shown in figure 2.3c. Where another 3 x 3 matrix containing 8-bit numbers is multiplied with the matrix existing out of the binary numbers. The center values of both matrixes can be ignored, so these are set to zero before multiplication. The multiplication results in another 3 x 3 matrix, of which all the values are summed. This leads to a decimal number, which is called the LBP-code (see figure 2.3) [6]. Another way would be by using equation 2.2 [3]. Which returns the same decimal number as the first strategy just explained. This way each set of pixels can be characterized by one out of 2^8 potential LBP-codes [6]

$$\text{LBP-code} = \sum_{n=0}^{n-1} = f(f_n - f_c)2^n \tag{2.2}$$

Where: n is the number of neighbors
f_n is the n'th neighbor pixel value
f_c is the value of the center pixel
2^n is the 8-bit number the binary number has to multiply with

The useful thing about LBP is that it is illumination invariant. When you change the lighting on the scene, all pixel-values might go up or down, but the relative difference between the pixels will remain the same irrespective of illumination variation [13]. This will apply, assuming that the change of illumination is constant over the complete set

of pixels that is evaluated. For example, if lighting only changed about a few but not all pixels we are evaluating, some pixel-values would change while others would not. Consequently, the relative difference between the pixels will change. This would have a different binary pattern as a result.

2.4. Multi-scale Local Binary Patterns

The extended LBP is very similar to the original LBP except for the way it selects its neighbors. First, a radius and number of neighbors are set. Then the neighbors are evenly selected in an angle on a circle of the radius centered around the center pixel (see figure 2.4) [12]. When different scalings are combined in one LBP, we can speak of a multi-scale LBP.

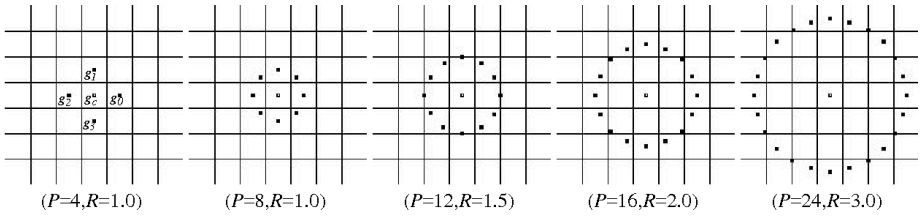

Figure 2.4: Extended LBP neighbors distribution.

If the coordinates of the central pixel are (0, 0) the coordinates of the neighbors would be (x, y) with its values for variables extracted from equations 2.3 and 2.4.

$$x = -r \cdot sin\left(\frac{2\pi n}{p}\right) \tag{2.3}$$

$$y = r \cdot cos\left(\frac{2\pi n}{p}\right) \tag{2.4}$$

Where: r Is the radius
n The n'th neighbor being evaluated at the moment
p The total number of neighbors

2.5. Multi Block Local Binary Patterns

The idea of Multi-Block Local Binary Patterns (MB-LBP) is to compare average pixel-values from pixel blocks instead of comparing pixel-values as in the basic LBP [13]. In Haar-like Features, rectangles are compared to each other. With MB-LBP the blocks of pixels, represented by their pixel-values average, that are being compared, are similar to the rectangles. With MB-LBP the center rectangle is compared to its neighboring other rectangles. It returns a binary code just as the basic LBP does (see equations 2.5 & 2.6 and figure 2.5)[16].

$$\text{MP-LBP} = \sum_{n=0}^{n-1} = s(g_n - g_c)2^n \tag{2.5}$$

$$s(x) = \begin{cases} 1, & \text{if x} \geq 0; \\ 0, & \text{if x} < 0, \end{cases} \tag{2.6}$$

Where: n is the number of neighbors
g_n is the n'th neighbor rectangle average pixel-value
g_c is the center rectangle average pixel value
2^n is the 8-bit number the binary number has to multiply with

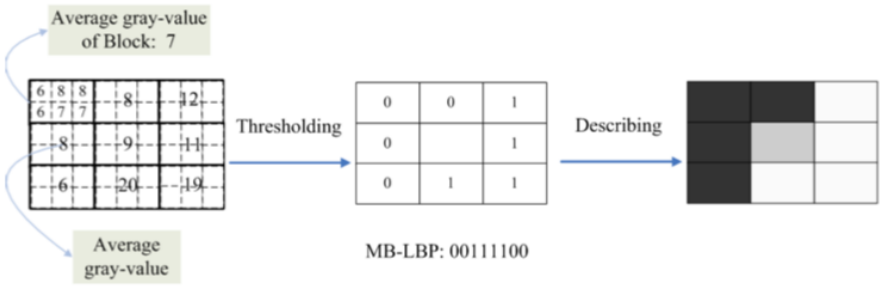

Figure 2.5: MB-LBP.

Figure 2.6 shows that MB-LBP features are more distinctive than Haar-like features and original LBP features. Furthermore, the complete set of MB-LBP features is much smaller than that of the Haar-like features. The ratio between the amount of MB-LBP and Haar-like features are approximately 1:20 [16].

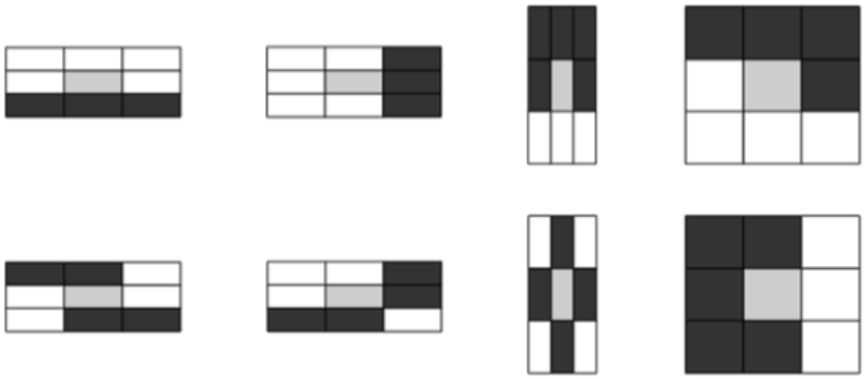

Figure 2.6: A randomly chosen subset of the MB-LBP features.

2.6. Color and LBP

LBP′s have been proven to be successful in the fields of texture classification [**?**] and face recognition [2]. However, it has not been as widely used in Visual Object Classes recognition. One of the reasons for this might be that almost all LBP variations operate in grayscale while color is of significant importance for the task of object-categorization within images. Most LBP′s tend to work in grayscale, adding color information could make recognition task more complicated. Namely, changes in the illumination make the production of a Color LBP-classifier more challenging [17]. Therefore some adjustments will have to be made to get an effective color LBP. This section will start out going over the potential illumination changes after which the Multi-scale color LBP′s used in this study are defined and illustrated.

2.6.1. Illumination Changes

Changes in illumination can be represented with the von Kries Model [15]. It is given by the following model:

$$\mathbf{f}^c = \mathcal{D}^{u,c}\mathbf{f}^u \tag{2.7}$$

Where:
$\mathbf{f}^c$ the image transformed
$\mathbf{f}^u$ the image taken under an unknown light source
u an unknown light source
c the canonical illuminant

$\mathcal{D}^{u,c}$ a diagonal matrix mapping colors taken under u to their corresponding colors under c

Equation (2.7) can also be represented as the following [14]:

$$\begin{pmatrix} R^c \\ G^c \\ B^c \end{pmatrix} = \begin{pmatrix} a & 0 & 0 \\ 0 & b & 0 \\ 0 & 0 & c \end{pmatrix} \begin{pmatrix} R^u \\ G^u \\ B^u \end{pmatrix} \tag{2.8}$$

However, shifts in the color values as a result of increased diffuse light is not covered in this model [14]. For this reason the model is extended to the diagonal-offset model represented with equation (2.9) [5].

$$\begin{pmatrix} R^c \\ G^c \\ B^c \end{pmatrix} = \begin{pmatrix} a & 0 & 0 \\ 0 & b & 0 \\ 0 & 0 & c \end{pmatrix} \begin{pmatrix} R^u \\ G^u \\ B^u \end{pmatrix} + \begin{pmatrix} o_1 \\ o_2 \\ o_3 \end{pmatrix} \tag{2.9}$$

Using the above mentioned two models the following five kinds of illumination changes can be expressed [14]:

light intensity change occurs when the image values change by a constant factors in all channels (i.e. a = b = c). It includes shadows and illumination geometry changes.

$$\begin{pmatrix} R^c \\ G^c \\ B^c \end{pmatrix} = \begin{pmatrix} a & 0 & 0 \\ 0 & a & 0 \\ 0 & 0 & a \end{pmatrix} \begin{pmatrix} R^u \\ G^u \\ B^u \end{pmatrix} \tag{2.10}$$

light intensity shift occurs when the image intensity values have an equal shift over all color channels (o1 = o2 = o3) & (a = b = c = 1).

$$\begin{pmatrix} R^c \\ G^c \\ B^c \end{pmatrix} = \begin{pmatrix} R^u \\ G^u \\ B^u \end{pmatrix} + \begin{pmatrix} o_1 \\ o_1 \\ o_1 \end{pmatrix} \tag{2.11}$$

light intensity change and shift is a combination of the models mentioned above. it occurs when both the image intensity values shift equally over all color channels and when the image values change by a constant factor in all channels.

$$\begin{pmatrix} R^c \\ G^c \\ B^c \end{pmatrix} = \begin{pmatrix} a & 0 & 0 \\ 0 & a & 0 \\ 0 & 0 & a \end{pmatrix} \begin{pmatrix} R^u \\ G^u \\ B^u \end{pmatrix} + \begin{pmatrix} o_1 \\ o_1 \\ o_1 \end{pmatrix} \tag{2.12}$$

light color change is represented in equation (2.8). It occurs when image values scale independently (i.e. allowing a ≠b ≠c).

light color change and shift is represented in equation (2.9). It occurs when image values scale independently (i.e. allowing $a \neq b \neq c$) and it models arbitrary offsets ($o_1 \neq o_2 \neq o_3$).

2.6.2. Multi-scale Color LBP

In Multi-scale Color Local Binary Patterns for Visual Object Classes Recognition [17] six Multi-scale Color LBP′s are introduced that add color information to the original LBP operator by dealing with the just above-mentioned illumination changes. These Multi-scale Color LBP′s are:

RGB-LBP: This operator is obtained by computing LBP over the red, green, and blue channels of a particular image independently. After which the results are combined.

nRGB-LBP: This operator is obtained by computing LBP over the normalized red, green, and blue channels of a particular image independently. After which the results are combined. The blue channel is redundant because the sum of the three channels will be equal to one. The nRGB-LBP is represented in equation (2.13).

$$\begin{pmatrix} r \\ g \\ b \end{pmatrix} = \begin{pmatrix} R/(R+G+B) \\ G/(R+G+B) \\ B/(R+G+B) \end{pmatrix} \tag{2.13}$$

The normalization results in the red and green channels being scale-invariant, which makes this operator invariant to light intensity changes (2.10).

Transformed color LBP: This operator is obtained by computing LBP over color channels from the transformed color space of a particular image independently.

$$\begin{pmatrix} R' \\ G' \\ B' \end{pmatrix} = \begin{pmatrix} (R-\mu_R)/\sigma_R \\ (G-\mu_G)/\sigma_G \\ (B-\mu_B)/\sigma_B \end{pmatrix} \tag{2.14}$$

Where:
μ the mean of the corresponding channel
σ the standard deviation of the corresponding channel

The subtraction and normalization make the operator scale- and shift-invariant, which

means it is invariant to light intensity change and shift (2.12). Furthermore, because al the channels are computed independently the operator is invariant to light color change and shift (2.9).

Opponent-LBP: This operator is obtained by computing LBP over all three channels of the opponent color space independently.

$$\begin{pmatrix} O_1 \\ O_2 \\ O_3' \end{pmatrix} = \begin{pmatrix} (R-G)/\sqrt{2} \\ (R+G-2B)/\sqrt{6} \\ (R+G+B)/\sqrt{3} \end{pmatrix} \tag{2.15}$$

The substraction makes the O_1 and O_2 channels invariant to light intensity shift (2.11).

nOpponent-LBP: This operator is obtained by computing LBP over two channels of the normalized opponent color space independently.

$$\begin{pmatrix} O_1' \\ O_2' \end{pmatrix} = \begin{pmatrix} \frac{O_1}{O_3} \\ \frac{O_2}{O_3} \end{pmatrix} = \begin{pmatrix} \frac{(R-G)/\sqrt{2}}{(R+G+B)/\sqrt{3}} \\ \frac{(R+G-2B)/\sqrt{6}}{(R+G+B)/\sqrt{3}} \end{pmatrix} \tag{2.16}$$

The normalization makes the O_1' and O_2' channels both shift- and scale-invariant (2.12).

Hue-LBP: This operator is obtained by computing LBP for the Hue channel of the HSV color space.

$$Hue = arctan\left(\frac{O_1}{O_2}\right) = arctan\left(\frac{\sqrt{3}(R-G)}{R+G-2B}\right) \tag{2.17}$$

The subtraction and division make the operator scale- and shift-invariant, therefore it is invariant to light intensity change and shift (2.12).

This paper [17] concluded that not only did all the Multi-scale Color LBP′s outperform the original LBP (section 2.3) but also the Multi-scale LBP (section 2.4). The best three Multi-scale Color LBP′s where the Hue-, nopponent- and opponent-LBP.

3

Method

The goal of this study is to compare the accuracy of the Multi-scale Color LBPs to each other and to that of the Multi-Block LBP, which is used as a baseline. Ideally, the measurements and processing of the results are mimicked as much as possible as is done in the Miami study. The only factor that should differ is the type of LBP that is used. Only then can be concluded that whatever the outcome may be, it is depended on the type of LBP that has been used and not some other factor which has been overlooked. In this paragraph, every step that has been taken will be illustrated.

3.1. Data

To be able to recognize opponents, a classifier has to be trained. To do so, a dataset is required. To mimic the circumstances in the Miami study [3] as much as possible, the same dataset is used. It was obtained via U. Visser who was willing to share the directory containing the datasets. The dataset includes four positive datasets, namely the red, blue, yellow, and black datasets. Each of these positive datasets containing approximately fifty images. All of these images are cropped to the targeted object, which would be the jerseys that the NAO robots are wearing. This is what the LBP should be able to detect within a picture(see figure 3.1). The negative dataset consists of 2000 images containing anything but the jerseys that are supposed to be detected (see figure 3.2). Both the positive and the negative images are taken from the upper camera from the NAO robot. They will have to be taken in the following various scenarios:

- Different lightings
- Distance range
- Different ball positions and backgrounds
- Still or moving robots
- Still or moving observer robot
- Different carpets

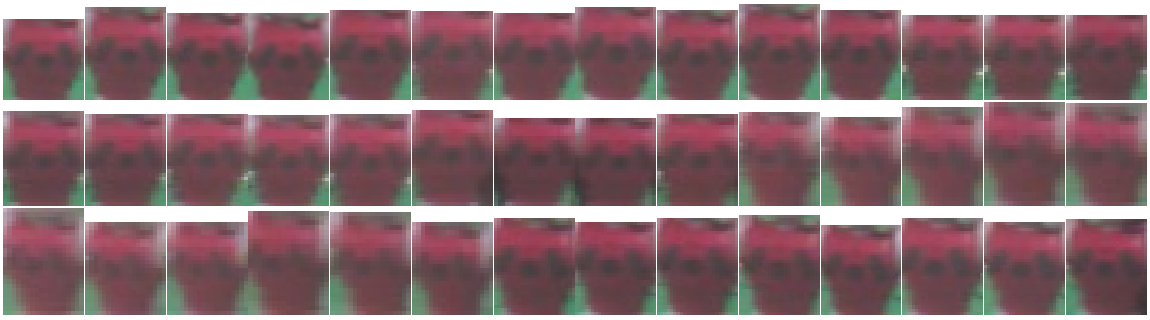

Figure 3.1: Red jersey data.

Figure 3.2: Negative data.

3.2. opencv_createsamples

For each positive dataset fifty images will not be enough. Therefore opencv_createsamples from the OpenCV library is used to generate more positive images until every positive dataset contains a thousand images. opencv_createsamples takes the positive images and then randomly rotates them, changes the intensity, and places them on a background image. These background images are obtained from the negative image-dataset.

The range of randomness is given by the user with command line arguments of the opencv_createsamples application. An example of a opencv_createsamples command used in this study is:

```
$perl_createsamples.pl positives.txt negatives.txt samples 3000 ”opencv
createsamples -bgcolor 0 -bgthresh 0 -maxxangle 1.1 -maxyangle 1.1 -
maxzangle 0.5 -maxidev 40 -w 10 -h 20”
```

- **positives.txt**: contains directories to positive images
- **negatives.txt**: contains directories to negative images
- **samples**: name of the output file containing the positive samples for training
- **num**: number of positive samples to generate
- **bgcolor**: background color controlled by bgthresh
- **bgthresh**: background color threshold
- **maxxangle**: maximal rotation angle towards x-axis, must be given in radians
- **maxyangle**: maximal rotation angle towards y-axis, must be given in radians
- **maxzangle**: maximal rotation angle towards z-axis, must be given in radians
- **maxidev**: maximal intensity deviation of pixels in foreground samples
- **w**: Width in pixels of the output samples
- **h**: Height in pixels of the output samples

The code that is used to compute positves.txt and negatives.txt are represented in Appendix B.

3.3. Further Transformation of Data

In this study, multiple Multi-scale Color LBP′s are implemented. Each one of these LBP′s requires different color channels of both the positive and negative samples. This section will expand on how these color channels are obtained. All code examples will be in C++. The complete codes can be found in Appendix C. Examples of these color channels are displayed in the results.

RGB-LBP: OpenCV has a function called `split()` that takes in an image and returns a Mat-file containing the red, green and blue color channels. An example would be:

```
vector<Mat> rgbChannels(3);
split(image, rgb_channels);
```

To obtain the blue channel one would do so by using: `rgb_channels[0]`, for green: `rgb_channels[1]` and for red: `rgb_channels[2]`.

nRGB-LBP: First, the RGB color channels are obtained by using `split()` in the same way as described in the section above. The normalized RGB channels are then computed by using equation (2.13). Where each pixels value is divided by the sum of the pixel's value over all channels. This is done by first calculating the sum of all pixel values as such:

```
cv::Mat intensity_f(rgb_channels[0] + rgb_channels[1] + rgb_channels[2]);
cv::Mat intensity;
intensity_f.convertTo(intensity, CV_8UC1);
```

Then the `divide()` function is used to divide the RGB channels by the sum of pixel values. An example of computing the normalized blue channel would be:

```
cv::Mat b_normalized_f;
cv::divide(rgb_channels[0], intensity_f, b_normalized_f);
cv::Mat b_normalized;
b_normalized_f.convertTo(b_normalized, CV_8UC1);
```

Transformed Color LBP: After `split()` is used to obtain the RGB channels. For each channel, the mean and standard deviation is calculated. They are obtained by using the function `meanStdDev();`. Both are returned as scalars so it will need to be used Like so:

```
cv::Scalar mean, stddev;
cv::meanStdDev(rgb_channels[0], mean, stddev);
float Mean = mean.val[0];;
float stddev = stddev.val[0];
```

Finaly as seen in (2.14) Each pixel is subtracted by the mean of the corresponding channel. After which it is divided by its standard deviation. An example of the code for obtaining the transformed red channel is:

```
cv::Mat trans_Blue = (rgb_channels[0] - mean[0]) / stddev[0];
```

Opponent-LBP: All opponent channels are calculated with equation (2.15), where O_1 is computed by subtracting each pixel value in the red channel by the value of the same pixel in the green channel. After that, all pixel values are divided by the square root of two.

```
cv::Mat O_1 = opponentChannels[0].at<uchar>(y, x) =
saturate_cast<uchar>(0.5f*(255 + g - r));
```

O_2 is obtained by adding each pixel value in the red channel by the value of the same pixel in the green channel and then subtracted this by the value of that pixel in the blue channel multiplied by two. After that, all pixel values are divided by the square root of 6.

```
cv::Mat O_2 = opponentChannels[1].at<uchar>(y, x) =
saturate_cast<uchar>(0.25f*(510 + r + g - 2*b));
```

Finally, O_3 is obtained by taking the sum of every pixel's value over all channels and then dividing each pixel value by the square root of three.

```
opponentChannels[2].at<uchar>(y, x) = saturate_cast<uchar>(1.f/3.f
* (r + g + b));
```

nOpponent-LBP: Firstly the opponent channels are computed in the way just described in the section above. The normalized opponent channels are then obtained by using equation (2.16) where O_1' is calculated by dividing each pixel value in the O_1 channel by the pixel's value in the O_3 channel.

```
cv::Mat nO_1;
cv::divide(O_1, O_3, nO_1);
cv::Mat b_normalized;
nO_1.convertTo(n_O_1, CV_8UC1);
```

O_2' is calculated by dividing each pixel value in the O_2 channel by the pixel's value in the O_3.

```
cv::Mat nO_2;
cv::divide(O_2, O_3, nO_2);
cv::Mat b_normalized;
nO_2.convertTo(n_O_2, CV_8UC1);
```

Hue-LBP: Is obtained by using the function `cvtColor`, which converts images from one color space to another, and again `split()`. `cvtColor` is used to convert the image

from a RGB to a hsv color space. `split()` is then used to split the converted image into three channels of which the first one is the hue channel. To do this the following code is used:

```
Mat hsv, hue;
vector<Mat> hsvChannels(3);
cvtColor(frame, hsv, CV_BGR2HSV);
split(hsv, hsvChannels);
hue = hsvChannels[0];
```

3.4. Training

The next step is the actual training of the cascades. 70% of the dataset is used for training, 15% for testing and 15% for validation. The training is executed by using the train cascades tool form the OpenCV library An example of a opencv_traincascade command used in this study is:

```
$opencv_traincascade -data Classifer -vec samples.vec -bg negatives.txt
-numStages 20 -minHitRate 0.999 -maxFalseAlarmRate 0.4 -featureType LBP -
numPos 1000 -numNeg 2000 -w 15 -h 15 -precalcValBufSize 6000 -precalcIdxBufSize
6000
```

- **data**: Where the trained classifier should be stored. This folder should be created manually beforehand
- **vec**: vec-file with positive samples (created by opencv_createsamples utility)
- **bg**: Background description file. This is the file containing the negative sample images
- **numStages**: Number of cascade stages to be trained
- **minHitRate**: The minimal desired hit rate for each stage of the classifier
- **maxFalseAlarmRate**: Maximal desired false alarm rate for each stage of the classifier
- **featureType**: Type of features. In this case LBP
- **numPos**: Number of positive samples used in training for every classifier stage
- **numNeg**: The number of negative samples used in training for every classifier stage.
- **w**: Width of training samples (in pixels)

- **h**: Height of training samples (in pixels)
- **precalcIdxBufSize**:Size of buffer for precalculated feature indices in Mb
- **precalcValBufSize**: Size of buffer for precalculated feature values in Mb

opencv_traincascade will return an xml file containing all the cascades. In section 2.6.2 and 3.3 was mentioned that some Multi-scale color LBP′s use multiple color channels on which LBP is computed. Every color channel is trained independently and thus has it's own cascades.xml file.

3.5. Classification

Before a frame or image can be classified, it will first have to be split in the correct color channels. Which color channels it has to be split into depends on the LBP that is used. If the concerning LBP computes LBP over multiple channels, the cascades.xml files are used on each of these channels of the image that has to be classified. To determine if a jersey is detected, voting is used. This means that if in the majority of the color channels, a jersey is detected, the overall LBP-classifier detects a jersey. LBP′s using either three channels or one channel will always have a majority and thus will always be able to classify an image. Two-channel LBP′s detect a jersey if either both or just one channel detects a jersey.

3.6. DetectMultiscale

Once the classifier is trained, and the required color channels are obtained, everything is set to classifier an image. This is executed by using the detectMultiscale function from the OpenCV library. An example of a detectMultiscale command is:

```
$cascade.detectMultiScale( image, objects, SF, MNN, Size(w,h), Size(W,H));
```

- **cascade**: The xml file obtained from opencv_traincascade
- **image**: The image in which is to be detected
- **object**: A vector containing the coordinates of the object (If the object is detected)
- **MNN**: Minimum nearest neighbor
- **SF**: The sample window scaling factor. SF > 1
- **Size(w, h)**: Minimum sample window. Not smaller than trained sample window
- **Size(W, H)**: Maximum sample window. Not larger than the frame size

3.7. Evaluation Criteria

The detector will be evaluated using the following five criteria:

- Precision: The precision is calculated with equation 7. The true positives (TP) are divided by the true positives (TP) + the false positives (FP). Where the true positives are the jerseys detected of which the classifier is supposed to detect, and the false positives are the jerseys or objects detected that should not have been classified as a opponents jersey.

$$\text{Precision} = \frac{TP}{TP + FP} \tag{3.1}$$

- Recall: The recall is calculated with equation 8. The true positives (TP) are divided by the true positives (TP) + the false negatives (FN). Where the false negatives are the opponent jersey that should have been detected but where not.

$$\text{Recall} = \frac{TP}{TP + FN} \tag{3.2}$$

- Detection range: The distance the object should be detected by the classifier.
- Average Detection Speed: The time the opponent detector takes to detect an opponent in an image.
- Training speed: How long the opponent detector takes to be trained.

15% if the data is used for validation, 15% for testing and the remaining 70% is used as the training set. 10-fold cross validation will be used. To make sure that the data is not misrepresented k-folds should be applied. For example, using ten folds would mean the data is split into ten equal parts of which each time one of the folds is selected to be the test data and the remaining data is used for training. This is executed ten times with each time a different fold being selected as the testing set. Finally, the average is taken over all testing results. This ensures that the data is not misrepresented by an unfortunate selection for a test set. Moreover, this way, all the data can be used for both testing and training.

3.8. Validation Training Parameters

In this section, the validation process of the parameters used in the opencv_traincascade is discussed. Tweaking these parameters will optimize the performance of the classifier. More information on these parameters can be found in section 3.4. To parameters are validated one by one. While validating a parameter, all the others are fixed. As mentioned in section 3.6, the performance is measured by recall, precision, and F1. The parameters validated are the number of stages, window size, false alarm rate, and hit rate.

With validating the number of stages parameter, one increases the number of stages starting from one. The expectation is that one will encounter underfitting at this point. Here the precision will be lower than the recall while increasing the number of stages the precision increases while the recall decreases. Eventually, the precision exceeds the recall. This is where the parameter is considered approximately optimal. After this point, increasing the number of stages will result in overfitting.

With window size, the width and height parameters are implied. It is expected that small window sizes will result in high false positives and recall because features extracted from a small window are limited. Big windows might fail to detect any jerseys.

Within the opencv_traincascade the false alarm rate (FAR) is represented as 'maxFalseAlarmRate'. And it is defined by the maximally tolerated percentage of false positives made by the classifier. A FAR of 0.5 is equivalent to a random guess. Decreasing the classifier will use more features in each stage to eliminate negatives. At a certain point, this does not improve much of the results. This would be the value the FAR is set to.

Lastly, the hit rate (HR) is represented within the opencv_traincascade as the 'minHitRate' parameter. It represents the minimum accepted percentage of true positives detected by the classifier as positives in each stage. Set the HR to high, and it will cause overfitting. Too low will result in underfitting as the HR increases the speeds of the classifier decreases.

3.9. Validation Detection Parameters

The parameters used in the detectMultiscale function should also be validated to optimize the results but also increase the speed of the detection.

Firstly the scale factor (SF) parameter. Every round it scales up the searching window by its value until it would otherwise exceed the width and height of the image. Decreasing the SF will result in the classifier using more windows to fulfill scanning an image. This produces a high detection rate; however, it does lower the precision. Furthermore, as a result of the classifier using more windows, the complete process of scanning the entire image will take longer.

A situation that can occur while scaling up the window size is that multiple labels are set on the same object. This is where the MNN (Minimum nearest neighbor) parameter proves to be useful. It groups the overlapping labels into one single averaged label. Increasing the MNN parameter will increase precision; however, the recall decreases because now, more true positives are discarded.

4

Results

In this section, the results of this study are presented. The product of splitting images into multiple color channels are laid out, and a fraction of the samples obtained from the opencv_createsamples is given.

4.1. Color Channels

All the color channels are collected from the same four images. These images are displayed in figures 4.1, 4.2, 4.3 and 4.4. These images have been chosen because together, they contain the four jerseys that have been used in this study, and thus giving a better overview of the effect of the color channels in all categories. This section will present all the color channels corresponding to their LBP′s. The code with which the color channels have been obtained can be found in Appendix C.

Figure 4.1: Original RGB image of blue jersey.

Figure 4.2: Original RGB image of red jersey.

Figure 4.3: Original RGB image of black jersey.

Figure 4.4: Original RGB image of yellow jersey.

All the color channels are displayed in grayscale except for the merged normalized RGB channels in figures 4.29, 4.30, 4.31 and 4.32.

RGB-LBP: In this section figures 4.5 to 4.16 In this section, figures 4.5 to 4.16 are representations of the RGB channels from the original images. While comparing the red channel of the yellow jersey (4.16) with the green (4.8) and blue (4.12) channels, it is noticeable that the jersey is much darker in the blue channel than in the other two. From this, one could conclude that yellow has very low pixel values within its blue channel and high values in the green and red channels. Furthermore, The red jersey within the red channel (4.6) and the blue jersey in the blue channel (4.13) have higher pixel values than they have in their other channels. This is expected because objects of a particular color will have high pixel values if expressed in a channel from that same color. Lastly, it is noticeable that the black jersey has low pixel values in every channel.

Figure 4.5: Red channel of blue jersey.

Figure 4.6: Red channel of red jersey.

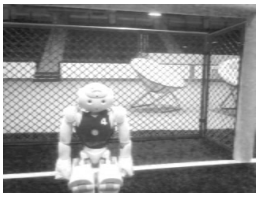

Figure 4.7: Red channel of black jersey.

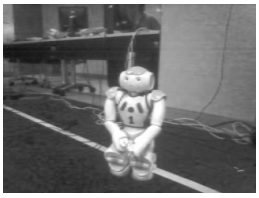

Figure 4.8: Red channel of yellow jersey.

Figure 4.9: Green channel of blue jersey.

Figure 4.10: Green channel of red jersey.

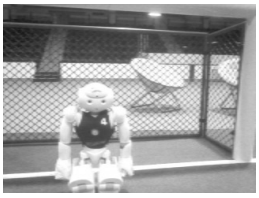

Figure 4.11: Green channel of black jersey.

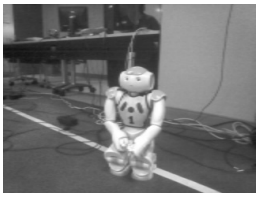

Figure 4.12: Green channel of yellow jersey.

Figure 4.13: Blue channel of blue jersey.

Figure 4.14: Blue channel of red jersey.

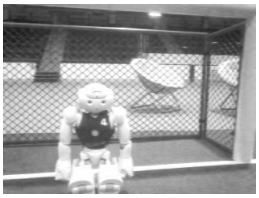

Figure 4.15: Blue channel of black jersey.

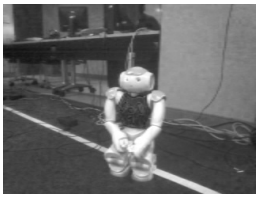

Figure 4.16: Blue channel of yellow jersey.

nRGB-LBP: Figures 4.17 to 4.28 represent the Figures 4.17 to 4.28 represent the normalized RGB channels. Firstly, the normalized green channel in figures 4.21 to 4.23 exists mainly out of big white spots. This is a consequence of the field being green and thus having high pixel values in the normalized green channel. Furthermore, within the normalized red channel (figures 4.17 to 4.20) it is very clear where the NAO is located within the image. An explanation is that it has low pixel values for all jersey colors except for the normalized red jersey. Moreover, the green field has low pixel values as well, causing the NAO robots white arms and legs, which have high pixel values, not to fade away in the background. Lastly, two points that where also noticeable within the RGB channels are the black jersey having low pixel values in all normalized channels (4.19,4.23 and 4.27) and the yellow jersey has high pixel values in the normalized red(4.20) and green(4.24) channels and very low pixel values in the normalized blue channel (4.28).

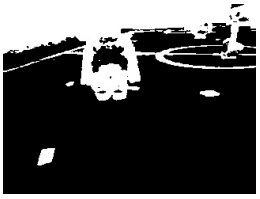

Figure 4.17: Normalized R channel of blue jersey.

Figure 4.18: Normalized R channel of red jersey.

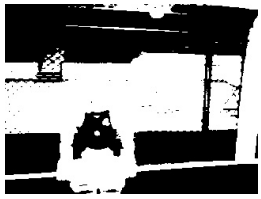

Figure 4.19: Normalized R channel of black jersey.

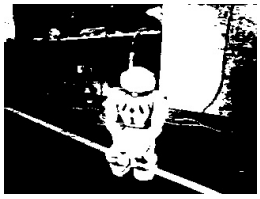

Figure 4.20: Normalized R channel of yellow jersey.

Figure 4.21: Normalized G channel of blue jersey.

Figure 4.22: Normalized G channel of red jersey.

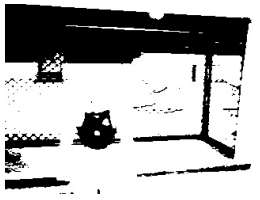

Figure 4.23: Normalized G channel of black jersey.

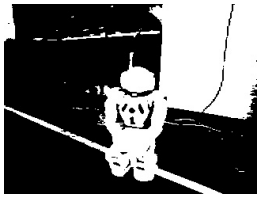

Figure 4.24: Normalized G channel of yellow jersey.

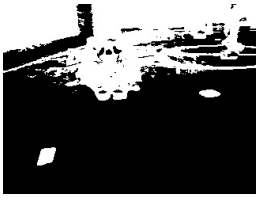

Figure 4.25: Normalized B channel of blue jersey.

Figure 4.26: Normalized B channel of red jersey.

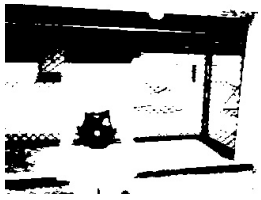

Figure 4.27: Normalized B channel of black jersey.

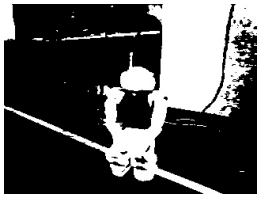

Figure 4.28: Normalized B channel of yellow jersey.

In figures 4.29, 4.30, 4.31 and 4.32 the normalized RGB channels are merged together.

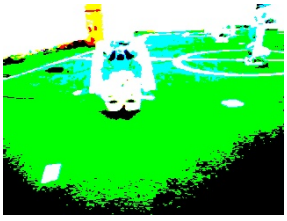

Figure 4.29: Merged normalized RGB of blue jersey.

Figure 4.30: Merged normalized RGB of red jersey.

Figure 4.31: Merged normalized RGB of black jersey.

Figure 4.32: Merged normalized RGB of yellow jersey.

Transformed-LBP: The transformed channels seem to be similar to the RGB and nRGB channels. Within the transformed green channel the field has higher pixel values compared to the other channels (4.37 to 4.40). The black jersey has low pixel values in all channels (4.35, 4.39 and 4.43). The blue jersey has high pixel values in the transformed blue channel (4.41) and low pixel values in the other two channels (4.33 and 4.37). The yellow jersey has low pixel values in the transformed blue channel (4.44), and high pixel values in the other two channels (4.36 and 4.40). Finally the red jersey seems to have slightly higher pixel values in the transformed red channel (4.34) than in the other two channels (4.38 and 4.42).

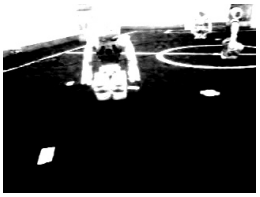

Figure 4.33: Transformed red channel of blue jersey.

Figure 4.34: Transformed red channel of red jersey.

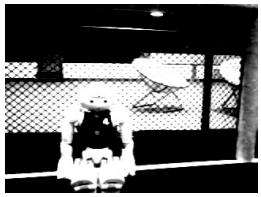

Figure 4.35: Transformed red channel of black jersey.

Figure 4.36: Transformed red channel of yellow jersey.

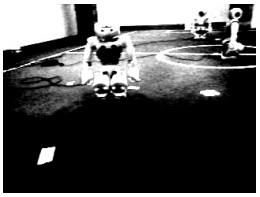

Figure 4.37: Transformed green channel of blue jersey.

Figure 4.38: Transformed green channel of red jersey.

Figure 4.39: Transformed green channel of black jersey.

Figure 4.40: Transformed green channel of yellow jersey.

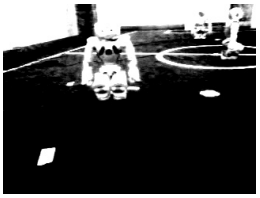

Figure 4.41: Transformed blue channel of blue jersey.

Figure 4.42: Transformed blue channel of red jersey.

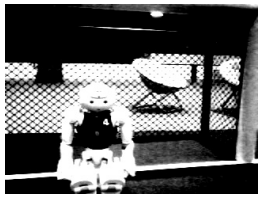

Figure 4.43: Transformed blue channel of black jersey.

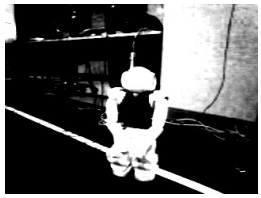

Figure 4.44: Transformed blue channel of yellow jersey.

Opponent-LBP: In figures 4.45 to 4.56 the opponent channels of the four original images are presented. One of the first things noticed is the O_3 channel (4.53 to 4.56) being a lot clearer than the other two channels. The O_3 channel is more alike the RGB and nRGB channels. Some jerseys seem to fade away in the background within the O_1 and O_2 channels (4.47, 4.48, 4.50 and 4.51). The blue jersey is a bit clearer in O_1 and O_2 (4.45 and 4.49), however still difficult to see. Finally, in figure 4.46 and especially in figure 4.52 the jerseys stand out significantly.

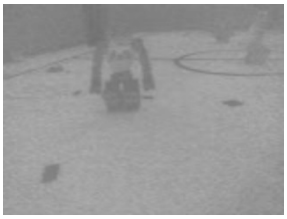

Figure 4.45: O_1 channel of blue jersey.

Figure 4.46: O_1 channel of red jersey.

Figure 4.47: O_1 channel of black jersey.

Figure 4.48: O_1 channel of yellow jersey.

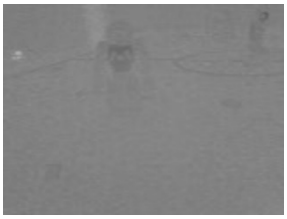

Figure 4.49: O_2 channel of blue jersey.

Figure 4.50: O_2 channel of red jersey.

Figure 4.51: O_2 channel of black jersey.

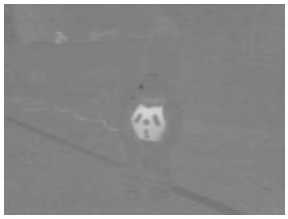

Figure 4.52: O_2 channel of yellow jersey.

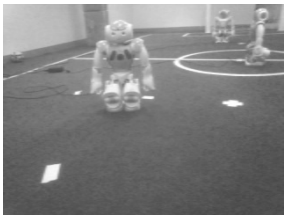

Figure 4.53: O_3 channel of blue jersey.

Figure 4.54: O_3 channel of red jersey.

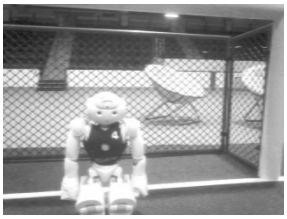

Figure 4.55: O_3 channel of black jersey.

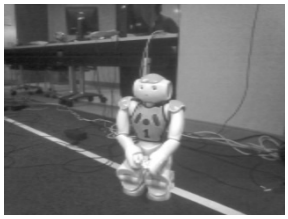

Figure 4.56: O_3 channel of yellow jersey.

nOpponent-LBP: In figures 4.57 to 4.64 the normalized opponent channels of the original images are presented. Firstly, the blue jersey is not distinguishable from its background in both the normalized channels (4.57 and 4.61). Secondly, the red and black jerseys are distinguishable from its background (4.58, 4.59, 4.62, and 4.63); however, it is not clear that one is looking at an NAO robot. Also, noticeable is the NAO's wearing yellow jerseys are relatively easy identifiable in both channels (4.60 and 4.64). Finally, when comparing representations of the jersey colors by the two normalized opponent channels, there is almost no discrepancy between them. Only the yellow jersey seems to have more high pixel values in the nO_2 channel then in nO_1 (4.60 and 4.64).

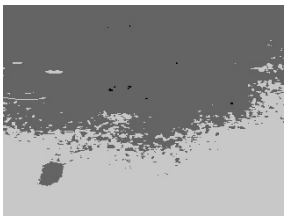

Figure 4.57: Normalized O_1 channel of blue jersey.

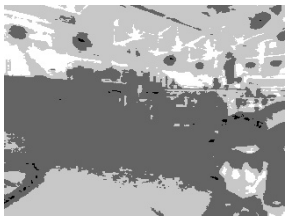

Figure 4.58: Normalized O_1 channel of red jersey.

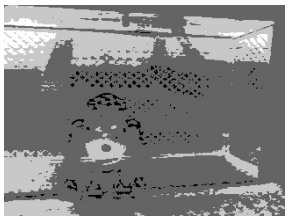

Figure 4.59: Normalized O_1 channel of black jersey.

Figure 4.60: Normalized O_1 channel of yellow jersey.

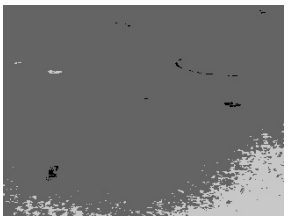

Figure 4.61: Normalized O_2 channel of blue jersey.

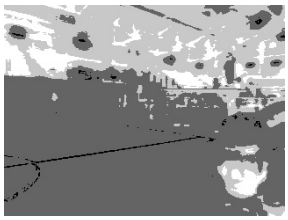

Figure 4.62: Normalized O_2 channel of red jersey.

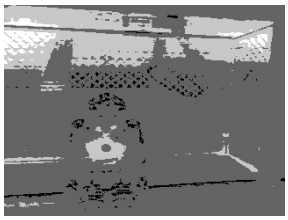

Figure 4.63: Normalized O_2 channel of black jersey.

Figure 4.64: Normalized O_2 channel of yellow jersey.

Hue-LBP: Figures 4.65 to 4.68 present the hue channels of the original images. In figures 4.65 ,4.66 and 4.68 the jersey is clearly distinguishable from the arms and legs of the NAO robot. in 4.67 it is less clear. The yellow jersey () has low pixel values, the red jersey (4.66) has high pixel values, and the blue jersey (4.65) is somewhere in between.

Figure 4.65: Hue channel of blue jersey.

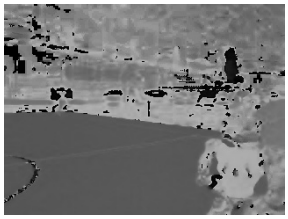

Figure 4.66: Hue channel of red jersey.

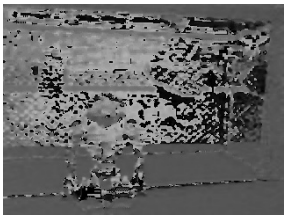

Figure 4.67: Hue channel of black jersey.

Figure 4.68: Hue channel of yellow jersey.

Finalizing this section, the statement must be made that to be able to visualize the nRGB-, Transformed-, Opponent- and nOppoennt-LBP some adjustments have been made. First of all, is the conversion to grayscale. And secondly, the pixel values had to be multiplied by a scalar, to be able to represent the relative proportions between the pixel values. The relative difference between the pixel values within a color channel where small. imwrite() would end up saving a completely white or black image. Multiplying these values by a scalar was a way of figuratively stretching out the relative difference among the pixel values and thus making it possible to visualize these relative differences.

4.2. Samples from opencv_createsamples

In figure 4.69, a fraction of the samples created by the opencv_createsamples function is displayed. As mentioned in the method section, after create_samples is given the directories to the positive and negative datasets in the form of positive.txt and negative.txt it takes an image from this negative dataset and then proceeds with placing one of the positive images over it. This produces an image containing the object that is to be detected within a background in which the object would generally be found, and thus a new positive image is produced. In figure 4.69 is displayed how these positive images are placed upon the background images. create_samples produces samples in grayscale, which is a problem when working with color LBP′s. This is further expanded upon in the discussion.

Figure 4.69: Samples produced by opencv_createsamples.

5

Discussion

This study started with the goal of following through with the complete process that is described in the method section. However, complications with the installation and running of the OpenCV library from source appeared to be more considerable obstacle than was expected. This resulted in the training of the LBP cascades not being completed within the bounds of time that were given for this study. This was, without doubt, the most significant complication the study came across. Therefore, it is brought to mind to all that wish to reproduce or expand on this study to assign this task as one of the first to be carried out. Another recommendation, within the reason of possibility, would be to operate in Linux instead of Mac OS. This study started out using Mac OS until after approximately one, and a half month later, the decision was made to start using a Linux Virtual Machine.

Another point of discussion would be the data created by create_samples. The function does generate new positive samples as it should; however, the problem is the images produced are in grayscale, which would be a problem since this study is using color LBP's and thus needs positive samples to be colored. The expectation is that this obstacle can be overcome by altering the source code in createSamples.cpp and utility.cpp. An attempt to do so has been made, however not thoroughly because the end if this study approaching, priorities had to be taken. So if one were to reproduce or expand on this study and the size of one's own dataset leads to wishing to use create_samples also, it might be beneficial to ascertain if this is indeed a possibility. Another solution, which will produce even better results, is not using create_samples at all but instead supplying all

positive images oneself [1]. This would be achieved by using the NAO robots upper camera. If time allows it, this would be the preferred approach of creating a positive dataset.

The following point of discussion is the size of the dataset. It seems that more massive datasets result in higher accuracy because they contain more variety in representations of the object that is to be detected. Therefore expanding on the amount of positive and negative images could be beneficial to the accuracy of the classifier. Moreover as mentioned in the Miami study, during the training of the cascades with more massive datasets, an acceptable degree of accuracy can be obtained with a fewer number of stages in comparison to having used a smaller dataset.

In this study, all images used are in jpg format; however, the images the classifier is trained with should be uncompressed. Formats like jpeg or jpg can cause compression noise or loss of image information. Therefore it would be best to switch to a png format.

Additional to the positive dataset discussion points are those of the negative dataset. The ideal negative dataset would contain every single object that could occur in the surroundings of an NAO robot during a RoboCup SPL match multiple times. This would reduce the number of false positives detected by the robot. The negative dataset used in this study is acceptable but could be better. It contains a few images of which it is questionable if they should be included in the dataset. Examples of these images are presented in figure 5.1. There you can see that some images contain an object that would not usually be in the surroundings of an NAO robot during a match. Moreover, the last two images contain a jersey, which is of a negative effect on the results if this is also the jersey color the classifier has to detect. Further research might compare the results of the trained cascades with and without these images included. Another expansion on this study would be to include other colored jerseys in the negative images; then the classifier is supposed to detect.

Figure 5.1: Questionable negative images

[1] https://docs.opencv.org/3.3.0/dc/d88/tutorial$_t raincascade.html$

Finally, a point of discussion for future research. Zhu et al. [17] concluded that a combination of the three best Multi-scale Color LBP_s (Opponent-, nOpponent- and Hue-LBP) had even greater accuracy than the three LBP_s independent. The expectation is that the combination of these three LBP_s is executed by voting over all the channels combined. It might be interesting to take a look at combining even more Multi-scale Color LBP_s and seeing what the results are

6

Conclusion

Based on what has been shown in the results, it seems that the RGB-, nRGB- and transformed-LBP would have better accuracy than the other LBP_s brought forward by this study. The black jersey can be detected by having low pixel values in all channels. The yellow jersey can be identified by having high pixel values in the red and green channels and low pixel values in the blue channel. The red jersey can be detected by having higher pixel values in the red channel and low pixel values in the other two. The blue channel can be identified by having low pixel values in the red channel and high pixel values in the other two channels. However, this is contradictory to the results of the study: Multi-scale color local binary patterns for visual object classes recognition, in which the conclusion is that the best three Multi-scale Color LBP_s where the Opponent-, nOpponent- and Hue-LBP. It may be that the conversion to grayscale loses some information within the color channels resulting in the wrong interpretation of the channels.

In the introduction, the research question of this paper was presented as follows:

Does the performance shown by the NAO robots within the RoboCup Standard Platform League increase when using Multi-scale Color Local Binary Patterns relative to that of using Multi-Block Local Binary Patterns?

As to answering the research question: Multi-scale Color Local Binary Patterns will outperform the Multi-block LBP used in the study of Miami[3]. Among the Multi-scale Color Local Binary Patterns, the Opponent-, nOpponnent- and Hue-LBP will have the highest accuracy. Eventhough the color channels shown in the result section might seem contradictory to this statement. However, the conclusion from Zhu et al. [17] still forms a

strong argument for this claim. As mentioned in section 2.6.2 all the Multi-scale Color LBP's brought forward by this study, outperform not only the original LBP but also the Multi-scale LBP. And indeed, amongst each other Opponent-, nOpponnent- and Hue-LBP had the best accuracy. On top of that, it seems an LBP that includes color coding would have higher accuracy in the detection of objects that are almost only distinguishable by their color relative to an LBP working in grayscale.

Appendices

A

The RoboCup, SPL and NAO

A.1. RoboCup

The RoboCup, also known as the International Robot Soccer World Cup, is an international competition. It was founded in 1997 as an attempt to promote AI and to advance the development in robotics and AI using soccer as a means of motivation [10]. There are multiple RoboCup soccer leagues:

- Humanoid
- Middle Size
- Simulation
- Standard Platfrom
- Small Size

This study focusses exclusively on the Standard Platform League (SPL), which will be expanded upon in the preceding section on the next page.

A.2. Standard Platform League

The RoboCup SPL is a league, in which all teams compete with identical robots. There is no way to control the robots externally. With other words, they need to act entirely autonomously [1]. The current standard platform used is the humanoid NAO manufactured by SoftBank Robotics, of which further information will be given in the next paragraph.

[1] https://spl.robocup.org

The soccer-field that is used is of length 10.4 m, and its width is 7.4 m. The surface of the field is as flat as possible (see figure A.1).

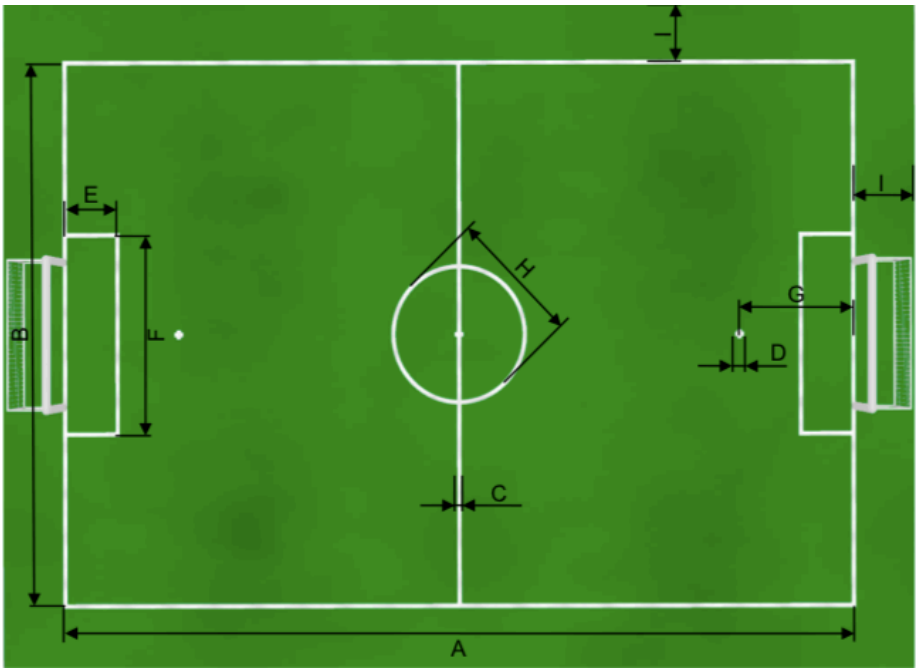

Figure A.1: SPL field.

Where:
A Field length = 9000 mm
B Field width = 6000 mm
C Line width = 50 mm
D Penalty cross size = 100 mm
E Penalty area length = 600 mm
F Penalty area width = 2200
G Penalty cross distance = 1300
H Center circle diameter = 1500
I Border strip width = 700

As far as coloring, a few things are certain: The field is green; The lines on the field are white; The posts and top crossbar of the goals are white; The net and the support structure for the net are either white, gray, or black; The ball has a black and white soccer-ball print[2].

The lighting conditions differ per location. The league moves towards using natural lighting. So if the possibility is there the field will be placed near windows. Otherwise, ceiling lighting is used to guarantee most of the field will not be darker than 300 Lux. The lighting intensity does not necessarily have to be even across the whole field. The lighting ratio between the darkest and brightest area is aimed at ten to one. Lighting irregularities occurring during a match will not be a reason for stopping the game. A few examples of lighting irregularities occurring would be the sun streaming through a window are lightbulbs turning off[2].

[2] http://spl.robocup.org/wp-content/uploads/downloads/Rules2019.pdf

A.3. NAO

The match is played by NAO robots manufactured by Soft-Bank Robotics (see figure A.2)). Here you see that NAO has two cameras, one bottom, and one top camera. This study will focus solely on the view of the top camera [3].

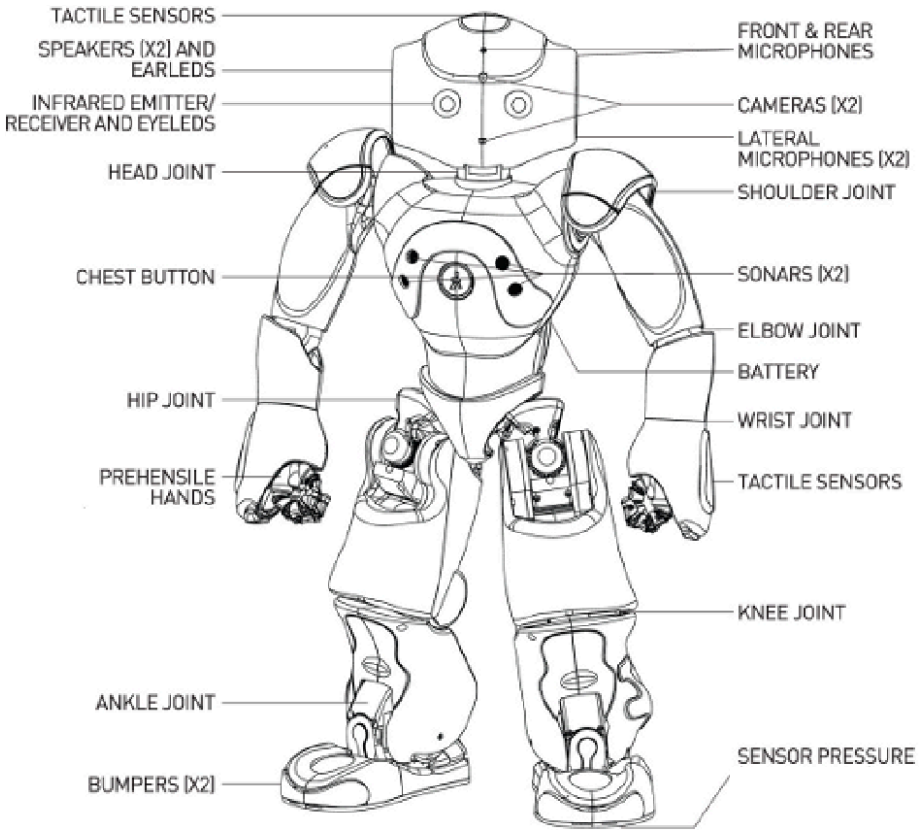

Figure A.2: Hardware of NAO.

Two teams play a match. Each team exists out of 5 players. One of these is de keeper, which always has jersey number '1'. The other players are called field players and will always have a jersey number from the set 2, 3, 4, 5, 6. Number 6 is a substitute player which could potentially enter the game later during the match [2].

B

positves.txt & negatives.txt

This code is used to obtain the txt files containing the directories to the positive and negative images that are used in the creating of samples with the opencv_createsamples application. The following code is in python:

```
import cv2
import glob

f = open(”positives.txt”,”w+”)
for file in glob.glob(’positive_images/*jpg’):
   f.write(”%s” % file)

f = open(”negatives.txt”,”w+”)
for file in glob.glob(’negative_images/*jpg’):
   f.write(”%s” % file)
```

C

Transformed Data Code

BGR-LBP:

```
static void BGR_LBP( const Mat image, vector<Mat> BGRChannels )
{
   split(image, BGRChannels);
}
```

nBGR-LBP:

```
// This code was partly obtained from: https://stackoverflow.com/
// questions/24417082/normalizing-colorchannels-of-and-image-by-
// -intensity-values-opencv

static void nBGR_LBP( const Mat image, vector<Mat> nBGRChannels )
{
   vector<Mat> BGR_channels(3);
   split(image, BGR_channels);

   cv::Mat intensity_f(BGR_channels[0] + BGR_channels[1] +
   BGR_channels[2]);
   cv::Mat intensity;
   intensity_f.convertTo(intensity, CV_8UC1);

   cv::Mat b_normalized_f;
   cv::divide(BGR_channels[0], intensity_f, b_normalized_f);
   cv::Mat b_normalized;
   b_normalized_f.convertTo(b_normalized, CV_8UC1);
   cv::Mat b_norm;
   nBGRChannels[0] = b_normalized * 255.0;

   cv::Mat g_normalized_f;
   cv::divide(BGR_channels[1], intensity_f, g_normalized_f);
   cv::Mat g_normalized;
   g_normalized_f.convertTo(g_normalized, CV_8UC1);
   cv::Mat g_norm;
   nBGRChannels[1] = g_normalized * 255.0;

   cv::Mat r_normalized_f;
   cv::divide(BGR_channels[2], intensity_f, r_normalized_f);
   cv::Mat r_normalized;
   r_normalized_f.convertTo(r_normalized, CV_8UC1);
   cv::Mat r_norm;
   nBGRChannels[2] = r_normalized * 255.0;
}
```

Transformed Color LBP:

```
static void Trans_LBP( const Mat image, vector<Mat> TransChannels )
{
   vector<Mat> rgb_channels(3);
   split(image, rgb_channels);

   cv::Scalar Redmean, Redstddev, Greenmean, Greenstddev,
   Bluemean, Bluestddev;
   cv::meanStdDev(rgb_channels[0], Bluemean, Bluestddev);
   cv::meanStdDev(rgb_channels[1], Greenmean, Greenstddev);
   cv::meanStdDev(rgb_channels[2], Redmean, Redstddev);

   float rMean = Redmean.val[0];
   float rStddev = Redstddev.val[0];
   float gMean = Greenmean.val[0];
   float gStddev = Greenstddev.val[0];
   float bMean = Bluemean.val[0];
   float bStddev = Bluestddev.val[0];

   TransChannels[0] = (rgb_channels[0] - rMean) / rStddev * 255;
   TransChannels[1] = (rgb_channels[1] - gMean) / gStddev * 255;
   TransChannels[2] = (rgb_channels[2] - bMean) / bStddev * 255;
}
```

Opponent-LBP:

```
// This code was partly obtained from: https://github.com/opencv
// /opencv/blob/2.4/modules/features2d/src/descriptors.cppL126

static void Opponent_LBP( const Mat image, vector<Mat> opponentChan-
nels )
{
   if( image.type() != CV_8UC3 )
        CV_Error( CV_StsBadArg, ïnput image must be an BGR image of
        type CV_8UC3)¨;

   opponentChannels.resize( 3 );
   opponentChannels[0] = cv::Mat(image.size(), CV/8UC1);
   opponentChannels[1] = cv::Mat(image.size(), CV/8UC1);
   opponentChannels[2] = cv::Mat(image.size(), CV/8UC1);

   for(int y = 0; y < image.rows; ++y)
       for(int x = 0; x < image.cols; ++x)
       {
           Vec3b v = image.at<Vec3b>(y, x);
           uchar b = v[0];
           uchar g = v[1];
           uchar r = v[2];

       opponentChannels[0].at<uchar>(y, x) = saturate_cast<uchar>(0.5f
           *(255 + g - r));
       opponentChannels[1].at<uchar>(y, x) = saturate_cast<uchar>(0.25f
           *(510 + r + g - 2*b));
       opponentChannels[2].at<uchar>(y, x) = saturate_cast<uchar>(1.f/3.f
           *(r + g + b));
       }
}
```

nOpponent-LBP:

```
static void nOpponent_LBP( const Mat image, vector<Mat> nopponentChan-
nels )
{
   vector<Mat> opponentChannels(3);
   Opponent_LBP(image, opponentChannels);

   cv::Mat intensity_f(opponentChannels[2] + opponentChannels[1] +
   opponentChannels[0]);

   cv::Mat O1_normalized;
   cv::divide(opponentChannels[0], intensity_f, O1_normalized);
   cv::Mat O1_norm;
   O1_normalized.convertTo(O1_norm, CV_8UC1, 255.0);

   cv::Mat O2_normalized;
   cv::divide(opponentChannels[1], intensity/f, O2_normalized);
   cv::Mat O2_norm;
   O1_normalized.convertTo(O2_norm, CV_8UC1, 255.0);

   nopponentChannels[0] = O1_norm;
   nopponentChannels[1] = O2_norm;
}
```

Hue-LBP:

```
static void Hue_LBP( const Mat image, Mat hueChannel )
{
   Mat hsv;
   vector<Mat> hsvChannels(3);
   cvtColor(image, hsv, CV_BGR2HSV);
   split(hsv, hsvChannels);
   hueChannel = hsvChannels[0];
}
```

Bibliography

[1] Timo Ahonen, Abdenour Hadid, and Matti Pietikäinen. Face recognition with local binary patterns. In *European conference on computer vision*, pages 469–481. Springer, 2004.

[2] Timo Ahonen, Abdenour Hadid, and Matti Pietikainen. Face description with local binary patterns: Application to face recognition. *IEEE Transactions on Pattern Analysis & Machine Intelligence*, (12):2037–2041, 2006.

[3] Ibrahim Abdullah Alsayyari. *Real-Time Object Detection On Humanoid Robots For The RoboCup Soccer SPL Using Cascaded Classifiers*. PhD thesis, University of Miami, 2017.

[4] Qing Chen, Nicolas D Georganas, and Emil M Petriu. Real-time vision-based hand gesture recognition using haar-like features. In *2007 IEEE instrumentation & measurement technology conference IMTC 2007*, pages 1–6. IEEE, 2007.

[5] Graham D Finlayson, Steven D Hordley, and Ruixia Xu. Convex programming colour constancy with a diagonal-offset model. In *IEEE International Conference on Image Processing 2005*, volume 3, pages III–948. IEEE, 2005.

[6] Stamos Katsigiannis. *Fuzzy Local Binary Patterns*, pages 149–175. 01 2014. ISBN 978-3-642-39288-7. doi: 10.1007/978-3-642-39289-4_7.

[7] Hiroaki Kitano, Minoru Asada, Yasuo Kuniyoshi, Itsuki Noda, Eiichi Osawa, and Hitoshi Matsubara. Robocup: A challenge problem for ai. *AI magazine*, 18(1):73–73, 1997.

[8] Hiroaki Kitano, Milind Tambe, Peter Stone, Manuela Veloso, Silvia Coradeschi, Eiichi Osawa, Hitoshi Matsubara, Itsuki Noda, and Minoru Asada. The robocup synthetic agent challenge 97. In *Robot Soccer World Cup*, pages 62–73. Springer, 1997.

[9] Takeshi Mita, Toshimitsu Kaneko, and Osamu Hori. Joint haar-like features for face detection. In *Tenth IEEE International Conference on Computer Vision (ICCV'05) Volume 1*, volume 2, pages 1619–1626. IEEE, 2005.

[10] Sanmit Narvekar, Ruohan Zhang, and Peter Stone. Fast and precise black and white ball detection for robocup soccer. *RoboCup 2017: Robot World Cup XXI*, 11175:45, 2018.

[11] Timo Ojala, Matti Pietikäinen, and David Harwood. A comparative study of texture measures with classification based on featured distributions. *Pattern recognition*, 29 (1):51–59, 1996.

[12] Timo Ojala, Matti Pietikäinen, and Topi Mäenpää. Multiresolution gray-scale and rotation invariant texture classification with local binary patterns. *IEEE Transactions on Pattern Analysis & Machine Intelligence*, 24:971–987, 2002.

[13] Matti Pietikäinen, Abdenour Hadid, Guoying Zhao, and Timo Ahonen. *Computer vision using local binary patterns*, volume 40, pages 4, 36. Springer Science & Business Media, 2011.

[14] Koen Van De Sande, Theo Gevers, and Cees Snoek. Evaluating color descriptors for object and scene recognition. *IEEE transactions on pattern analysis and machine intelligence*, 32(9):1582–1596, 2009.

[15] Johannes von Kries. Influence of adaptation on the effects produced by luminous stimuli. In *MacAdam, D.L. (Ed.), Sources of Color Vision*. MIT Press, Cambridge, 1970.

[16] Lun Zhang, Rufeng Chu, Shiming Xiang, Shengcai Liao, and Stan Z Li. Face detection based on multi-block lbp representation. In *International Conference on Biometrics*, pages 11–18. Springer, 2007.

[17] Chao Zhu, Charles-Edmond Bichot, and Liming Chen. Multi-scale color local binary patterns for visual object classes recognition. In *2010 20th International Conference on Pattern Recognition*, pages 3065–3068. IEEE, 2010.

Acknowledgement

I would like to expess my sincerest gratitude to my thesis supervisor Mr. Dr. A. Visser for supporting me during my thesis and always promptly ready to help me out, which was very much appreciated. Also, my special thanks to Mr. Dr. S. van Splunter, my course coordinator, providing me with tips on how to schedule my task, and for always staying positive. This was appreciated a great deal. Finally, I would like to thank Mr. Dr. U. Visser for being so kind with providing me with the dataset used in my thesis.