Improving Visual Processing in RoboCup SSL by Learning Color Segmentation with Neural Networks

Ernesto Torres, Alfredo Weitzenfeld

Robotics Laboratory
ITAM
San Angel Tizapán, México DF, CP 0100
etorresvid@itam.mx, alfredo@itam.mx
http://robotica.itam.mx

Abstract. Optimizing processing in the vision system is crucial for real-time performance of robots in RoboCup's Small-Size League (SSL). We describe in this paper our current approach to improve visual processing in ITAM's Eagle Knights SSL team. We describe our use of a neural network to classify camera image pixels to a discrete set of color classes that is robust under different light conditions. We show how we can improve the recall time of the neural network to achieve vision processing of over 30 fps using high resolution images. We present our solution and compare to previous methods showing improvements in real time image segmentation and varying light conditions.

1 Introduction

Many vision-based robotic systems require object identification through color as in the case of RoboCup Small-Size League (SSL). In such domain, object identification requires classifying each pixel in an image from a discrete set of color classes. To achieve this classification a calibration step is initially done based most often on a constant threshold model involving six threshold values, two for each dimension in the color space (RGB, YUV, etc.). These values define a small cube used to group the different pixels into values taken from a discrete set of color classes. The values for these thresholds that are mostly based in the RGB color space may be set in different ways, including manually [1], using decision trees [2], trained by neural networks [3], and also by dynamically adjusting the initial color calibration using automatic color calibration [4, 5, 6]. Other approaches, including hierarchical models, can be found in [7, 8, 9, 10]. As opposed to most of the basic image segmentation methods intended to specify color thresholds during calibration or dynamically adjusting them throughout the game, our approach is based on training a neural network as a *pixel classifier* to be applied throughout the game.

A number of teams also calibrate images using the HSV color space [11, 12] where instead of manually defining constant thresholds in the RGB color space they can take advantage of GUI tools simplifying color classification and then convert this representation to the RGB color space of the cameras using a standard conversion algorithm. Manual calibration is simpler than in RGB since HSV separates color hue

from light intensity and color saturation. Yet, once color space values are set, it becomes hard to adapt to variations in lighting conditions in time or even throughout the field. Again, the main advantage of our approach is that we adapt to changes in light conditions in time rather than applying constant color classes thresholds set during calibration.

In general, using manual calibration involves capturing images from different regions of the field and defining the color boundaries between color classes. For this reason it does not offer the tolerance required to adapt the vision system to different and varying lighting conditions such as shadows on the field especially when they are not taken into consideration during the calibration process. The objective of this paper is to describe our current work in developing a robust and fast color segmentation method tolerant to light changes and resistant to noise from adjacent color regions in the context of Robocup SSL. Having already developed various color vision models for our robots similar to those previously mentioned, we decided to use a backpropagation algorithm to train a multilayer neural network to improve our color segmentation process.

In the following sections we describe our color calibration, neural network architecture, describe experiments performed under varying light conditions and then compare our results to other approaches.

2 Color Calibration

We use an RGB color calibration based to classify five different colors classes: orange, blue, yellow, green and white, corresponding to the ball or to a robot color patch having official size according to SSL rules. We perform an initial manual calibration step to set pixels according to the five color classes of interest. The purpose of this initial calibration is to have a reference set of color classes that can be used in our case to train the neural network as will be explained in the next section. Final pixel color assignment will be determined in real-time during the neural network recall stage.

While different color calibration methods may be used, a relatively simple and accurate way is to do a manual calibration where pixel values are captured from a set of color classes we are interested in identifying. For each region of interest in the image we click with the mouse on a particular pixel to generate a small cube of approximately 8 pixels around the original one. All the RGB values that are part of this small cube are included into the group of pixels that will be used to define a color class and therefore each one of these values (formed by the combination of each dimension: Red, Green and Blue) will be used as a sample. Any pixel in the cube that is not included into any of the five color classes is discarded. The complete process is described in Figure 1.

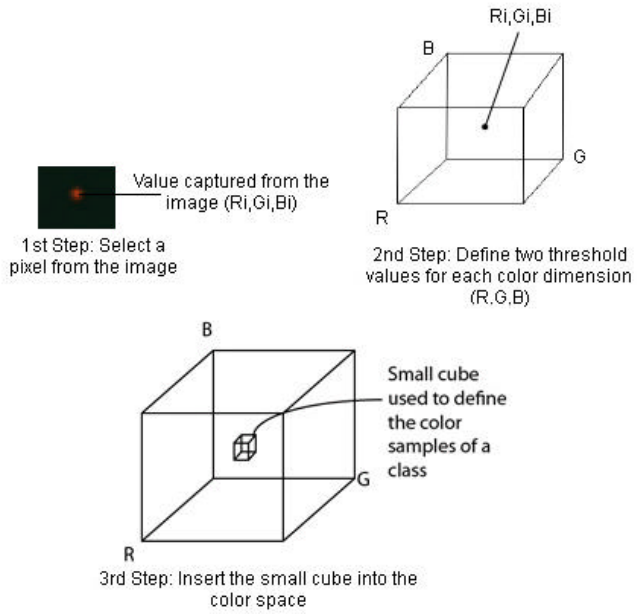

Figure 1. The diagram describes our manual calibration process to assign pixel values from a set of color classes defined in the color cube.

3 Neural Network Architecture

We use for pixel classification a multi-layer neural network architecture trained using a back propagation algorithm [13]. The input layer consists of three neurons each corresponding to a different color dimension in RGB, i.e. Red, Green, and Blue; while the output layer consists of five neurons each corresponding to one of five colors of interest, i.e. Orange, Blue, Yellow, Green, and White, as shown in Figure 2.

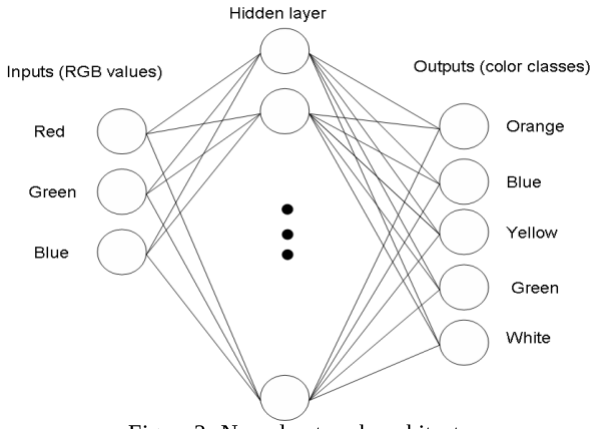

Figure 2: Neural network architecture

Training samples are defined using 1 byte (256 bits) for inputs and floating point values between "0" and "1" for outputs as described in Figure 3.

Inputs			Outputs				
Red	Green	Blue	Class1	Class2	Class3	Class4	Class5
[1-255]	[1-255]	[1-255]	[0,1]	[0,1]	[0,1]	[0,1]	[0,1]

Figure 3: Input and output data format used for classification of one RGB pixel value into a one of five color classes.

Samples are generated by initially capturing and segmenting one image from the camera according to the calibration process defined in the previous section. The sample list contains only values of pixels that are segmented into one of the five color classes. We eliminate any repeated values to reduce the number of samples since they do not provide any additional information and will only slow down the neural network training process. For initial classification we place several color patches across the field to provide as many different color values as possible on different image regions.

The number of neurons chosen for the hidden layer shown in Figure 2 depends on the number of input and output neurons and also on the number of samples used for training. We use the heuristic described in equation 1 to calculate the number of hidden neurons based on the total number of samples (S), the number of input neurons (I) and the total output neurons (O).

$$H=(S/3-O)/(I+O+1) \quad (1)$$

Note that the size of the hidden layer may vary depending on the particular number of samples used during training. The number of neurons in the hidden layer in addition to the learning rate in the back-propagation algorithm may be later adjusted minimize test errors.

The back-propagation algorithm uses a linear transfer function for the input layer and a sigmoid transfer function for the hidden and output layers. After completing the training process we obtain the resulting connection weights for the recall process. For each RGB value used as input to the neural network we will get a decimal output value between zero and one for the corresponding possible color class. Note that we may get output values for different colors at the same time. The greatest decimal value in the output is used to decide into which color class the input is classified.

One key challenge in using a neural network to segment a complete image is that the recall process can greatly affect real time performance. Processing time will depend directly on the size of the neural network and most importantly on the number of connections in the network. Considering a training set of approximately 225 samples this will result in a hidden layer of 10 neurons according to the heuristic described in equation 1. Thus such neural network will include 3 input neurons, 10 hidden neurons and 5 output neurons. The number of connections (C) in the neural network may be computed by applying equation (2).

$$C = (I+1)(H) + (H+1)(O) \quad (2)$$

For a neural network having 3 inputs, 5 outputs and 10 neurons in the hidden layer, there will be a total of 95 connections. To classify each pixel, 95 multiplications will be needed (output from one neuron is multiplied by a connection weight to obtain the partial input to the neuron). Additionally, 95 additions are required to compute the summed input for each neuron in the following layer. In order to achieve 30 fps using a 640x480 resolution image 9,216,000 recalls per second will be required, each consisting of 95 multiplications and 95 additions. This approach is rather inefficient because we would need to do a very large number of mathematical operations per second.

We considered a number of alternatives to reduce the neural network recall procedure. One alternative we considered was to subtract the background image [14] in order to avoid processing the full set of pixels in the image. Yet we still needed to process the remaining set of pixel through the neural network.

An alternative to avoid having to process the neural network for each pixel during real time image segmentation is to initially classify all colors in the RGB color space into one of the five color classes of interest, i.e. Orange, Blue, Yellow, Green, and White. In this approach we can initially classify all RGB values in the color space by processing the neural network and storing the corresponding resulting output color in a new 256x256x256 cube, as shown in Figure 4. The 'outputcolors' cube would then store all output color values obtained from processing the 'neuralnetwork'. Although this requires extensive processing, we are computing all these values just once during calibration.

```
for(R=0;R<256;R++)
  for(G=0;G<256;G++)
     for(B=0;B<256;B++)
        outputcolors[R][G][B]= neuralnetwork(R,G,B);
```

Figure 4. Store in 'outputcolor' the complete color space as computed from 'neuralnetwork'.

After 'outputcolors' has been computed from 'neuralnetwork' we can simply obtain any pixel color value during real-time image segmentation by reading the stored value in 'outputcolors' without having to repeatedly process the neural network. This approach reduces image processing time since for each pixel we only need to do a single memory access instead of the full evaluation of the neural network. In the expression described in equation 3 'finalcolor' is obtained by reading from 'outputcolors' with the 'red', 'green' and 'blue' values from the corresponding pixel in the image.

```
finalcolor = outputcolors[red][green][blue]      (3)
```

Finally, our image segmentation procedure allows the user to discard pixels that do not correspond to any of the five color classification classes. Such pixels are considered as noise but would still be segmented into one of the five color classes. To avoid this problem the user can set during calibration certain threshold values for each

of the five colors under which classification is discarded, i.e. if the value of the output color classification is under certain threshold value then that pixel is considered as noise.

4 Experiments and Results

To test the classification algorithm we used a 640x480 image obtained from the camera as shown in Figure 5.

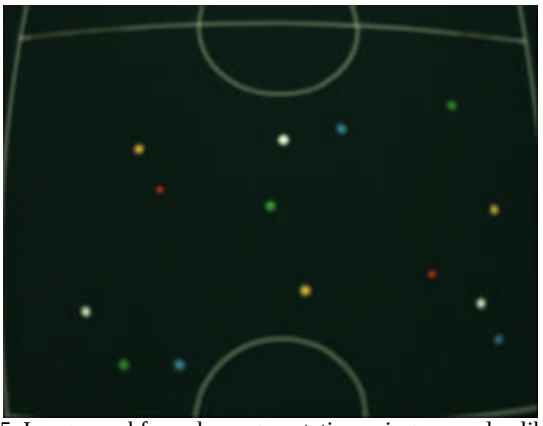

Figure 5. Image used for color segmentation using manual calibration.

Figure 6 shows the image after manual calibration. Data from both images (Figure 5 and 6) are used to train the neural network. Note that all pixels in black in Figure 6 correspond to discarded color values.

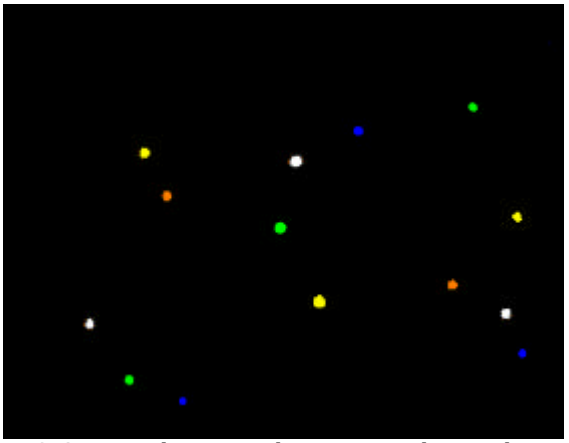

Figure 6. Segmented image used to generate color samples.

4.1 Segmentation Results

Our neural network architecture uses three input layer neurons, ten hidden layer neurons, and five output layer neurons. We used 660 samples for training and 162 samples for testing. The learning rate chosen was 0.01. Figure 7 describes the training and test errors while figure 8 presents a diagram of the corresponding learning curves for RMS and Maximum errors using 5000 epochs and performing one test every 100 steps.

Max. Training Error	0.07198
RMS Training Error	0.01934
Max Test Error	0.07128
RMS Test Error	0.02002

Figure 7. Maximum and RMS training and testing errors.

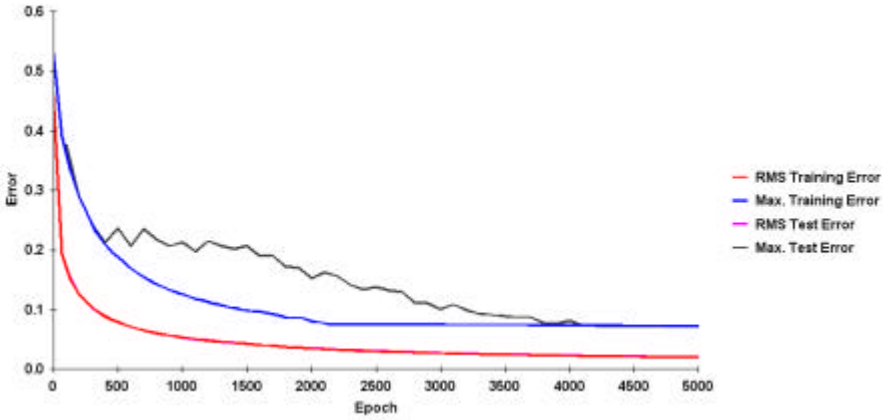

Figure 8. Training and testing error curves.

The time required to evaluate during calibration the complete color space into the neural network was from 4 to 5 minutes using an Intel Centrino Duo 2.20 GHz laptop computer with 3.5GB RAM. Figure 9 shows the result of image segmentation using the neural network method. Note the similarity with the image used for calibration shown in Figure 6.

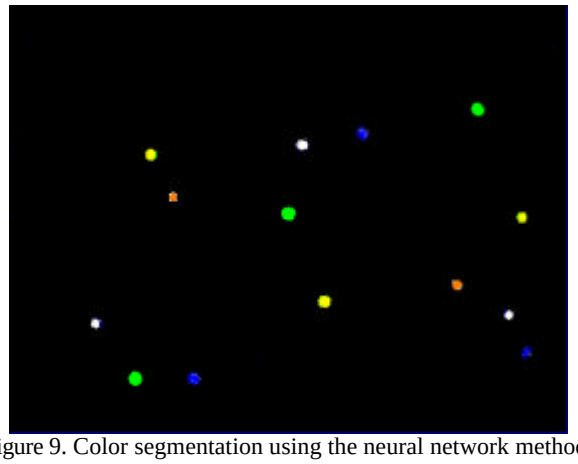

Figure 9. Color segmentation using the neural network method.

In Figure 10 we show the results from applying the different recall procedures to segment an image: (1) applying the recall procedure to each pixel of the image, (2) performing background substraction [14] and applying the recall procedure only to the remaining pixels in the image, and (3) applying the recall procedure to the RGB color space initially during calibration and then accessing the stored value later during the game. Note how only the last method results in over 30 fps as needed to achieve the desired robot performance in real SSL soccer games.

	Image Segmentation Procedure	**Processing Time**
1	Applying the neural network recall procedure to all image pixel.	2 to 3 fps
2	Performing background subtraction and then applying the neural network recall procedure only to the remaining pixels.	12 to 16 fps
3	Applying the recall procedure to full RGB color space initially during calibration and then accessing stored values later during the game.	Over 30 fps

Figure 10. Image segmentation processing time for the three pixel recall methods.

4.2 Light adaptability

To test how our neural network classification method responds to changes in light conditions we compared image segmentation against our original constant color threshold. For constant light conditions both segmentation methods responded comparably as shown in Figure 11 for the raw image previously shown in Figure 5. We analyzed the number of color regions correctly segmented.

Original light conditions	Number of patches recognized	Total number of patches
Segmentation using RGB thresholds	14	14
Segmentation using the neural network	14	14

Figure 11. Number of patches recognized using constant RGB thresholds and neural network segmentation for original light conditions.

We then compared both methods reducing by 50% the light intensity in the environment and making it non uniform across the field. Figure 12 shows the raw image obtained for the same color patches under the reduced light conditions. Note how it becomes very hard for even us to perceive the different patch locations and their corresponding colors.

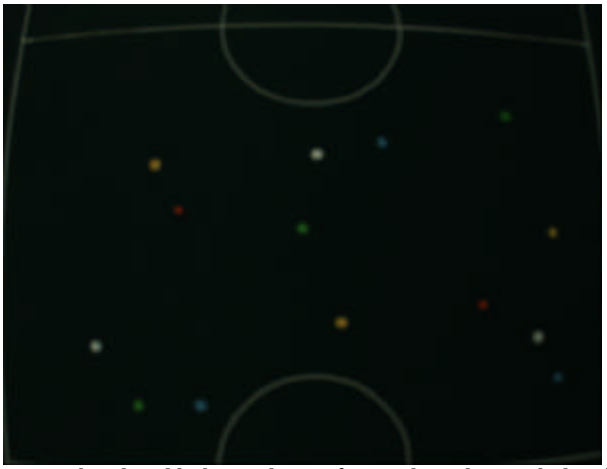

Figure 12. Raw image with reduced light conditions for similar color patch distribution.

Figure 13 shows the result of the segmentation process using the constant RGB threshold method and the neural network procedure. Note how it becomes very hard for the constant RGB threshold method to identify patches. On the other hand, our neural network method identifies correctly almost all patches.

Changing light conditions	Correct number of patches recognized	Total number of patches
Segmentation using RGB thresholds	3	14
Segmentation using the neural network	13	14

Figure 13. Number of patches recognized using constant RGB thresholds and neural network segmentation for reduced light conditions.

Figure 14 shows the resulting image segmentation using the constant RGB threshold method under reduced lighting conditions. Note how the quality of segmentation drastically drops.

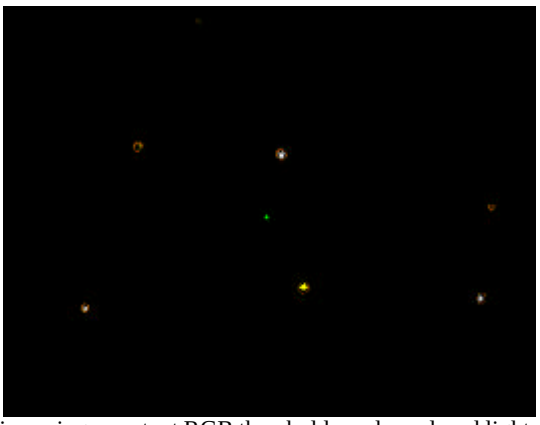

Figure 14. Segmentation using constant RGB thresholds under reduced light conditions.

Figure 15 shows the resulting image segmentation using our neural network method under reduced lighting conditions. Note how the quality of segmentation stays almost the same.

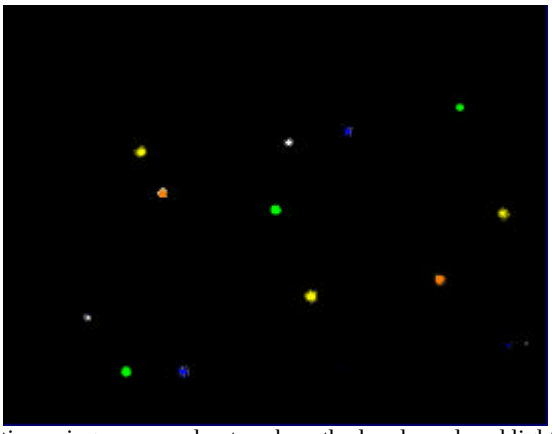

Figure 15. Segmentation using our neural network method under reduced light conditions.

Note how our neural network method only fails to recognize a single white patch located near the right lower corner of the image (between the orange and blue patches) as shown in Figure 15.

Conclusions

We have presented in this paper a new method for real time color segmentation using a neural network. The method can be applied to high resolution images achieving more than 30 frames per second under different light conditions. We compared this approach to other methods used for color segmentation in SSL. In order to apply the proposed algorithm five steps must be followed:

1. Take color samples using a manual calibration.
2. Define the neural network architecture and train it using the set of samples.
3. Initially apply the recall procedure to all pixels in the RGB color space.
4. Store the resulting color mapping in memory.
5. Segment the images using the stored color mapping.

This segmentation method can be applied to other robotics domains requiring more robust image segmentation results under variable light conditions. After using different colors in the neural network we realized that some colors are harder to classify with the neural network. To reduce the error that the neural network could have during training it is important to adjust the architecture to the particular problem. Compared to manual color calibration done in the HSV space and then converted to RGB (methodology commonly used by many teams in the Small Size league, see [11, 12]) our method relies on a learning process rather than a manual color calibration method that may be difficult to adjust. While both approaches achieve real time performance required for robot soccer, another advantage in using our current method is that the training process needs to be done only once during initial setup and then can applied throughout the full competition, something that we plan to test more extensively.

We plan to explore different ways to improve the method. One improvement would be to define more effectively color threshold values during calibration to avoid color matching errors resulting from manual calibration. Another improvement would be to define more precisely the color spaces among adjacent colors such as orange and magenta or yellow and white in order to avoid errors in their classification while reducing the noise added in the transitions between those colors. We also would like to test the method under the HSV color space and compare to our current results using the RGB color space. Finally, we plan to further test the method under diverse lighting conditions including shadows and natural light [15].

Acknowledgements

Supported by the French-Mexican LAFMI, the ACI TTT Projects in France and the UC-MEXUS CONACYT, CONACYT grant #42440, and “Asociación Mexicana de Cultura” in Mexico.

References

1. Bruce, J., Balch, T., Veloso, M.: Fast and inexpensive color image segmentation for interactive robots. In: IROS'00. (2000) 2061-2066
2. Brusey, J., Padgham, L.: Techniques for obtaining robust, real-time, colour-based vision for robotics. In: RoboCup 1999. LNAI 1856, Springer (2000) 243-256
3. Claudia Gönner, Martín Rous, Karl-Friedrich Kraiss: Real-Time Adaptative Colour Segmentation for the RoboCup Middle Size League.
4. Ketill Gunnarsson, Fabian Wiesel, and Raúl Rojas. The Color and the Shape: Automatic On-line Color Calibration for Autonomous Robots (2005).
5. Cameron, D., Barnes, N.: Knowledge-based autonomous dynamic color calibration. In: Robocup 2003, Padua, Italy (2003).
6. Mayer, G., Utz, H., Kraetzschmar, G.: Toward autonomous vision self-calibration for soccer robots. In: IROS'02. (2002) 214-219.
7. Dahm, I., Deutsch, S., Hebbel, M., Osterhues, A.: Robust color classification for robot soccer. In: Robocup 2003, Padua, Italy (2003).
8. Kestler, H.A., Simon, S., Baune, A., Schwenker, F., Palm, G.: Object Classification Using Simple, Colour Based Visual Attention and a Hierarchical Neural Network for Neuro-Symbolic Integration. In Burgard, W., Christaller, T., Cremers, A., eds.: Advances in Artificial Intelligence. Springer (1999) 267–279.
9. Simon, S., Kestler, H., Baune, A., Schwenker, F., Palm, G.: Object Classification with Simple Visual Attention and a Hierarchical Neural Network for Subsymbolic- Symbolic Integration. In: Proceedings of IEEE International Symposium on Computational Intelligence in Robotics and Automation. (1999) 244–249-
10. Bruce, J., and Veloso, M.: Fast and accurate vision-based pattern detection and identification. In: Proceedings of the IEEE International Conference on Robotics and Automation, Taiwan (2003).
11. Stefan Zickler, Michael Licitra. RoboCup SSL 2005 Team Description: Wingers. University at Buffalo (2005).
12. Kan Kanjanapas, Karu Chongsiripinyo, Poomyos Wimonkittiwat, Prempreedee Kitirattrakarn, Saorasith Intrasinghathong , Siwadol Matayakul, Tossapon Larppichet, Veerasak Nichayapun, Saran Potewiratananond, Kitti Lertlapwasin, Witthawas Boonyapinyo, Natsuda Laokulrat, Chatavut Viriyasuthee, Wittaya Wannasuphoprasit1, Manop Wongsaisuwan Plasma-Z 2007 Team Description Paper (2007).
13. Rumelhart, D. E., Hinton, G. E., and Williams, R. J. Learning internal representations by error propagation, in *Parallel Distributed Processing: Explorations in the Microstructure of Cognition*, ed. D. E. Rumelhart, J. L. McClelland, and PDP Research Group, vol. 1, *Foundations*, Cambridge, MA: The MIT Press, pp. 318–362 (1986).
14. Piccardi, M.: Background subtraction techniques: a review. In: SMC (4), IEEE (2004) 3099–3104.
15. Mayer, G., Utz, H., Kraetzschmar, G.: Playing robot soccer under natural light: A case study. In: RoboCup 2003 International Symposium Padua. (2004).