Efficient object recognition for robot soccer

using the Nao Robot

Max Bügler
Supervisor: Nico Roos

May 12, 2010

Abstract

This paper presents an object recognition algorithm for robot soccer which is optimized to run on the Aldebaran Nao robot. The aim is to recognize the ball and the goals which have distinct colors. Another objective is to keep CPU utilization to a minimum since other routines will have to run simultaneously for the robot to operate. Different methods for image clustering and shape matching are presented and compared. Experiments showed that the graph based image segmentation method is the most efficient clustering method tested. For shape matching a linear neural network returned the best results.

1 Introduction

From the standpoint of object recognition, robot soccer is a good example of a real life problem which must be simplified for low CPU utilization. The objective of this paper is to describe a multi-stage algorithm that will allow recognition of the relevant objects. Namely the orange ball, other players and the goals. Goals are rectangular frames with sky-blue and yellow color, as defined by the RoboCup Technical Committee [2]. Other robots in the game can either be located similarly to the ball, since the round head covers of the robots also have distinct colors, or by using the ultra sonic sensors of the robots. Due to this observation this paper focuses on recognition of the ball and goals.
In order to recognize objects in an image it needs a low amount of noise, a representation of objects in the scene and an algorithm to match objects. Furthermore motion prediction is needed to increase efficiency.
The following stages for processing are investigated in this paper:

1. pre-process image to reduce noise,
2. cluster image to retrieve visible objects,
3. match retrieved objects with the desired objects,
4. predict motion of objects,
5. transform image coordinates to real life coordinates.

The usage of computational resources is an essential issue in each of these steps since resources are needed for other processes needed to operate the robot. Since the light conditions in the game area are not strictly defined (the only rule considering light is that the only lights are above the game area) the algorithm has to be resistant to different light set-ups.

1.1 Color representation

Colors are represented by one byte value for each color channel (red, green, blue). The brightness of a color is defined as the square root of the sum of squares:
$V(r, g, b) = \sqrt{r^2 + g^2 + b^2}$

2 Preprocessing

The Nao robot uses two pinhole cameras with each a maximum resolution of 640x480 to record images. Inevitably the recorded images will have a certain level of noise. In order to be able to cluster the image data properly, the image needs to be denoised. Since the clustering algorithm

Max Bügler
Supervisor: Nico Roos

Figure 1: The Nao robot.

used is not very noise sensitive, a Gaussian filter with a low smoothing factor $\sigma = 0.5$ is sufficient to allow successful clustering (see Figure 2). A Gaussian filter works by averaging the color values of adjacent pixels in a certain area around each pixel [1]. The higher the parameter σ is chosen the bigger the area and the smoother the result (see Figure 3). Gaussian filtering runs in linear time.

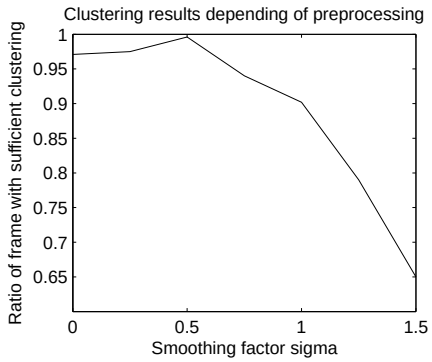

Figure 2: Clustering performance depending on smoothing factor.

3 Image Clustering

The most important step of the algorithm is to segment the image data into clusters representing the real objects visible in the scene. In order to obtain those clusters the edges separating the object must be found. There are different methods to do so. The most common approach is the use of an edge detection algorithm [3]. A more advanced approach is graph based image segmentation [4].

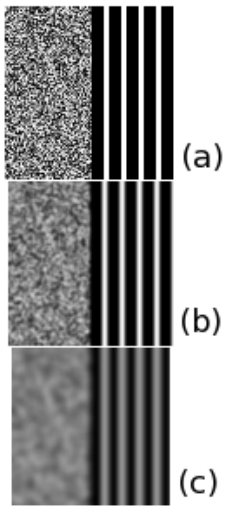

Figure 3: (a) input image. (b) output of the Gaussian filter for $\sigma = 0.8$. (c) output of the Gaussian filter for $\sigma = 2.0$.

3.1 Methods based on edge detection

Edge detection is the analysis of an image in order to find points or lines where the image data significantly changes. The detected edges should represent the outlines of objects in the scene. Experiments with the following three algorithms have been conducted: La Place, Sobel and the Canny edge detector [3].
While the Sobel algorithm approximates the gradient of the image to find edges, the La Place algorithm approximates the second derivative. Both algorithms work by shifting 2 dimensional matrices over all pixels in the image to determine a value for the pixel covered by the center cell of the matrix (see Figures 4 and 5).

The result is calculated by multiplying each matrix value by the brightness value (defined in Section 1.1) of the pixel it is covering and summing up the products. In case of the the La Place algorithm the resulting values directly translate to the intensity of the edge at the current position. In case of Sobel on the other hand, where two matrices are used, two values are returned for each horizontal and vertical direction. A downside of these algorithms is that the edges are not necessarily connected.
The Canny edge detector follows a different approach to detect edges. It works similar to the Sobel algorithm but also calculates the direction of

an edge instead of just its directional components. This makes it possible to follow edges through the image and therefore create connected edges instead of loose points. (see Figure 6)
All three algorithms result in a second image showing only the calculated edges. In order to determine the desired clusters for further processing the areas enclosed by the edges need to be calculated. This task is hard to realize with the La Place and Sobel edge detectors since the resulting edges are not necessarily connected. Therefore using a simple flood filling algorithm is not sufficient. Although the Canny algorithm tries to create connected outlines the parameters need to be tuned depending on the light conditions in order to produce a usable result. For all of the edge detection methods the connected areas in the resulting image need to be calculated. This step is unfortunately $O(n^2)$ in runtime, where n is the number of pixels.

-1	-1	-1	-1	-1
-1	-1	-1	-1	-1
-1	-1	24	-1	-1
-1	-1	-1	-1	-1
-1	-1	-1	-1	-1

Figure 4: La Place edge detector matrix.

-1	0	+1	+1	+2	+1
-2	0	+2	0	0	0
-1	0	+1	-1	-2	-1
Gx			Gy		

Figure 5: Sobel edge detector matrices. G_x is the horizontal and G_y the vertical component.

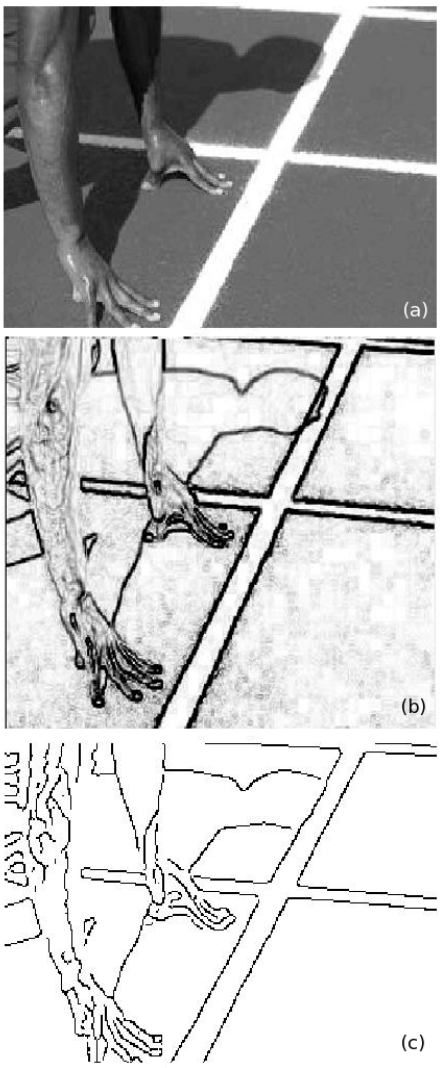

Figure 6: (a) input image. (b) output of the Sobel algorithm. (c) output of the Canny edge detector.

3.2 Graph based image segmentation

To reduce the runtime of the algorithm, graph based image segmentation, another approach to the problem, was analyzed. This algorithm is based on representing the image as a graph, where each node represents one segment (containing one pixel at the beginning) where the edges represent adjacency. Each edge has a weight w which is a non-negative measure for the dissimilarity between neighboring segments. To determine whether two nodes in the graph should be joined, the inner difference of the pixels contained in one node is compared to the difference to the second node. The minimum internal difference $\mathrm{MInt}(C_1, C_2)$ is calculated by obtaining the largest weight of the minimum weight spanning tree over the pixels of each segment and taking the minimum of those.

The algorithm consists of the following 5 steps:

The input is a graph $G = (V, E)$, with n vertices and m edges (initially a grid graph spanning over all pixels). The output is a segmentation of V into components $S = (C_1, ..., C_r)$.

1. Sort E into $\pi = (o_1, ..., o_m)$, by non-decreasing edge weight.
2. Start with a segmentation S^0 , where each vertex v_i is in its own component.
3. Repeat step 4 for $q = 1, ..., m$.
4. Construct S^q given S^{q-1} as follows. Let v_i and v_j denote the vertices connected by the q-th edge in the ordering, i.e., $o_q = (v_i, v_j)$. If v_i and v_j are in disjoint components of S^{q-1} and $w(o_q)$ is small compared to the internal difference of both those components, then merge the two components otherwise do nothing. More formally, let ${C_i}^{q-1}$ be the component of S^{q-1} containing v_i and ${C_j}^{q-1}$ the component containing v_j . If ${C_i}^{q-1} \neq {C_j}^{q-1}$ and $w(o_q) \leq \text{MInt}({C_i}^{q-1}, {C_j}^{q-1})$ then S^q is obtained from S^{q-1} by merging ${C_i}^{q-1}$ and ${C_j}^{q-1}$. Otherwise $S^q = S^{q-1}$.
5. Return $S = S^m$.

Although the algorithm only makes greedy decisions the results satisfy global properties (see Figure 7) and the runtime is nearly linear in number of pixels [4]. Scaling the $\text{MInt}({C_i}^{q-1}, {C_j}^{q-1})$ function causes the algorithm to create smaller or bigger segments. Experiments showed that this can be used to optimize the algorithm for the specifics of the camera used and the size of the object to be found. They do not need to be changed for different light situations. This makes the algorithm attractive for the task.

3.3 Conclusions

Although the common edge detectors Sobel, La Place and Canny work in linear time and produce good results, they have to be post processed in order to find the connected areas in the image. Since that extra step has quadratic runtime the second approach using graph based image segmentation proved to be faster doing the complete task in near linear runtime. Moreover this algorithm is less sensitive to noise in the image and far more consistent given different lighting situations. Therefore it was considered best for image clustering.

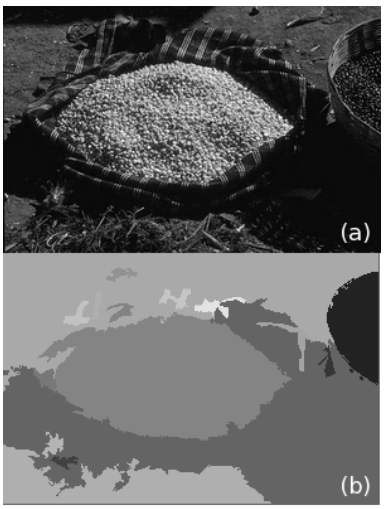

Figure 7: (a) input image. (b) segmented output [4].

4 Shape Matching

To determine which of the segments in the image represent the objects relevant to the game a shape matching algorithm is needed. In case of robot soccer the relevant objects are the ball and the goals which only require a simple shape matching algorithm.

4.1 Matching rectangular and oval objects

The simplest approach that was examined uses two functions, which measure ovality and rectangularity. To measure the ovality $B(S)$ of a segment S, the ratio of points inside a oval, defined by the bounding rectangle (see Figure 8) of the segment, is subtracted by the ratio of points outside of that oval. Since the goal is no filled rectangle but a frame (see Figure 8), the second function $R(S)$ has to subtract the ratio of points inside the frame from the ratio of points on the frame.

$$\text{InOval}(P) = \begin{cases} 1, & \text{if P is inside} \\ & \text{of the oval area,} \\ -1, & \text{otherwise.} \end{cases}$$

$$B(S) = \frac{1}{n} \sum_{i=1}^{n} \text{InOval}(S_i)$$

where n is the number of pixels in S and S_i is the ith pixel of the segment S.

$$\text{InFrame}(P) = \begin{cases} 1, & \text{if P is inside} \\ & \text{of the frame area,} \\ -1, & \text{otherwise.} \end{cases}$$

$$R(S) = \frac{1}{n} \sum_{i=1}^{n} \text{InFrame}(S_i)$$

where n is the number of pixels in S and S_i is the ith pixel of the segment S.

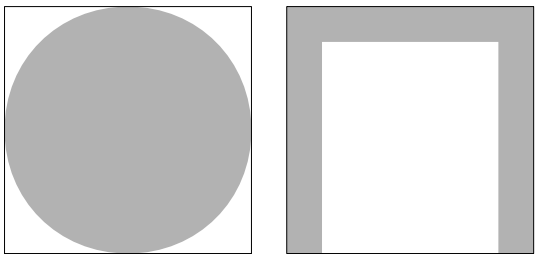

Figure 8: Ball shape (left) and goal shape (right)

Experiments showed that this method is well able to distinguish goal and ball from other objects in the scene. To determine whether one of the objects is currently visible the functions need to be thresholded in some way. It turned out that there are no fixed thresholds to be found. After optimizing the settings to an specific environment the method returned good results. Unfortunately only small changes in the environment causes the results to decrease strongly. This is caused by the shaded lower half of the ball. The lower the light intensity the higher the similarity between the upper and lower area of the ball. This causes the shape of the segment to change slightly. In order to avoid that problem other methods of shape matching have been investigated.

4.2 Polygon matching

Another approach to shape matching is polygon matching, where the outline of objects is compared. To be able to do so, the segments have to be converted to polygons. This requires all the outline pixels of the segment to be calculated. Unfortunately this process is $O(n^2)$ in runtime, where n is the number of pixels, and it is problematic for non solid objects like a torus, where multiple outlines exist. Furthermore the pixels in the outline need to be sorted to create a continuous outline.
To compare the resulting polygons the turning angle function [5] (see Figure 9) was used since it is invariant to rotation and scale. It has a value for each point of the polygon. Following the points around the polygon the function changes at each point by the change in angle at that point. It can then be shifted for comparison to obtain invariance to rotation. Although this approach returns good and stable results, the low efficiency of the polygon creation algorithm, especially when multiple outlines are present, is strongly increasing processing times.

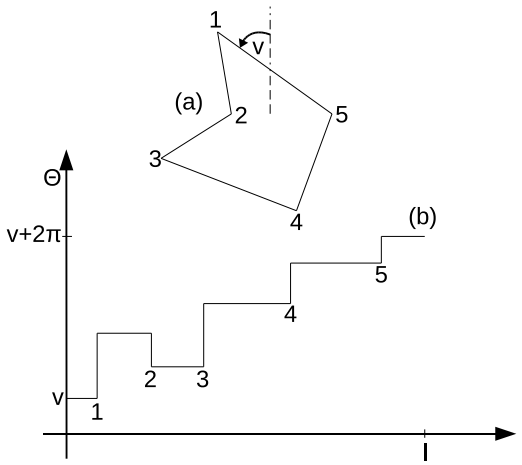

Figure 9: (a) segment shape. (b) resulting turning angle function.

4.3 Using neural networks for shape matching

Since neural networks can be used to solve nearly every classification problem they can also be used in multiple ways to recognize the segments in the image. First the results of the two functions defined in section 4.1, the euclidean distance between desired color and segment color and the segment size have been used as input for a multi layer network. This network was trained with three outputs, which represent the ball and the two goals, to return 1 if an object is represented by the segment data inputted and 0 if not. The training was performed using back propagation. This approach unfortunately showed

the same problems as described in section 4.1.
As a second approach a linear neural network (Perceptron) was trained getting a scaled representation of the segment and its color difference as input. To obtain the scaled representation the shape of the object is projected on a two dimensional array of input neurons with constant size (see Figure 10). During experiments results varied depending on the number of input neurons used for the shape. Further experiments showed that the average dimensions of the object in the image data are best for the scaled representation. So for objects of different sizes separate networks with different numbers of input neurons should be used.

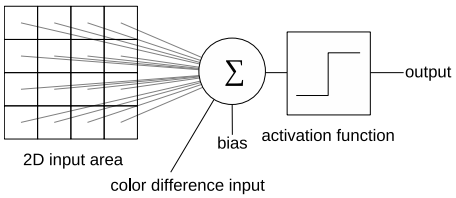

Figure 10: Linear neural network structure.

4.4 Conclusions

All the methods tested returned good results. Although the polygon matching based on the turning angle function is the only algorithm tested that is totally invariant to rotation it needs the polygons in the scene to be calculated. That unfortunately can not be done in linear time. Since the ball is round and the goals are stationary rotational invariance is considered less important than avoiding an extra step with quadratic complexity. Both the functions described in section 4.1 and the neural network approach are successful in detecting the objects, but the linear functions have to be thresholded in order to determine whether an object is visible. A linear neural network could be trained to solve the problem with very low error rates .
Figure 11 shows a comparison of the discussed shape matching methods. The left part of the graph shows the success rates of recognizing an object when it is present, while the right part shows the success rates of recognizing the absence of an object rather than pointing to another object.

5 Improving Efficiency

Since the object recognition is only one of many routines that will be executed on the robot, the goal is to make it as efficient as possible. All the subroutines were already selected based on speed such

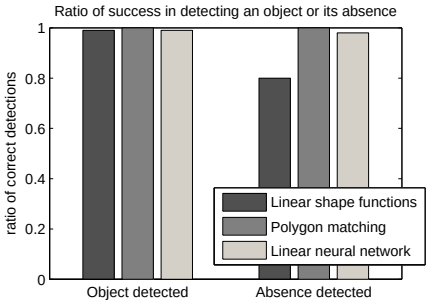

Figure 11: Shape matching error rates.

that the algorithm runs in near linear time. In order to improve it further two big improvements were implemented. In order to compare the methods and measure the improvement, a 3D scene was rendered into a video of 15 seconds runtime. The performance was measured by calculating the average time needed to process one frame.

5.1 Dynamic programming

While the image segments are created by the graph based image segmentation algorithm (section 3.2) information like the central point and the average color require very little computation. So instead of calculating this information later on demand, by iterating over the pixels in the segment, the sum of the following properties of each of the contained pixels is stored in each segment:

- red color value,
- blue color value,
- green color value,
- x position,
- y position.

These summations can be calculated dynamically during the construction of the segments. The average color and mean position for a segment can later be calculated by dividing these numbers by the number of pixels contained in the segment. Also the extreme values are stored, namely minimum and maximum x and y values, to define the dimensions of the segment. This reduced the average runtime by 15% (see Figure 16).

5.2 Motion prediction and partial image processing

In the case where the locations of all important objects are known the area of the image to be processed can be reduced. This could strongly

reduce the processing time. In order to estimate an object's position two methods have been investigated. The first method assumes the object to be moving on a straight line. In order to estimate that line two prior positions of the object need to be known. The vector between these positions is now assumed to be the change of position for the next frame (see Figure 12).
The second method tested assumes objects may follow a curved trajectory. Therefore the change in the first derivative of the prior three positions of the object is used to predict the next position (see Figure 13).

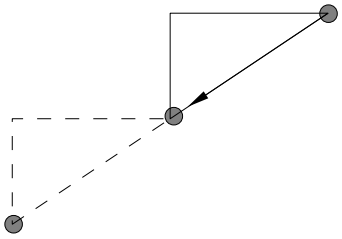

Figure 12: Linear motion prediction. The dashed triangle is the predicted movement.

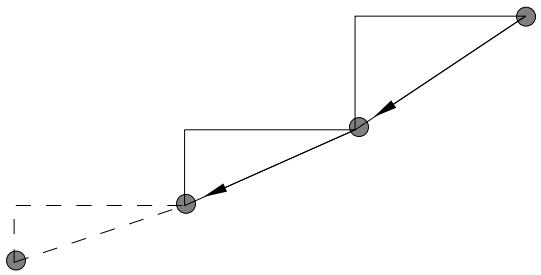

Figure 13: Non-linear motion prediction. The dashed triangle is the predicted movement.

Experiments showed equal results for both methods (see Figure 14). This can be explained by the little time between the frames and the therefore little movement. Since the non linear prediction might still be useful for lower frame rates it was kept in the algorithm as an option although by default linear prediction is used.

The predicted positions of the objects are used to reduce the part of the image to be processed. Since the expected positions and the dimensions of the objects are known an area can be calculated which will contain all of the objects in the next frame. This is only possible if all objects were

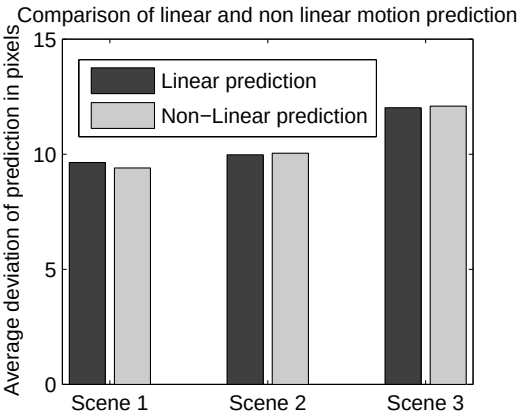

Figure 14: Comparison of prediction methods.

visible in the previous frame. Since the prediction is not always precise a safety area around the minimal area is kept which is determined by scaling up the dimensions of the objects by a factor of 1.5 when calculating the area (see Figure 15).

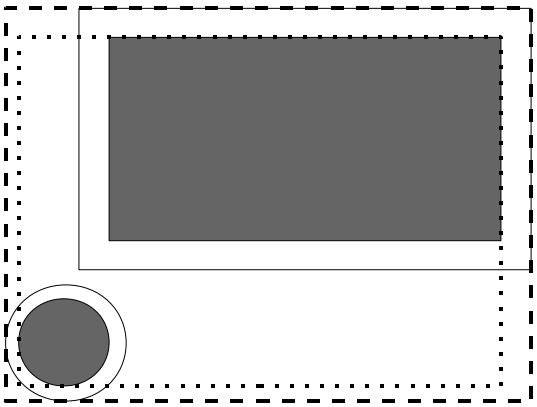

Figure 15: Partial image processing area. The dotted rectangle is the area where objects are predicted to be. The dashed rectangle is the safety area.

Although this method improved the speed of the algorithm drastically, the improvement is not constant. So partial image processing is only applicable when all objects to be detected are visible. In case of robot soccer that is usually the enemy's goal and the ball. If one of the objects is out of sight the entire frame needs to be processed in case the object appears somewhere.

5.3 Conclusions

Both tested improvements were able to speed up the algorithm by about 50% on the example scene but only the 15% speed improvement caused by the dynamic programming part is a constant improvement (see Figure 16). The partial image processing can only be used when all important objects have been detected in the previous frames.

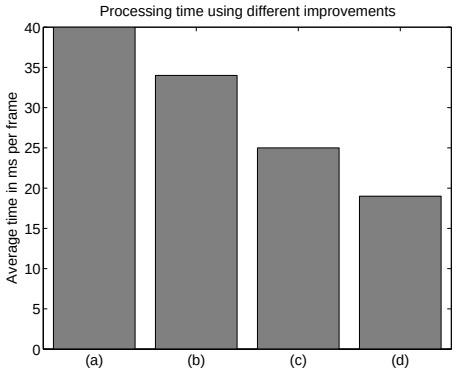

Figure 16: Comparison of processing time with different improvements. (a) Reference - without improvement, (b) dynamic programming, (c) partial image processing, (d) both improvements.

6 Nao implementation

In order to make the algorithm usable on the Nao robot, it has been implemented in C++ and compiled for the Linux platform. An alternative would have been to implement it in Python, which was tested to be too slow for the complex algorithm. Anyway the C++ code can be called from python files which are mainly used to control the robot. Images are captured from the cameras using the NaoQi package which is provided by the robot. Based on the specifications of the robot's cameras the obtained object coordinates need to be transformed to coordinates representing the objects position in the real world (the game area).

6.1 Handling the two cameras

The Nao Robot is equipped with two cameras, one on the forehead above the eyes and the other one near its mouth (see Figure 17). Unfortunately it is not possible to capture images from both cameras simultaneously since only one camera input is available. The switching between the cameras is handled on hardware level and therefore causes a certain delay. The number of switches should therefore be minimized. In order to determine from which camera to capture a set of simple rules was defined:

- If the ball is detected on the top camera only capture that camera.
- If the ball is detected on the bottom camera only the bottom camera is captured until the robot needs to aim for the goal. At this point the top camera is captured to determine the location of the goal.
- If the ball is not visible switch between cameras at a certain rate until it gets visible.

These rules allow the algorithm to work with a minimum number of switches between the cameras.

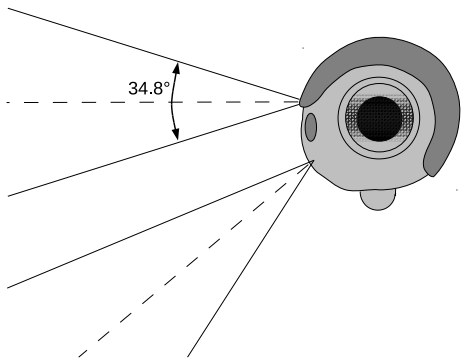

Figure 17: Diagram of the two cameras and the fields of view [6].

6.2 Coordinate transformation

Since robot soccer is played on a planar field the transformation from the coordinates on the camera images to real life coordinates is linear. Furthermore the goal will always be located on the top camera and only the x-coordinate is important for aiming. In order to walk towards an object the robot needs to keep it horizontally centered in the camera image, given the robot is looking forward. So the most important information for an objects location is the horizontal deviation from the center. Furthermore for being able to kick the ball, the distance to the ball should be returned. Both informations can be obtained by the following two linear transformations:

$$x_r = (\frac{x_c}{\frac{x_{\max}}{2}} - 1)\frac{x_1 + \frac{y_c}{y_{\max}}(x_2 - x_1)}{2}$$

$$y_r = y_1 + \frac{y_c}{y_{\max}}(y_2 - y_1)$$

where

x_c	is the x coordinate of the object in the camera image,
y_c	is the y coordinate of the object in the camera image,
$x_{\max}$	is the width of the camera image,
$y_{\max}$	is the height of the camera image,
x_r	is the x coordinate of the object relative to the robot,
y_r	is the y coordinate of the object relative to the robot.

x_1, x_2, y_1 and y_2 are illustrated in Figure 18.

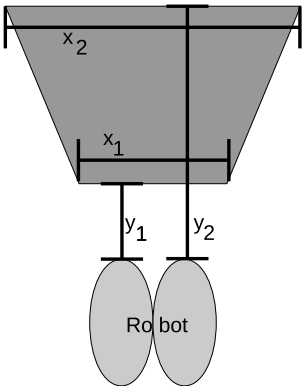

Figure 18: Diagram of the robot and its field of view in top perspective.

These formulas transform the rectangle, bounding the camera image, to the trapezoid field of view on the planar game area (see Figure 18). The parameters x_1, x_2, y_1 and y_2 need to be set according to the active camera.

6.3 Capture resolution

Since the resolution of the captured images has a huge influence on the processing time of frames it has to be minimized. The cameras in the Nao robot allow a maximum resolution of 640x480. For the simple objects to be found experiments showed that 160x120 is the lowest resolution with sufficient results (see Figure 19).

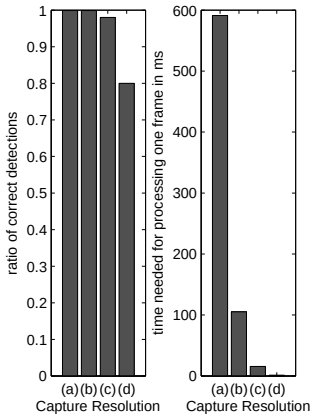

Figure 19: Comparison of results (left) and runtimes (right) with different capture resolutions. (a) 640x480 (b) 320x240 (c) 160x120 (d) 80x60

7 Conclusions

This paper described an efficient algorithm for object recognition which is optimized for robot soccer using the Aldebaran Nao robot. Several methods for the needed steps were investigated and the most effective ones were chosen. The image data obtained from the robot's cameras is preprocessed with a weak Gaussian blur in order to remove noise. This step has runtime complexity $O(n)$, where n is the number of pixels. To obtain a representation of the objects visible in the image data graph based image segmentation was chosen to be most effective. It has been improved using dynamic programming to precalculate values needed for later steps. This step has runtime complexity $O(n\log n)$. Determining which of the segments obtained are the relevant objects requires a shape matching algorithm. Creating polygons from the obtained segments gives the best results, but required too much computational resources. A linear neural network approach was chosen because it balanced the quality of the results, and the required computational resources.
Since the Nao Robot has two cameras, but can only use one at a time, a set of rules has been defined to switch between the cameras. Furthermore a linear transformation from the cameras to the real coordinates relative to the robot was described.
The robot's cameras can capture in different resolutions up to 640x480. Since the resolution has huge influence on the runtime of the algorithm, experiments were conducted to find the lowest resolution required for sufficient results. This

showed to be 160x120.
To run the algorithm on the Nao robot it was implemented in C++ and compiled for Linux. The resulting algorithm has runtime complexity $O(n\log n)$ where n is the number of pixels in the image. Although this algorithm allows stable recognition of the relevant objects robot soccer remains a challenging task for both robots and developers.

8 Future work

The algorithm is currently extended for more complex tasks like recognizing cartoon characters and locating license plates. Furthermore it is investigated whether the clustering algorithm can be used to create and cluster two dimensional Kohonen maps. It would also be interesting to investigate whether it can be applied to higher dimensional data like a set of multiple frames. It is possible that using different kinds of color representations, like HSV, will improve the results when additional objects need to be detected (like the corner poles or other robots).

References

[1] Coll, A. Buades B. and Morel, J. M. (2005). A review of image denoising algorithms, with a new one. *Multiscale Model. Simul.*, Vol. 4, No. 2, pp. 490–530.

[2] Committee, RoboCup Technical (2009). Robocup standard platform league (nao) rule book. Technical report.

[3] D.B, Shin M.C Goldgof and K.W, Bowyer (2001). Comparison of edge detector performance through use in an object recognition task. *Computer Vision and Image Understanding*, Vol. 84, No. 1, pp. 160–178.

[4] Felzenszwalb, Pedro F. and Huttenlocher, Daniel P. (2004). Efficient graph-based image segmentation. *International Journal of Computer Vision*, Vol. 59, No. 2.

[5] Rangaraj M. Rangayyan, Juliano Daloia de Carvalho, Denise Guliato and Santiago, Sergio Anchieta (2006). Feature extraction from the turning angle function for the classication of contours of breast tumors. itab2006 Ioannina, Greece.

[6] Red documentation. the developer documentation for nao. http://academics.aldebaran-robotics.com/docs/reddoc/.