Fast and Accurate Robot Vision for Vision based Motion

Pieter Jonker, Jurjen Caarls, Wouter Bokhove

Pattern Recognition Group, Department of Applied Physics,
Delft University of Technology
Lorentzweg 1, 2628 CJ delft, The Netherlands
pieter@ph.tn.tudelft.nl

Abstract. This paper describes the vision module from the soccer playing robots of the Dutch Team. Fast vision is necessary to get a close coupling with the motion software in order to allow fast turning and dribbling with the ball without loosing it. Accurate vision is necessary for the determination of the robot's position in the field and the accurate estimation of the ball position. Both fast and accurate are necessary for the goalkeeper, but also when one robot passes the ball to another. While the Dutch team has pneumatic kicking devices that allows catching a ball smoothly, fast an accurate vision is mandatory. We use lens undistortion, a new color segmentation scheme and a shape classification scheme based on linear and circular Hough transforms in regions of Interest. We use a severe calibration procedure to get very good distance and angle measurements of the known objects in the field of view of the robot. For the keeper robot we use a Linear Processor Array in SIMD mode, that is able to execute the entire robust vision algorithm within 30ms. However the same software was programmed for the other robots with a WinTV framegrabber on the on-board Pentium of the robot. With optimizing for speed we also remained within 25ms, however, omitting the circular Hough transform for the ball and processing in a separate thread the Linear Hough transforms for self-localization on lower rate of about 50msec. The angular errors at 0 °, 20 ° and 30° heading are about 0.6 °, 0.5° and 0.4° up to 4,5 meters. The distance error at 0° heading is 5% up to 3 meters.

1 Introduction

The robot that is mainly used in the Dutch Team is the Nomad Scout robot [1]. with a Pentium 233MHz running Linux. The robots are equipped with a WinTV PCI framegrabber. In this case the Pentium performs the image processing. As tracking the ball, team-mates and competitors was one of the most difficult tasks in past RoboCup matches, we equipped the goalkeeper with an IMAP-Real-time Vision System from NEC's Incubation Center [2]. The cameras used are NTSC color cameras with fixed zoom manual iris lenses with a (wide) opening angle of 94°. The IMAP-RVS is a Linear Processor Array with 256, 8 bits data processors in SIMD mode, color framegrabbing hardware and a RISC control processor on a PCI board. It is a parallel architecture specially made for real-time image processing, where a single column of an image is mapped onto one data processor. The system is programmed in

P. Stone, T. Balch, and G. Kraetzschmar (Eds.): RoboCup 2000, LNAI 2019, pp. 149-158, 2001.

1DC, C extended for data parallel processing, simplifying to make equivalent software for robots equipped with and without IMAP. The modules with and without IMAP have the same software interface to the other modules.

Fig. 1. The IMAP-Vision System board

The software architecture of the soccer robots (Figure 2) shows three identical robots connected to each other via a communication module. The basic layer of the software consists of virtual devices. The middle layer contains the intelligent basic skills of the robot and the World Knowledge Module, in a sense the robot's memory of its past, present and its assumptions about its direct future world. Within the top level, intelligent decisions are made to adapt the robot to the changes in the game [3].

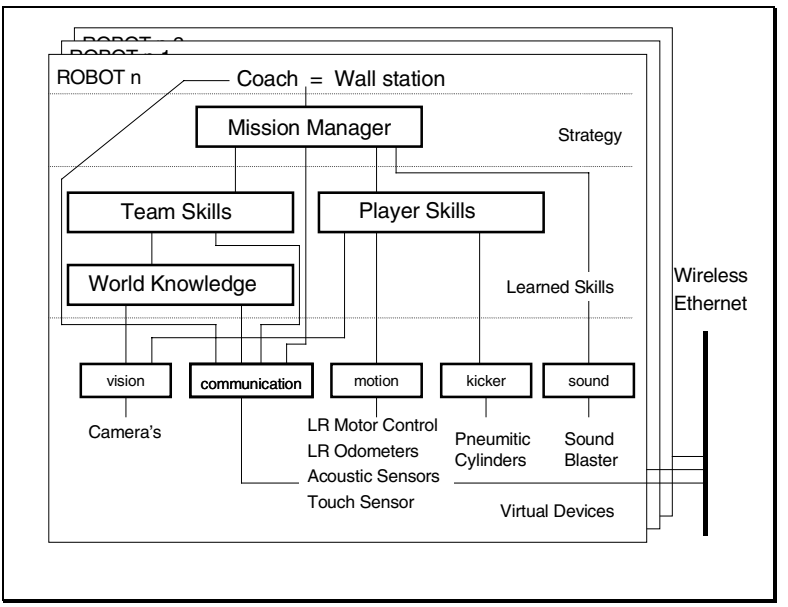

Fig. 2. Software architecture of the autonomous soccer playing robots

2. Vision for object recognition and tracking

This paper deals with the Vision Module of the robots of the Dutch Team. This module provides measurements to the Player Skills Module and the World Module and through that indirectly to the Team Skills Module. The Vision Module provides angles and distances of

known objects, like ball, goals, mates, opponents, corners, white lines and field borders. The operations that we use to realize this within the vision module are:

- Specular reflection reduction, white balancing and lens undistortion
- Segmentation in color-space; label each pixel according to YUV-information
- Search for the ball shape using a Circular Hough Transform
- Search for white lines (and wall/field-transits) using a Linear Hough Transform
- Find the corners and goal - wall transits for auto positioning using a re-segmented image

2.1 Specular reflections, white balancing, lens distortions

In order to get rid of most specular reflections a Polaroid filter is used on al cameras. To delude the auto white balance of the camera, a piece of white is visible in its VOF. Blooming is reduced by the dark filter, and the Polaroid itself takes away some of the reflections itself. Due to the wide opening angle of 94 ° of the color camera, all images are distorted. This distortion can be found by a suitable calibration method. We use two ways to circumvent the effect of the distortion: To do the image processing operations and then compensate the Regions of Interest (ROIs), features and points found for the known distortion, or to undistort the entire image and then do all subsequent operations. Using Tsai's [4] algorithm for camera calibration, as explained later, a shift in X- and Y-direction of each pixel can be calculated. These X and Y shifts are stored in Look-Up-Table images that are used to undistort the image. However for some pixels in the output image no data exist (figure 3b). Those are filled with the majority vote of neighbor data. The result is shown in figure 3c.

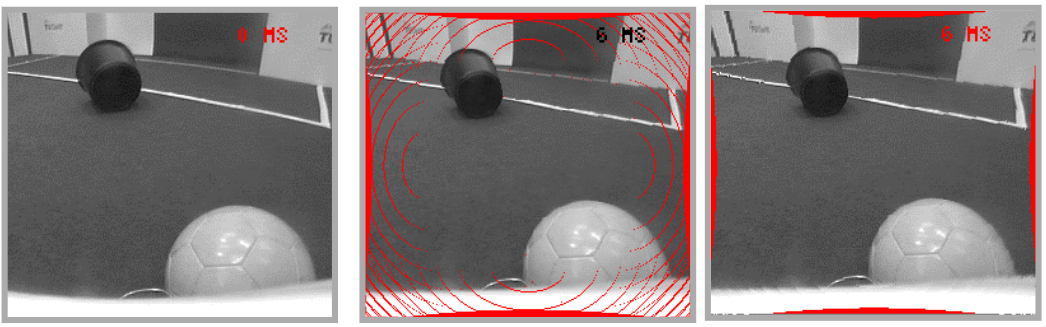

Fig. 3. Lens undistortion: a) original , b) Corrected image, c) Positions with missing data filled in.

2.2 Segmentation in color-space

For the segmentation of objects in color space we use YUV images. The benefit of the YUV space is that the UV space contains almost all color-information, meaning that a 2D image can be used to do the color segmentation. Moreover, many framegrabbers support YUV extraction in hardware. In principle the transformation from RGB to YUV is given by (Y, U, V) below,

however if the framegrabber is not supporting this, the pseudo space (Y', U', V') can be used, as only color differences important.

$$\begin{pmatrix} Y \\ U \\ V \end{pmatrix} = \frac{1}{128} \begin{pmatrix} 38 & 75 & 15 \\ -22 & -42 & 64 \\ 64 & -54 & 10 \end{pmatrix} \begin{pmatrix} R \\ G \\ B \end{pmatrix} + \begin{pmatrix} 0 \\ 128 \\ 128 \end{pmatrix} \qquad \begin{pmatrix} Y' \\ U' \\ V' \end{pmatrix} = \begin{pmatrix} 1 & 1 & 1 \\ 1 & 0 & -1 \\ 0 & 1 & -1 \end{pmatrix} \begin{pmatrix} R \\ G \\ B \end{pmatrix}$$

Figure 4a shows a typical scene. Figure 4b and 5 show the UV space of this scene with in greyvalue overlay the intensity of the pixels found at that position in the UV space. So very white spots in e.g. the red area are redish pixels with high intensity. Two methods can be used to do the color segmentation.

In figure 4b the method is shown that is often used, see e.g. [5], in which in U and V direction, a region of interest is made for each object to be found. This is fast but not robust.

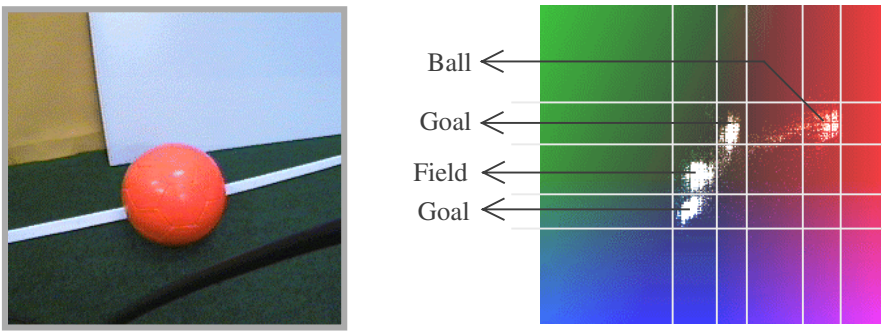

Fig. 4. a) Original typical scene, b) UV Colorspace with square ROIs to color classify objects

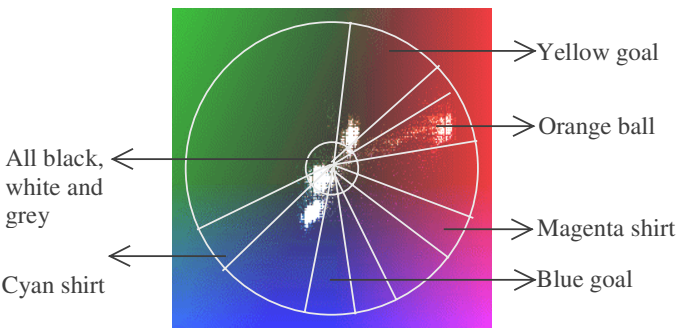

Fig. 5. Color classification method based on a simplified HSI bounding box, or pizza slices in YUV

Figure 5 shows our method, based on the observation that similar colors can be found at the same angle from the center. We simplified the original HSI colors-pace, so that no multiplications are needed when represented in YUV. The majority of the pixels from the orange ball can indeed be found in the Region Of Interest spanned in Figure 4b, however the pixels in the tail also belong to the ball.

Typically, these are the pixels on top and at the bottom of the ball, where the color is less saturated due to the illumination from above and the shadow beneath. For the correct determination of the distance, e.g. based on the position where the ball hits the floor, these pixels are important. When segmenting the UV space in pizza slices, more pixels can be classified correctly and robustly to the same color, as the blob of pixels of each color class will move along the radiant during the many lighting conditions in the various game situations. In addition an inner circle is used for each color class to exclude all pixels that have a tendency too much to the grey, white, or black. For black and white objects we do an additional classification based on thresholding in the intensity image. Our color calibration procedure is based on drawing a region of interest around the object of interest in the input image, looking at its mapping in UV space and moving the two hands and inner circle and examining the result in the segmented image. This can be repeated for several play situations, until a robust segmentation is obtained. From this color calibration procedure we derive two Look-Up Tables, one for Y and another for (U,V), whereas each known object with its color class (or color segment) is assigned a bitplane in this LUT. In this way we can determine in one pass the color class of a pixel. Hence it remains possible that a pixel can be assigned to more than one colorclass. We perform:

Class = LUT_Y [Y] & LUT_UV [U,V], with & a bitwise logic AND

Per image from the grabber we first classify all pixels in this way. During this classification we form on the fly a projection of the classes on that column (i) and row (j) of the image:

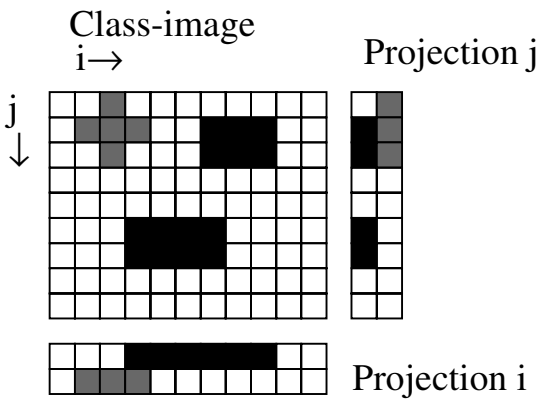

Fig. 6. Fast pixel classification procedure allowing initially multiple class labels

Then to detect each object separately, we use subsequent ROI passes in which we consider the projection boundaries as boundaries of a Region of Interest that we investigate. E.g. a red object on j=(0-2) and i=(1-3), a black object on j=(1-2) i=(3-8) and / or j=(5-6) i=(3-8). For each found ROI we re-project in order to separate objects until we are done. This results in bounding boxes around each object of a certain color class.

In order to be more robust against noise, we count in the ROI projections per row and column the pixels belonging to a certain color class. E.g. for a row we perform:

$$count_j = \sum_i \left(class\_image[i, j] \; has \; class\_bit == 1 \right)$$

Only the columns and rows that exceed a certain threshold are considered further. Moreover, we use minimum values for ROI sizes to filter out small noise objects. From the bounding boxes found the point at the bottom in the middle is used to determine the distance (this point is expected to be on the ground) and the middle-points of the side edges are used to calculate the angle of the object, using transformation formulas with calibrated parameters.

2.3 Searching and tracking the ball using a circular Hough Transform

The ball is found by fitting a circle through the edges of the orange blob, which is found during color segmentation. This feature makes it possible to estimate the correct position of the ball even if it is up to 70% occluded by other robots. Fitting a circle is done using a Circular Hough Transform in three dimensions [6]. This tries to find the three parameters by which a circle is determined: the center of the circle (XC, YC) and its radius R:

$$(x - X_C)^2 + (y - Y_C)^2 = R^2$$

The 2D Circular Hough Transform uses a fixed search-radius and tries to find the center of a circle with this radius. Therefore only a 2D Hough space is enough to find the correct circle. We use a Sobel edge detector on the orange blob, which also provides directional information. This is used to allow votes only to the XC and YC in the direction of the edge. As all edge-points vote to the correct center, a peak-detection-filter will find the correct center and hence the correct circle. The value of the fixed radius is determined by a recursive loop. The first time an orange blob is located in the image, using the calibration as described below, a radius of the ball for each position in the image can be found. This is used as start value for the recursive loop, which in general only needs one or two iterations.

2.4 Self-localization

For self-localization of the robot fixed markers are needed. The white lines and the wall/field transition in the field are good candidates. For a very fast Hough Transform we can't use all the points in the image. So in a pass from the top to the bottom of the image we look for white <-> non-white transitions. We select the first and last 2 points per column resulting in 4*320=1280 points to be transformed. The camera image suffers from lens distortion, so straight lines in the Field of View of the camera aren't straight in the image. Therefore we first transform all points using the formulas that define a virtual camera as described in 2.5. With all those points in Hough Space we search for peaks. Those peaks mark the lines that can be seen in the image. Note that here the IMAP Linear Processor Array and the WinTV grabber implementations differ in the sense that lens distortions are only compensated in the WinTV version for the image lines that are needed, whereas in the IMAP-RVS the whole image is undistorted. To find the corners and goals in an easy way, we have to use coordinates that are corrected for the camera tilt. To be fast, we only transform the first 20 lines of the image, as

the horizon lies within this area. With a histogram on the horizontal axis of the image, we can look for transitions from white to green and vice versa. The angle to those transition lines can be found. For an accurate repositioning algorithm for the keeper we need as many transition as can be seen. So we find each corner pole pair, as well as the wall-goal, goal-wall transitions.

2.5 Calibration procedure for lens distortion, angle and distance.

For the calibration of the lens distortion parameters we use the algorithm of Tsai [4]. First we go from pixel-coordinates (X_f,Y_f) to image-coordinates in mm: (X_d,Y_d) , d from distorted.

$$\begin{aligned} X_d &= d`x \cdot (X_i - C_x) \\ Y_d &= dy \cdot (Y_i - C_y) \end{aligned} \qquad \text{with} \qquad d`x = d_x \frac{N_{cx}}{N_{fx}} \tag{1}$$

(X_d,Y_d) : coordinates in mm of the distorted CCD image plane. (0,0) is the optical axis
d_x center to center distance between adjacent CCD sensors in the scanning direction X
d_y center to center distance between adjacent CCD sensors in the Y direction; double this value when capturing only half of the video-frame.
N_{cx} number of sensor elements in the x direction
N_{fx} number of pixels in each line as they are acquired by the framegrabber.
(C_x,C_y) : center of the lens distortion in pixels.

We only take the quadratic distortion into account, to undistort:

$$\begin{aligned} X_u &= (1 + k_1 r^2) X_d \\ Y_u &= (1 + k_1 r^2) Y_d \end{aligned} \qquad \text{with} \quad r^2 = X_d^2 + Y_d^2$$

The Tsai algorithm now calibrates C_x,C_y and k_1. For this we use a calibration pattern as described below. The results are used to shift the pixels as explained above.

The camera on the robot has an angle of 30 degrees with the floor. To ease the calibration of the distance we first transform the image such that the resulting image is the image if the camera (C) would look straight ahead. We call this the virtual camera (VC) image. This is done by projection of the 2D image of C onto the image of VC. Coordinates in VC are expressed as (Xv,Yv). Assuming Yu=0 and Yv=0 on the horizon H (see figure 7), we obtain:

$$Z_H = Z_Q = \frac{f}{\cos(\varphi)} \qquad Z_P = \frac{f}{\cos(\varphi)} - Yu_P \cdot \sin(\varphi) \qquad \mathrm{Yv_P} = Yu \cdot \cos(\varphi)$$

With:

$$\frac{Yv_Q}{Yv_P} = \frac{Z_Q}{Z_P} \Leftrightarrow Yv_Q = \frac{Yu \cdot \cos(\varphi) \cdot \dfrac{f}{\cos(\varphi)}}{\dfrac{f}{\cos(\varphi)} - Yu_P \cdot \sin(\varphi)} = \frac{Yu \cdot \cos(\varphi)}{1 - Yu_P \cdot \dfrac{\sin(\varphi) \cdot \cos(\varphi)}{f}}$$

For Xv_Q we have:

$$Xv_Q = \frac{Xu}{1 - Yu_P \cdot \frac{\sin(\varphi)\cdot\cos(\varphi)}{f}}$$

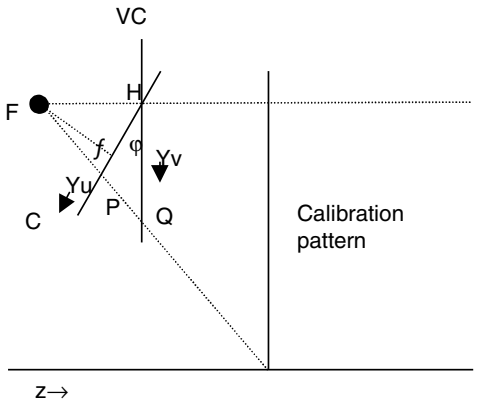

Fig. 7. Position of the real camera C en virtual camera VC in the world. H is the intersection of the horizon with the cameras. P and Q are the projections of the bottom side of the pattern onto C en VC respectively. f is the perpendicular distance of the focus point F to camera C.

These formulas are linear in the denominator and we can calibrate this using the camera on the robot and a dot pattern perpendicular to the floor, so that the VC should view a square grid of dots. With the known coordinates of the dots in the real world, the found coordinates in C and the knowledge that the dots form a regular grid, a least squares fit is used to find the denominator. Yv keeps an unknown scaling factor cos(φ), and if Yu_H and Yv_H are not equal 0, an extra scale factor in X and Y will appear and hence Y will obtain an additional shift. However, these linear transformations in Xv and Yv are of no concern in our additional calibration procedure! After transformation the image will look like Figure 8.

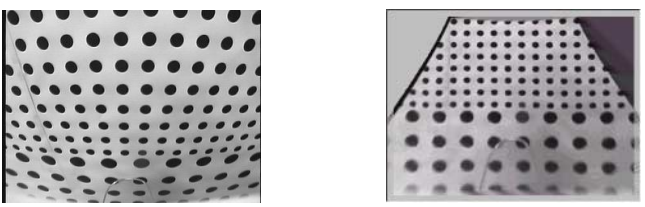

Fig. 8. Effect of calibrating for lens distortion and transformation to VC. The right image has a rectangular grid. It is not square because of the scaling cos(φ) in Yv

With the coordinates in VC (Xv,Yv) we are now able to calibrate the angle.

The formula for tan(φ), taken into account the unknown linear transformation from (X_v,Y_v) to world coordinates in the tilt-calibration, as can be deduced from figure 9, now is:

$$\tan(\varphi) = \frac{X_p}{Z_P - Z_F} = \frac{X_Q}{f} = \frac{a \cdot Xv_Q + b}{f}$$

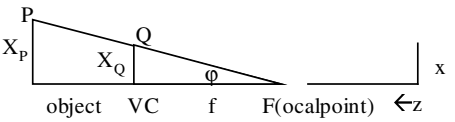

Fig. 9. Using these lines one can find the formulas for tan(φ). f is the focal length. (x,z) are world coordinates

Using the known calibration points, this is again a linear formula, that can be found using a least squares method. Z_F can be found in a distance calibration procedure as described below. However, if $Z_P >> Z_F$ then Z_F can be neglected.

The virtual camera concept can be used very well for the calibration of the distance. The formula that gives the relation between the distance and Yv is almost linear:

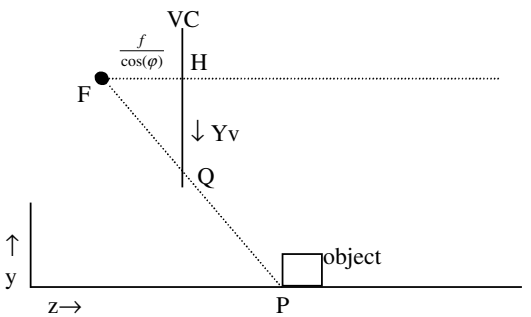

Fig. 10. f is the focal-length of C. (y,z) are world coordinates with a zero point on the floor

$$\frac{Z_P - Z_F}{Z_Q - Z_F} = \frac{Y_F}{a \cdot Yv_Q + b} \quad \Leftrightarrow \quad Z_P = Z_F + \frac{\frac{f}{\cos(\varphi)} \cdot Y_F}{a} \cdot \frac{1}{Yv_Q + b'}$$

This formula is only nonlinear in b'. The calibration with known points is done as follows:

- Choose a b'. Calculate with least squares the best fit. Do the same with b"= b'± h.
- Choose the b" with the least residual squared error, and step in that direction.

Use step halving on h, until the best b' is found. The distance now is:

$$Z = Z_{F,best} + a_{best} \cdot \frac{1}{Yv + b'_{best}}$$

The found $Z_{F,best}$ can be used in the angle calibration.

3 Speed and Accuracy

The Pentium 200MHz with IMAP and WinTV (foreground and background) vision tasks taking 256x240 and 320x240 images respectively:

Image Processing Routine	IMAP-RVS	WinTV (fg)	WinTV (bg)
Undistortion of the lens	6.5 ms		
Color classification of all objects	7.7 ms	10 ms	
Circular Hough Transform for the ball	3.9 ms		50 ms
Find objects (WinTV + lens undistortion)	2.6 ms	6 ms	
Linear Hough Transform for field lines	4.2 ms		50 ms
Find accurate corners and goals	2.1 ms	2 ms	
Total	**27 ms**	**18 ms**	**100 ms**

The Pentium load using respectively WinTV / IMAP is 27% / 3% at 15 / 30 frames/sec.

We measured that at:	1.5 m	3 m	4.5 m
The angular error at view angle 0 ° is:	0.0 °	0.6 °	1.0 °
The angular error at view angle ± 20 °is:	-0.3 °	0.0 °	0.5 °
The angular error at view angle ± 30 °is:	-0.8 °	0.4 °	-0.1 °

The distance error at	0.25 m	0.5 m	1m	1.5m	2m	2.5m	3m
and view angle 0 ° is:	-2 cm	-2 mm	1 cm	6 cm	-4 cm	-2.5 cm	+6 cm

This work was partially sponsored by the Dutch Foundation for Pure Research NWO.

4 References

[1] http://www.robots.com/nsuperscout.htm

[2] Y. Fujita et. al. 'A 10 GIPS SIMD Processor for PC based Real-Time Vision Applications', Proc. of the IEEE int. Workshop on Computer Architectures for Machine Perception (CAMP'97), pp22-32, Cambridge, MA, USA, 1997

[3] J. Lubbers, R.R. Spaans, E.P.M. Corten, F.C.A. Groen, 'AIACS: A Robotic Soccer Team Using the Priority/Confidence Model', in 'Proceedings Xth Netherlands/Belgium conference on AI', 19 November 1998, p. 127-135.

[4] Roger Y. Tsai "A Versatile Camera Calibration Technique for High-Accuracy 3D Machine Vision Metrology Using Off-the-Shelf TV Cameras and Lenses". IEEE Journal of Robotics and Automation, Vol. RA-3, NO. 4, August 1987

[5] J.Bruce, T. Balch, M. Veloso, "Fast Color Image Segmentation using Commodity Hardware" see: http://parrotfish.coral.cs.cmu.edu/fastvision/

[6] E.R.Davies, "Machine Vision: Theory, Algorihms, Practicalities" Academic Press, 1997. ISBN 01-12-206092-X

[7] O. Faugeras, "Three-Dimensional Computer Vision", The MIT Press, 1996, ISBN 0-262-06158-9