ChipVision2 – A Stereo Vision System for Robots Based on Reconfigurable Hardware

Lars Asplund, Henrik Bergkvist, and Thomas Nordh

Mälardalen University

Abstract. A system utilizing reconfigurable hardware of 1 million gates and two CMOS cameras is used in an image analysis system. The system is a part of a sensor system for a robot, and can deliver data about the robots position as well as relative distances of other objects in real time.

1 Introduction

Many algorithms used in digital signal processing for image analysis require large computational resources. The most commonly approach is to use a DSP (Digital Signal Processor), such as Texas Instruments TMS320C6201 and C6701, Philips TriMedia TM1100 and Analog Devices Sharc ADSP 21160M. These processors are variations of SIMD architectures, and they contain several processing units. By the internal pipelining and by using the processing units in an optimal way quite high throughputs can be achieved. Standard PCs are naturally used for image analysis, but for real-time applications these systems has in the past not been powerful enough.

Reconfigurable hardware has most often been regarded as a means for speeding up standard processors or DSP's. Operating systems implemented in an FPGA as an accelerator can in a Real-Time system guarantee that the system is fully predictable [16].

There have also been attempts to use reconfigurable hardware for Image Processing, [12] and [1].

The current project aims at building a stereo vision system (a vision system using image analysis in configurable hardware – FPGA, Field Programmable Gate Array) in the new robot design, Aros. ChipVision will analyze the output from two digital cameras in order to find objects (football, goal, opponents, lines etc.). The output from ChipVision will be data to the main computer in the robot about distance and angles to found objects and the estimated position and direction of the robot itself.

The design is based on an estimated throughput of 15-30Hz. ChipVision is mounted on a free rotating disc, rotated by a stepper motor. Communication with other parts of the robot is by means of an optical CAN–bus.

2 Overview of the Camera System

The FPGA used in this work is equivalent to one million gates and is mounted on a six layer PCB. On the board there is also 64 MB RAM and 32 MB Flash

A. Bredenfeld et al. (Eds.): RoboCup 2005, LNAI 4020, pp. 359–370, 2006.

EPROM. The two CMOS cameras are from OmniVision Technologies Inc. type OV7610 and are directly coupled to the FPGA. The output from the camera is either RGB or YUV. In the current design only 8 wires are used for data transfer, and the format of data used is RGB. The cameras are used in non-interlaced format and the colour filter pattern is the Bayer-pattern, this implies that the green-value is new for each pixel, but the red and blue are alternating. We treat two pixels as one pixel, and thus giving full information of both Red, Green and Blue. The camera chip also has an I^2C bus. This serial bus uses two wires and it can be used to control the camera. Typical controls are gain and white balance.

The internal hardware design (further on referred to as the program) of the FPGA is stored in the Flash–memory in a so called bit–file. A CPLD (Complex Programmable Logic Device) handles the loading of the program into the FPGA. There is a selector–switch with which one of eight programs can be selected for loading. This selection can also be performed by the micro controller (AVR ATmega16) mounted on the PCB for the disc. One of the eight programs is a program for loading the Flash. Thus, each camera–disc can have seven different programs stored in the Flash, thus seven different algorithms can be selected during run-time. The time to load a new program is less than one second.

In this paper three different programs are described. The first program is using one camera for finding the ball and the other camera for finding white lines. The two cameras have different gain–settings for the individual colours. A stereo matching algorithm is the second program. Detected feature transitions from a row for both cameras are stored in a CAM for fast comparison and matching. The stereo matching program successfully detects and matches feature transitions and depth information, this can be obtained from two camera images in real-time. The third program implements Harris and Stephens combined corner and edge detection algorithm, and it uses both cameras. With a frame–rate of 13Hz corners are outputted from left and right cameras.

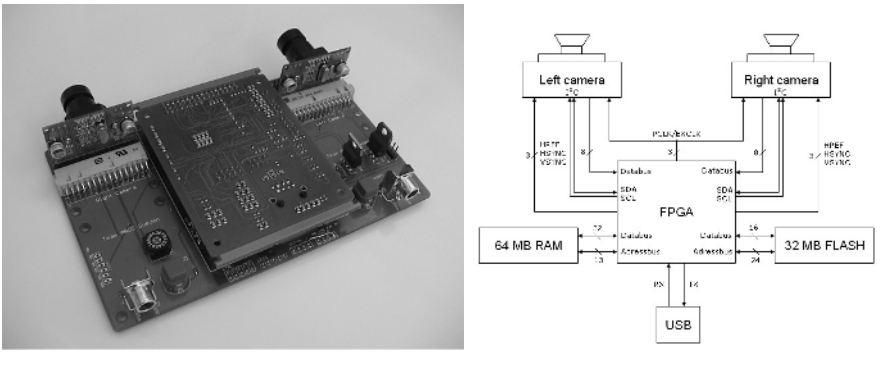

Fig. 1. PCB holding the FPGA-board and the two cameras and the corresponding block diagram

3 Algorithms for Image Analysis

The proposed solution is based on algorithms that are suitable for hardware implementation, i.e. the inherent parallel nature of an FPGA can be fully utilized. In cases where an algorithm is described in a sequential language such as C, there are tools that can convert the C program into a hardware description language such as VHDL [7]. In ref [2] a language called Handel–C has been used. Another way is to use a language that has been developed for the application domain, such as SA–C, a Single Assignment variant of C. In ref [4] this language has been used to implement image analysis algorithms in an FPGA. The performance is in ref [5] compared to the performance of a Pentium processor. The FPGA has in this test been used as a co processor to a rather slow processor. Due to communication the performance in some of the tests are not convincing, but for other tests, the FPGA outperforms the Pentium processor. In ref [2] a true comparison between a high speed DSP (TMS320C44) and a 400k gate FPGA is shown. The number of clock cycles in the FPGA are 64 for a cluster analysis. The corresponding number in the DSP is 16000 cycles. With cycle times of 30 ns for the FPGA and 33 ns for the DSP the time per cluster is 21μs for the FPGA and 5.3 ms for the DSP.

The vision system presented in this paper is aimed to be used in a robot system, and there are numerous industrial applications where vision is an important part. A survey of applications and tools is given in the paper presented by Malamas et al [10].

The size of an FPGA today is far above 1 million gates. This amount of gates allow the full system to be implemented on one FPGA. This can be compared to the ARDOISE system [3], which is a modular system based on smaller FPGA's.

In [9] a hardware setup which resembles our system is described. Instead of using an analogue video source we are using a camera with a digital output. Since our system communicates with the host computer using the CAN–bus, there is no need for a second FPGA. Due to increased performance of circuits, our FPGA and the memory chips have better performance both in speed and number of gates and memory size.

In [8] the ISODATA algorithm is presented, and this algorithm does a mapping from RGB–colour information into classes. The algorithm is used as a first stage and the number of bits in the further processing is reduced by a factor of six. The filter solution in this project is based on a two–dimensional filter, corresponding to the filter used in [11].

For the higher level analysis some version of the object recognition models presented in [6] is used. Still for high performance it is important to have a pipe–line architecture and in the stage after classifying the data there are tables for start and stop pixels for all possible objects.

3.1 Overview of Algorithms

The following steps are the main algorithms of the image analysis. First the RGB representation is transferred into HSI representation. One reason for this

is that the analysis should not be sensitive for differences in light or shading effects. The HSI representation includes a separate variable/channel for intensity I. H is the variable which represents colour tone. H is usually represented by a disc with 0-255 levels . By this representation the colour is described with in one dimension instead of three. From this representation the interesting objects could be thresholded out using their colour values.

Every pixel is given a number indicating their class, depending on their class and the class of the neighbors the pixels are labeled. Noise reduction is performed to minimize the number of pixels with erroneous object classification.

3.2 RGB to HSI Conversion

Mathematical formulas for transformation from RGB to HSI. These formulas take for granted that the RGB values are normalized to [0, 1].

$$I = \frac{R+G+B}{3} \tag{1}$$

$$S = I - \frac{3}{R+G+B}\min(R,G,B) \tag{2}$$

$$H = \arccos\left[\frac{\frac{1}{2}(R-G)+(R-B)}{\sqrt{(R-G)^2+(R-B)(G-B)}}\right] \tag{3}$$

If $B > G$ then $H = 360^o - H$.

For Robocup the Saturation is not required since the colours are well separated in the Hue-circle. Thus only the H channel (and maybe the I channel) are relevant for finding interesting objects. S never needs to be calculated. H is a disc with radius 0-360 and can be represented as 256 gray levels (8 bits).

The equations above are not well suited for implementation in VHDL. The formula 3 is quite complex. Since the resulting H–value is to be represented as an 8–bit value the following algorithm which is a simplification of the Kender's algorithm [17] has proved to have high enough accuracy. In this implementation the full circle is represented between 0 and 252.

Fig. 2. The original pictures from the camera (RGB)

```
Max = MAX (R,G,B);       H'=42*(MID(R,G,B)-MIN(R,G,B))/Max
if R=Max             if G=Max            if B=Max
 if G>B               if R>B              if R>G
   H=H'                 H=84+H'             H=168+H'
 else                 else                else
   H=252-H'             H=84-H'             H=168-H'
```

By sorting the three values of R,G,B the right sector (6 in total) of the colour circle can be found. The range 0 to 255 is not suitable since it can not be divided by 6. The closest value is therefore 0 to 252. Within in each sector the linear expression H' is accurate enough for finding the H-value within the sector.

3.3 Segmentation

The segmentation will take place concurrently as the calculation of H.

1. Is the pixel white? $I > Th_{white}$ gives white
2. Is the pixel black? $I < Th_{black}$ give black
3. Calculate H for pixel N.
4. Segment using multiple thresholds, etc.
   - $x > H$ or $H > X$ Red
   - $y < H < Y$ Orange
   - $z < H < Z$ Green
   - $u < H < U$ Yellow
   - $v < H < V$ Light blue
   - $r < H < R$ Magenta

Each pixel is now represented by its class ID. There are eight different colour classes and one background class. These can be represented by 4 bits. What is left is a picture matrix with 16 gray levels.

3.4 Noise Reduction

This algorithm describes how noise reduction is performed. This step is performed once the segmentation is completed. A median filter sets a pixels class value based on its neighbor's class values. Of this follows that the image will be more homogeneous in terms of object classes. This since erroneous pixels will be adjusted. If this step is performed, over segmentation, which would complicate the classification, will be avoided.

For each pixel, look in a $n \times n$ neighborhood and assign the current pixel the group belonging of which most of its neighbors belong to.

So, given a pixels x and y coordinate, loop around its neighbors and let counters describe the number of pixels belonging to each class. Set the current pixels class belonging to the class which has the largest number of members in the current surrounding.

Fig. 3. Pictures from the right camera after multi segmentation and noise reduction

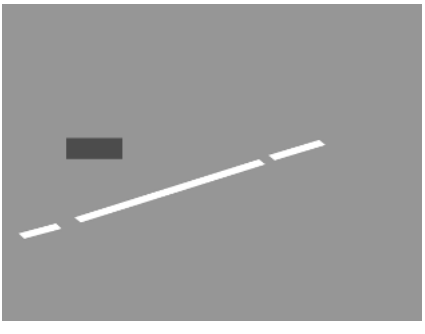

Fig. 4. Final result with found straight lines and ball

3.5 Ball Finder

The Ball Finder is using the output from the noise reduction stage.

When two consecutive red points is found, the min and max x-values are updated and the row is marked as a row with red on it. If red points are found and whose x–values differs more than 50 pixels from previous min and max values they are discarded as noise. When red is found on a row the max y–value is updated, and if red was not found on the previous row the min y–values is updated as well. After the last pixel in the row, the max and min values are checked and if big enough the ball is found. If no red points are found on a row, but was found on the previous row all max and min values are reset. If no ball is found a min–value greater than max is reported.

3.6 Linealyzer – A simple Non-horizontal Straight-Line Finder

When a line segment, a green-white-green edge, has been detected the row, width and center of that segment is stored in a FIFO queue. After a segment is read, check if there is a stored line with a center that differs at most ten pixels, if not, start a new line and save x, row and width of the first line. If the segment is a

part of a previous line, update the center of last line and the number of rows that the line consists of.

When eight consecutive rows have been found, compute the slope of that line part. When sixteen consecutive rows have been found compute the slope of the line part and compare it against the slope of the first part. If the slope is within an error marginal of two pixels the line is considered as a straight line. Mark the line as valid and update the data for the endpoint of the line. Compute the slope of every eight–row block and compare it to the first block. If the slope is within the error marginal update the endpoints, otherwise don't update the line anymore.

When the FIFO is empty, the memory is searched for bad lines, i.e. lines that have been started but not found at least sixteen consecutive rows with the same slope. A line is considered bad when starting row plus number of rows is less than the current row in the picture and is not marked as valid.

All lines start– and endpoints are reported at the beginning of the next frame.

3.7 Stereo Matching

Three FIFO queues (FIFO_l, FIFO_r, FIFO_res) and one content addressable memory, CAM are used in the stereo matching.

First detect all feature transitions for both cameras and store the row, x position and colour in FIFO_l and FIFO_r. When all transitions in a row have been detected, the first eight (or as many that has been found) transitions and positions from FIFO_l are read to memory and the transitions are stored in CAM for fast comparisons.

The first transition of FIFO_r is compared to the transitions in the CAM. If a match is found, the corresponding transitions position is loaded and if the difference in x position is within the predetermined range (150 pixels in our case to detect objects that is about 3 dm away) it is considered a good match. If it is a good match, the transition is stored with the row number, disparity and mean x value of the two images and is stored in FIFO_res for further analysis. If it isn't a good match, the next possible match in the CAM is checked in the same

Fig. 5. Pictures from left and right camera for Stereo matching

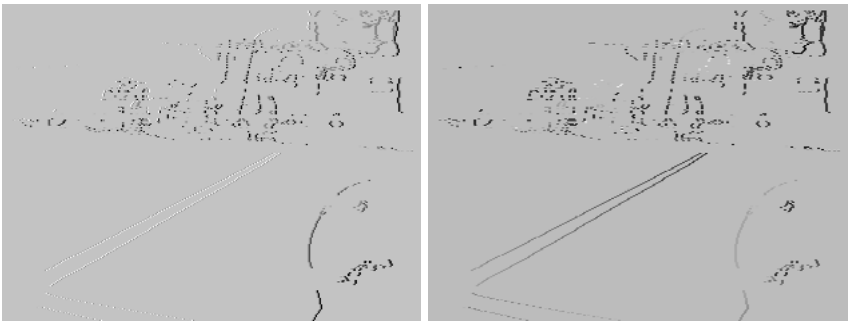

Fig. 6. Left image shows points found in both right and left camera. Right image shows the distance as gray-scale.

way. When a good match is found, the transition at that and lower addresses are shifted out CAM, and the CAM is filled with any transitions that are left at the same row in FIFO_l.

If no good match is found a check for partial matches is started. A partial match could happen when for example, the ball lies close to a line and one camera sees ball-line and the other sees ball-grass-line, or just a mismatch in sensitivity in the cameras.

For a partial match, first the transition from a colour in FIFO_r is searched for and if found, the to colour in FIFO_r is searched. Only if both from and to colours are found any match is considered good. When both from and to colours are found, two transitions is stored in FIFO_res, first the transition that matched the from colour and then the transitions matching the to colour, effectively inferring one extra transition in the original point. If no good match is found the next entry in FIFO_r is analyzed, and if necessary the entries in CAM is updated with new points from FIFO_l.

The pictures in fig (5) are the starting images for the stereo matching. The result is shown in fig (6) and it shows all matched edges to the left and to the right the distances are displayed. White is close and black is far away. The shift to the ball is 72 pixels. The shift to the beginning of the line is 69 pixels and to the end 30 pixels.

There are some errors in the background and in the ball due to the differences in colours and lighting in the pictures

3.8 Harris and Stephens Combined Corner and Edge Detector

For Robocup 2005 a new algorithm has been implemented in the vision system that complement existing functionality. The widely–used Harris and Stephens combined corner and edge detector [14] has been implemented in hardware. It's based on a local autocorrelation function and it performs very well on natural images. The hardware implementation in this paper takes as input RGB-signals from the two synchronized OV7610 cameras in real–time. Pixels whose strength is above an experimental threshold are chosen as visual features. The purpose is

to extract the image feature in a sequence of images taken by the two cameras. The obtained data is detected feature–points on each image and can be future analyzed by for example, feature matching algorithms, stereo vision algorithms and visual odometry.

Harris and Stephens combined corner and edge uses small local window w to analyze the image I given by the mathematical formula 4. Note that the small local window only traverses the image with small shifts:

$$E(x, y) = Ax^2 + 2Cxy + By^2 \tag{4}$$

where:

$$A = X^2 \otimes w, B = Y^2 \otimes w, C = (XY) \otimes w \tag{5}$$

where X and Y are approximated by:

$$X = I \otimes \begin{bmatrix} -1\ 0\ 1 \\ -1\ 0\ 1 \\ -1\ 0\ 1 \end{bmatrix} = \partial I/\partial x \tag{6}$$

$$Y = I \otimes \begin{bmatrix} -1 & -1 & -1 \\ 0 & 0 & 0 \\ 1 & 1 & 1 \end{bmatrix} = \partial I/\partial y \tag{7}$$

The response to this function is noisy caused by the rectangle window, so we introduced as Harris and Stephens declared a smooth circular Gaussian window equ (8) instead:

$$w_{u,v} = e^{-(u^2+v^2)/2\sigma^2} \tag{8}$$

With a lot of potential corners detected from the function described above, we then apply the experimental threshold to sort out the true corners from the less distinctive ones. Finally a corner filter has been applied by only storing the most distinctive corners within a 3x3 pixel window sliding over the final feature set. This filter eliminates all corners that are to close to each other and will reduce the amount of data to analyze in the next step.

To implement this algorithm on the 1 million gates FPGA used on the vision system the VHDL language has been used. But to parallelize Harris and Stephens combined corner and edge detector; the system was first implemented in Matlab to test different approaches, local window sizes and thresholds. In Matlab real images from the vision system could be analyzed and also detected corners could be plotted on the analyzed images for comparison.

To gain real-time speed of the system the algorithm was designed as a pipeline, so each step execute in parallel. This means that it takes some cycles for the first two pixels to traverse the entire pipeline, but when the first two pixels has been analyzed the next two pixels will come the immediately at the end of next cycle. So when the system is running two pixels will be analyzed every cycle, note that one cycle is not the same as one tick on the system clock.

Three different window generators are used; the first one is before the calculation with derivative masks, next is before the factorization stage and the last one is before the comparison stage. Each window generator store values incoming from the stage before and will not generate any output until the entire window is full. When the window is full the values within it will be transferred to the next stage and the window will be shifted. The size of the last two window generator gave us a satisfied result and was decided when the algorithm was tested in Matlab. The size of the first window generator was not optional because it must have the same size as the derivative mask which is 3–by–3.

In general we can say that the first real stage of the pipeline is to calculate the pixels within the first small window with the derivative masks described by formula 6 and 7 above. The next step in the pipeline is to factorize (5) and apply the Gaussian filter (8). The stage after that is to calculate formula 4 which will give us a corner strength value. And the last stage is to compare corners with the experimental threshold and filter our corners that are to close to each other on the image.

When testing and design of a parallel version of Harris and Stephens corner and edge detector was complete the system was implemented in VHDL. When each stage of the pipeline was implemented the stage was tested in ModelSim and the result was always compared and verified with Matlab as reference.

When the entire algorithm was completely implemented, the system was tested in reality. To test it an image capture and corner plotting program was implemented and the result can be seen in Figure 7).

From the resulting images we can clearly see that many of the corners that have been found in both images are the same corners. This will facilitate for feature matching algorithms to match pair corners from both images. When corners in both images have been pair, stereo vision algorithm can be applied and visual odometry can be obtained from a sequence of paired features. More research will be done on these subjects in the near future.

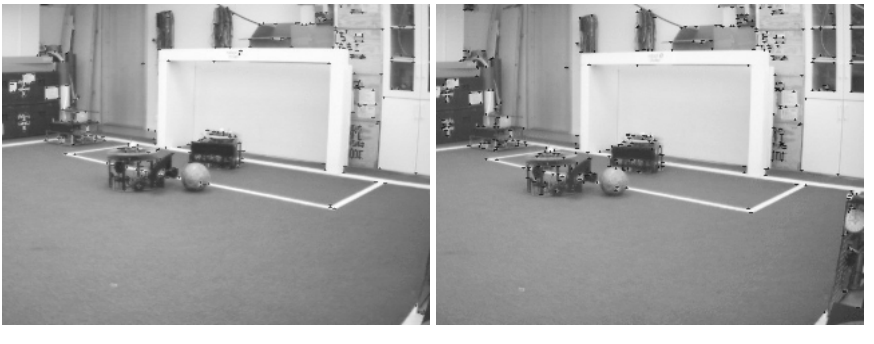

Fig. 7. The corner detection result of one of the first real tests of the system. These two images is a pair out of a sequence of images analyzed by the hardware implemented algorithm.

The frame rate obtained when the system is running was approximately 13.7 frames per second on our 50 MHz FPGA.

4 Results and Discussion

For any kind of robot the sensor system is crucial for its observation of the environment. Of various sensors used in Robocup, vision is the most powerful. The main way vision is implemented today is on ordinary computers. Although a modern PC has very high performance there is always a trade–off between frame–rate and resolution.

The results from this study shows that; by using an FPGA with only 1 million gates, it is possible to achieve a frame–rate of 13Hz on a stereo–camera setup, where all corners are detected in real–time. There are plans of implementing the stereo matching in the FPGA as well as a variant of the ICP–algorithm, which can be used for both localization and movement of the robot. A modified version of the ICP–algorithm (Iterative Closest Point)[15], can also be used to monitor the movements of other robots as well as the ball.

The limitation of frame–rate to 13Hz is due to the number of hardware multipliers, and the clock–frequency of the FPGA. By increasing the number of multipliers the frame–rate can be turned up to the maximal 25Hz, which is then limited by the frame–rate of the cameras.

Acknowledgments

A special thanks to Xilinx Inc., for their support in terms of circuits.

References

[1] K Benkrid, D Crookes, J Smith and A Benkrid, "High Level Programming for Real Time FPGA Based Video Processing", IEEE Symposium on Field-Programmable Custom Computing Machines, FCCM2001, April 2001.

[2] P M Curry, F Morgan and L Kilmartin, "Xilinx FPGA Implementation of an Image Classifier for Object Detection Applications", IEEE Signal Processing Society, International conference on image processing, 2001, pp. 346-349.

[3] D Demigny, L Kessal, R Bourgiba, N Boudouani, "How to Use High Speed Reconfigurable FPGA for Real Time Image Processing", Fifth IEEE International Workshop on Computer Architectures for Machine Perception (CAMP'00), 2000.

[4] B. Draper, W. Najjar, W. Bøhm, J. Hammes, R. Rinker, C. Ross, J. Bins, "Compiling an Optimizing Image Processing Algorithms for FPGAs", IEEE International Workshop on Computer Architecture for Machine Perception (CAMP). Padova, Italy, Sept. 11-13, 2000.

[5] B.A. Draper,W Bøhm, J Hammes, W Najjar, R Beveridge, C Ross, M Chawathe, M Desai and J Bins, "Compiling SA-C Programs to FPGAs: Performance Results", International Conference on Vision Systems, Vancouver, July 7-8, 2001. p. 220-235.

[6] S Herrmann, H Mooshofer, H Dietrich and W Stechele, "A Video Segmentation Algorithm for Hierarchical Object Representation and Its Implementations", IEEE Trans on Circuits and Systems for Video Technology, Vol. 9, No. 8, Dec. 1999.
[7] M Jacomet, J Goette, J Breitenstein and M Hager, "On a Development for Real-Time Information Processing in System-on-Chip Solutions", University of Applied Sciences Berne, Biel School of Engineering and Architecture, Technical Report.
[8] M Leeser, N Kitayeva and J Chrisman, "Spatial and Color Clustering on an FPGA-based Computer System", Proc. SPIE, Vol. 3526, Nov. 1998, pp. 25-33.
[9] W Luk, P Andreou, A Derbyshire, D Dupont-De Dinechin, J Rice, N Shiraz and D Siganos, "A Reconfigurable Engine for Real-Time Video Processing", FPL, Springer Lecture Notes in Computer Science, 1998, pp. 169-178.
[10] E.N. Malamas, E.G.M. Petrakis, M Zervakis, L Petit and J.D. Legat, "A Survey on Industrial Vision Systems", Applications and Tools, Technical Report.
[11] K.J. Palaniswamy, M.E. Rizkalla, A.C. Sihna, M. El-Sharkawy and P Salama, "VHDL Implementation of a 2-D median filter", IEEE, 1999.
[12] L Petit and J.D. Legat, "Hardware Techniques for the Real-Time Implementation of Image Processing Algorithms", Microelectronics Laboratory, University Catholique de Louvian, Belgium.
[13] L Larson, "An EPLD Based Transient Recorder for Simulation of Video Signal Processing Devices in a VHDL Environment Close to System Level Conditions". In Proceedings of the Sixth International Workshop on Field Programmable Logic and Applications, volume 1142, pages 371-375. Darmstadt, Sept. 1996. LCNS
[14] C. Harris and M. Stephens, "A combined corner and edge detector", In M. M. Matthews, editor, Proceedings of the 4th ALVEY vision conference, September 1988, pp. 147-151.
[15] P.J. Besl and N.D. McKay, "A Metod for Registration of 3-D Shapes", IEEE Trans. on Pattern Analysis and Machine Intelligence. vol. 14, no. 2, February 1992, pp. 239-256.
[16] L. Lindh, "FASTHARD - A Fast Time Deterministic Hardware Based Real-Time Kernel", IEEE press, Real-Time Workshop, Athens, January, 1992.
[17] J. Kender, "Saturation, Hue and Normalized Color", Carnegie-Mellon University, Computer Science Dept., Pittsburgh PA, 1976.