Acceleration of Blob Detection Within Images in Hardware

by

Alexander Bochem,
Rainer Herpers and Kenneth B. Kent

TR 09-197, December 15, 2009

Faculty of Computer Science
University of New Brunswick
Fredericton, NB, E3B 5A3
Canada

Phone: (506) 453-4566
Fax: (506) 453-3566
Email: fcs@unb.ca
http://www.cs.unb.ca

Contents

List of Figures

Abstract

This report presents the implementation and evaluation of a computer vision task on a Field Programmable Gate Array (FPGA). As an experimental approach for an application-specific image-processing problem it provides reliable results to measure gained performance and precision compared with similar solutions on General Purpose Processor (GPP) architectures.

The project addresses the problem of detecting Binary Large OBjects (BLOBs) in a continuous video stream. For this problem a number of different solutions exist. But most of these are realized on GPP platforms, where resolution and processing speed define the performance barrier. With the opportunity of parallelization and performance abilities like in hardware, the application of FPGAs become interesting. This work belongs to the MI6 project from the Computer Vision research group of the University of Applied Sciences Bonn-Rhein-Sieg. It address the detection of the users position and orientation in relation to the virtual environment in an Immersion Square[1]*.*

The goal is to develop a light emitting device, that points from the user towards the point of interest on the projection screen. The projected light dots are used to represent the user in the virtual environment. By detecting the light dots with video cameras, the idea is to interface the position and orientation of the relative position of the user interface. Fort that the laser dots need to be arranged in a unique pattern, which requires at least five points.[29] For a reliable estimation a robust computation of the BLOB's center-points is necessary.

This project has covered the development of a BLOB detection system on a FPGA platform. It detects binary spatially extended objects in a continuous video stream and computes their center points. The results are displayed to the user and where validated for their ground truth. The evaluation compares precision and performance gain against similar approaches on GPP platforms.

[1] http://www.cv-lab.inf.fh-brs.de

1 Introduction

Computer Vision problems in general require the processing of a large amount of data for specific application areas. The data to be processed shows similarities since most of the time vision tasks are reduced to a specific application environment with more or less stable conditions. Detection of humans or human faces was a problem where much effort has been invested in the past.[7, 12, 25, 21] Although the invented approaches showed lots of improvement, a common problem in computer vision is still the trade-off between precision and performance.[26] It has been shown that the evolution of computer architecture reached a physical border. The strategy of scaling down semiconductor elements and increasing the number of circuits is no longer profitable. Therefore one way to increase processing power is done by parallelization and application-specific processing devices. The Graphic Processing Unit (GPU), which was originally designed for simple monitor output, nowadays performs most of the image processing tasks on computers. Their architecture is designed to solve various computations that are required in graphics processing.

The idea of parallelization to increase the processing performance is not new and has been realized in many different ways. Some approaches used to build up clusters of homogeneous computers for balancing the work load.[1, 22, 9] Another solution, which is applied in many mainframe computer architectures, is vector processing. These units are specialized to perform the same operation on a mass of similar data in parallel. The downside of parallelization in software is the overhead that is required for synchronization, memory management, and deadlock avoidance. Realizing the same functionality in hardware requires less overhead while performing faster.

More performance gains can be achieved if the algorithms are realized in hardware also known as ASIC (Application Specific Integrated Circuit). ASICs are used for high-customized solutions to process computationally-intensive problems. Their disadvantage is that once the circuit is created, it can not be changed afterwards. Here is where FPGAs can go one step further. These chips contain many free configurable logic units. The program, which is typically implemented in a Hardware Description Language like VHDL or Verilog, configures the logic units of the FPGA to implement the algorithm. A special characteristic is that the computation will be as fast as it would be designed on a ASIC chip.

This characteristic and the fact that the prices for FPGAs are still decreasing are strong reasons to use them for application specific solutions. The vendors of FPGAs provide their own development environments. But if a developer prefers to work with high-level languages, such as Handel-C, or wants to use different simulation tools, it is possible to extend these development

environments with third-vendor products. Handel-C is a C-based programming language, which will be translated into Verilog or VHDL by special compilers.

The presented work in this report is motivated by a project named *6 Degree-of-Freedom Multi-User Interaction-Device for 3D-Projection Environments* (MI6) of the Computer Vision research group from Bonn-Rhein-Sieg University of Applied Sciences. Their work is targeting the development of a passive detection system to improve human-machine interfaces (HMI) in virtual environments.

Nowadays the interaction with virtual environments works with standard-input devices, such as keyboards or mice. But a computer mouse only has a relative position on the desk in relation to its position on the screen. It can not serve as a representative for the user's position in virtual environments, because the mouse does not provide information about its absolute position. In the common use-case the mouse sends data which describes its movement in X- and Y-direction. It is probably possible to use the mouse wheel to navigate on the Z-axis, but this does not solve the issue about the absolute position.

With a device that does not require the user to navigate by active manipulation the perception of the virtual environment by the user could become more realistic. The user could manipulate the ambiance without worrying about the handling of any input device.

This is where the detection of the position of the interactive device and its orientation becomes interesting. It would allow the user to interact with the environment without the training phase for teaching him to handle a specific 3D-input device. The representation of the system would change dependent on the motions of the user carrying the 3D-input device. If he changes his orientation to another point of interest, the system could just move this point to the center of the user's field of vision. Another benefit is the integration of the user into the virtual environment. This could be used for improving automated teaching and testing situations, such as those required in bicycling-driving tests for children in Germany.

One problem is that the detection requires a fast computation but also a very precise one. If the calculation of position and orientation takes too long, the system will become locked up. In the current project the detection of the user's position and orientation, in relation to the virtual environment he is looking at, is addressed.

But a challenging constraint is that the detection system has to be passive, which means that

Figure 1. Immersion Square

it will not detect the user itself. Instead the application of a light emitting device, which points from the user towards the point of interest to the projection screen, is a possible solution.[18, 29] By detecting the light dots the idea is to determine the position and orientation of the user, based on the position of the detected light dots on the projection screens. To achieve an appropriate usability for the user, the processing has to fulfill real-time conditions. A change in the position or orientation should change the representation of the virtual environment without a distracting delay for the user. In addition, the applied light sources for the pointing device will work in infrared range, to not distract the user by light dots on the projection screen. This allows the application of infrared cameras which track only the light dots on the projection screen. These cameras create greyscale image material which will be used for the BLOB detection.

This project covers the development of an FPGA based detection system for BLOBs in a continuous video stream and the computation of their center points. The system has to perform these processes in real-time with reliable precision. To estimate the ground-truth of the center points

a representative amount of digitized video material containing sets of BLOBs will be used as test data. The BLOB detection system will be compared against conventional software approaches in terms of performance and scalability. The development of a light emitting device and the required algorithms, for the computation of user's position and orientation, are goals of the MI6 project and not addressed in this work.

2 State of the Art

A Field Programmable Gate Array contains blocks of logic gates which are freely configurable. While satisfying the desire of application-specific solutions, their performance is similar to hardware circuits. Strictly speaking, by implementing a program with a preferred Hardware Description Language it becomes a hardware circuit.

FPGAs have become an attractive opportunity for various kinds of automated processing. During the past, the number of Logic Transistor Units (LTU) on the chip is increasing while the purchase costs are going down. It has been shown in [10] that FPGA architectures are well qualified for image processing tasks.

Real-time image processing requires the fast processing of large amounts of similar data. The customized implementation of a algorithm on FPGA usually performs better than the same algorithm on a standard x86 architecture. Metaphorically speaking, the algorithm becomes a hardware circuit and every computation step is performed in an independent module of logic gates. In addition, the ability to parallelize the execution of the algorithm makes the application of FPGAs even more interesting, especially for computer vision tasks[20].

Another fact that affects the rising interest of FPGAs is the increasing number of tools for design, development and testing of such architectures and that they have become much more usable in the last few years. Together with the cumulative community of developers there are good reasons for the increasing usage of FPGAs[8].

One of the first approaches in the field of computer vision, where FPGAs have been applied, was image convolution[4]. Other experimental solutions addressed the implementation of specific image processing algorithms[2], distortion correction of raw camera data[23, 24] or application problems like vision systems for mobile robots[6]. They showed that the technology does fulfill real time constraints while also staying attractive for low-cost solutions. In [28, 27] a BLOB-detection system has been established on an FPGA architecture to detect multiple BLOBs in images. But they stated that problems of imprecision due to hardware inaccuracy's were not yet solved. Also they did not take advantage of the parallelization opportunities of FPGAs.

BLOB detection is a corner stone for object detection and recognition. Much effort has been put into the exploration of new algorithms for BLOB detection, which allow to choose from a wide range of methods. But there still seem to be room for improvement[11].

A critical aspect of BLOB detection is the precision of detected pixels at the BLOB's border. Usually a BLOB shows a descent gradient of intensity at the border. In threshold based detection this causes the detection of false-positives and lead to imprecise results if the image material contains to much noise. The number of identified pixels, which belong to the BLOB, estimate the BLOB's characteristics. A possible solution is the parallel processing with detection procedures, in different image resolutions[16]. But this require to make multiple copies of the image data and transform it into different image resolutions. In addition the results for the multiple copies have to be merged to one general result. Other approaches use fixed parameters for the detection of BLOBs. For example the reduction of the application environment to a fixed background[17] to apply foreground-background segmentation. These methods are vulnerable for changing parameters, like illumination conditions or changing perspectives. For the desired system the foreground-background segmentation is not a useful method. The synchronization of the original image with the tracked image from the projection surface would require a index or marker that is visible on the image. That again is an undesired manipulation of the image material, which could distract the user.

A precise computation of the BLOB's center-point is very dependent on the precision of the BLOB detection. Especially for BLOBs that are not perfectly circular- or squarely-shaped, the detection and computation methods need to be exact. Precision is an important factor because with only a few number of false-positive pixels the computed center point gets shifted. This will cause a large error in the computations for the position and orientation of the light emitting device.

A common method to compute center-points of BLOBs is a bounding box which refers to the minimum and maximum positions in the XY-coordinate system. The method of inner circle creates a circle of maximum size that fits into the BLOB area without intersecting the area around it. Both methods do not solve the problem of precision, in reference to not perfectly circular- or squarely-shaped BLOBs (Fig.2). A very common method is center-of-mass that refers to the number of pixels in relation to the coordinates of the pixels[11]. It computes the center of the BLOB based on the number of pixels and weights them by a related value, for example the brightness of the pixel.

It is possible to increase precision by applying a higher resolution, but this is the point where GPP architectures reach their performance barrier. This is a big problem of computer vision. With more data to analyze, the maximum frame rate goes down and the system is not able to

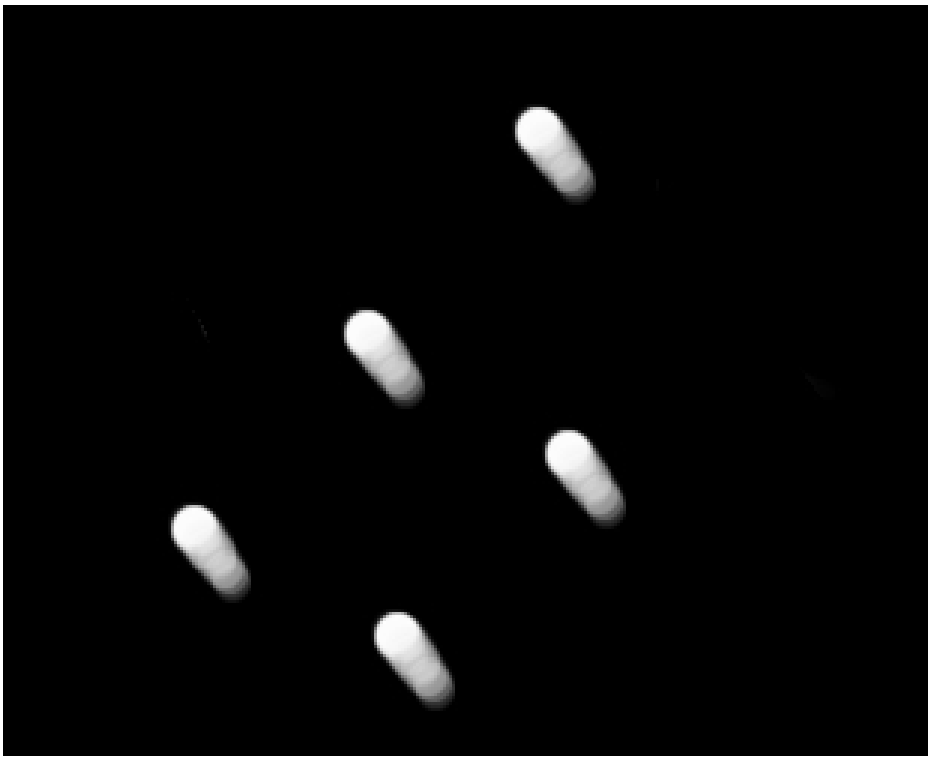

Figure 2. Example of a BLOB image containing blur

achieve real-time processing speed any longer. In [10] it is shown that FPGA architectures have the abitily to work around this problem. It does not solve the performance versus the amount of data balancing issue, but it can process common algorithms much faster and therefore places the barrier much higher. Another aspect is the parallelization of algorithms. This opportunity is not exclusive only for FPGAs or hardware in general, but a software parallelization like OpenMP, MPI or CORBA always creates more overhead for synchronization, dead lock prevention, and resource management.

The disadvantage of working with FPGA architectures is the required effort for the fine grained design. It has to be taken care of the missing abstraction which comes with high-level languages like C++ or JAVA. And parallelizing just any possible image-processing algorithm will not necessarily lead to better performance. It has to have meaningful parts, which can be processed in parallel.

3 BLOB Detection

The aim of the BLOB detection for our requirements is to determine the center point of the BLOBs in the current frame encoded in XY-coordinates. In the proposed project, a BLOB consists of white and light gray pixels while the background pixels are black. This results from the fact that, the cameras which track the laser dots work in the infrared range and do not recognize image scenes from the projection of the immersion square. This is also a reason for not using foreground-background segmentation in the system. For being independent from the camera technology of the application environment sample images have been created. With artificial modeled input data like in Figure 3 the system has been tested for functionality and the precision of center point computation. The number of BLOBs in the video frames can vary, which complicates the

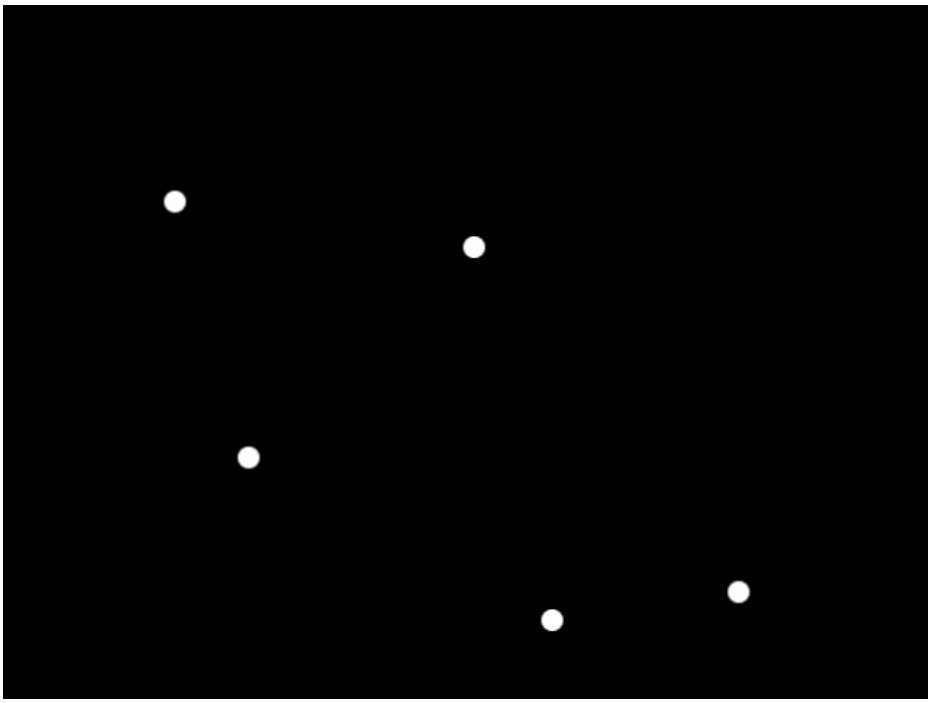

Figure 3. Example of a BLOB image

conditions for the detection approach. This problem is caused by the construction of the application environment. With the three-sided cubicle as a projection surface, the user can change his orientation to any of the screens. If the user faces a point at the corner of the cubicle the light dots will be distributed on two different video frames. This requires the processing of multiple video streams in parallel, as another demand for the BLOB detection system.

To simplify the problem, an upper bound for the number of BLOBs to detect has been defined. Two simple constraints are sufficient to decide if a pixel belongs to a BLOB:

1) Is the brightness value of the pixel above a specified threshold?

2) Is the pixel adjacent to pixels of a detected BLOB?

The threshold is represented as a natural number value. It can be configured by the user or computed by averaging arbitrary attribute values of all pixels in the current frame. Usually the color value or the brightness value is used for this estimation process. The averaging requires the application of a frame buffer to allow multiple processing of the same frame or a continuous adjustment of the threshold, while performing the BLOB detection with some initial threshold value.

To combine detected pixels to BLOBs a test of pixel adjacency needs to be performed. One way is to check every single pixel on the frame for its adjacency to pixels which are already detected and assigned to a BLOB. For the adjacency it is common to distinguish between a four pixel neighborhood and an eight pixel neighborhood, illustrated in Figure 4. The image shows two representative outtakes of pixel matrices containing either white or black pixels. Adjacent pixels are labeled with the same index number and specify the object which they do belong to. In the left image four BLOBs have been detected. The adjacency condition of four pixel neighborhood checks for the horizontal and the vertical axis in the image. On the right image the same pixels are detected in two BLOBs. In the eight pixel neighborhood pixels that are adjacent on the diagonal axis are part of the same BLOB. With this adjacency condition, it is necessary evaluate all pixels

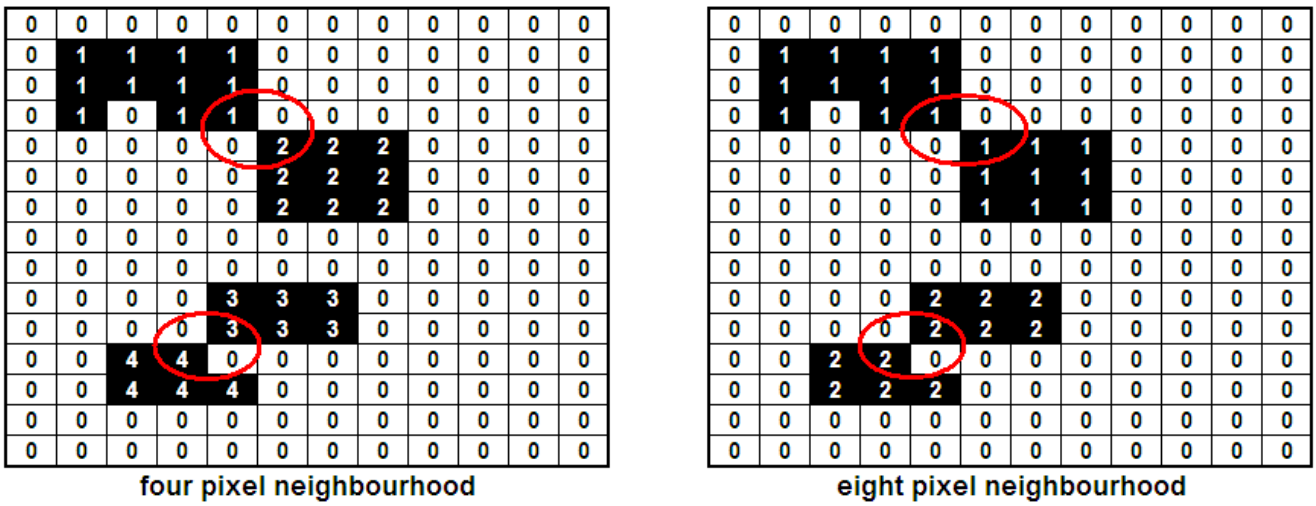

Figure 4. Four and Eight Pixel Neighborhood

of the frame, which are detected. Not only the adjacency of pixels need to be proofed during

the detection. Having BLOBs with uncommon shapes like in Figure 5 the adjacency proof for detected BLOBs need to be performed as well. A sequential scanning process would detect the upper part of the object in the sample image as two independent BLOBs. The number of steps

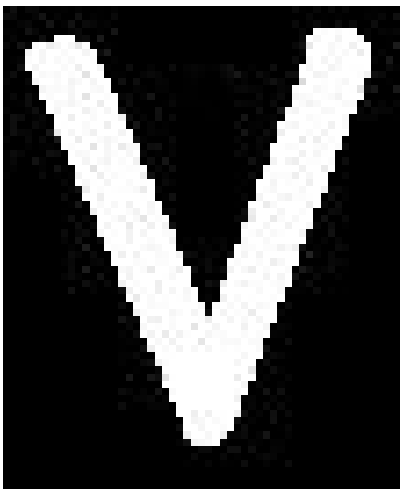

Figure 5. Example of blob shape requires late merge

for estimating the adjacency to BLOBs can be reduced. A data compression method to allow this is the run-length-encoding (RLE)[28]. RLE based detection searches for runs in the single line of a frame. A run contains the coordinates of the first pixel above the threshold in a line of the frame and the number of all of the following pixels in the line adjacent to it, which are above the threshold as well. A frame line can contain multiple runs, for example if two BLOBs intersect on their Y-axis and differ on X-axis.(Fig.6) Each BLOB consists of two runs at least. The maximum number of runs in one frame is restricted to its resolution, since the concept of RLE is based on the number of image lines. Considering a minimum distance of one pixel between two runs and a minimum length of two pixels for a run, the number of runs for one image line is not greater than N.

$$N = \left[\left(\frac{horizontal\ resolution}{2} \right) - 1 \right]$$

The definition of the minimum-length and -distance for runs is defined in the design of the system. The detection of runs is very similar to the pixel based detection of blobs. It scans a single line of a frame for pixel values above a specified threshold. If a pixel is detected its coordinate is stored as starting position of a run. From there on the algorithm adds all adjacent pixels in the same line, which are above the threshold, to the run, by increasing a counter for the length of it. This procedure need to be performed for all image lines. The RLE-based BLOB-detection has the advantage that, the detection of runs can be done in parallel.

By extending the detection of pixels to detection of runs, the proof of adjacency for the blob detection later on can stop if the neighborhood condition of one pixel in the current run and the corresponding BLOB is fulfilled. Also, it is not necessary to store coordinates of all pixels of the current frame, which allows to save memory during execution.

The ability to parallelize part of the detection procedure is a feature that is perfectly qualified for

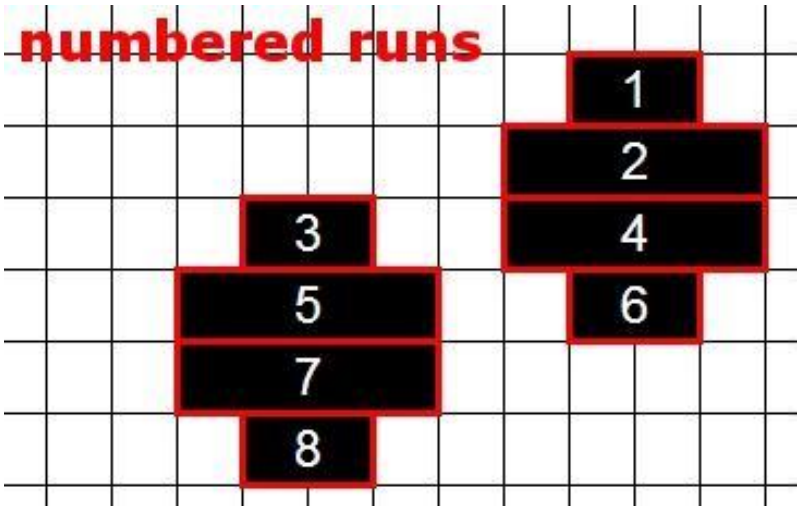

Figure 6. Run Length Encoding based BLOB Detection

FPGA architectures. FPGAs have some constraints due to their support of basic programming theorems. The BLOB detection has to work without recursion, because it can not be implemented on FPGA architectures. Necessary features for context switching are not offered like in GPP systems.

For the computation of the BLOB's center-point different methods could be applied. Bounding Box measures the center-point of the BLOB by checking for the minimal and maximal XY-coordinates of the BLOB. If a new pixel is adjacent or in the range of the estimated XY coordinates of the BLOB, it will be added to the BLOB and an adjustment of the min-/max-values is performed. For the computation of the center-point the bounding box offers not enough information about the BLOB to extract very precise results. The computation can be implement very efficiently and will not cause big performance issues.

$$\text{BLOB's } X\_center\_position = minX\_position + \frac{maxX\_position \text{ - } minX\_position}{2}$$

$$\text{BLOB's } Y\_center\_position = minY\_position + \frac{maxY\_position - minY\_position}{2}$$

Division by two can be realized by a bit shift and mathematic operations, such as addition and subtraction, are not computationally expensive either.

The center coordinates are strongly affected by the pixels at the BLOB's border. With a threshold based detection the pixels at the border can show flickering. This effect becomes even stronger for BLOBs in motion. With reference to the light emitting device for our virtual environment, the angle of the light beams to the projection surface change the shape of the BLOBs and increase the range of pixels less intensity. The movement of the device by the user will cause motion blur on the BLOB's shape. These effects will increase the flickering of pixels at the BLOB's border and will cause a flickering in the computed center-point. For reducing these flickering effects another method for the computation of the BLOB center point is the center-of-mass method. All pixels of the detected BLOB are taken into account for the computation of the center point. The algorithm applies the coordinate values of the detected pixels as weighted sum and calculates an averaged center coordinate.

$$\text{BLOB's } X\_center\_position = \frac{\sum X\_position \ of \ all \ BLOB \ pixels}{number \ of \ all \ BLOB \ pixels}$$

$$\text{BLOB's } Y\_center\_position = \frac{\sum Y\_position \ of \ all \ BLOB \ pixels}{number \ of \ all \ BLOB \ pixels}$$

To get an even higher precision the brightness values of the pixels could be applied as weights as well, which increases the precision of the center-point of a BLOB.

$$\text{center position} = = \frac{\sum \ pixel \ position * pixel \ brightness}{\sum pixel \ brightness}$$

Since the available video material is converted into greyscale, the border of the BLOBs can show a significant flickering, dependent on the threshold value and the color gradient. To avoid a similar flickering in the computed center-position of the BLOB, the application of the proposed averaging is recommended. This can be realized by applying a running summation of the values during the detection phase. The division by the number of pixels can be done after all merging procedures for the BLOBs has been executed.

The computation of the center point by inner-circle has not been taken into account here. The reason is that it offers the same advantages and disadvantages as a bounding box.

4 Implementation

4.1 Tools & Environment

For the implementation of the BLOB detection system, an Altera DE2 development and education board with a Cyclone II FPGA has been used. It contains 33,215 logic elements (LEs) for high-volume applications with large complexity. It comes along with several interfaces to receive and send data in various representations. Altera provides a comprehensive development environment, the Quartus II Web Edition (Fig.7), which contains a broad set of tools for design, implementation and simulation of FPGA systems.

Altera also offers a library containing components with various kinds of functionalities. These are known as blocks of intellectual property (IP) and can be customized by the developer. But the application of IPs is restricted to licensing conditions. In regular case they cannot be used for commercial products. In the field of FPGA development there are particularly two dominating hardware description languages. Very High Speed Integrated Circuit Hardware Description Language (VHDL) and Verilog HDL[5, 3, 19]. VHDL can be used to design a system in Behavioral, Structural or RTL Dataflow Description or as a combination of all three of them, which we refer to as Mixed Description.

The Behavioral Description defines processes which execute on changing conditions like signals or register values. Processes define concurrent blocks of execution, the decision how to order the execution depends on the connections between them. The Structural Description describes the configuration of entities but not the functionality. It uses logic gates to design general-circuit components. Register-Transfer-Level (RTL) Description is based upon the data-flow of the system. A combination of all three description methods is also possible and specified as Mixed Description.

Verilog was developed at the same time as VHDL, however only meant for simulation of hardware systems. A first standardized version became published as IEEE 1364-1995[13], also known as Verilog-95. The last significant extension in 2001 changed the language abilities to a so-called Hardware Description and Verification Language[14]. Todays standard is described in IEEE 1800 and is formally known as SystemVerilog[15]. The proposed development environment does allow the combination of components written in either one of the hardware description languages. A successful integration by the synthesizer depends on the correct definition of the module's interfaces.

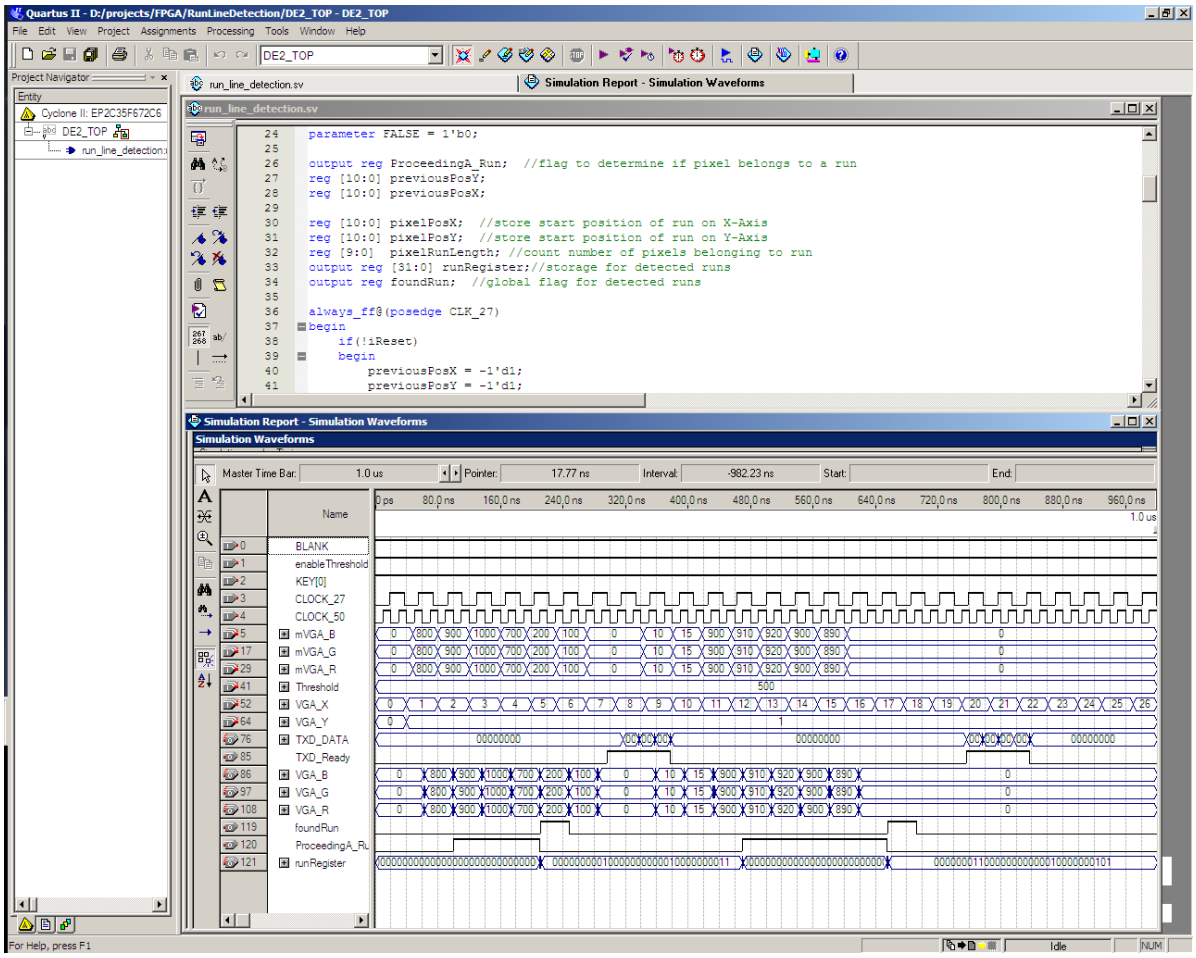

Figure 7. Quartus II Dev. Environment. Code editor in upper window, Design Tool for waveform simulation in lower window

The image material containing the BLOBs was provided either from a standard DVD player or similar media devices, which had an analog composite output interface. For testing and evaluation of the BLOB detection the provided image material was scaled to a 640x480 resolution. This was required since the analog-digital converter of the DE2 board was driven to convert the image material into the same resolution. The test wise execution of the BLOB detection during implementation phase, with different resolution scales, showed a slight error in precision due to the scaling of input images. For continuous validation of the performed manipulation on the image material, the digitized frames have been converted to RGB and displayed on the VGA output. This allows an visual evaluation on the binarized image material and the impact of the threshold value on the precision of the BLOB detection. The digitized video stream is converted pixel wise from one-channel YCbCr format into a three-channel RGB model and transformed to greyscale

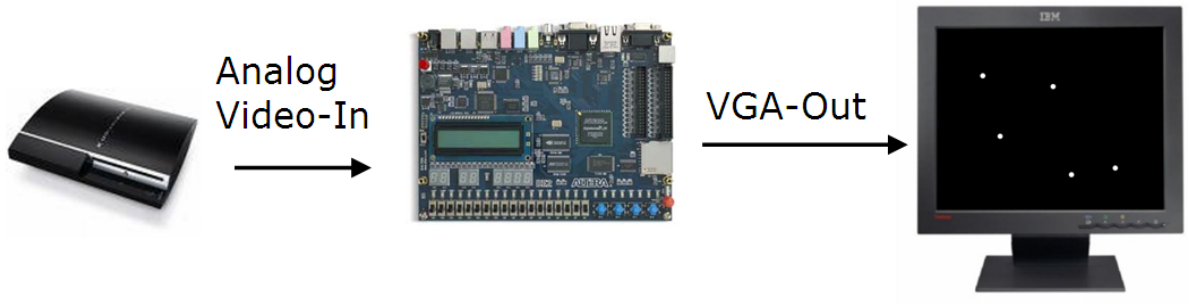

Figure 8. source and destination of image processing

image data. The transformation into greyscale data can be configured on the DE2 board during runtime. So it is either possible to run the BLOB detection with greyscale values or RGB color values. As mentioned earlier the tracked image data from the video cameras is already in greyscale format. This first approach was realized based upon an existing demonstration program which was available from the Altera homepage.[2]

4.2 BLOB Detection

As a first approach the threshold based BLOB-detection for single pixels has been implemented(Fig.9). The number of track-able BLOBs was reduced to one for the first revision. This simplification of the problem-space allows one to forget about to check for adjacency of multiple BLOB parts or the existence of multiple BLOBs in the frame. If a relevant pixel was detected only the adjacency proof with already stored pixels needs to be applied. Every pixel in the current frame is checked for its corresponding brightness value. This value is computed out of the three values of the RGB channels containing the image data. If the pixel's brightness value reaches the defined threshold value it is recognized as BLOB relevant. The detected pixels are added to a data structure containing the attributes of a BLOB, if the pixel is adjacent to the BLOB. With every pixel added the BLOB's coordinates are adjusted. If the end of the frame is reached, the center point of the BLOB is computed and shown on the seven-segment-display.

4.3 Bounding Box

The bounding box method requires four registers for minimum and maximum X- and Y-coordinates for each BLOB. These set of registers from here on are referred to as "container". For detecting multiple BLOBs a sufficient number of container is required. With respect to the mentioned problem of uncommon BLOB shapes the number of container has to be larger than the number of BLOBs to detect. The condition for a pixel belonging to a BLOB is fulfilled if it matches the

[2]http://www.altera.com/support/examples/dsp-builder/exm-color-space-converter.html

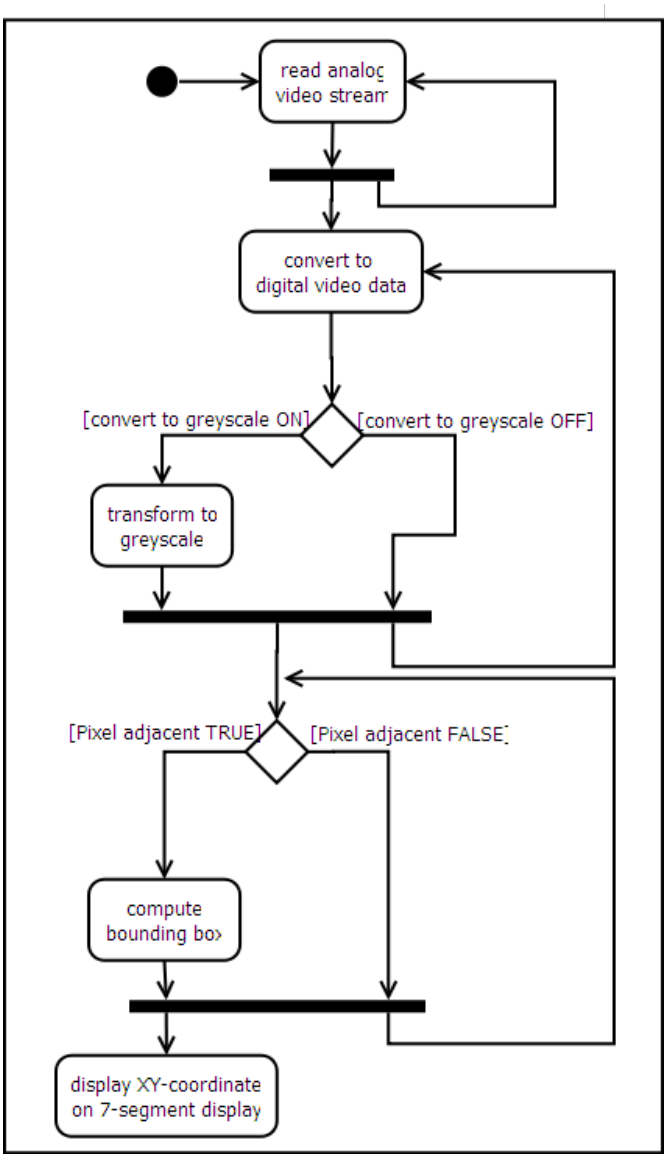

Figure 9. Bounding Box based BLOB Detection

inner region of the bounding box or is adjacent to it. The values for the bounding box of a BLOB are initialized with the first detected pixel.

To estimate the starting point of a frame the pixel XY-coordinates are checked for a tuple of (0,0) coordinate values. This was possible because of the sequential processing of the digitalized video-stream. For the output of the computed center-points the seven-segment-display has been used. The coordinates are represented as hexadecimal values and were computed and displayed for every frame.

The outline of the SystemVerilog code shows the detection of BLOB pixels by the threshold and the manipulation of the BLOB's bounding-box attributes. First the brightness value of the current pixel is checked for the threshold condition. Then the attributes of the BLOB's bounding box are updated. The lower if-block checks if a new frame has started. If that is the case the center point of the blob is computed and transmitted to the display.

Pixel Detection by Threshold

```
.
.
//value > Threshold => pixel belongs to BLOB
if((iVGA_R+iVGA_G+iVGA_B) > threshold)
begin

        //check if pixel belongs to current frame
        if(currentFrame == countFrames)
        begin
                //compute coordinates for bounding box
                if(iPosX < minX) minX = iPosX;
                if(iPosX > maxX) maxX = iPosX;
                if(iPosY < minY) minY = iPosY;
                if(iPosY > maxY) maxY = iPosY;
        end
end // end threshold

//compute difference between minX, minY, maxX, maxY
diffX = ((maxX - minX)>>1); //>>1 == division by 2
diffY = ((maxY - minY)>>1); //>>1 == division by 2
.
.
//send BLOB attributs to output, if new frame starts
if(currentFrame < countFrames)
begin
        othrPosX = (minX + diffX);
        othrPosY = (minY + diffY);
        currentFrame = countFrames;
        maxX = 0;
        minX = 1000;
        maxY = 0;
        minY = 1000;
end
.
```

.

The bounding box based approach has been extended to process multiple video streams in parallel. In the second revision it can process three video streams at the same time. Because of the limited number of composite-video inputs the system uses three times the same video stream. The output of each detection module can be shown on the seven segment display by enabling one of the Switches 14 to 16 on the board. For a processing of three different video streams, the system needs to be extended to support other communication interfaces or using further DE2 boards and connect them together.

In a third revision the detection has been extended to detect up to five BLOBs in one frame. A set of containers was created to store the attributes of the BLOBs. An additional adjacency proof was required to merge BLOB-parts during the detection phase. Dependent on the shape of the BLOBs it can happen that pixels of the same BLOB were split into two different BLOB containers. Therefore the container of each BLOB needs to be checked for adjacency. If that is the case, the attributes of one BLOB are updated while the other container is cleared.

Merge of Pixels into BLOB-Container

.

```
//MERGING
//MERGE CONTAINER 2 TO CONTAINER 1

        //if Container 2 is not empty, check for adjacency
        if(BLOB2empty==FALSE && BLOB1empty==FALSE)
        begin
                //Merge Container 1 and 2 if adjacent
                //check adjacency on Y axis
                if( BLOB2minY >= (BLOB1minY-1) && BLOB2minY <= (BLOB1maxY+1) &&
                        BLOB2minX >= (BLOB1minX-1) && BLOB2minX <= (BLOB1maxX+1) ||
                        //check adjacency on X axis
                        BLOB2maxY >= (BLOB1minY-1) && BLOB2maxY <= (BLOB1maxY+1) &&
                        BLOB2maxX >= (BLOB1minX-1) && BLOB2maxX <= (BLOB1maxX+1))
                        begin
                          //Update attributes of Container 1 if required
                                if(BLOB2maxX > BLOB1maxX) BLOB1maxX = BLOB2maxX;
                                if(BLOB2maxY > BLOB1maxY) BLOB1maxY = BLOB2maxY;
                                if(BLOB2minX < BLOB1minX) BLOB1minX = BLOB2minX;
                                if(BLOB2minY < BLOB1minY) BLOB1minY = BLOB2minY;

                                //clear content of Container 2
```

```
                                BLOB2maxX = 0;
                                BLOB2minX = 1000;
                                BLOB2maxY = 0;
                                BLOB2minY = 1000;
                                BLOB2diffX = 0;
                                BLOB2diffY = 0;
                                BLOB2empty = TRUE;
                        end
        end
        .
```

The disadvantage of the sequential processing of the video-data is that it restricts the scanning of the image lines to be done in sequential. A parallel detection of all lines is not possible without the extension of the system with an image-buffer unit. And the maximum frame-rate is restricted to the performance of the analog-digital-converter of the DE2-board. Even with an image-buffer the system would have to wait for the AD-converter until all the information of a full frame is stored.

4.4 Run-Length-Encoding

With the ability to access the whole data of one frame in parallel the Run-Length-Encoding based detection of the image becomes interesting. The adjacency check for the detected run can be processed after the current frame is scanned(Fig.10). The identified runs will be combined to binary large objects. RLE could be also applied for the sequential processing of the pixels to reduce the number of adjacency proofs during the detection phase. An RLE based detection is not realized in the current system. With the restricted sequential access to the image data it would not bring a great benefit for the detection performance. But even with it, the concept of center-point computation based on the bounding-box faces the problem of less precision than center of mass. The shortcomings in computation of the BLOBs' center-points, which are not perfectly circular- or square-shaped can not be solved with it.

While the center-of-mass computation allows more precise results, it requires more data as well. The mentioned weighting of the BLOB's pixels by their brightness value requires to store the information about the brightness values for the duration of the detection phase.

For evaluation of the achieved precision, the computed coordinates should be transmitted to a connected host PC. Any available communication interface of the DE2-board that provides sufficient bandwidth could be applied. The serial interface can provide up to 38,400 bits/second which will be enough while a BLOB's center-point can be encoded in 24 bit. The system would

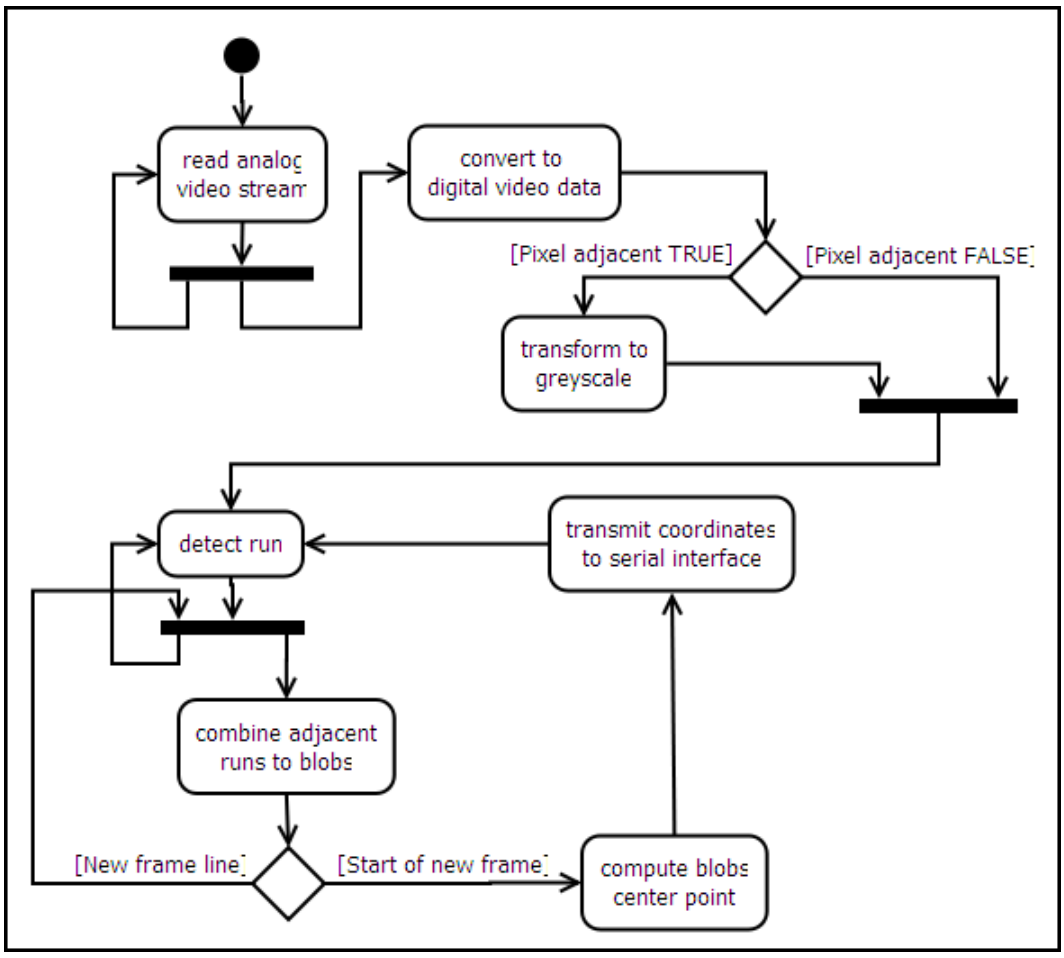

Figure 10. Run-Line-Encoded based BLOB Detection

be able to transmit the computed coordinates to the Host-PC for a processing speed of up to 80 frames per second. Under the condition that a frame does not contain more than 20 BLOBs.

4.5 Problems

At the beginning of the project, the literature search for related material with meaningful content became quite difficult. Even if the quantity of the work about image-processing systems based on FPGAs is quite high, only a few publications cover the topic around the applied solution sufficient enough to understand it in detail. Besides that, the number of persons in the area of the hardware and hardware-describing development is hardly noticeable, compared with the large community's of high-level languages like C++ or JAVA. Therefore, the availability of tutorials and documented program examples is extremely small, except for Altera's support homepage, which offers a considerable collection of sample programs around the functions for the DE2-board to test it. But test benches and waveform files to observe the internal proceedings, while simulating the system,

are not provided. The test programs do not provide any functionality to display internal values on the on-board displays or send it to one of the available interfaces.

At first it was necessary to analyze the available hardware designs by their source code and show selected register contents on the board's displays. This kind of training became particularly time-consuming, since all programs were written in Verilog and the author did not know much about it at the beginning. Also, the assigned tools for implementation and design required a longer training phase than was expected from usual development environments, such as NetBeans or Eclipse.

As a basis for the project, a sample program from Altera was used, that reads an analog video-signal on the video-in interface and converts it to an RGB signal for sending it to the VGA output. The analysis of the source code showed that, the manipulation of the video signal works straight sequential. The program offered a good starting point to apply some experimental changes on the video signal. First, a new hardware module in Verilog has been implemented to proof the pixels for a defined threshold value and to alternatively transfer the video signal into a greyscale format.

It should be made possible to change the threshold value during runtime, by the keys 1 for plus one and 2 for minus one on the board. The implementation of an appropriate routine turned out to be more difficult than expected. The value did not change with each depressing of the keys instead it jumped with different step-sizes between the minimum and maximum value.

As a first assumption it seems logical that the keys cause a jitter in the pin signals since they were not debounced. In order to eliminate the problem an additional register was used to represent a mutex. This should prevent a repeated execution of incrementing or decrementing of the threshold value within consecutive execution cycles. The desired success was missing in this solution. In addition the documentation of the DE2-board said that all keys are debounced with a Schmitt-Trigger, so this could not be the source of the problem.

To eliminate the problem a counting variable was used. This was set to zero by pressing one of the two keys on the board and prevented each further manipulation of the threshold value for the following clock cycles, until a defined number of clock cycles passed. This solution did not lead to the desired success as well. The reason for this could be an insufficiently low value for the number of clock cycles. In order not to lose to much time with the realization of this program feature, the switches 0 to 9 for the manipulation of the threshold value were used instead. These

make it possible to set the value in a range from 0 to 1023.

With the implementation of the BLOB-detection procedure, on the basis of the Run-Length-Encoding, it did come into problems, which could not be solved up to the project's end. The implemented module to recognize runs within frames could not be proofed for correct functionality. The assumptions to the ranges of values of the used input signals for the module were based only on the isolated expenditures of these values on the 7-segment display. Therefore there was no warranty that the internal processing of the analog video signal ran off in the expected sequential order. But since the bounding-box based approach was close to the expected outcome, the assumptions about the internal values should be correct. For the proof of the functionality of the module for detection of runs, a Waveform file (Fig.11) was provided. The defined input signals in the Waveform file reflecting the assumptions of the methods of analysis mentioned earlier. The simulation on the basis of the waveform file showed that the detection of the runs works

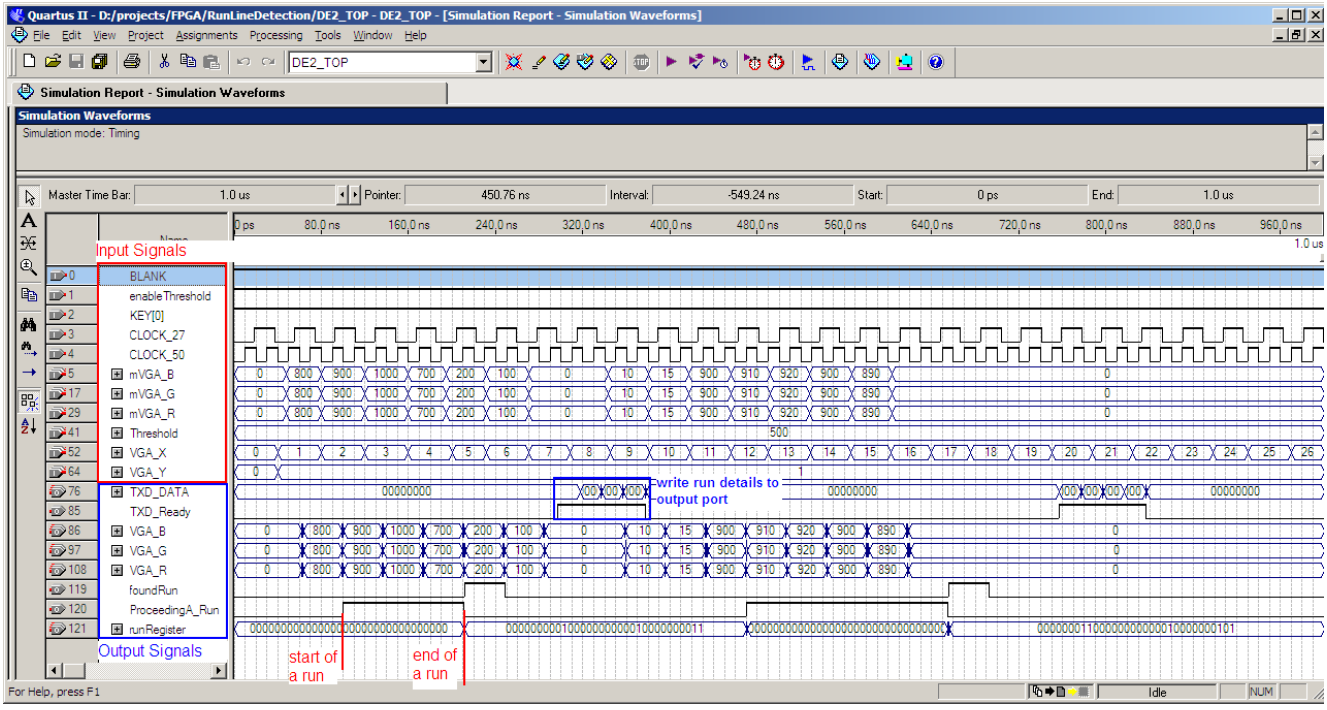

Figure 11. Waveform Simulation Report

correctly. However, simulation proceeds from a perfect switching attitude of the hardware and is not a reliable source to conclude to real-world execution. Signal rise- and fall-times are not considered which again do not allow any conclusion about the internal operational sequence of the synthesized algorithm. It is worth to mention that the created waveform file was reduced to an outtake of the estimated image data. The file represents only a single line with two disjoint

runs to detect. For the definition of a waveform file that simulates the input signals of a whole frame it would probably take about a week to create it. Since it would not necessarily lead to better consolidated findings about the internal proceedings such a file was not created.

In order to make a further evaluation of the module possible, the implementation of a communication interface to a standard PC has been started. This should allow to log the determined coordinates of the run for further processing of the BLOB's attributes later on. For a first proof of concept the serial interface should provide a sufficient bandwidth and offer the simplest techniques for data transmission. With the available 50 MHz input clock of the DE2-board a data transmission-rate of 38,400 bits per second could be achieved. For the transfer of data, concerning a run, not more than 32 bits are necessary. The resolution of the frame in the RGB signal is 640x480 pixel and for a run the maximum length was set to 254 pixel. Thus the information of a run can be transmitted within 32 bits.(Fig.12) To be able to evaluate the data conveyed by

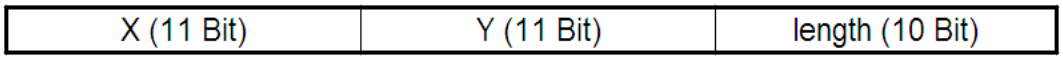

Figure 12. Attributes of a run encoded in 32 bit

the DE2 board an appropriate logging program was implemented, which writes the received bytes down as a bit-stream into a text file. Existing terminal programs of the Host-PC's operating system as well as freely available tools could not fulfill this requirement. Available applications would in most cases apply a conversion of the individual bytes into ASCII-format. There are other programs which place the received bytes in the hexadecimal format instead. With the selected encoding for runs or BLOBs to a 32 bit string, the implementation of an own parser program was necessary. It required a dissimilar splitting of the 32 bits into the XY-coordinates and additional attributes.

The implemented module for the serial communication worked in its first version. There it was supposed to send a simple character string of "Hello World!" to the connected Host-PC. The output could be observed on the hyper-terminal application of Windows XP. For the transmission of the runs data, the routine has been modified. The code listing shows the structure of the implemented buffering procedure to allow the transmission of run's data to the RS232 interface.

It executes with a clock rate of 50 MHz and checks the content of the TXD_Buffer_Flag register. If the bit at position 15 is set to 1 the register TXD_Buffer in the range [127:120] has to be transmitted to the serial interface. Meanwhile, it checks on every clock-tick if the run-detection module has found a new run. If yes, the data is appended to the end of the TXD_Buffer register

and five new flag bits are added to the TXD_Buffer_Flag register. The byte 2'h0A represents the carriage-return of the ASCII-Table and is used to separate the data at the receiving protocol application. **RS232 Transmission Buffer**

```
.
reg  [31:0]  tmpRun;
reg  [127:0] TXD_Buffer;
reg  [15:0]  TXD_Buffer_Flag;

always@(CLK_50) begin
        if(!iReset)
        begin //Initialize Buffer Registers
                TXD_Buffer = 128'b0;
                TXD_Buffer_Flag = 16'b0;
        end

        //If run detection found run, add to Buffer Register
        if((foundRun == TRUE) && (tmpRun != runRegister)) begin
                //copy register for check of duplicate in next iteration
                tmpRun = runRegister;
                //append register to Buffer
                TXD_Buffer = {TXD_Buffer[87:0], runRegister, 2'h0A};
                //set flags to enable writing
                TXD_Buffer_Flag = {TXD_Buffer_Flag, 5'b11111};
        end

        //if flag on MSB position ==1, Buffer Register contains data
        if(TXD_Buffer_Flag[15] == TRUE) begin
                //enable write signal
                oTXD_Start = 1'b1;
                //copy 8 bits to UART Register
                oTXD_DATA = TXD_Buffer[127:120];
        end
        else
        begin
                oTXD_Start = 1'b0; //disable write signal
        end

        //shift flag register by 1 bit
        TXD_Buffer_Flag = {TXD_Buffer_Flag[14:0], 1'b0};
        //shift buffer register by 8 bit
        TXD_Buffer = {TXD_Buffer[119:0],8'b00000000};
end
```

.

The simulation of the serial communication as part of the run-detection module worked properly. (Fig.11) For the transmission of real data it did not. The received data contained information that matches the expected data, but most of the time the number of bits before a carriage return where not 32. There were either more or less bits received. As a first problem, it could not been ruled out that the bit combination of 0000-1010 is a valid combination that could occur in one of the four bytes encoding the run details. Therefore the appended flag to mark the end of a transmission has been changed to 1111-1111. But it did not solve the described problem.

A possible solution to figure out why this problem occurs is to separate the procedure of data transmission from the BLOB-detection module. The transmission of each byte can be interrupted by short transmission breaks to avoid the overflow of the UART buffer on either the sender and/or receiver side. Let us assume that a BLOB contains around 30 runs under the condition that the resolution of the frame is 640x480 pixels. The current system is driven by a 27 MHz clock-rate and has a frame-rate of 88 frames per second on average. If in the provided video stream a frame contains 6 BLOBs the communication interface has to have a bandwidth of 506,880 bit per second.

506,800 bit per second = 30 runs x 88 frames x 6 BLOBs x 32 bits per BLOB

Apparently this is 13 times faster than the serial interface of the current implementation can handle. Till the end of the project phase a serial communication to transmit the detected runs could not been established. But it is also not really required since the computation of the BLOB center-point will be performed on the system as well.

For the transmission of the computed BLOB's center points 32 bits are required for each BLOB.(Fig.13) Instead of the length of a run the index of a BLOB and the frame index will be sent to the connected PC. The frame index is a continuous number that can be used to distinguish BLOB positions in consecutive frames. The BLOB index should relate to the same BLOB in consecutive frames, to allow computations on the user's movement. An appropriate communication between

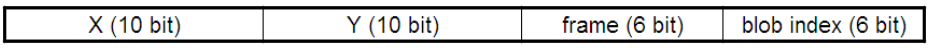

Figure 13. Attributes of a BLOB encoded in 32 bit

the DE2-board and Host-PC could not be established. Since the main target was the detection of BLOBs the solving of the serial-communication problems has been postponed.

As described in the last section the BLOB detection could be implemented for up to five BLOBs

in a single frame. But the missing serial communication did not allow to validate the BLOB detection on a continuous video-stream. The system can perform the detection, but since the computed center-points for the BLOBs are only visible on the seven-segment-display of the DE2-board, it is not possible to extract meaningful results. While performing the BLOB-detection the estimated center-point's positions changes to fast to allow the user to read the values.

Another problem occurred for the precision of the computed center point. The system showed a systematic error of -3 on the X-axis and +4 on the Y-axis. This error could be observed for all BLOBs on different input images. When this problem occurred the first time a simple workaround has been applied to eliminate this error. By shifting the computed center-point with this constant error-values the result was pushed closer to the expected outcome. Later on, it could be observed that the problem initially comes from the scale of the applied input image. The first BLOB images did not have a resolution of exactly 640x480 pixels. Since the applied video-input devices scale the image to a NTSC format with different resolution, this transformation shifts the positions of the BLOBs in relation to the RGB output. After changing the resolution of the input images to 640x480 pixels the systematic error changed but was not eliminated completely. The new error showed a constant shift of the BLOB's center point by +2 on X- and Y-axis.

During the first phase of the project a DVD Player with a USB interface was used to provide an input signal to the analog video-interface of the DE2-board. This input source was no longer available since the project location changed due to the study process of the author. At this time the video material should be provided with a regular DVD Player. This caused some problems because of missing experience in burning DVD's in video format. Even with famous mainstream programs like Nero 9© the creation process failed several times. As a momentary solution an available Sony Play Station 3© System has been applied as input source. A temporary workaround was to provide the input signal from a PC, that was connected to a VGA to Composite-Video converter. But the additional conversion caused a higher error in the computed BLOBs' center points and was useless for the project. The creation of a useful Video-DVD could be successfully completed and is used in the current system's setup.

5 Evaluation

Since the communication on the serial interface to the Host-PC could not be established a useful evaluation of precision of the center-point computation was not possible for continuous video-streams. The validation was therefore reduced to static images like in Figure 3. As mentioned earlier a systematic error in the computed center point could be observed, which was caused by scaling effects from the input source and a missing constant in the computation method. The first evaluation of the system showed a constant precision error(Fig.14). Figure 14 shows the difference

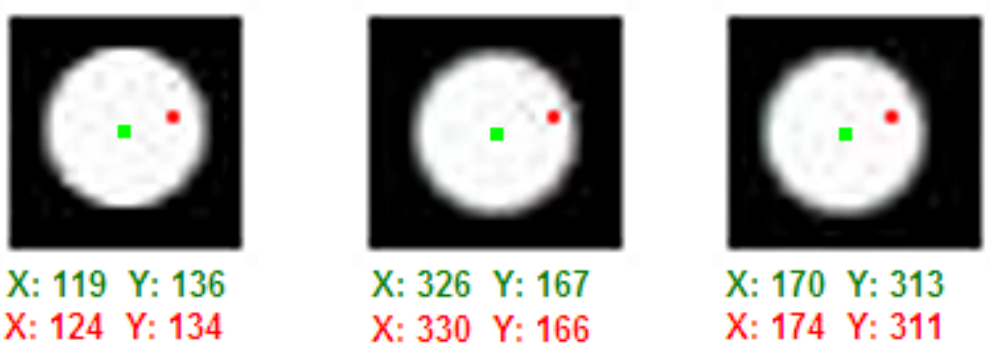

Figure 14. Center Point Precision Error

between the ground-truth center-point (green/top) and the center-point computed on the FPGA system (red/bottom). The coordinates for the center-points, which are referred to as ground-truth center-points, have been computed with a bounding-box approach on a GPP architecture. Since the provided images show perfectly-circular BLOBs only, the disadvantages of the bounding box method have not been taken into account. For further evaluation other images with different BLOB shapes have been used(Fig.15). For the processing of image frames, based on the sequential processing, a performance of up to 64 frames per second could be measured. This was limited by the performance of the analog-digital-converter, which translate the analog-video-signal input into digital-video-stream format.

To achieve higher frame-rates another approach to provide video frames to the BLOB-detection module needs to be realized. The BLOB detection system in the current version requires 8,449 logic elements, which allocate 25% of the FPGAs surface. The functionality of the system only covers the processing of a single video stream.

With the configuration changed to process three input streams in parallel the system requires

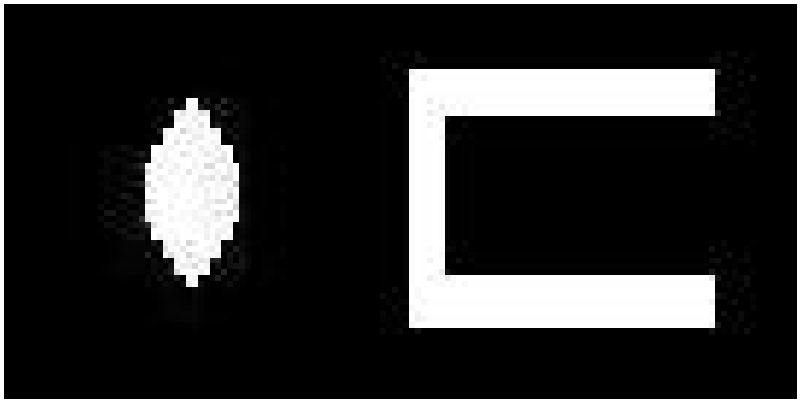

Figure 15. Diamond and U-shaped BLOB

21,786 logic elements which allocates 66% of the FPGA surface. This does leave some space for further functionality.

Compared to the performance of the same BLOB detection approach on a GPP architecture, which reached up to 91 frames per second, the FPGA approach does not offer an advantage. It took much longer to develop the system and systematic errors could not be eliminated. But this can not be taken as a serious outcome for an evaluation since the maximum performance is restricted to the speed of the underlying video-in to RGB-out transformation. For valid evaluation of the performance gain another input interface for the image or video material need to be applied. This requires the implementation of an device-specific module to receive and pre-process the image data.

6 Future Work

For the continuation of the project there are several steps which can be taken next. For a meaningful evaluation of the possible performance gain by the FPGA architecture, the problem of providing image and video data to the system needs to be solved first. Since the BLOB detection has been evaluated on static images only no statements can be made about precision error caused by motion blur.

This effect can happen dependent on how fast the user moves the light emitting device in the Immersion Square. The shape of the BLOBs will become elliptical with orientation to the direction of movement. This is another point which needs improvement to get more precise results in the center point computation. The detection of BLOBs has to be changed such that the processing of the whole frame is reduced to the area around the last computed BLOB coordinates. In addition, the intersection of the BLOB pixels of two consecutive frames could be used to eliminate false positives and to increase BLOB detection performance. This requires an additional image buffer to allow access to more than one frame.

An important next extension will be the RLE-based BLOB-detection method. It will allow us to take advantage of the parallelization aspect. But only under the condition that an appropriate frame buffer can be provided.

To analyze the extracted center points, but also for further processing, the transmission of the BLOBs' center-coordinates to a connected Host-PC is required. A test implementation for the serial interface has been realized and needs further improvement, if it will be used later on. This requires a module which can buffer the data, before sending it to the Host-PC. Otherwise the timing requirements of the RS232 chip can not be fulfilled. As mentioned before the data needs to be separated into blocks of 8 bit length or the transmission buffer will run into an overflow.

The current basis of the BLOB-detection system permits a sequential processing of the pixels while searching for BLOBs. This does of course not offer any point to parallelize the program execution. Also the access to the content of the frames is restricted to a single time. This can be solved by implementing a frame buffer that can hold one or more whole frames during the BLOB detection. This would provide random access to the frame instead of the sequential processing of the video data on single pixels.

A feature which is interesting for a future project would be the processing of different video

streams in parallel on one FPGA. But it requires the extension of the board interface to connect two more video sources to the system. The connection of two additional DE2 boards on the IDE interfaces could be a possible solution. These two boards can provide the already digitalized video stream from their analog-video interfaces.

In the current system the video stream is provided on the analog video interface. The theoretical upper bound of frames that can be processed is restricted to the provided frame rate of the connected video input-device. The DE2-board also contains an Ethernet interface which could be used to connect Gigabit Ethernet cameras to it. But this requires the implementation of our own API to put the camera frame packets into the correct order at the receiving DE2-board.

7 Summary

FPGA systems are an upcoming technology and of special interest for Computer Vision problems. These are still known as computationally intensive and require large processing power. Another reason are the decreasing prices for FPGA chips while the number of Logic Units increases.

With the growing number of projects and therefore developers as well, the entrance into this field will become easier. The vendors of FPGAs have recognized the indications of time and provide a larger support and free development and simulation environments.

It can not be concealed that the learning phase for VHDL or Verilog will consume more time than high-level languages like C or JAVA. But once the concepts are familiar, it will offer the ability to design individual solutions for all kinds of problem definitions. And these will be able to perform any required computation a hundred times faster than any C or JAVA program ever can achieve. In addition the higher level languages like Handel-C are more and more supported and will set the entrance barrier lower for new developers.

The implementation of a complete BLOB-detection system to detect multiple BLOBs in a continuous video stream could not be finished during the project. With the experimental approach of threshold based BLOB detection it has been proofed that the concept in general can be realized on a FGPA system. During the project many options of further directions for the implementation has come to mind.

Acknowledgments

Thanks to my advisors Prof. Rainer Herpers (Bonn-Rhein-Sieg University of Applied Sciences, Germany) and Prof. Kenneth B. Kent (University of New Brunswick, Canada) for their help and stimulating suggestions. I also want to thank the people from Matrix Vision and all who supported me in my research work, especially the community from alteraforum.com.

References

[1] Review of "highly parallel computing" by g. s. almasi and a. gottlieb, benjamin-cummings publishers, redwood city, ca, 1989. *IBM Syst. J.*, 29(1):165–166, 1990. Reviewer-Lorin, Harold R.

[2] P. C. Arribas and F. M. H. Maciá. Fpga implementation of santos-victor optical flow algorithm for real time image processing: an useful attempt. Univ. Politécnica de Madrid, 2002.

[3] J. Aylor, R. Waxman, and C. Scarratt. Vhdl - feature description and analysis. *Design & Test of Computers, IEEE*, 3(2):17–27, April 1986.

[4] A. Benedetti, A. Prati, and N. Scarabottolo. Image convolution on fpgas: the implementation of a multi-fpga fifo structure. In *FIFO Structure, 24 th. EUROMICRO Conference Volume 1 (EUROMICRO'98), August 25 - 27*, 1998.

[5] V. Berman. Standard verilog-vhdl interoperability. In *Verilog HDL Conference, 1994., International*, pages 2–9, Mar 1994.

[6] R. A. Bianchi and A. H. R. Costa. Implementing computer vision algorithms in hardware: an fpga/vhdl-based vision system for a mobile robot. In *Lecture Notes in Computer Science*. Springer Berlin / Heidelberg, 2002.

[7] W. Chuan-xu and L. Zuo-yong. A new face tracking algorithm based on local binary pattern and skin color information. In *Computer Science and Computational Technology, 2008. ISCSCT '08. International Symposium on*, volume 2, pages 657–660, Dec. 2008.

[8] B. Draper, W. Najjar, W. Bőhm, J. Hammes, B. Rinker, C. Ross, M. Chawathe, and J. Bins. Compiling and optimizing image processing algorithms for fpga's. In *in International Workshop on Computer Architectures for Machine Perception (CAMP)*, 2000.

[9] S. Garg, S. Garg, A. Willert, A. Willert, W. Rehm, and W. Rehm. Evaluating performance of heterogeneous clusters, 2002.

[10] J. Hammes, A. P. W. Bőhm, C. Ross, M. Chawathe, B. Draper, and W. Najjar. High performance image processing on fpgas. In *In Proceedings of the Los Alamos Computer Science Institute Symposium. Santa Fe, NM*, 2000.

[11] S. Hinz. Fast and subpixel precise blob detection and attribution. In *Proceedings of ICIP, 2005*, pages 11–14, 2005.

[12] R.-L. Hsu, M. Abdel-Mottaleb, and A. K. Jain. Face detection in color images, 2002.

[13] IEEE. Ieee standard hardware description language based on the verilog(r) hardware description language. *IEEE Std 1364-1995*, pages 1–675, Oct 1996.

[14] IEEE. Ieee standard verilog hardware description language. *IEEE Std 1364-2001*, pages 1–856, 2001.

[15] IEEE. Standard for systemverilog - unified hardware design, specification, and verification language. *IEC 62530:2007 (E)*, pages 1–668, 2007.

[16] A. F. Institute and A. R. J. François. Real-time multi-resolution blob tracking, April 2004.

[17] Isard and Maccormick. Bramble: A bayesian multiple-blob tracker. Compaq Systems Research Center, 2001.

[18] M. E. Latoschik and E. Bomberg. Augmenting a laser pointer with a diffraction grating for monoscopic 6dof detection. *Journal of Virtual Reality and Broadcasting*, 4(14):–, jan 2007. urn:nbn:de:0009-6-12754,, ISSN 1860-2037.

[19] G. Lipovski. Hardware description languages: Voices from the tower of babel*. *Computer*, 10(6):14–17, June 1977.

[20] J. W. MacLean. An evaluation of the suitability of fpgas for embedded vision systems. The First IEEE Workshop on Embedded Computer Vision Systems (San Diego), June 2005.

[21] S. Mckenna, S. Mckenna, S. Gong, and S. Gong. Tracking faces. In *In Proceedings of International Conference on Automatic Face & Gesture Recognition*, pages 271–276. IEEE Computer Society Press, 1996.

[22] E. Pinheiro, R. Bianchini, E. V. Carrera, and T. Heath. Load balancing and unbalancing for power and performance in cluster-based systems. In *In Workshop on Compilers and Operating Systems for Low Power*, 2001.

[23] L. Qiang and N. M. Allinson. Fpga-based optical distortion correction for imaging systems. ICSP, November 16-20 2006.

[24] L. Qiang and N. M. Allinson. Fpga implementation of pipelined architecture for optical imaging distortion correction. IEEE Workshop on Signal Processing Systems, October 2-4 2006.

[25] T. Serre, B. Heisele, S. Mukherjee, and T. Poggio. Feature selection for face detection. In *AI Memo 1697, Massachusetts Institute of Technology*, 2000.

[26] B. Tippetts, S. Fowers, K. Lillywhite, D.-J. Lee, and J. Archibald. Fpga implementation of a feature detection and tracking algorithm for real-time applications. pages 682–691. 2007.

[27] J. Trein, A. T. Schwarzbacher, and B. Hoppe. Fpga implementation of a single pass real-time blob analysis using run length encoding. MPC - Workshop, February 2008.

[28] J. Trein, A. T. Schwarzbacher, B. Hoppe, K.-H. Noffz, and T. Trenschel. Development of a fpga based real-time blob analysis circuit. In -. ISSC, Sept 13-14 2007.

[29] C. Wienss, I. Nikitin, G. Goebbels, K. Troche, M. Gőbel, L. Nikitina, and S. Műller. Sceptre - an infrared laser tracking system for virtual environments. In -, volume isbn 1-59593-321-2. Proceedings of the ACM symposium on Virtual Reality software and technology VRST 2006, 2006.