Active Tracking with Accelerated Image Processing in Hardware

Alexander Bochem

Publisher: Dean Prof. Dr. Wolfgang Heiden

University of Applied Sciences Bonn-Rhein-Sieg,
Department of Computer Science

Sankt Augustin, Germany

January 2011

Technical Report 01-2011

Hochschule
Bonn-Rhein-Sieg
University of Applied Sciences

ISSN 1869-5272

Bonn-Rhein-Sieg University of Applied Sciences

University of New Brunswick

Master Thesis

Active Tracking with Accelerated Image Processing in Hardware

by

Alexander Bochem

Student ID: 3316683 (UNB)
Matrikel-Nr. 9007689 (BRSU)

1 November 2010

B-IT Master Studies
Autonomous Systems

Bonn-Rhein-Sieg
University of Applied Sciences

Fraunhofer Institute for
Autonomous Intelligent Systems

Master Studies
Computer Science

Faculty of Computer Science
University of New Brunswick

Faculty of Computer Science
Bonn-Rhein-Sieg University of Applied Sciences

Grantham-Allee 20
53754 Sankt Augustin, NRW
Germany

Phone: +49 (2241) 865-195
Fax: +49 (2241) 865-8195
Email: sekretariat@mail.inf.h-brs.de
http://www.inf.h-brs.de

Faculty of Computer Science
University of New Brunswick

540 Windsor Street
Fredericton, NB, E3B 5A3
Canada

Phone: +1 (506) 453-4566
Fax: +1 (506) 453-3566
Email: fcs@unb.ca
http://www.cs.unb.ca

Abstract

This thesis work presents the implementation and validation of image processing problems in hardware to estimate the performance and precision gain. It compares the implementation for the addressed problem on a Field Programmable Gate Array (FPGA) with a software implementation for a General Purpose Processor (GPP) architecture. For both solutions the implementation costs for their development is an important aspect in the validation. The analysis of the flexibility and extendability that can be achieved by a modular implementation for the FPGA design was another major aspect. This work is based upon approaches from previous work, which included the detection of Binary Large OBjects (BLOBs) in static images and continuous video streams [13, 15]. One addressed problem of this work is the tracking of the detected BLOBs in continuous image material. This has been implemented for the FPGA platform and the GPP architecture. Both approaches have been compared with respect to performance and precision. This research project is motivated by the MI6 project of the Computer Vision research group, which is located at the Bonn-Rhein-Sieg University of Applied Sciences. The intent of the MI6 project is the tracking of a user in an immersive environment. The proposed solution is to attach a light emitting device to the user for tracking the created light dots on the projection surface of the immersive environment. Having the center points of those light dots would allow the estimation of the user's position and orientation. One major issue that makes Computer Vision problems computationally expensive is the high amount of data that has to be processed in real-time. Therefore, one major target for the implementation was to get a processing speed of more than 30 frames per second. This would allow the system to realize feedback to the user in a response time which is faster than the human visual perception. One problem that comes with the idea of using a light emitting device to represent the user, is the precision error. Dependent on the resolution of the tracked projection surface of the immersive environment, a *pixel might have a size in cm^2. Having a precision error of only a few pixels, might lead to an offset in the estimated user's position of several cm. In this research work the development and validation of a detection and tracking system for BLOBs on a Cyclone II FPGA from Altera has been realized. The system supports different input devices for the image acquisition and can perform detection and tracking for five to eight BLOBs. A further extension of the design has been evaluated and is possible with some constraints. Additional modules for compressing the image data based on run-length encoding and sub-pixel precision for the computed BLOB center-points have been designed. For the comparison of the FPGA approach for BLOB tracking a similar implementation in software using a multi-threaded approach has been realized. The system can transmit the detection or tracking results on two available communication interfaces, USB and RS232. The analysis of the hardware solution showed a similar precision for the BLOB detection and tracking as the software approach. One problem is the strong increase of the allocated resources when extending the system to process more BLOBs. With one of the applied target platforms, the DE2-70 board from Altera, the BLOB detection could be extended to process up to thirty BLOBs. The implementation of the tracking approach in hardware required much more effort than the software solution. The design of high level problems in hardware for this case are more expensive than the software implementation. The search and match steps in the tracking approach could be realized more efficiently and reliably in software. The additional pre-processing modules for sub-pixel precision and run-length-encoding helped to increase the system's performance and precision.*

Acknowledgements

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. This work is supported in part by Matrix Vision, Altera, CMC Microsystems and the Natural Sciences and Engineering Research Council of Canada.

Contents

List of Figures

List of Tables

1 Introduction

The acceptance of software applications is highly dependent on the usability of the software interface by the user. Finding different ways for the design of user interfaces to improve the interaction with software and optimizing the representation of the application data is addressed in various Computer Science research projects. The more intuitive a user can operate with a software interface the more benefit can be taken out of the digitization. This has lead to several different solutions for Human-Machine Interaction interfaces (HMI). First approaches had the motivation to integrate the user into a representation of the virtual world, known as virtual reality (VR) [19, 24]. In recent years the direction has changed to the integration of digital data into context with the real world, known as augmented reality (AR) [7, 57]. In 1994 the combination of VR and AR has been invented by Paul Milgram and Fumio Kishino, known as virtuality continuum (VC) [43].

The definition of the term "virtual reality" has been used in Antonin Artaud's[1] *The Theatre and Its Double (1938).* It describes an environment "in which characters, objects, and images take on the phantasmogaric force of alchemy's visionary internal dramas" [21]. Today, a user would usually think of a computer-designed virtual environment that allows the navigation in 3D.

For the representation of virtual environments one can choose between different technologies, such as computer monitors, Head Mounted Displays (HMD), and the Cave Automatic Virtual Environment (CAVE). The CAVE concept uses an arrangement of the screens that has the form of a cubicle [19]. The Bonn-Rhein-Sieg University of Applied Sciences applied this concept for the invention of an immersive visualization environment which is mobile and low cost. The "Immersion Square" [28] can operate with common PC hardware and has three projection walls for the visualization of the VR.

[1]http://en.wikipedia.org/wiki/Artaud

One problem which is disturbing the immersive cognition for the user of the virtual reality is caused by the characteristics of the available input devices. A common input device such as a keyboard or mouse, has to be operated actively by the user. This does restrain the user from diving into the virtual reality. For a better immersive effect the application of multi-modal input devices has become usual. Omni directional treadmills and wired gloves belong to this group of equipment. They have the advantage that the user can act more natural to create inputs for the virtual environment. The glove even allows the navigation in three dimensional space. But all those input devices have the same problem for the Immersion Square environment. They give no information about the user's position or orientation in the cubicle. This information would be helpful to increase the immersive effect for the user and improve the HMI of the system.

This thesis work is based on a BLOB detection system with a Bounding Box and a Center-of-Mass based approach for the computation of the center points using an FPGA. It covers the design and implementation of a tracking solution for the estimated BLOB center points in hardware and in software. Those two approaches are compared with respect to performance and precision. It covers the implementation of a pre-processing module to compress image data according to run-length-encoding. In addition the accuracy of the detection and tracking system has been extended with sub-pixel precision. The system's interface has been extended with a USB communication module. This has been compared with the existing serial communication module.

The remainder of this thesis is organized as follows. Section 2 gives a brief overview of the related works in immersive environments, tracking in computer vision, and parallel programming for multi-core architectures. Section 3 introduces previous work where this thesis work is based upon. In Section 4 the system design and applied methods are presented. Section 5 shows how the design of the approaches are implemented. Section 6 provides details of the applied tools and equipment. Section 7 investigates on the performance and precision characteristics of the approaches and sets them in relation to implementation costs. Section 8 finally concludes the paper and Section 9 gives suggestions for future work.

2 Background

The following Section will give an overview of the related research work and applied technologies in this thesis. First section covers the definition of "Immersive Environments" and the existing problems of their human-machine interfaces. The second section is an overview of different solutions for tracking problems in image processing. In section three the motivation for this work using FPGAs for image processing is presented. Section four gives an introduction into parallel programming and the application of Intel's Threading Building Blocks library in particular.

2.1 Immersive Environments

The term "immersion" in virtual reality refers to the effect a user can experience when confronted with an artificial environment. This can include perception with any of the human senses, but is usually realized as a visual solution. For the visualization of a virtual environment one can usually choose between a computer monitor, a Head Mounted Display (HMD), or back projection approaches.

Using computer monitors to create immersion has the problem that one monitor is relatively small in comparison to a virtual environment. But for using multiple monitors one has to accept that the frame of the monitors might disturb the immersive effect for the user. Therefore a computer monitor is for sure a low priced solution but not very impressive. With a HMD the virtual reality can achieve a much higher immersion. One issue of HMDs is their weight, that makes them uncomfortable to wear for an extended time period. HMDs are also expensive, and therefore not recommendable for application environments with multiple users. For such cases the application of one or several back projection screens should be preferred. This concept is known as a Cave Automatic Virtual Environment (CAVE) and is usually realized in the form of a big cubicle [19]. The Bonn-Rhein-Sieg University of Applied Sciences has adapted the CAVE concept to realize a low cost solution for immersive environments in the form of a mobile platform, called "Immersion Square" [28]. The problem that comes with such an immersive system is known as Off-axis projection. Without any knowledge about the position of the user and the point he is looking at, the system can not simulate a perspective view for him. The perspective of one particular object will not change if the user changes their position in the cubicle. But this kind of user interaction is desired for application purposes like in [27]. Common input devices for computers, such as a mouse, or even for virtual reality, like gloves or treadmills, can not serve this purpose. They do not offer information about absolute position in the real world, instead representing the user in the virtual world.

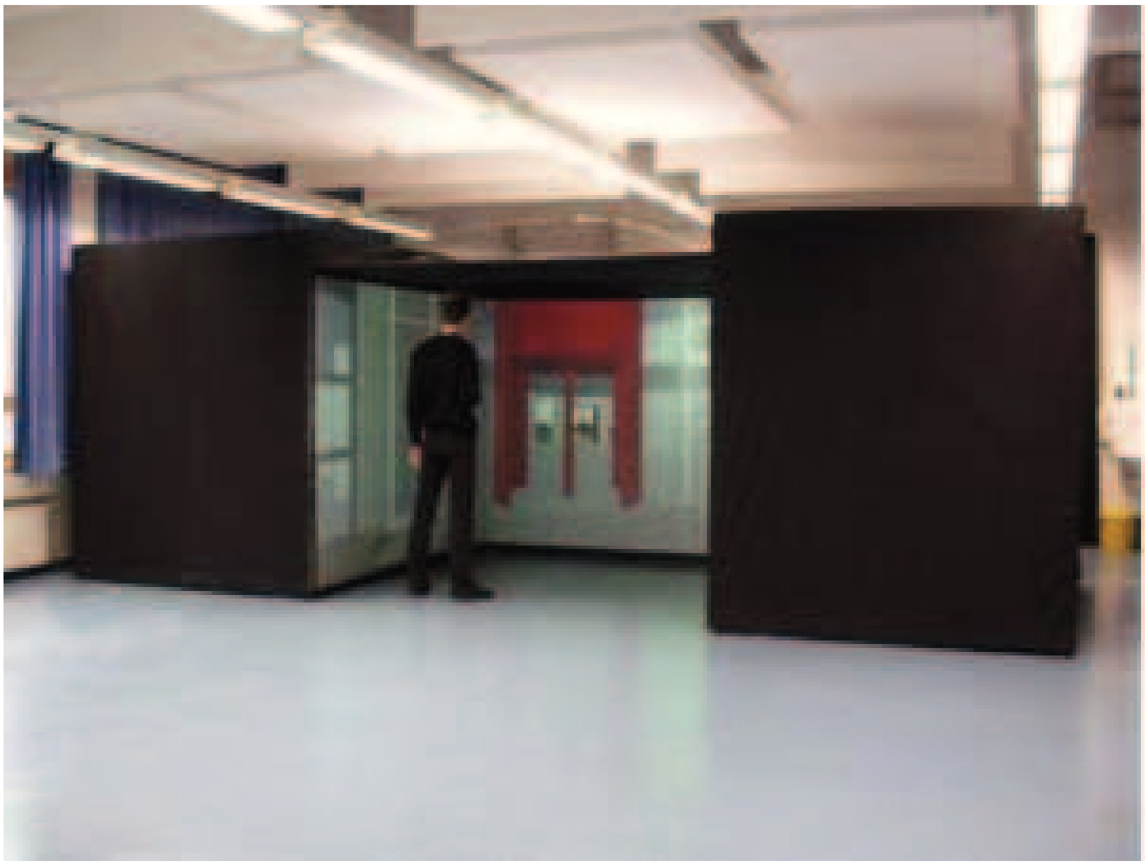


Figure 1. Immersion Square at Bonn-Rhein-Sieg University of Applied Sciences.

In a research project at the Bonn-Rhein-Sieg University of Applied Sciences named "*6 Degree-of-Freedom Multi-User Interaction-Device for 3D-Projection Environments (MI6)*" the invention of alternative ways for user interaction has been addressed by the Computer Vision research group. The idea is the application of a customized device to estimate the position and orientation of the user in the Immersion Square (Figures 1,2). This information shall be used to manipulate the virtual world. For the representation of the user's position and orientation a light emitting device will be developed, that is creating a pattern of light dots onto the projection screens. The device is carried by the user and points towards the projection walls. Using a unique pattern, it is possible to compute the required information about the user if the center points of the BLOBs can be estimated, as shown in [56, 39].

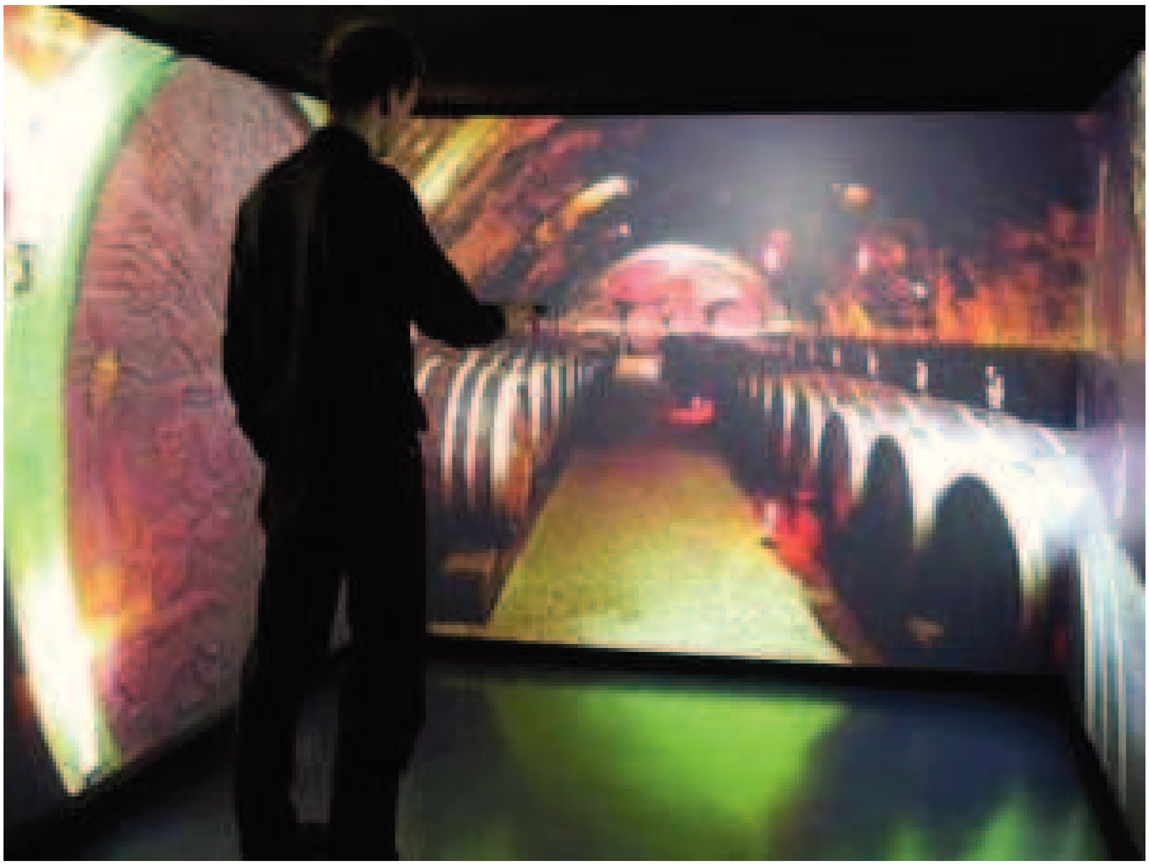


Figure 2. Immersion Square at Bonn-Rhein-Sieg University of Applied Sciences.

The light emitting device will use a laser diode which work in the infrared range. This has the advantage that it is not recognizable for the human eye. The projection screens will be recorded with digital cameras and acquire the input data for the detection of the BLOBs. Precision is a major requirement for the target application.

Dependent on the size of a pixel an error of a few pixel, for the center point of a BLOB, will cause a big offset in the computed position of the user. This problem will increase with the distance between user and projection wall. For a sufficient response time of the Immersion Square for the user, the image processing for the BLOB detection has to be fast. The intent is the development of a reliable system with high precision and performance.

2.2 Tracking

One intent of tracking in computer vision is to identify the motion of objects in image scenes. Once an object is detected in the processed image material, this object is recognized as the same object in the following consecutive frames. For identifying the object several different attributes of the object can be applied.

A simple way to track an object is to estimate the distance between objects in consecutive frames. Under the assumption that an object at time t should be the closest to an object at time $t+1$, the Euclidean distance between the object's center points can be applied to determine the object. For optimal results the performance of the camera has to be faster than the movement of the object in the image scene. According to the Nyquist rate this means the frame rate has to be more than two times faster, than the object can move. The distance can be expressed in terms of pixel per second (pps). If an object would move with 1 pps. the camera should at least run with 2 frames per second (fps). The size of the area in the object, which is covered by one pixel, depends on the camera resolution and the distance of the object to the lense of the camera.

Additional attributes to improve the tracking reliability would be the size of the object, its shape or color. Those are usually combined in a so-called "feature list" and have to be extracted from the image scene. Again the benefit of those attributes depends on several things, like the object, the surroundings, and the applied technology. The appearance of the object can change dependent on the position and orientation of it or if the environment changes. For example, the change of illumination can show the object in a completely different shape, because of shading or color recognition.

Binh [12] showed a useful combination of object attributes and invented a robust framework for object tracking. The feature list for the tracked object is updated continuously while processing the image material. Regarding Binh's statement that existing tracking approaches cannot handle objects in motion properly and therefore lose track of the object after some period of time, the approach combines machine learning and a robust strategy to handle tracking failures and recovery. The concept of the framework is very complex and because of the large memory requirements for the machine learning process, not applicable for small sized FPGA platforms. The experimental results showed a maximum performance of 25 fps, which is satisfactory for a GPP approach. To apply the mentioned attributes the camera system has to be calibrated for the application environment. Calibration is always required to guarantee good image quality and especially useful if the relationship between the real world and the representation world is needed.

Tracking can help to scale down computation time. With gained knowledge about the object's position from previous frames, the area of the image to be processed can be decreased. Another way would be the segmentation of the objects from a static background scene. This method is known as foreground-background segmentation and has been applied frequently for object detection and tracking.

The authors of [2] combined foreground-background segmentation with a dynamic feature extraction of the detected objects to allow tracking of multiple moving objects. With additional information about the objects center point, its area and the main intensity in relation to the image background, the approach showed reliable results for moving objects in clear scenes. However, their invented similarity function to allow continuous tracking of multiple moving objects, failed for occluded objects and showed errors for multiple objects with large differences in size. Their achieved performance of 24 frames per second worked on a very low resolution of 320x480.

Occlusion is a major problem for detection and tracking applications. Especially in public environments, where the object's movement and the appearance of new objects, is not predictable. In [23] the tracking of moving objects, which are partially occluded, has been addressed using a Kalman Filter. The approach required the creation of a 3D model out of the applied 2D image data. Therefore the objects need to be detected without occlusion or the model of the occluding objects had to be taken into account as well. The tracking precision could be improved for moving objects if the occluding objects are static. However the tracking of moving objects, which are partially occluded by other moving objects is still unsolved. Because of noise or influences that have not been modeled the tracking is more likely to fail. The authors left the extraction of 3D information out of 2D images automatically as an open problem.

The problem of object tracking on a FPGA has been addressed in [18] by creating a modified algorithm for a particle filter with multiple features. Particle filters have the advantage to handle clutter in image scenes and recover from temporary distractions like occlusion. The problems are that particle filters are computationally expensive and that a particle filter for a circuit design requires a specific modification of its algorithm. Although the approach showed confident performance of 54 fps for a resolution of 640x480, the results only discussed the detection and tracking of single objects. This does not meet the requirements for the proposed application task in our project.

The tracking of multiple moving objects in existing approaches requires complex image processing. In [58] the authors showed that a simple foreground-background segmentation does not suffer the requirements of precise object detection. Without precise detection reliable tracking is not possible, because the applied templates to identify objects fail. With additional pre-processing, e.g. shadow removal, the approach can be improved. However, besides the fact that the tracking did not work if occlusion in-between moving objects occurred, the approach could not perform faster than 12 frames per second and therefore, is not useful for real-time performance requirements.

Tracking of multiple objects in real-time has the issue of high performance requirements. For increasing precision the processing steps have to be refined by more accurate matching and feature extraction methods. In [59] the application of an improved tracking approach for multiple objects managed to gain precision but suffered in processing performance. The system managed to process up to 10 frames per second for a resolution of 320x240.

The proposed approaches show that the trade off between precision and performance is yet unsolved for the problem space of object tracking. While the high-level solutions perform pretty good with respect to precision, they are not qualified for implementing in hardware. For breaking down the complexity of those approaches to allow the description in a hardware description language, essential changes and trade offs in the functionality are required.

All proposed tracking approaches use the direct visual representation of the object to be tracked in the processed image material. Those applications can be referred to as passive tracking. Passive tracking uses existing attributes of the object without applying any additional features. One problem of passive tracking is the requirement for good illumination conditions. If the image scene is too dark, chances are good that the object can not be tracked nor detected. To overcome this problem the application of artificial features, for example beacons or light sources, is one opportunity. In [55] the object to be tracked is replaced by light emitting devices. This allows the application of tracking methods even in environments with bad or variable changing illumination conditions. Approaches based on this concept are referred to as active tracking.

The application of active tracking can overcome the discussed illumination problems. As shown in [56] with a sufficient number of dots, projected from the light emitting device, it is possible to estimate position and orientation of the light source. In passive tracking approaches the quality of illumination has a big impact on the object appearance. With bad illumination the object's edges can become blurred and parts of the object might not be detected. The application of active tracking can reduce the complexity of the image material and improve the tracking precision. Existing approaches in that area take advantage of the higher light spectrum that comes with an active light source. A light emitting device as representation for the object allows one to record the image scene in infrared. The resulting image material only contains the dots from the light source. Due to the simplicity of the image material the processing of it is usually referred to as Binary Large OBject (BLOB) detection. BLOBs are not necessarily light dots, but simple objects in images that can be identified as one object due to connectivity or color. By using a unique pattern of light dots, it is possible to compute the position and orientation of the light source, based on the center points of the light dots. This approach requires a precise calibration of the camera. Only if the relation between the pixel size and the size of the light dots is known, the estimation of position and orientation is possible. To simplify the recording and calibration, a plain projection surface is usually applied from where the light dots are tracked. The problem here is the need for a high resolution in the image material to allow precise results. However, with a higher resolution the processing performance decreases. Existing solutions as in [39] are not able to be performed in real-time. Due to the visual perception of the human eye, this would mean having a frame rate above 30 frames per second.

The application of active tracking, using a light emitting device, shortens the technologies for recording the image scene from which the developer can choose. For this application case the usage of a Charge-coupled device camera should be avoided, since those tend to show a smear effect on the image for active light sources. The light source causes an overload effect on the sensor nodes of a full column. This column will show up with much brighter pixel values, compared to its neighbours. One solution is to use a complementary metal-oxide-semiconductor (CMOS) camera instead, which will not show a smear on active light sources. Another way is to apply some kind of normalization, such as a Gaussian Filter, to reduce the brightness of that particular image column based on its neighbour column.

2.3 FPGA Image Processing

Digital Image Processing takes place in various fields of today's life. Whereas the solutions for each problem are different, they all have the automated processing of digital image data in common. The rapid spread of automated image processing into research and industry was possible because of the large computational power that had become available in workstations and personal computers about 20 years ago. One problem in computer vision is that a general solution for different image processing tasks is not feasible. The heterogeneity of the data that comes with digital image processing requires context sensitive solutions. This is one particular reason for the huge amount of research work that has been done and is going on in that area. Out of these circumstances the image material that is usually processed in particular vision problems, looks very similar in general. This similarity has been used to increase processing performance by specialized hardware units, like specialized Digital Signal Processors (DSP) and Graphic Processing Units (GPU).

The first DSPs were expensive and have been used for special applications, like radar & sonar or space exploration in the 1960s [49]. The functionality of a DSP can vary from filtering over sampling to compression, dependent on its design. With the revolution of personal computers in the 1980s, DSPs have become affordable even for consumer products. DSPs can be found in any field from scientific research over biomedical devices to industrial applications. Many of today's embedded systems contain DSPs with specialized functionality. In consumer products DSPs can be found in camcorders or digital cameras, for example. They provide standard operations to process signal data, while having a low power consumption because of their customized design [49]. Without DSPs for image processing the success of the Television even might not have been possible at all. A problem with DSPs is that they are restricted to their design. The usual DSP can not be programmed with a functionality that has not been provided in its design during production. This gives limitations on the usability and flexibility for image processing tasks with changing conditions or different application purposes.

The invention of dedicated hardware to accelerate graphic processing on computers goes back to the late 1970s and early 1980s. Until then, visualization of graphics in computers has been performed on General Purpose Processors (GPP). The tasks in image processing do not require the whole functionality of a GPP architecture. Especially floating point computation is one of the most applied methods. GPUs were designed especially for floating point operations, to relieve the General Purpose Processor. A graphic card contains more than one Floating Point Unit (FPU) and allows one to process image data in parallel, if there is no dependency between the data. Today's graphic cards are referred to as "many-core" architectures and might contain several hundred processing cores. One advantage of GPUs is that an image processing task can be implemented in a high level programming language, like C/C++ or .NET. The different functional units of a GPU are usually accessible through an Application Programming Interface (API). Those APIs are provided by the vendor of the graphics card and offer massive parallel computation performance to the software developer. One problem with the API as a programming interface is that every vendor supports its own and they are not compatible. An application using an API cannot be executed on a GPU architecture from a different vendor. However, the compiler for the APIs do allow the developer to choose if the program should be compiled for the GPU or for common execution on a GPP. This feature simplifies the evaluation and comparison of sequential processing on a GPP with the parallel execution on a GPU. Having in mind that today's computers contain between two and eight cores, the comparison becomes rather complex since all available resources should be taken into account for providing reasonable results. Both hardware architectures have one common issue. The optimal usage of available resources is highly dependent on the quality of the implementation. Sequential programming is still the standard for how programs and algorithms are realized, although multi-core architectures are available for around a decade now.

The problem with GPP and GPU architectures is that they are not well suited for many application areas outside of personal computers. For example in the automotive industry, or for mobile devices, the product environment can not provide sufficient power supply or space to fit the hardware. The alternative of using a DSP instead might cause issues because of the limited functionality. One solution might be customized hardware, such as an Application Specific Integrated Circuit (ASIC). An ASIC can provide better performance compared to GPP and GPU approaches while having a very low power consumption. For years ASICs have been used for highly customized integrated circuits in different application areas [16]. One problem with ASICs is their missing flexibility once the design has been fabricated into a chip. The economic effort to design an ASIC is only worthwhile for mass production starting from several thousand units. That is where the application of a Field Programmable Gate Array (FPGA) might be the better choice. The FPGA is a grid of freely programmable logic blocks which can be combined through interconnection wires. This allows the hardware designer to have a flexible and also application specific hardware design. While FPGAs have been used for prototyping only in the beginning, the revolution of personal computers helped this field become attractive for consumer and industry products as well. The decreasing sizes for chip circuit elements allowed manufacturers to create FPGAs, which are large enough to fit even complete processor designs. Those advantages have been perceived in the computer vision area [25] and used in various research projects [9, 8, 29, 54]. The FPGA architecture allows to create a hardware design that can process data in parallel like a multi-core architecture on a GPU. This again is well suited for computer vision problems, as have been addressed in [41]. It has to be kept in mind that, although ASICs and FPGAs have the design of hardware in common, the circuit space and power consumption of an FPGA is still several times higher than that of an ASIC.

The design abilities of FPGAs lead to the invention of hardware/software co-design, as one field of research in computer science. The intent of hardware/software co-design is to split computational problems into hardware and software tasks and to explore the optimal balance in terms of resources, performance, and power consumption. For a complex system design it might become nearly impossible for a human developer to find every possible point of optimization. Therefore, vendors for FPGA development environments try to address the problem with automated tools, such as Catapult-C, Altium Designer, and Altera Quartus II. Until now, those tools can not solve the issue completely, but they perform much better than humans [42].

2.4 Parallel Programming for Multi-Core Architectures

With the invention of multi-core architectures the design concepts for application software development were forced to take a change. The pure sequential implementation of an algorithm can not run any faster, regardless of how many cores there will be available. For better performance, the program or algorithm has to be analyzed for sections that can be executed in parallel. The problem which can appear with multi-core architectures is similar to the problem of massive parallel systems and vector processing machines.

The most important thing the developer has to look at are data dependencies in the process to be implemented. This analysis procedure is referred to as identification of the "critical path". The term refers to the largest set of dependent calculations inside the program, that have to be sequentially executed. The theoretical concept to describe those dependencies have been published in 1966 by Bernstein [11]. It has been largely adopted for grid computing, specialized parallel systems, and similar approaches. The violation of the conditions can cause a "race condition" and leads to various effects, from wrong results to complete system crashes. There are several methods to work around this problem. One is known as "mutual exclusion" or "mutex". The developer implements an artificial barrier around the memory region, that contains the data which is manipulated. This allows to give control over the data in the order of execution, according to the critical path. Another method is the synchronization of the program execution right before and after a parallel section.

Parallelism is a design problem that can be solved on different levels of a computer architecture. The first one was used to increase processing performance of computers. By increasing the word size the central-processing-unit (CPU) can read, the number of instructions can be reduced. This is referred to as "bit-level parallelism" and has not been used since the invention of 64-bit processors. The downside of this method is that software programs need to be compiled with a compiler that supports the architecture of the CPU to take advantage of the word size. Another concept is known as "instruction level parallelism". It takes a set of instructions, that will be executed next in the program, and groups them into parallel executable sections. The condition is that there are no data dependencies between the variables of the parallel parts. A high level concept which is usually applied for parallelism nowadays is "task parallelism". It allows to perform the execution of the program in different processes and if available on different processors in parallel. One problem of task parallelism is the organization and control of memory access. The concept distinguishes between "shared memory" and "distributed memory". The shared memory is usually at a central location in the architecture and available to all tasks. While the distributed memory is located in the processor cores of the multi-core architecture and only available for the task of each core. Although the physical distribution might suggest it, the presence of concepts like direct-memory-access (DMA) can compensate for the architecture and might allow shared memory in the processor cores [20].

While the architectures and the computational problem for parallel programming is different, most of the problems are similar. This has lead to several different solutions, such as programming languages and libraries to simplify the implementation of parallelism. For software programming on cluster and multi-machine systems the application of Common Object Request Broker Architecture (CORBA) [45] or Internet Communications Engine (ICE) [26] is common. The goal behind CORBA and ICE is the combination of software in different programming languages and on multiple platforms. Other concepts like the Message Passing Interface (MPI) [44] or Open Multi-Processing (OpenMP) [47] have been around for while and are well established. MPI is an Application Programming Interfaces (API) to establish communication in clusters or multi-processor systems even between programs written in different programming languages. OpenMP is an API as well, but it works based upon multi threading. OpenMP is designed to implement task parallelism for multi-processor platforms. A concept which is much newer and not yet widely used is Intel(R) Threading Building Blocks (TBB) [34]. Intel has developed TBB to provide a Standard Template Library (STL) that supports their multi-core processors using the high-level programming language C++. TBB takes over responsibility to balance the workload of the created tasks onto the cores. It provides compiler directives to declare parallel sections, data structures to exchange data between tasks, control types to create mutual exclusion and atomic operations. Atomic operations are commands that cannot be interrupted during their execution. For balancing the workload, TBB has an integrated task scheduler. The TBB library is available from Intel as closed source commercial product, which requires a license fee. But at the same time Intel participates in the development of the "TBB for Open Source", which has nearly the same functionality as its commercial counterpart.

One advantage of TBB is that is does not require the usage of native threading packages, such as Windows threads or POSIX threads. It uses "work stealing" to move tasks between processing cores to balance the workload. TBB can be used by developers without being an expert in thread based programming paradigms [34, 35, 36, 48].

3 Previous Works

The work described in this thesis is based upon two projects, also known as "Research & Development" (R&D) 1 and 2. Those projects have addressed other problems and covered earlier design tasks, which have been mandatory for the MI6 project. This Section will introduce the main aspects and results of those projects.

3.1 R&D1 - Hardware Acceleration of Image Processing

The goal of the R&D1 project was to analyze the feasibility of computer vision tasks in hardware. One advantage of designing algorithms in hardware is the ability to parallelize processing. In this work a representative image processing method has been selected for the implementation on a FPGA platform. It had to be analyzed for parallel design. The implementation had been evaluated for performance and precision. With respect to the overall project background from the MI6 project, the approach covered the detection of multiple BLOBs in a continuous video stream [14]. The implemented design is processing frame pixel data in sequential and identifying BLOBs based on a fixed threshold for the brightness value. Adjacent pixels, that have a higher value than the threshold, are merged to a single BLOB. For the computation of the BLOB's center points, a Bounding Box based method has been applied. The results have been made available by showing them on the seven-segment display of the Altera DE2 development & education board [50]. For directing the video stream into the system, the analog-video input has been used. This decision turned out to become the bottleneck of the system, since the available analog-digital converter between the AV-input and the FPGA could not perform faster than 30 frames per second. Because of this shortcoming, the system could not compete with a similar implementation on a GPP architecture. The precision evaluation has been performed with reference image material, for that the BLOB's center points were analyzed by hand.

3.1.1 BLOB Detection

The image material in the proposed application task from Section 2.1 will have a black background, because the cameras that will record the projection surface of the Immersion Square work in the infrared range. The light dots will appear as circular shaped areas of white and light gray pixels, referred to as a Binary Large OBject (BLOB). Dependent on the viewing angle of the user and the speed of his motion, the intensity and shape of the BLOBs can vary. Figures 3, 4, and 5 showing some examples of how the expected image material will appear. The light dots will be arranged in a fixed pattern.

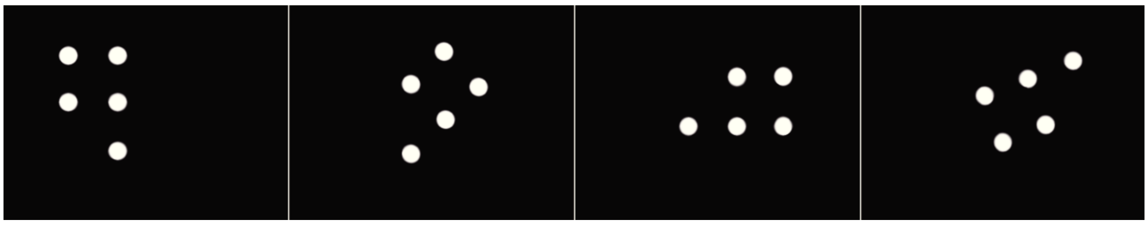


Figure 3. Example of clear shaped perfectly circular BLOBs

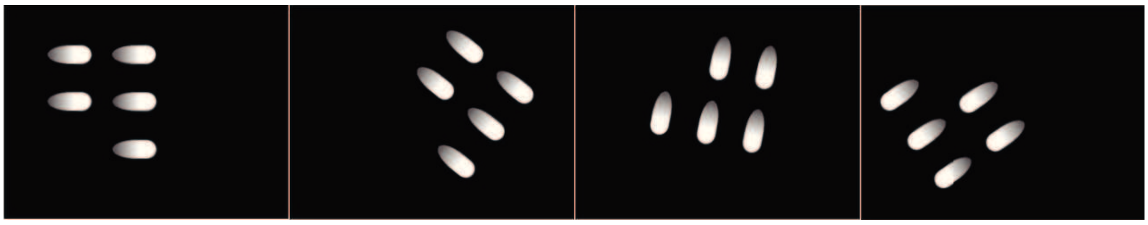


Figure 4. Example of BLOBs with blurring by horizontal or vertical motion

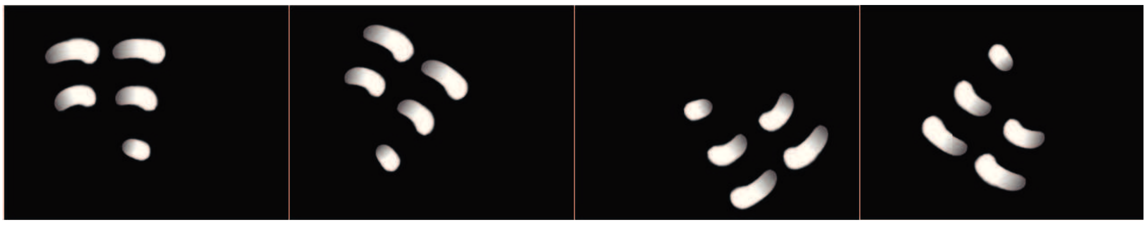
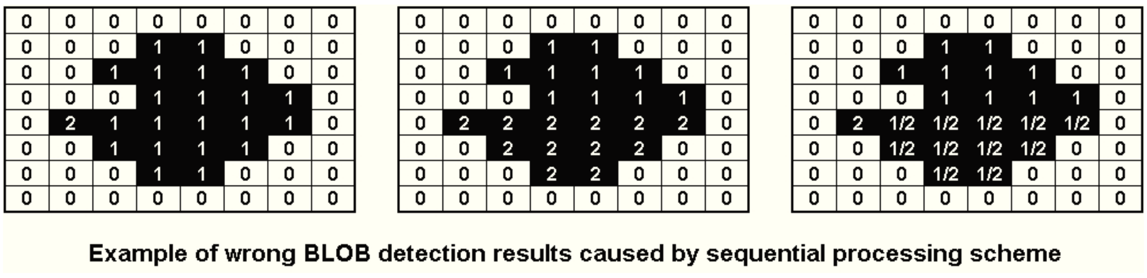


Figure 5. Example of BLOBs with blurring by rotation

Those BLOBs, which are perfectly circular shaped, have the best conditions for the detection task. Their area shows pixels of similar brightness value and the edges are very sharp. But this case will only occur if the user is watching at a specific spot on the projection screen, which means he does not move the light emitting device or moves it slow enough to not cause a blurring effect. The definition of "slow enough" depends on the actual frame rate and resolution of the camera, which records the projection screen.

Having perfect circular shaped BLOBs with clear edges decreases the possibility that one BLOB might be labeled as two different BLOBs. That can happen if the edges of a BLOB in two consecutive rows or lines are too far apart, to allow a valid adjacency check of the pixels. The Figure 6 gives an example of a BLOB that has been labeled inconsistent.



0	0	0	0	0	0	0	0
0	0	0	1	1	0	0	0
0	0	1	1	1	1	0	0
0	0	0	1	1	1	1	0
0	2	1	1	1	1	1	0
0	0	1	1	1	1	0	0
0	0	0	1	1	0	0	0
0	0	0	0	0	0	0	0

0	0	0	0	0	0	0	0
0	0	0	1	1	0	0	0
0	0	1	1	1	1	0	0
0	0	0	1	1	1	1	0
0	2	2	2	2	2	2	0
0	0	2	2	2	2	0	0
0	0	0	2	2	0	0	0
0	0	0	0	0	0	0	0

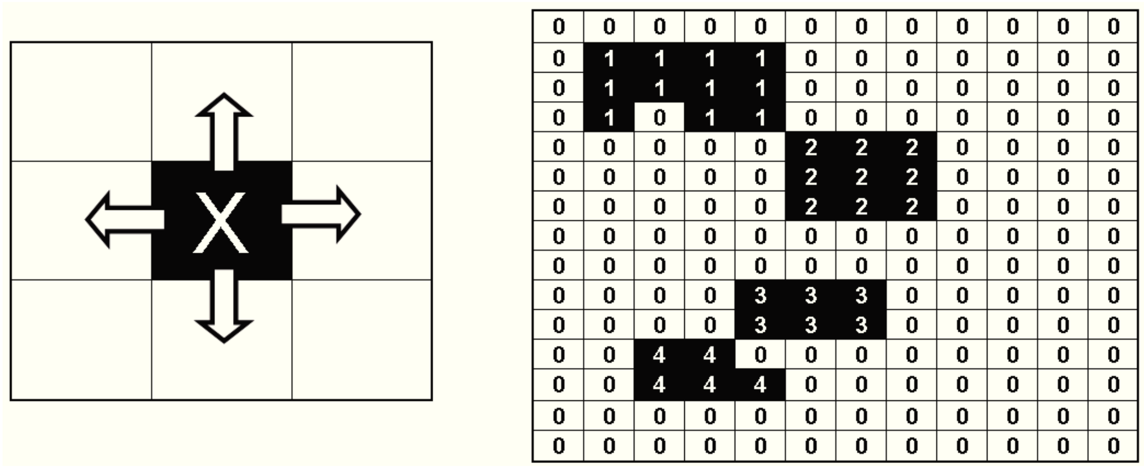
0	0	0	0	0	0	0	0
0	0	0	1	1	0	0	0
0	0	1	1	1	1	0	0
0	0	0	1	1	1	1	0
0	2	1/2	1/2	1/2	1/2	1/2	0
0	0	1/2	1/2	1/2	1/2	0	0
0	0	0	1/2	1/2	0	0	0
0	0	0	0	0	0	0	0

Example of wrong BLOB detection results caused by sequential processing scheme

Figure 6. Example of pixel labeling by sequential image processing

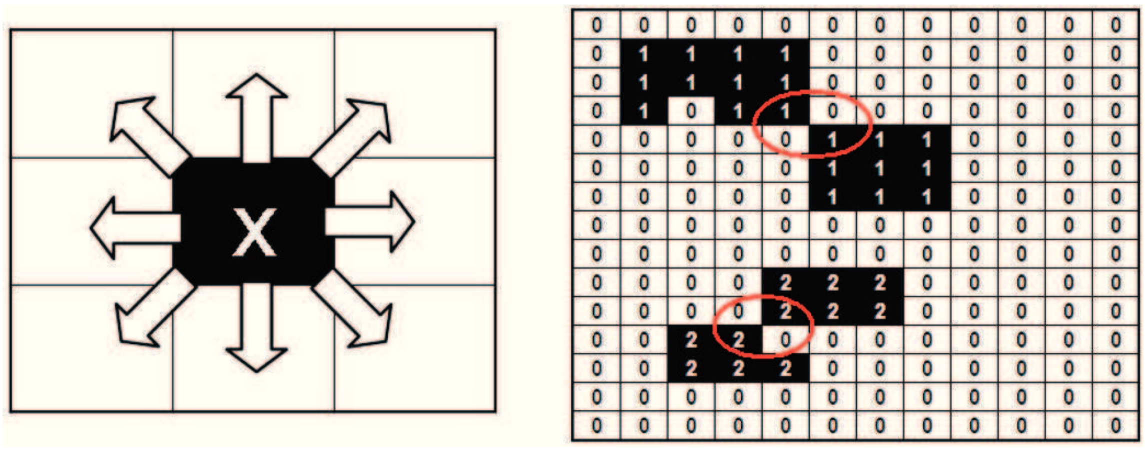
For the BLOB identification process the pixels that are above the specified threshold value will be checked for their adjacency. Two common methods to evaluate adjacency have been looked at, four-pixel neighbourhood and eight-pixel neighbourhood. An example of both is given in Figure 7 and 8. The four-pixel neighbourhood checks for adjacency only on the horizontal and vertical axis of the current pixel. With the eight-pixel neighbourhood the diagonal axis are taken into account as well.

The authors have decided to realize the BLOB detection using the eight-pixel neighbourhood for the adjacency check. This was motivated by the small difference in the implementation work for both methods, while gaining more reliability with the eight-pixel neighbourhood. As soon as the motion speed by the user with the light emitting device passes the point where the camera's frame rate is able to record



0	0	0	0	0	0	0	0	0	0	0	0
0	1	1	1	1	0	0	0	0	0	0	0
0	1	1	1	1	0	0	0	0	0	0	0
0	1	0	1	1	0	0	0	0	0	0	0
0	0	0	0	0	2	2	2	0	0	0	0
0	0	0	0	0	2	2	2	0	0	0	0
0	0	0	0	0	2	2	2	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	3	3	3	0	0	0	0	0
0	0	0	0	3	3	3	0	0	0	0	0
0	0	4	4	0	0	0	0	0	0	0	0
0	0	4	4	4	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0

Figure 7. Four pixel neighbourhood



0	0	0	0	0	0	0	0	0	0	0	0
0	1	1	1	1	0	0	0	0	0	0	0
0	1	1	1	1	0	0	0	0	0	0	0
0	1	0	1	1	0	0	0	0	0	0	0
0	0	0	0	0	1	1	1	0	0	0	0
0	0	0	0	0	1	1	1	0	0	0	0
0	0	0	0	0	1	1	1	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	2	2	2	0	0	0	0	0
0	0	0	0	2	2	2	0	0	0	0	0
0	0	2	2	0	0	0	0	0	0	0	0
0	0	2	2	2	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0

Figure 8. Eight pixel neighbourhood

the light dots in a perfectly circular shape, the BLOBs might show a blur effect. The BLOBs with blur will look similar to the examples from Figures 3, 4, and 5 but it depends on the direction of the user's movement. The user, standing in the Immersion Square, can move the device in the three dimensional coordinate system of the real world, while the image material from the recorded projection walls is only in two dimensions. If the motion goes along the horizontal or vertical axis of the screen, the BLOB can show an elliptical shape, like in Figure 4. If the user moves the device along the orthogonal axis (Z axis, Fig. 9), the size of the BLOBs and their distance to each other will change. Rotating the device on the horizontal or vertical axis (X/Y axis, Fig 9) again leads to elliptical shaped BLOBs. If the user rotates the light emitting device on the orthogonal axis, the BLOB might even show up in a curved elliptical shape due to the pattern of light dots, as can be seen in Figure 5.

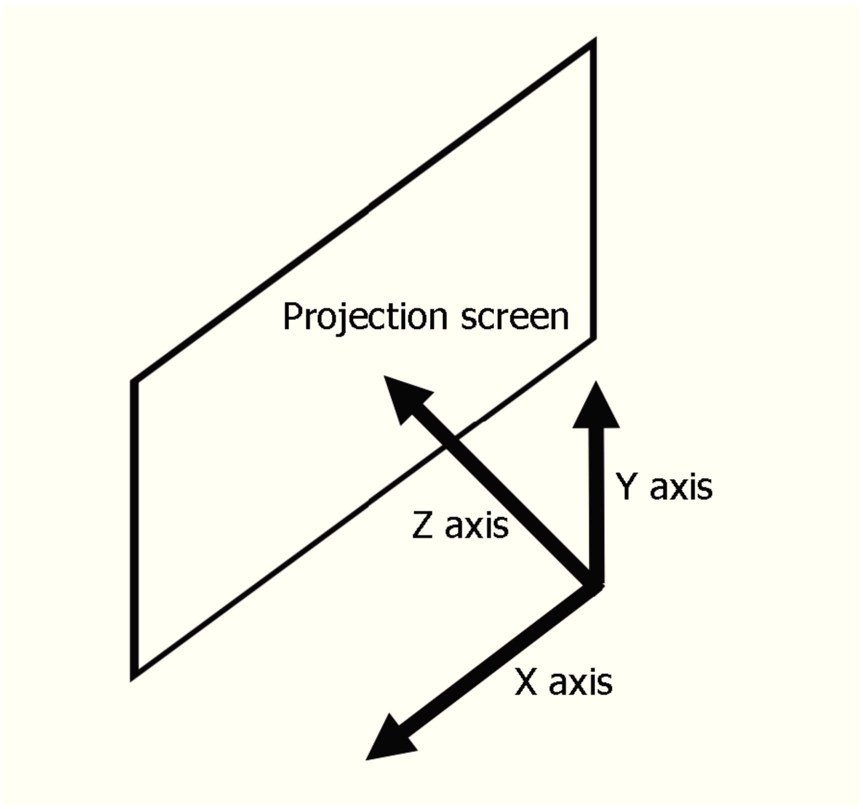


Figure 9. Navigation space for user in Immersion Square

In the detection approach, only the elliptical shape has been taken into account. This was based on the fact that the possible effect of a curved elliptical shape by the BLOB has not been expected in the requirements analysis of the project.

3.1.2 Bounding Box

After the BLOBs have been detected, the next step was the computation of their center points. For this approach the center points have been estimated based on a Bounding Box that encloses the BLOB. The required attributes are the minimum and maximum coordinates of the pixels on the horizontal and vertical axis. For computing the center points the formula is as follows.

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

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

Given the described image material Figure 10 shows the expected results for this approach. The center point results are very dependent on the pixels at the BLOB's edges and the general shape of the BLOB.

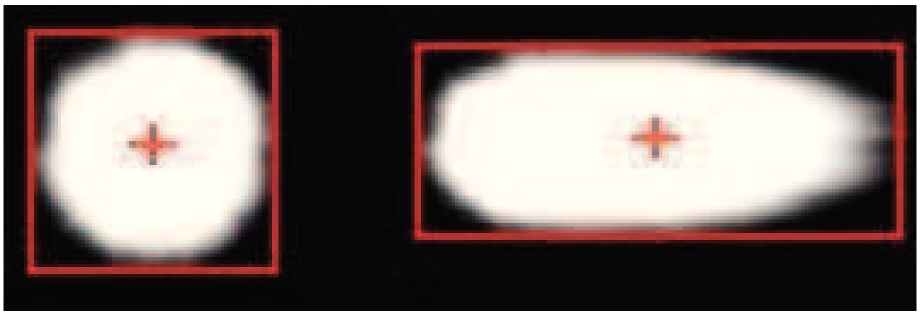


Figure 10. Example result for BLOB center point by Bounding Box

3.1.3 Run-Length Encoding

For simplifying the BLOB detection process, it was planned to realize a pre-processing of the frames that combines adjacent pixels on a frame line. This method is called run-length-encoding (RLE). The pixel data of a single frame line is searched for pixels that have its brightness value higher than the specified threshold. After the first pixel is found, the coordinates of that pixel and its brightness value is temporarily stored. The data structure for that is referred to as a "run". A run consisting of a single pixel has a length of one. Each adjacent pixel on the line which has a higher brightness value than the threshold as well, is added to the run. This is done by adding the brightness value to the run and increasing its length by one. A run ends if the next processed pixel in that line has a brightness value below the specified threshold. Each run in the processed line will have a unique index with respect to the detected runs in the current frame. Figure 11 gives an example of a frame that has been processed by the RLE method.

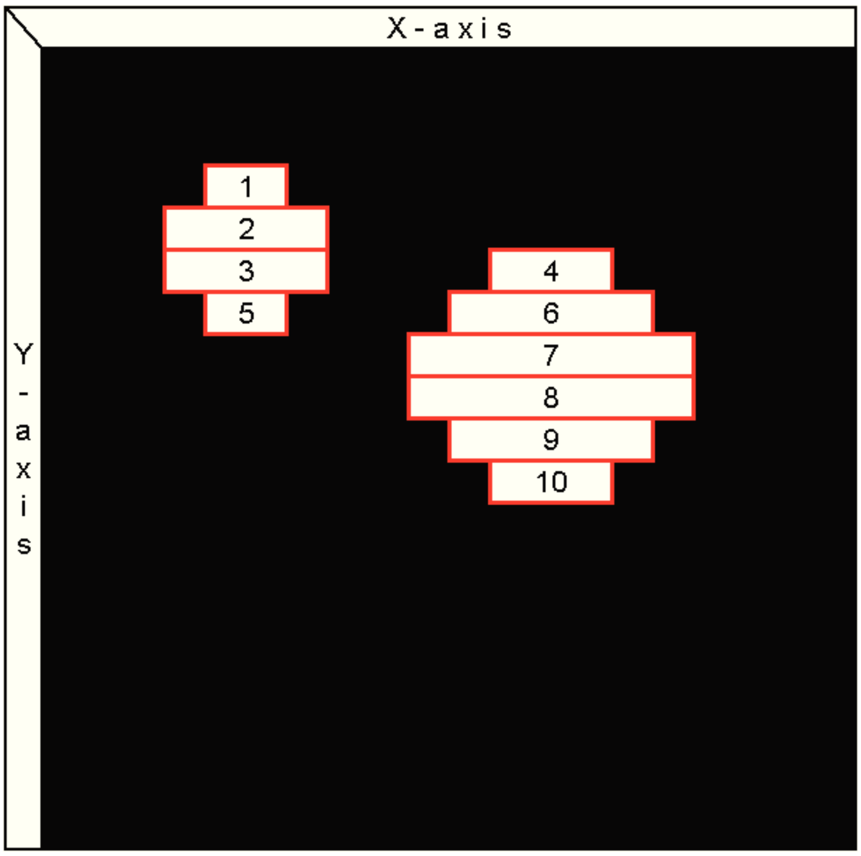


Figure 11. Run-Length-Encoding based on frame pre-processing

The RLE method would allow to process the whole image data in parallel. After the runs in the frame are detected, they can be merged to BLOBs based on their connectivity. The parallelization of the system design could not be covered in the R&D1. Therefore the implementation of RLE had been postponed.

3.1.4 System Design

The hardware design has been developed for the Altera DE2 development and education board [50]. This board uses a Cyclone II FPGA that contains 33,215 logic elements (LE). It is equipped with several communication technologies and I/O interfaces. The different modules of the hardware design have been implemented in Verilog. For the module which controls the AD-converter and receives the digitized video signal, a demonstration project from Altera has been reused and modified for the application purpose. The input video stream had to be provided in NTSC format and was converted into RGB format with a resolution of 640x480. The connection and process flow between the different modules are visualized in Figure 12.

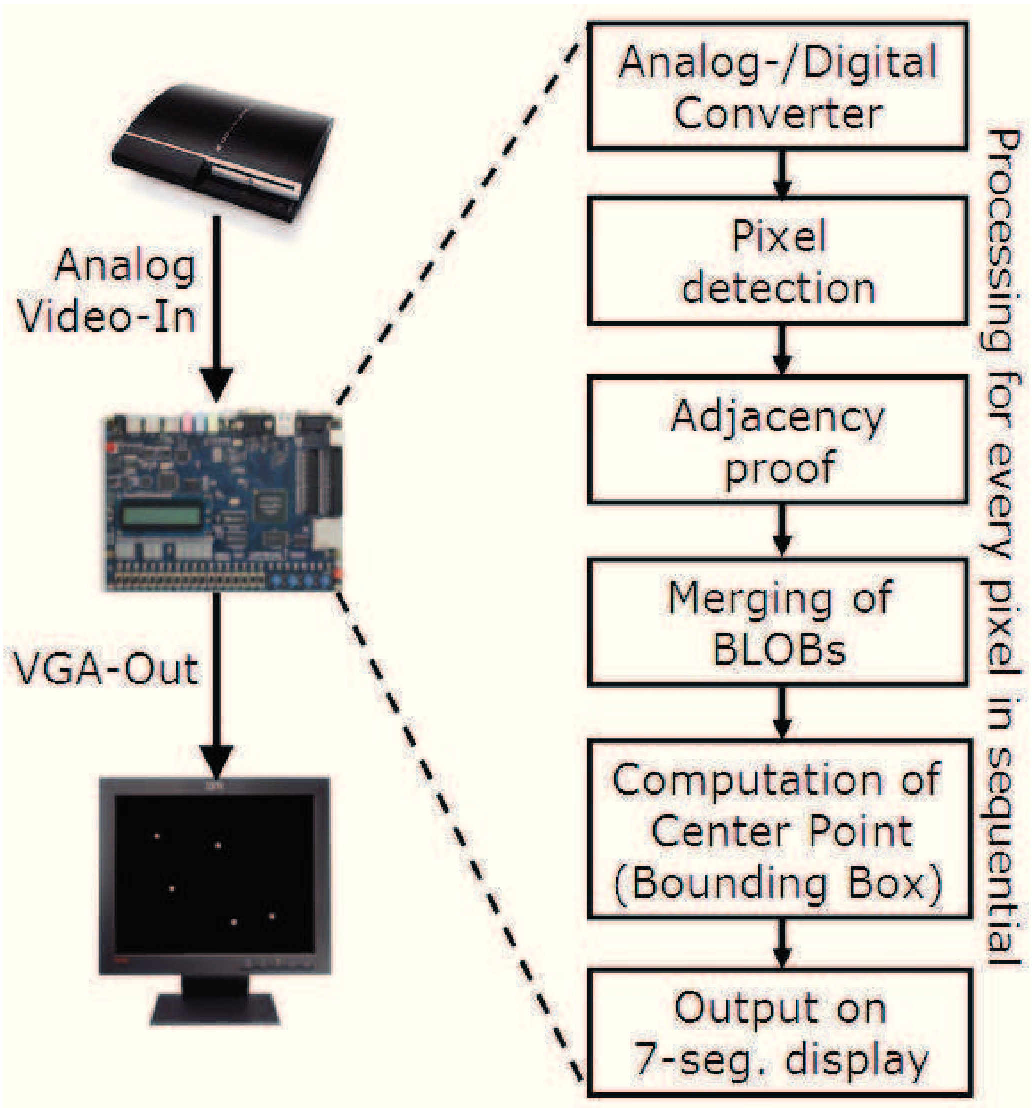

Analog Video-In
VGA-Out
Analog-/Digital Converter
Pixel detection
Adjacency proof
Merging of BLOBs
Computation of Center Point (Bounding Box)
Output on 7-seg. display
Processing for every pixel in sequential

Figure 12. System design for R&nD1 BLOB detection approach

The "Pixel detection" module performs the threshold check with the received data for identifying relevant image data. In the "Adjacency proof" module, the pixels are sorted into data containers. The hardware design uses a structure of registers, allocated in the FPGA, to keep the attribute data for each BLOB. Because of the usage of internal registers, the amount of containers to store BLOB data for the current frame had to be fixed. This design can detect up to five BLOBs in a single frame. To increase the system reliability the module ”Merging of BLOBs” performs an additional adjacency check for all detected BLOBs after the end of the frame is reached. This late merging eliminates data for a single BLOB that has been labeled different and stores it into one container. The module ”Computation of Center Point” uses the BLOB attributes, stored in the container structure, and computes the center points. The results are shown on the seven-segment display. In addition the video stream that is feed into the system on the AV-input, is displayed on the VGA output of the board. For manual observation of the image material, this output has been connected to a monitor. The transition from an analog-video input to a digital RGB output signal is part of the reused demonstration program from Altera.

3.1.5 Evaluation

For the validation of the BLOB detection system the design has been tested by simulation and by execution on the target platform. The applied development environment supports functional and timing simulation for the hardware design using Waveform files (Section 6.4). Because of the required time effort to create Waveform files, only the functionality from the pixel detection to the center point computation has been simulated. The Waveform file contained data for six frame lines with a line size of ten pixels. The simulation of the design showed correct results. It was planned, for the platform evaluation of the BLOB detection approach, to compare the computed results with predetermined ground truth values. This would have allowed an automated precision evaluation. Since the serial communication module could not be finished, the validation for the precision has been made for static images. The results during the first evaluation period showed a systematic error. All computed center points from the FPGA design showed a constant offset of minus three on the X-axis and plus four on the Y-axis (Figure 13). This error showed up for all applied image material.

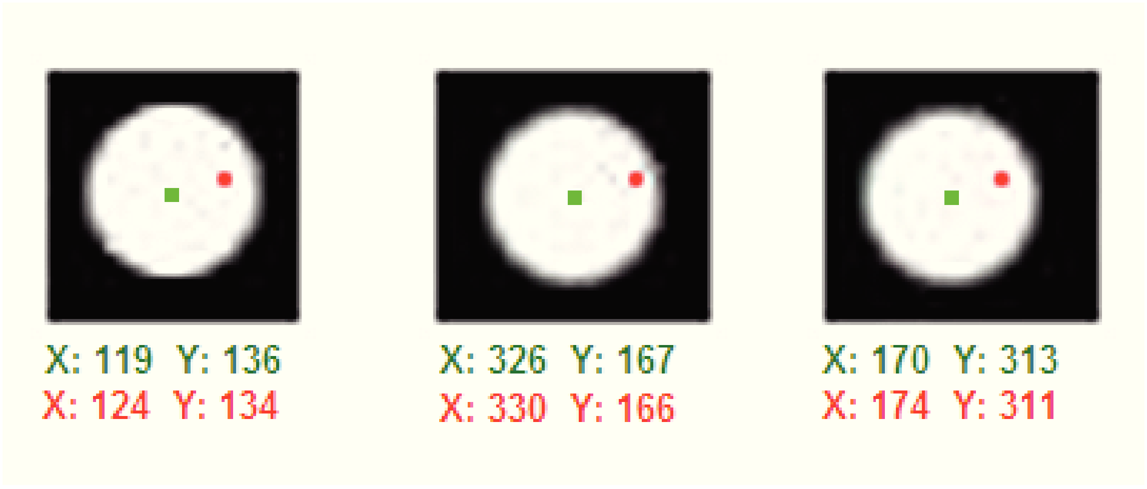


Figure 13. Center point precision error

The green/top coordinate values belong to the green dot in the BLOB, while the red/bottom coordinates describe the position of the red dot. The green results are the so called "ground truth" values and are expected as the correct result. The red results are the outcome of the BLOB detection system, that show the offset error. For the estimation of the ground-truth values a Bounding Box approach on a GPP architecture has been applied. The correctness of those results have been validated by hand. The origin for that offset was caused during the transmission of the image material into the BLOB detection system. For providing the image material on the AV-input a standard DVD player has been used. This DVD player provided the video signal in NTSC format, which means the input image was scaled down. The AD-converter of the FPGA design transformed the video stream into RGB format with 640x480 pixels. This scaling back and forth was part of the problem for the offset error. With the chosen input interface, this issue could not be eliminated. One alternative would have been the implementation of a module for a different input interface technology. That was not possible in the given time for the R&D1 project. The other reason for the offset error was the format of the images on the DVD. All images had a different resolution between 640x480 and 800x600. This was the main problem for the offset in the center point results. By using input images in the final RGB resolution of 640x480, the offset error could be reduced to one pixel. That was the best precision that could have been accomplished, with the two scaling processes of the image material.

For the evaluation of the system's performance, the number of processed frames per second has been counted. The maximum performance of the BLOB detection system was restricted by the processing speed of the analog-digital-converter, that pre-processes the video stream from the AV input interface. In this design the observed performance reached 64 frames per second. The hardware design required approximately 25 % of the FGPA resources with the functionality of processing a single video stream. The modular design of this approach allows the configuration to process three video streams in parallel, which requires 66 % of the FPGA resources.

3.1.6 R&D1 Conclusion

During the R&D1 project a BLOB detection system could be successfully designed in hardware. The approach has been evaluated for precision and performance. The design allows the detection of up to five BLOBs in a continuous video stream. The identification of the relevant pixel is working based on a fixed threshold for the brightness value of the pixels.

The implementation of a serial communication module to validate and process the BLOB detection results remained unsolved and was part of the subsequently R&D2 project. With the evaluation, using static images, problems like motion blur could not be tested truthfully. For the blurred BLOBs, like the examples from 4 and 5, the center point should be closer to the brighter area of the BLOB. A Bounding Box based computation places the center point in the center of the whole BLOB area or even outside of the BLOB for the curved shaped examples. That kind of precision is not acceptable for the application task in the Immersion Square. A small offset in the center point of the BLOBs can cause a huge offset for the estimated position and orientation of the user. The deliberation of other methods that allow a more precise computation of the BLOB's center points are part of the R&D2. One of the main advantages using FPGAs is the ability to design certain parts of the system in parallel. The design of the R&D1 work does sequential processing of the frame data and does not use any memory buffer for temporary storing image data. Using a frame buffer would allow to process all lines of one image in parallel. Realizing such a frame buffer with the given target board would require the application of the external memory modules on the board, since the internal register space of the FPGA is too small to store more than one frame. But a frame buffer should allow to store at least two frames since the reading of the next frame from the input source has to continue while the BLOB detection processes the latest received frame. In addition, only a frame buffer would allow to apply methods like foreground-background segmentation or random access and repeated access to the image data. The implementation of RLE to pre-process the image data has not been covered in the R&D1. It is part of the thesis work. Although the design supports processing three video streams in parallel, the provided DE2 board has not enough AV inputs. If the BLOB detection system for all three projection screens of the Immersion Square should run on a single platform, the application of different technology interfaces is required. In addition, the AV input has been identified as the bottleneck of this approach. It is therefore not likely that the AV input will be used for the final approach. Possible alternatives could be the application of cameras with Ethernet or USB interfaces. A customized camera that is connected on the available pin interface of the DE2 board, would be possible as well. Those solutions require the implementation of a customized circuit that controls the attached devices and receives the image data.

3.2 R&D2 - Acceleration of Image Processing in Hardware

In the R&D2 project the work scope focused on the improvement of the BLOB detection and a comparison of different methods for the computation of its center point [15]. In order to automate the observation of the system at runtime and for further processing of the results, the system has been extended with a serial communication module. This permits storing the data on a connected host computer and visualizing the detection results during runtime. It was planned to include a pre-processing module to combine line adjacent pixel based on a run-length-encoding concept. This module could not be finished within the project timeline and has been postponed. To overcome the bottleneck issues from the AV input of the first approach, a Charge-Coupled Device (CCD) camera has been attached to the target platform (Section 3.2.3). For comparing the Center-of-Mass and the Bounding Box computation methods, their performance and precision have been evaluated.

3.2.1 Center-of-Mass Computation

In order to achieve a higher precision for the center point computation the application of a Center-of-Mass (CoM) based method has been considered. For the CoM based method the BLOB detection module needs to store more information about the identified pixels. The standard CoM method uses the coordinate of the pixels as a weighted sum to estimate a mean center point for the BLOB.

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

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

This algorithm helps to find the center point of a BLOB with respect to its mass. But it does not use the information about the brightness of the pixel, which belongs to the BLOB. Therefore the CoM algorithm had to be modified to fit the proposed application case. Using the brightness values as an additional weight has increased the precision of the CoM approach. It shifts the estimated center point of the BLOB more to the concentration of those pixels with the highest brightness.

$$\text{BLOB's } X\_center\_position = \frac{\sum X\_pixel\ position * pixel\ brightness}{\sum pixel\ brightness}$$

$$\text{BLOB's } Y\_center\_position = \frac{\sum Y\_pixel\ position * pixel\ brightness}{\sum pixel\ brightness}$$

The evaluation of the precision on a GPP architecture showed promising results (Figure 14). Like someone would expect, CoM and Bounding Box showed similar results for BLOBs with perfect circular shape. But for the BLOBs with motion blur, the modified Center-of-Mass based approach was closer to the expected optimal result.

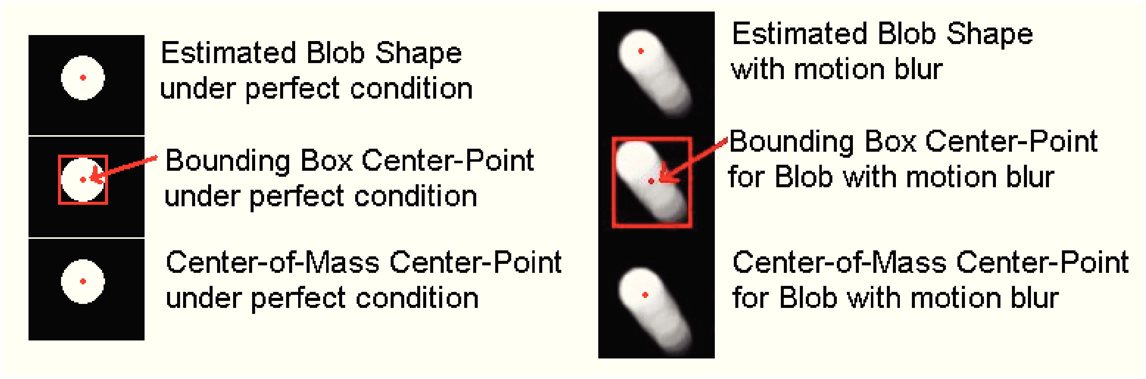

Estimated Blob Shape under perfect condition
Bounding Box Center-Point under perfect condition
Center-of-Mass Center-Point under perfect condition
Estimated Blob Shape with motion blur
Bounding Box Center-Point for Blob with motion blur
Center-of-Mass Center-Point for Blob with motion blur

Figure 14. Example of Center Point Inaccuracy. Bounding box vs. Center-of-Mass.

Other methods have been taken into account but did not show promising advantages. One would have been the inner-circle method [17]. This one places a circle inside of the BLOB, that does not intersect with the BLOB's edges. The concept is the same as Bounding Box and would lead to similar precision results.

3.2.2 Serial Communication Interface

The results of the BLOB detection are required for the estimation of the user's position and orientation in the Immersion Square. This computation was supposed to be performed on the computer, that runs the software of the virtual environment of the Immersion Square. That required the transmission of the BLOB detection results to the computer. The DE2 board had different interface technologies available, such as USB, Ethernet, IRDA and RS232, to send the data. The IRDA interface has not been used because of the required line of sight to the receiving station. In addition a regular computer usually has no IRDA interface. USB and Ethernet would have been selected if a high bandwidth would have been required. But the BLOB detection results only contain the X-/Y-coordinate and an index to distinct the identified BLOBs in one frame. Another problem with USB and Ethernet is the higher protocol overhead, that is required to establish communication. The RS232 controller on the DE2 board offers a maximum bandwidth of 120 Kbit/s [53]. Having the BLOB data encoded in 29 bits the serial communication would be sufficient for the system requirements. The implementation of a control module for the RS232 chip to establish the data transmission was promising because of the smaller code overhead that is required, compared to USB and Ethernet.

3.2.3 CCD camera

One of the concluding statements of the R&D1 work was that the input interface for the image material has created a bottleneck in the design. The only way to solve this was to use a faster input device. Terasic provides a CCD camera that can be directly attached to the DE2 development board (Figure 15) [51]. The camera has a sensor with a resolution of five megapixel and can operate as photo or video device. Its sensor is arranged in a Bayer pattern format and can be read out line wise. The specification of the camera promised a maximum read-out speed of 150 frames per second, which would have been sufficient for a comprehensive performance evaluation of the BLOB detection system. It has a I2C bus based configuration interface to setup the camera controller and an internal phase-locked-loop (PLL). The PLL is used for the read out procedure of the sensor and can be disabled if an external clock is available. The controller of the D5M camera reads one pixel per clock tick and processes the sensor in sequential line order.

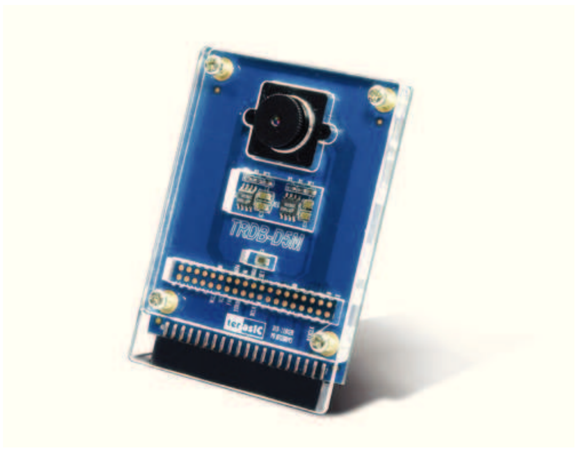


Figure 15. D5M Digital Camera from Terasic.

Having a sensor with a Bayer pattern format required the transformation of the frame data into RGB format. Based on the RGB format the pixel's greyscale representation can be computed. That pre-processing was needed to provide image material which is similar to the expected image material from the application in the Immersion Square. The Bayer pattern is a color pattern that is specialized to the anatomy of the human eye. A human eye is more sensitive to green than to red or blue colors. Therefore, the sensor has two green nodes, one red and one blue for each pixel. Each four nodes are combined to one pixel in RGB format (Figure 16).

3.2.4 System Design

Except for the additional input device the project used the same equipment as in the R&D1 project. The D5M camera required a different pre-processing as compared to the previous hardware design. With the modular design from the R&D1 the pre-processing module for the CCD camera could be placed in the same project design. Figure 17 gives an overview of the processing flow in the updated design. It is not

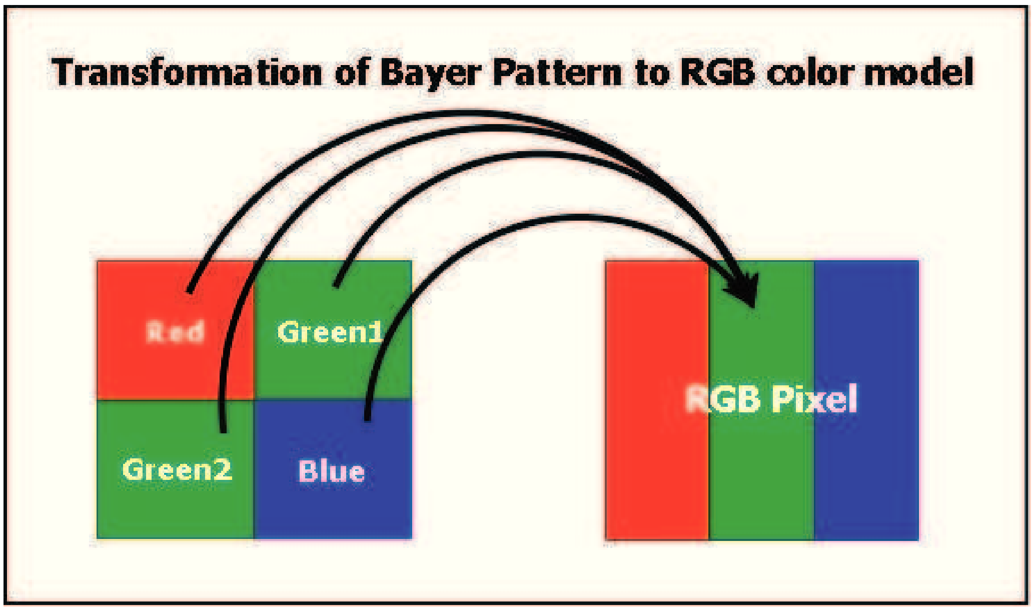


Figure 16. Applied Transformation for Sensor Data of D5M Camera.

possible to run both input devices at the same time, the system has to be configured to use either one of them.

The hardware design has been extended to eliminate runtime conditions between the different modules. Instead of connecting all the modules in the design directly with its predecessor and follower module, the data transmission has been buffered in first-in/first-out (FIFO) modules. The FIFO is a dual-clocked memory unit, created in the internal memory of the FPGA. This permits different modules running with different clock speeds and still be able to transmit data from one module to the other. The FIFO module is provided by Altera as one of many available Intellectual Property (IP) cores, within the development environment [4]. An example for the connection between the modules is given in Figure 18. The synchronization of a read process is based on two signal lines. One is going from the FIFO to the reading module and has a value of one if the FIFO is empty. The other one is going from the reading module to the FIFO and is used to send a read request. The write process is similar, where the writing module sends a write request and the FIFO might respond with a FIFO-full response, if no memory space is available.

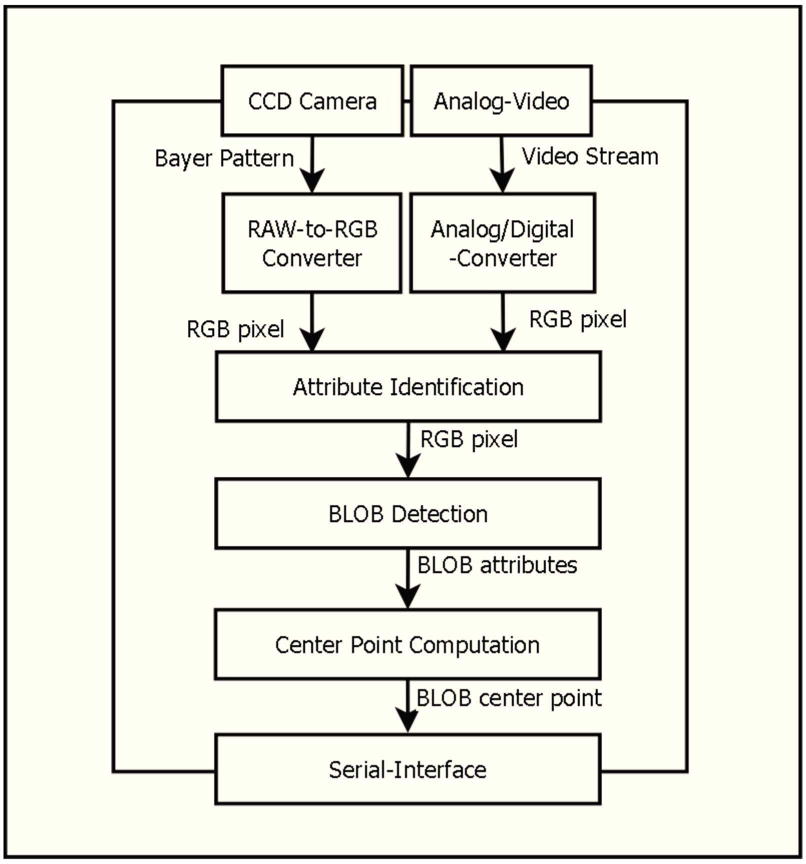


Figure 17. Schematic of the System Design.

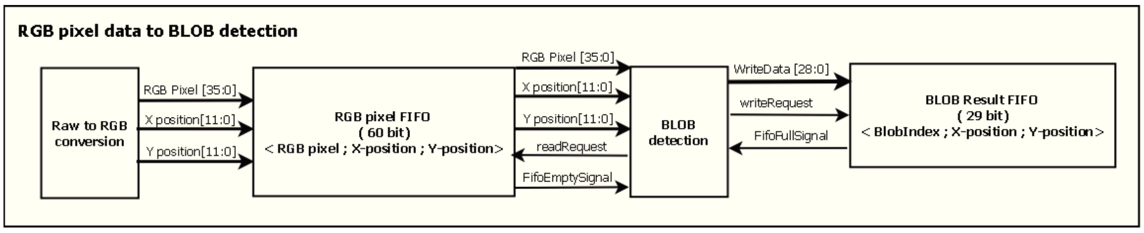


Figure 18. Example of Data Transport via FIFOs between Modules.

To gain more control of the processing flow inside the modules, their internal design has been modified. Each module is setup as a state machine, which requires the whole functionality of the module to be split in certain steps. Every state in a state machine can only perform operations that have no direct dependencies. The attribute identification module performs the threshold check of the pixel's brightness and forwards relevant pixels to the FIFO module from where the BLOB detection module reads its input data. In addition it checks for the start of a new frame and sends a EOF flag, encoded as pixel data. This EOF flag is used in the other modules to identify the end of a frame and drive the process execution. Figure 19 shows the module design for the attribute identification. Several states, such as INIT, IDLE, READ_FIFO, WRITE_FIFO and WAIT_FIFO, can be found in other modules again. Those have been used for the read and write procedures with the FIFOs as data buffers in between the modules.

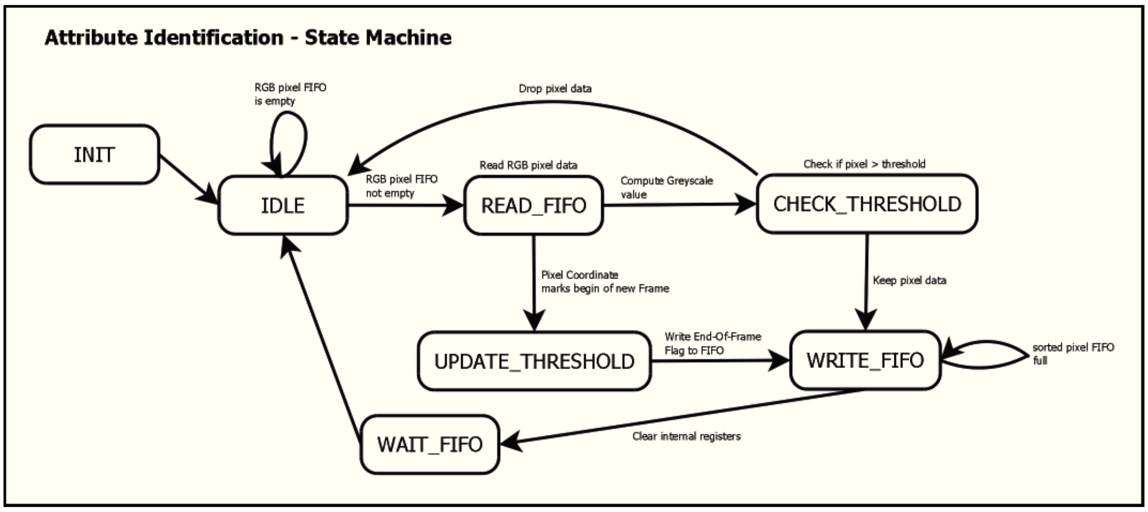
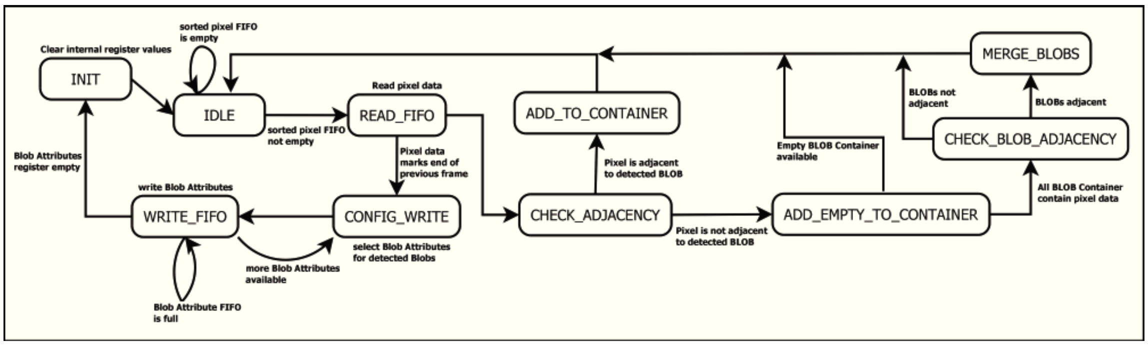


Figure 19. State machine for attribute identification.

Two different methods for center point computation have been examined in the project. The Bounding Box based approach has been reused from the previous R&D1 and modified according to the requirements of the FIFO buffers. For the Center-of-Mass based computation the data to be stored and passed along from the attribute identification to the BLOB detection had to be extended. The CoM based approach required storing the brightness values of the pixels during the attribute identification. This data has been used to compute the median center point of the BLOB, weighted by the brightness values of its pixels. Similar to the attribute identification, the BLOB detection module contains several additional states for the FIFO read and write procedures. It performs the adjacency check, based on the eight-pixel-neighbourhood condition (Section 3.1.1) and sorts the pixels into the

BLOB containers. The results are written into the FIFO from where the serial communication module obtains its input data. Figure 20 gives an overview of the state machine design for the BLOB detection module.



Clear internal register values
INIT
sorted pixel FIFO is empty
IDLE
sorted pixel FIFO not empty
Read pixel data
READ_FIFO
Pixel data marks end of previous frame
ADD_TO_CONTAINER
MERGE_BLOBS
BLOBs not adjacent
BLOBs adjacent
CHECK_BLOB_ADJACENCY
Empty BLOB Container available
Blob Attributes register empty
write Blob Attributes
WRITE_FIFO
CONFIG_WRITE
select Blob Attributes for detected Blobs
more Blob Attributes available
Blob Attribute FIFO is full
Pixel is adjacent to detected BLOB
CHECK_ADJACENCY
Pixel is not adjacent to detected BLOB
ADD_EMPTY_TO_CONTAINER
All BLOB Container contain pixel data

Figure 20. State Machine for BLOB Detection.

A limitation of FPGAs is the division of large numbers. The implementation of division within the module can lead to a slow performance of the module. This occurs because of the large amount of registers that need to be wired inside the FPGA to store the number. Therefore it is common to pipeline such critical operations in separate modules or use IP cores if available. The Altera IP cores library provides a core for division of integer numbers, which has been applied for the current design. The core has been integrated into a block design file, that can be instantiated as a sub-module in the associated module. Figure 21 is showing the block design diagram for the division operation, that has been used in the Center-of-Mass based computation module.

3.2.5 Evaluation

For the verification and validation of the BLOB detection results two different input sources have been applied. The S-Video input for the verification of detection accuracy and precision of the different center point computation methods. The CCD camera has been used for performance benchmarking of the system design. Both alternatives allow the observation of the image material while performing the BLOB

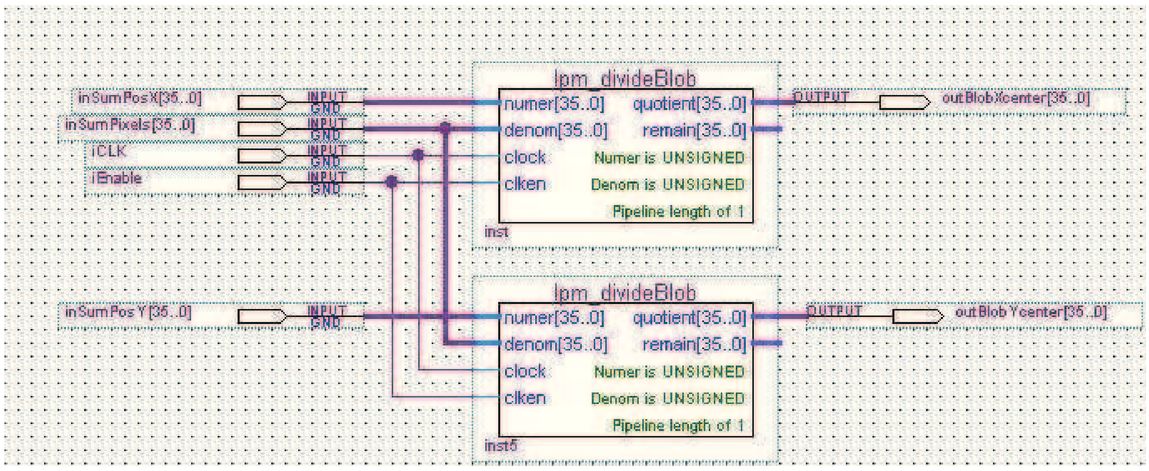


Figure 21. Block diagram for division part in Center-of-Mass computation.

detection, but only up to a certain speed. For the performance measuring, the visualization of the captured image data from the CCD camera on the VGA display had to be disabled. This was necessary because the VGA controller module did not support higher frame rates.

Based on the specified application area, the shape of the BLOBs to be detected has been estimated as perfect circular and circular shapes with blur effect. The perfect circular shape is given in the proposed application task if the light source, which will be tracked on the projection surface of the Immersion Square, is kept still or the motion is slower than the frame rate of the applied camera device. With faster movement of the light source a blur effect will occur. The angle between light source and the projection surface is another factor which can cause a distortion of the BLOB shape. This distortion might look similar to a blur effect. If the angle is small enough the BLOB's shape will deform into an elliptical form. But in contrast to the blur effect the brightest spot of the BLOB will still be in the center. For both problems the solution is the Center-of-Mass based computation of the BLOB's center points, which has been applied in the system design. Samples of the applied image material for evaluation of the precision is given in the Appendix.

For applying the BLOB detection system it needs to be configured before using the BLOB detection results for further processing. For the applied image material the computed results are shifted by a constant factor on the X and the Y axis. A calibration of the system for each new application environment is expected by the user, to confirm the proper results.

Precision

For the computation of the BLOBs' center point the Bounding Box and the Center-of-Mass based method showed the exact same results for the clear BLOBs with a perfect circular shape. If a BLOB is not showing any blur effect the, applied method for computing the center point has no influence on the precision. This holds for all applied threshold values. A summary is given in Table 1 and results are visualized in Figure 22.

	Bounding Box		**Center-of-Mass**	
Threshold	**Y**	**X**	**Y**	**X**
0x190	234	311	234	311
0x20B	234	311	234	311
0x286	234	311	234	311
0x301	234	311	234	311
0x37C	234	311	234	311
0x3E0	234	311	234	311

Table 1. Results for Center Point Computation with Clear Shape BLOBs.

The table shows what was expected already. If a BLOB is not showing any blur effect the applied method for computing the center point has no big effect on the precision. In the given results it can be observed that the estimated center point for both methods is at the exact same position.

The Bounding Box and Center-of-Mass computation showed different results for the image material showing BLOBs with blur effect. The Center-of-Mass results turned out to be closer to the BLOB's center point. Results for one particular example are shown in Table 2 and are visualized in Figure 23. Again a proper

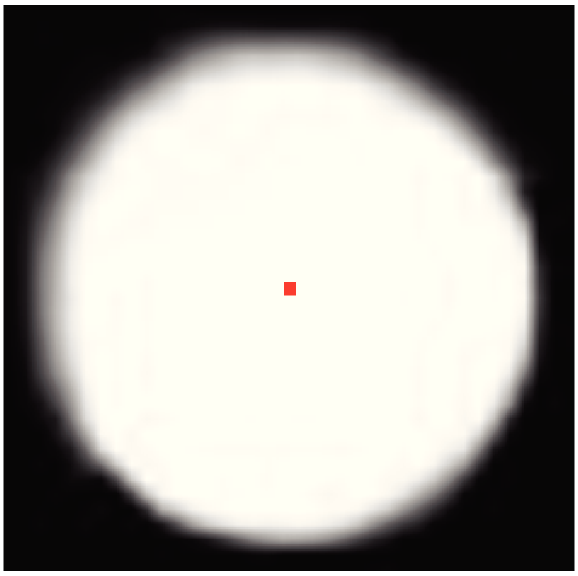


Figure 22. Clear Shape BLOB Center Point Computation.

calibration of the system is required to guarantee correct results.

	Bounding Box		Center-of-Mass	
Threshold	Y	X	Y	X
0x190	226	308	229	307
0x20B	227	308	229	307
0x286	228	307	230	306
0x301	231	305	233	304
0x37C	236	300	238	300
0x3E0	241	299	241	298

Table 2. Results for Center Point Computation with Blur Shape BLOBs.

Center point computation with Center-of-Mass shows higher precision for BLOBs with blur effect, compared to Bounding Box. With the applied visualization on the VGA output the frame rate of the blob detection was restricted to 12 frames per second. The threshold values which have been used for evaluation of precision are in-between the value range of the BLOBs in the applied image material. This value range is dependent on the applied image material and can not be simply reused for any image source or material. The estimation of the value range is a configuration requirement before using the BLOB detection system. For threshold values above

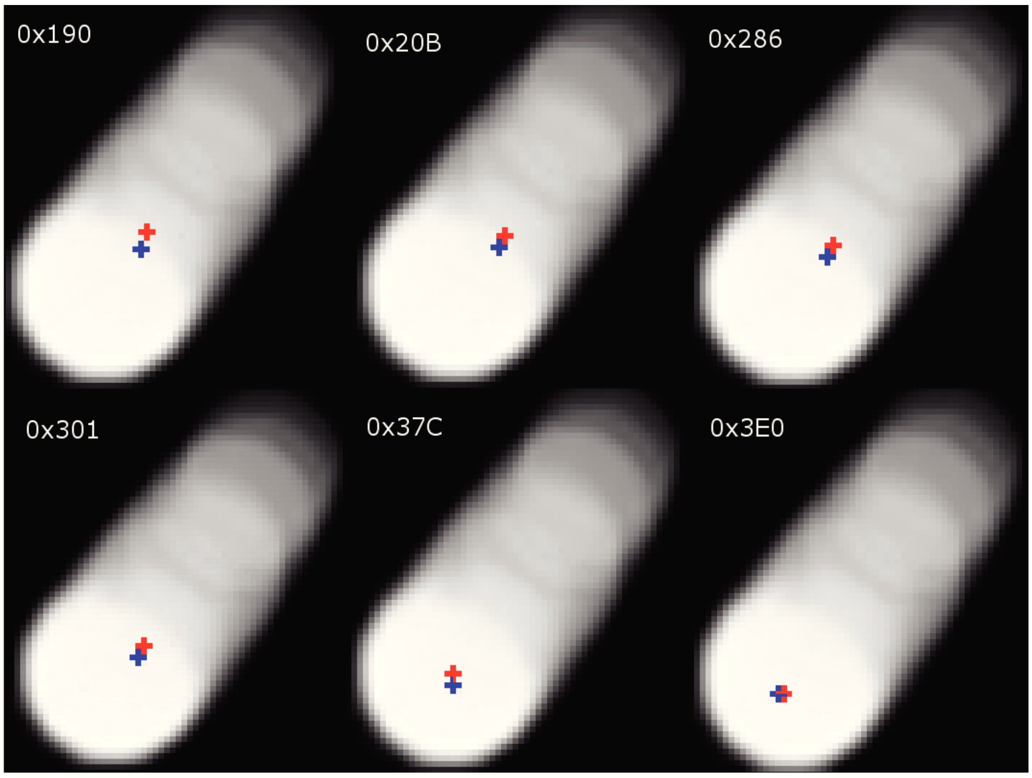


Figure 23. Blur Shape BLOB Center Point Computation.

or below the value range the system was not able to detect all BLOBs in the image material accurately.

Performance

The system performance has been evaluated on a fixed environment setup. Reported values about performance and resource allocation are given in Table 3.

The performance result "fps." refers to the obtained frame rate during the benchmarking. The "camera speed" is the particular clock rate that is used to read out the CCD sensor. "System Speed" is the clock rate for the BLOB detection module during the performance test and "Max. System Speed" describes the maximum possible clock rate for the implemented design.

For the evaluation with monitor output a faster frame rate would have been possible in theory, but it would have required time consuming configuration of the camera

	Monitor Output		No Monitor Output	
	BB	CoM	BB	CoM
Speed (fps.)	12	12	46	50
Camera Speed (MHz)	25	25	96	96
System Speed (MHz)	40	50	125	125
Max. System Speed (MHz)	72	65	140	189
Allocated Resources on the FPGA				
Logic Elements	7,850	14,430	5,884	13,311
Memory Bits	147,664	273,616	113,364	239,316
Registers	2,260	2,871	1,510	2,078
Verified	*	*	*	*
Bounding Box (BB) Center-of-Mass (CoM)				

Table 3. Resource Allocation and Benchmark Results.

settings and the VGA controller module. For monitoring the image data while performing the BLOB detection the VGA module has to run synchronized with the image capturing module. The applied CCD camera has been used for the evaluation of faster frame rates. It will not be used in the target application for the active tracking device of the MI6 project, because of its missing ability to record infrared light. For this reason it was reasonable to not put more effort into the integration of the CCD camera than was required for the performance measuring.

The CCD camera itself has an average speed of 48 frames per second for the applied configuration parameters. While both BLOB detection approaches would have been able to perform on faster frame rates, the estimation of the maximum performance was again restricted by the input source. The same problem about slow input sources existed in [14] as well and did not permit testing the system for maximum performance. The resource allocation shows that Center-of-Mass requires about twice as many logic elements and memory bits compared with Bounding Box.

3.2.6 R&D2 Conclusion

The project results offer several opportunities for future work. The additional preprocessing of the image data with RLE could increase the center-point precision.

With foreground-background segmentation of consecutive frames, the blur effect of moving BLOBs could be reduced. In addition, with the calibration of the average BLOB-size, the detection result could be checked for missing or false-positive pixels.

It has been shown in Section 3.2.5 that the BLOB detection could not be tested for its maximum performance. Both available input sources turned out to be a bottleneck for the acquisition of image material. A recommended step would be the integration of the BLOB detection approach into a camera with an onboard FPGA. With direct access to the sensor of a high performance camera, the BLOB detection can be evaluated for its maximum processing speed.

The detection of BLOBs in its current implementation is working for up to five BLOBs. The number is based on the required points for estimation of position and orientation of the user in a CAVE environment [56]. The extension for detecting more BLOBs is possible. For the detection of BLOBs the system does not track the movement of the BLOBs. The index for the detected BLOBs is based on the order of how the frame is processed. This works usually from the top-left corner to the bottom-right corner of the frame. The continuous tracking of BLOBs would be another possible functionality to improve the system. This would allow estimation of the speed and direction of the BLOBs movement as well.

For the continuation of the MI6 project the BLOB detection system needs to be tested with the Immersion Square [28]. This requires the extension of the FPGA system with a specialized infrared camera as the input source or the integration of the BLOB detection into an infrared camera.

With Bounding Box and Center-of-Mass two common methods for the estimation of a BLOB center point are now available in hardware. The evaluation has shown reliable precision results with respect to the given application area. As it has been estimated in [14] the Center-of-Mass based approach shows higher precision and is the recommended solution for the given application task.

The BLOB detection approach is working for threshold based and specialized for BLOBs which consist of white and light-grey pixels on a black image background. The BLOB detection has turned out to be non-critical with respect to performance and timing requirements for the proposed application task. Both available input sources have been identified as a bottleneck for the system's processing speed. This has proven the FPGA design to be well qualified for the proposed computer vision problem, if faster input sources can be provided.

For the evaluation and further processing a module for transmitting results on the serial interface has been designed, tested and applied. The maximum performance of the serial interface is high enough for even faster frame processing rates, as described in Section 3.2.2. The output format of the computation results can be easily changed in the system design. Or if further information about the BLOBs is computed on the FPGA system and needs to be transmitted as well, such as direction of movement or speed of the BLOBs.

This project has shown that a BLOB detection system can be successfully implemented and evaluated on a FPGA platform. Validation and verification showed reliable results and the advantages of image processing tasks designed in hardware. The work required a higher time effort compared to the implementation of a similar system in a high-level language, such as C++ or Java. But for customized solutions with high demand on performance and precision specialized hardware outperforms high-level software implementations. FPGAs are proven to be a good way for short prototyping development cycles to decrease time-to-market.

4 System Design

The general system design is based upon the approach of the R&D2 (Section 3.2). It supports both input devices, analog-video and the CCD camera. In addition the design has been extended to use a customized industrial Gigabit Ethernet camera with Complementary Metal-Oxide Semiconductor (CMOS) sensor from Matrix Vision (Section 6.6). Due to the performance requirements and availability, the CCD camera was mainly used during the development phase of the project. The measurement of the performance and precision has been done using the CCD camera and the mvBlueCOUGAR-X [38]. Figure 24 gives an overview of the modular design, that has been extended by the additional processing parts of this work. For more accuracy in the estimated BLOB center points the system has been extended by a four digit sub-pixel precision. In order to increase the flexibility of the system an additional USB communication module for the transmission of the detection and tracking results has been designed.

For the comparison of the tracking approach, this functionality had to be realized as a software application and a hardware design. In the following the design of both approaches and the concept of additional pre-processing modules are presented.

4.1 Tracking in Software

For the given application task it had to be taken into account that some of the light dots might not show up for a certain amount of time. This can happen because the light dots are spread over two sides of the Immersion Square, if the user is looking at one of the cubicle's corners. Another reason can be, that the illumination time for a particular frame was too short for the light dots to show up or they show up too dark. Therefore, the idea was to not only perform the tracking based on the latest frame data, but also keep a history of the neighbours of each BLOB. The general flow of the tracking procedure is shown in Figure 25.

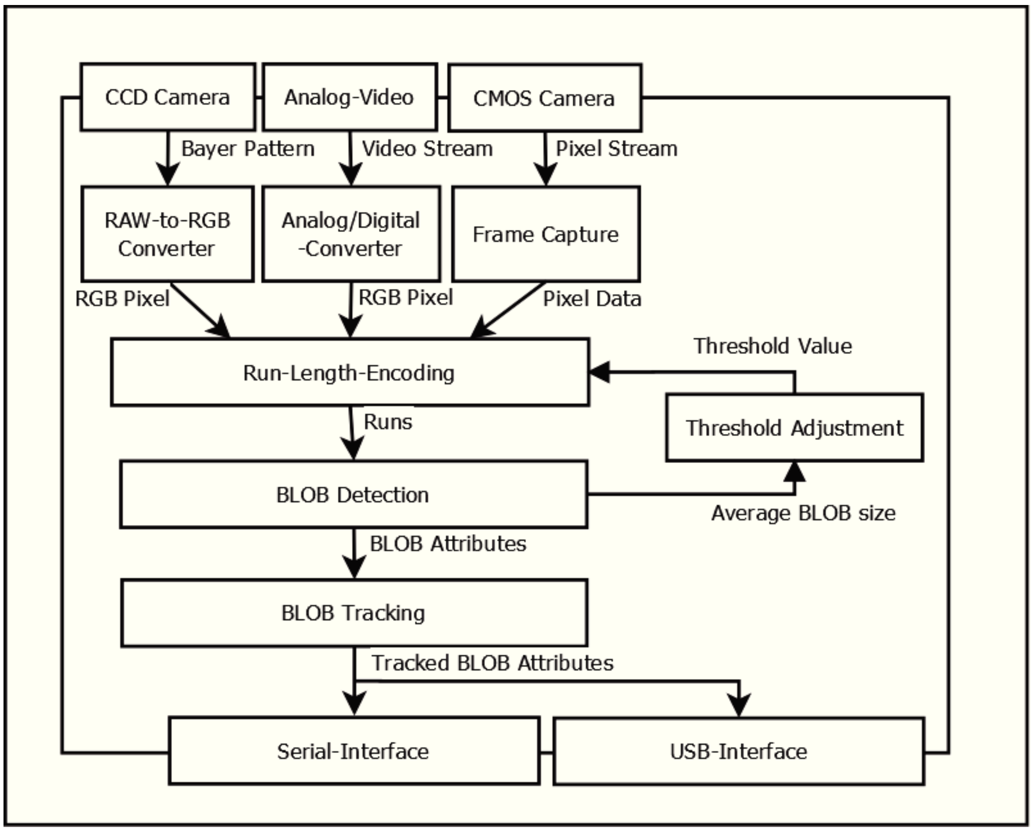


Figure 24. Schematic of the System Architecture, Including the Tracking Approach.

The first step of the tracking algorithm is to compute the euclidean distance between the BLOB results of the previous and the latest frame. Based on the distance, the BLOBs of the previous frame are matched to the latest frame. Every old BLOB can be assigned only once to one new BLOB and vice versa. To assure that an old BLOB is not just assigned to a new BLOB, because there are no other BLOBs left, the euclidean distance is only applied up to a certain distance. This distance is less than half the averaged distance between all the BLOBs in the frame. The value for this reference distance is continuously updated every time the tracking method has processed the latest set of BLOB results. This means that the tracking by euclidean distance fails if the BLOBs between two consecutive frames move greater than half the distance between the BLOBs. The reference points used to estimate the distances are the center points of the BLOBs.

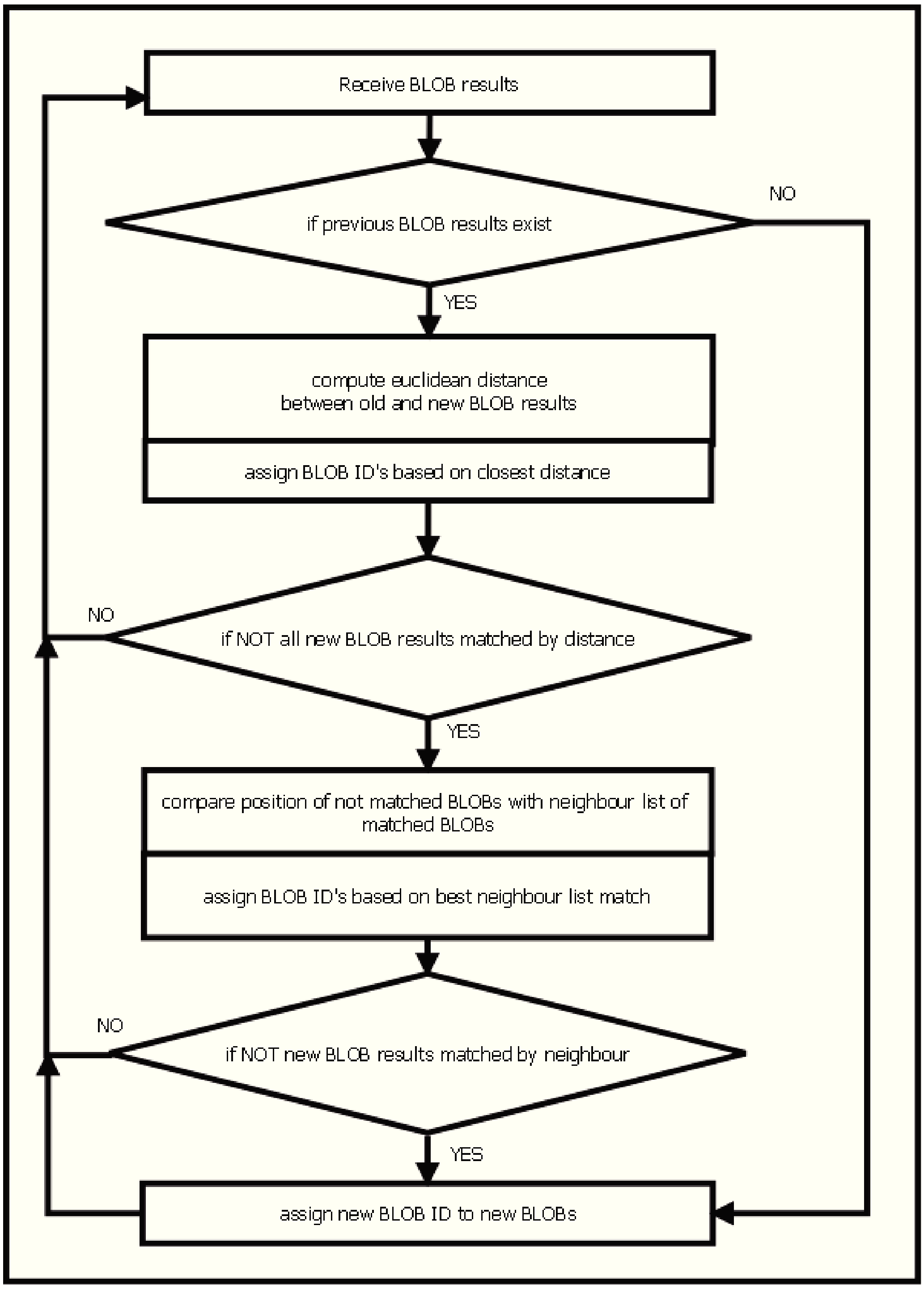

Receive BLOB results
if previous BLOB results exist
NO
YES
compute euclidean distance between old and new BLOB results
assign BLOB ID's based on closest distance
NO
if NOT all new BLOB results matched by distance
YES
compare position of not matched BLOBs with neighbour list of matched BLOBs
assign BLOB ID's based on best neighbour list match
NO
if NOT new BLOB results matched by neighbour
YES
assign new BLOB ID to new BLOBs

Figure 25. Processing Flow of the Tracking in Software.

For the BLOB results of the latest frame that could not be matched by euclidean distance, a feature list of the matched BLOBs about their distance to their neighbour BLOBs is considered. For each new and unmatched BLOB, their position is compared to the expected position of the neighbours from the BLOBs that could be matched. The algorithm performs the search for the new BLOB and selects the BLOB ID in the neighbour list of the matched BLOB which is most likely the previous position of the new BLOB. For every new BLOB that could not be matched an unused ID is assigned.

A BLOB that just shows up once will not be displayed or recorded in the output file. Every BLOB in the list of tracked BLOBs has a counter that is increased with every successful tracking and decreased for every time the tracking fails. If a BLOB is tracked five times, it gets recorded to the output file and displayed on the screen. This should make sure that if a BLOB could not be tracked correct for a few frames, it would not appear as a new BLOB. Once a BLOB is tracked more than twenty times, it is classified as valid. This should make sure that the reference BLOBs reach a stable state where the values can be expected to be correct. Only "valid" BLOBs will be used for the neighbour list classification. The counter will stop increasing after one-hundred successful trackings. The values were selected with respect to the frame rate, which was expected to be at least 30 frames per second. A BLOB would be displayed after 16.67 ms and classified to be valid after 66.67 ms. Regardless how often a BLOB is tracked successfully, if it cannot be matched to a new BLOB result, its visualization and recording is disabled immediately. The BLOB will stay in the list of tracked BLOBs until it can be matched again successfully by the neighbour feature list or the tracking counter reaches zero. Different values might be required for changing conditions on the application environment.

4.2 Tracking in Hardware

The design approach for the tracking in hardware works similar to the tracking in software. It uses the euclidean distance between the BLOBs center points of two consecutive frames to perform the tracking. The capabilities of an FPGA enabled the computation of the euclidean distance between one new BLOB and the set of old BLOBs in parallel. Other steps, such as the identification of the closest new BLOB, have to be done in sequence to avoid multiple assignments of the same BLOB ID. For control of the process flow, the algorithm has been broken into smaller steps that have no data dependencies. The processing of those different steps is driven by a state machine design. Figure 26 gives a simplified representation of this state machine. Some states, such as "write results to output fifo" consist of several substeps. The output of the results requires several steps, because all BLOBs are written to the output FIFO in sequence and terminated by a result value that specifies the end of the frame.

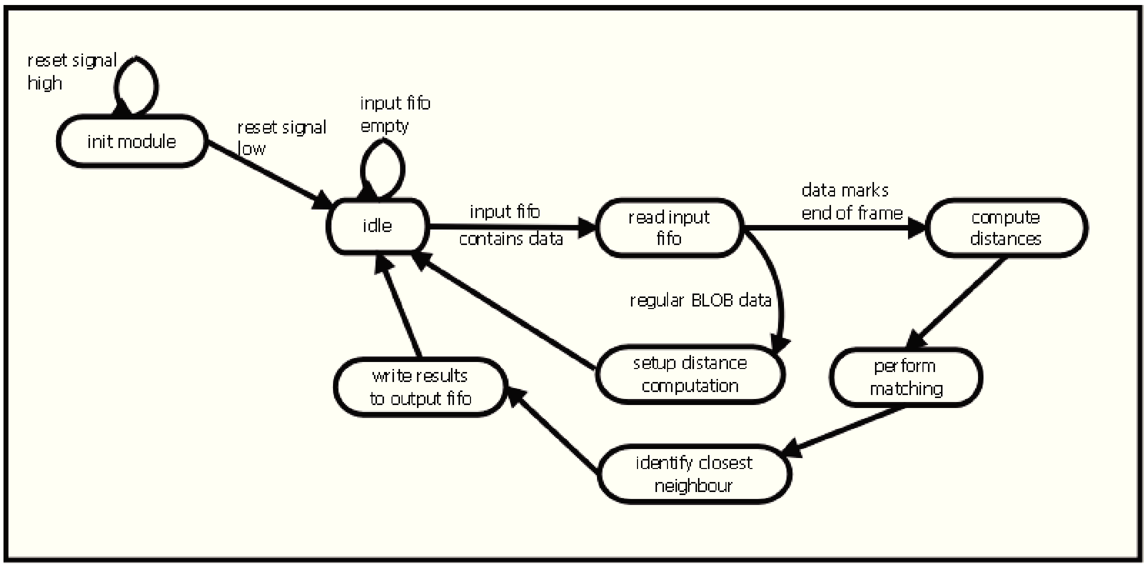


Figure 26. Tracking in Hardware State Machine Design.

The additional tracking procedure, to match the new BLOBs by a list of expected neighbour BLOBs if they could not be matched by euclidean distance, has not been designed in hardware. The software approach uses a flexible index based addressing, that changes dependent on the previous tracking result and nested loops. The number of neighbouring BLOBs can change which increases or decreases the length of the list. Such flexibility cannot be easily implemented in an HDL. Variable index based addressing is not supported by Verilog, since a variable is translated into a hardware register and not into a memory location, like in a software application [32]. The change of an index would mean the selection of a different register in the hardware circuit during runtime and the flexibility of the list of neighbours changes the processing flow of the hardware design. For an appropriate implementation in hardware, it would have been required to realize all possible combinations of matches for a fixed size of BLOBs and a maximum number of neighbours.

Such a design becomes inflexible for changes and has poor performance due to the number of registers and logic units that have to be allocated for the design. If a hardware design file acquires many resources on an FPGA the interconnection wires getting very long. This will cause the overall system's performance to decrease.

4.3 Variable Threshold Adjustment

The identification of the pixels that might belong to a BLOB in the processed frame is working based on a threshold value check. In the hardware design of the R&D1 and R&D2 the configuration of this value is done by the user using the I/O interfaces of the target platform. The threshold check is working similar to a high-pass filter. Only pixels which have a high enough brightness value are forwarded to the next processing step. In the given application case the brightness of the spots from the light emitting device can vary. Several factors will have an impact on the BLOBs, such as the angle between the light source and the projection surface, the distance between the light source and the projection surface or the speed that the user is moving the light source in the environment. This causes the BLOB's shape or the brightness value to change.

One problem with the "manual" configuration of the threshold value is that a human being is not able to change the system's settings fast enough for real-time requirements. He will not realize the absence of the BLOBs or that they are changing until several dozen frames have been processed already. Therefore, the idea is to automatically change the threshold value during runtime. The user will be provided with a setup feature to configure a minimum and a maximum value for the threshold. In addition the user configures a reference size for the diameter of the BLOBs on the X- and Y-axis. The intent of the automatic threshold adjustment is to change the value if the BLOBs size and the reference size are too far off. Doing this automatically allows the system to react to changes of the BLOBs within a few frames. Using minimum and maximum values will ensure that the system does not run into under- or overdrive. If the threshold value would change to greatly, the BLOB detection would stop working properly. With a threshold that is too high, no more pixels will be detected. Having a threshold that is too low, the whole frame will occur as one single BLOB.

For the continuous update of the threshold value, the module will receive an updated averaged BLOB size of the last frame processed. The automatic adjustment of the reference values shall be performed once the user has finished the calibration of the system after startup.

4.4 Pre-Processing Run Length Encoding

For the application of the Run-Length-Encoding (RLE) as a pre-processing task in the BLOB detection system, an analysis of the algorithm has been performed. It was analyzed for data dependencies to parallize parts of the process. The analysis identified five conditions that had to be checked for every pixel to be processed to achieve a reliable RLE of the image data.

1. Is the current pixel brightness higher than the threshold value?
2. Is the current pixel in the same frame line as the previous pixel?
3. Is the current pixel part of a new frame?
4. Is there a run in the same line, that has no end point so far?

5. Is the current pixel adjacent to that run?

With condition 1 the start pixel of a run or a pixel that belongs to that run is identified. A run cannot be longer than one single frame line. The condition 2 is required to terminate a run at the end of a frame line. If the beginning of a new frame is detected with condition 3 the module creates a data flag that notify the following modules about the start/end of a frame. If condition 3 is fulfilled the condition 2 is invalid and the run is terminated. The condition 4 is required if the detection of a run is in progress. This condition will remain ”true” until the end point of that run has been detected. With condition 5 the adjacency of the current pixel and the ongoing run is checked. It also allows the identification of a run’s end point. Every time a complete run has been detected, it is transmitted to the BLOB detection module for further processing.

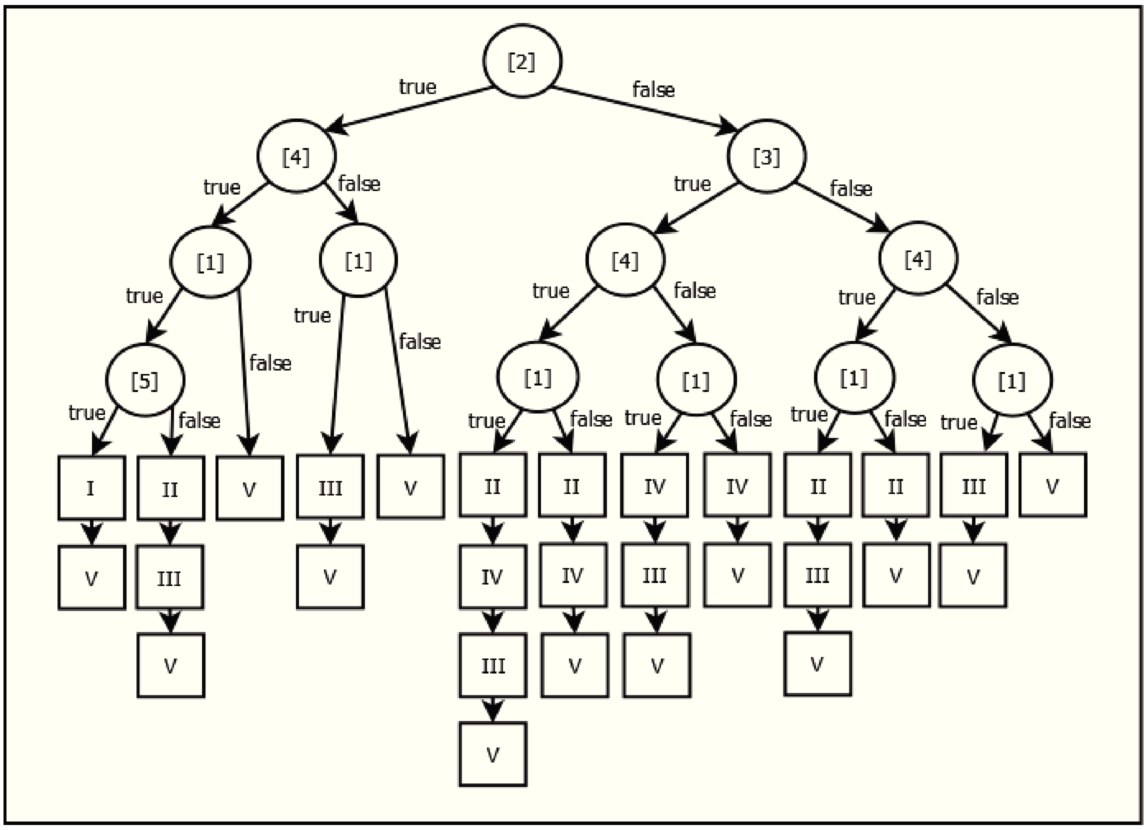


Figure 27. Decision Tree for RLE Conditions and Processing Steps.

Based on these five conditions a decision tree has been created, to identify patterns and dependencies in the processing flow (Figure 27). Different combinations of the conditions could be grouped together, since they require the same processing steps for a correct RLE encoding of the pixel data. This allowed an optimization of the processing steps for the implementation in hardware. The RLE processing has been separated into five steps, that are executed in different orderings, dependent on the condition check results.

I Add the current pixel to the ongoing run.

II Write the run to the output FIFO.

III Start a new run.

IV Write the end-of-frame flag to the output FIFO.

V Read a new pixel from the input FIFO.

For a correct result the input data for the module has to be provided for sequential processing. The processing concept of RLE allows the use of interlaced and non-interlaced frame data. The only requirement is that the order of the frame data stays consistent. For the RLE encoding the input data has to contain the brightness value for the pixel and its coordinates.

4.5 Subpixel precision

The previous BLOB detection approach was designed to use integer values for all kind of data values in the system. With the applied computation methods for estimating the BLOBs center points, the results usually showed a rounding error. To solve this problem the design should be extended by a subpixel precision for all values that are in relation to the BLOB detection results. One requirement was that this feature should be flexible to configure.

4.6 USB Communication Module

In the previous work the BLOB detection results had been transmitted on the RS232 interface. The bandwidth was sufficient for the existing approach, since the results did not contain very comprehensive information. The intent to extend the result values by subpixel precision and the system with a more common interface for a higher flexibility motivated the design of a communication module for the USB interface of the DE2 board. The module should support the same input data as the serial module and allow a higher data throughput. This would also allow to extend the content of the BLOB detection and tracking results, which are transmitted from the target platform to the connected host-PC.

5 Implementation

The Altera development environment Quartus II (ver. 9.0) has been used for the implementation of the hardware design modules (Section 6.2) for the DE2 and DE2-70 target platform. The tracking of the BLOBs in software has been implemented in C++ using Microsoft Visual Studio 2008 as the development environment. In the following Section the implementation concepts of the different modules are presented.

5.1 Tracking in Software Implementation

One requirement for the tracking in software was that it should work on the same input data as the tracking in hardware. In addition the bandwidth of the serial communication interface was too small for transmitting all pixel data of the BLOBs from the FPGA board to the connected host-PC. Therefore, the data used for the software tracking approach contained the center point of the each BLOB and the largest diameter on its horizontal and vertical axis.

For better work balancing of the software approach the Intel(R) TBB library version 3.0 has been used to create a multi-threaded application structure [34]. This supports separate processing tasks, such as receiving data or tracking and visualizing the BLOBs, to run in parallel. The received BLOB results are collected in a result list by the task that reads from the serial buffer. This list is passed over to the tracking task using a concurrent queue. The queue is a data buffer from the TBB library, that allows non-blocking read and write access.

For the graphical visualization of the tracked BLOBs, the Open Source Computer Vision (OpenCV) library version 8.3 for C++ has been used [1]. The visualization of the BLOBs on the host-PC serves for debugging and demonstration purposes.

5.2 Tracking in Hardware Implementation

The tracking has been designed in Verilog, using IP cores and block-design files to realize sub-modules. The several steps for computing the euclidean distance have been pipelined in a module, shown in Figure 28. Each block represents an IP core that performs a mathematical operation, such as multiplication, addition and taking the square root. Since the IP block for taking the square root was only available for floating point numbers, the coordinate values had to be converted accordingly. With respect to the detection module the tracking is working for six to eight BLOBs.

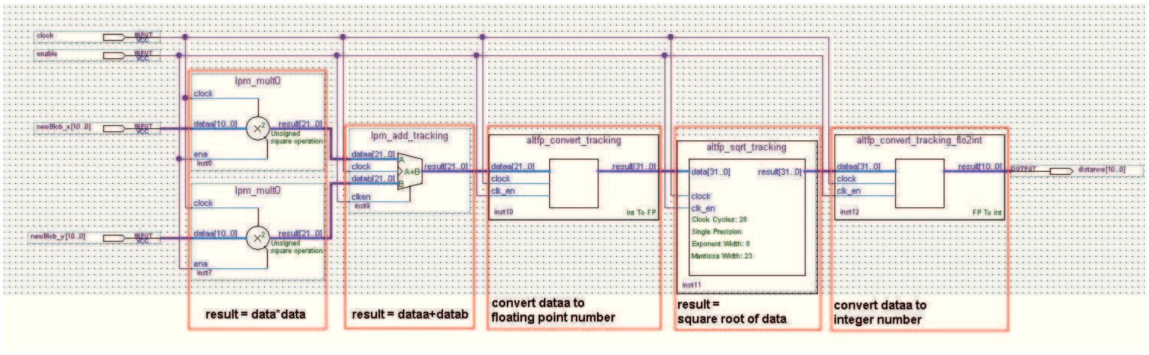


Figure 28. Pipelined Computation of Euclidean Distance.

All values that are feed into the system, processed and send out are register based integer values. The value range depends on the register size of the variable declaration. Assignments between variables of different register sizes in Verilog and VHDL are valid. However, the developer has to be aware that for assignments from larger register variables to smaller register variables, the larger variable gets trimmed down to the size of the smaller one from the most significant bit on. For assignments from smaller register variables to larger ones, the register bits outside the value range of the smaller variable will be undefined. To avoid such situations the developer should either use fill bits to equalize the size of variables, when assigning values or use only variables of equal size.

For the processing control of the tracking procedure the program logic has been realized as a state-machine based implementation. The computation of the euclidean distance for each new BLOB result and the tracked BLOBs of the previous frame is done in parallel. The evaluation of the results to perform the tracking had to be implemented in sequential. For straighten out runtime dependencies, some of those result evaluation steps have been encapsulated into nested state machines. This increases the number of processing steps inside the tracking module. However, it gives more control over the processing time for conditional checks and value assignments.

For the visualization of the tracking results the BLOB data is transmitted via RS232 to a connected computer. The received data is written into a log file and visualized in an OpenCV based C++ program.

5.3 Variable Threshold Adjustment Implementation

The module for the automatic threshold adjustment has two processing modes, "setup" and "running". Using the switches 2 to 4 of the DE2 board, the user can swap between both modes during runtime. The reference values for the size of the X-/Y-axis, the threshold value for the BLOB detection and the minimum and maximum values for the threshold are initialized with fixed values at startup. The module receives the averaged values for the BLOB's size from the BLOB detection module after each frame. If the size of the BLOB is not within the tolerance level, the threshold value is changed accordingly. The automatic adjustment of the threshold value only works in "running" mode. For setting a new initial threshold value, the user has to activate the "setup" mode with switch 2. Changing the initial threshold value also changes the reference values for the X-/Y-axis of the BLOB. The module uses the averaged values for the BLOB size of the detection module. Once the user sets the system back to "running" mode, the reference value for the BLOB size will remain fixed. For changing the maximum threshold value, switch 3 has to be enabled. Switch 4 is used to allow the user to configure the minimum threshold.

If all three switches are disabled, the system is in ”running” mode and performs the BLOB detection with the given configuration. The computed threshold value is transmitted to the BLOB detection module and updated every clock cycle. In all setup modes the configuration values are shown on the seven-segment display of the DE2 board. When the user is setting the initial threshold value, the averaged size for the BLOB's X- and Y-axis are shown as well.

5.4 Run-Length-Encoding Implementation

The implementation of the Run-Length-Encoding based pre-processing was done according to the optimized decision tree for the conditional checks and processing steps from Section 4.4. The conditional checks are performed in a single state in parallel. Based on the outcome the matching RLE processing step is executed in the following state. Each combination of results for the conditional check is mutually exclusive. This allows implementation of all processing steps in parallel in one state. Once a complete run is detected, the run's data is sent to the output FIFO before processing the next pixel data. Figure 29 gives an idea of the functional steps inside the module for RLE encoding of the BLOB data.

For the configuration of the criteria to identify runs in the processed image material all parameters are defined in the project settings file. Those have to be changed according to the conditions of the application environment before creating the hardware design file.

5.5 Subpixel precision Implementation

The declaration for the register sizes and the data channels between the modules in the existing BLOB detection approach was hard coded. In addition all values were defined as integers. This caused a rounding error for the computation of the averaged brightness of the BLOB and for the center point coordinates as well. A

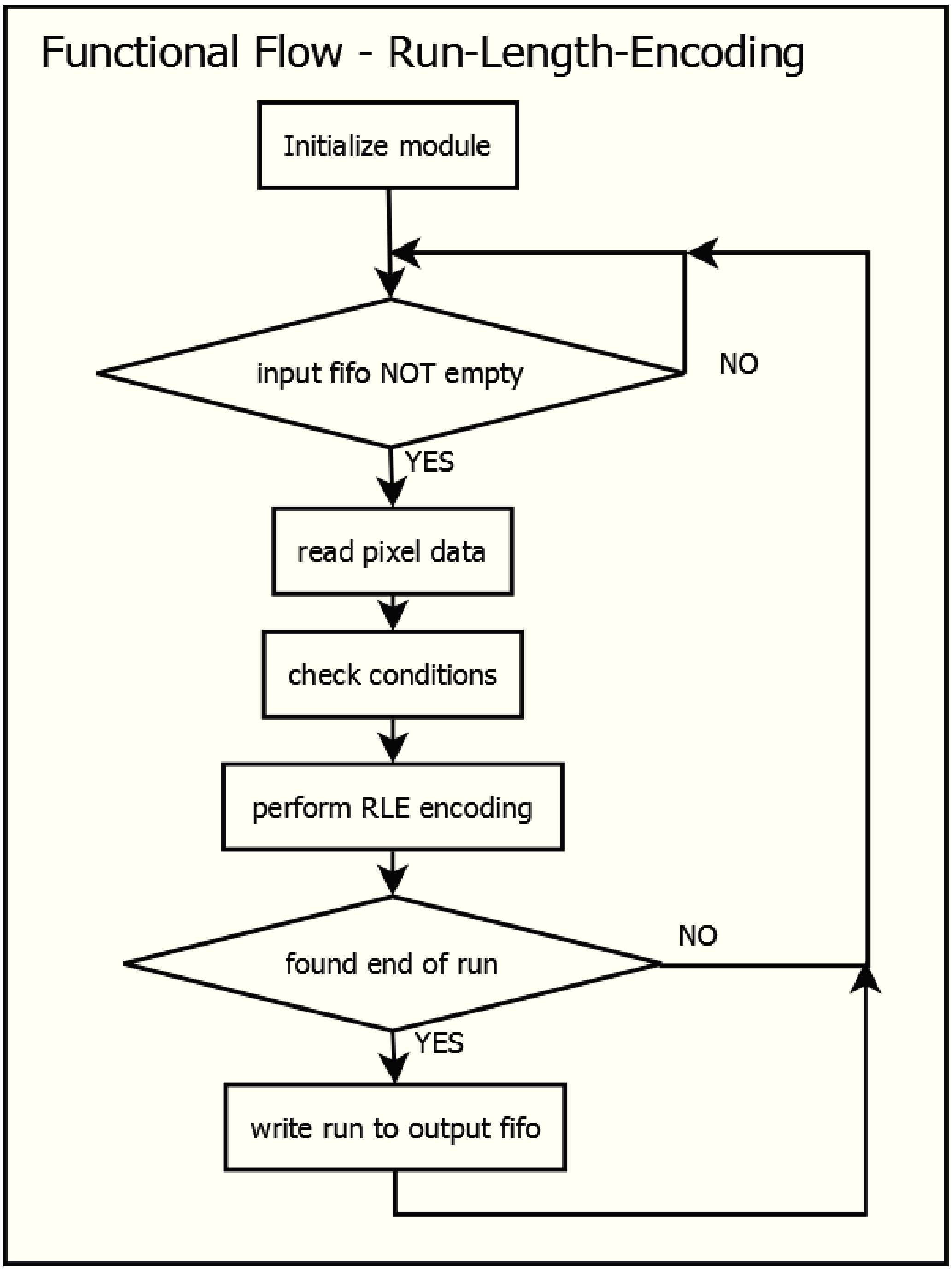


Figure 29. Simplyfied Representation of the Functional Process in the RLE Module.

declaration of floating point or fixed point values is not supported in HDLs right away. The handling has to be realized by developer or by applying IP cores from the Altera IP core library (Section 6.3).

The extension of the system with subpixel precision has been combined with the centralized configuration of the hardware design. This allowed a variable adjustment for the different hardware modules in a single settings file. The subpixel precision works in fixed-point arithmetic. For this approach the implementation has been done for the fixed-point configuration with four decimal places.

5.6 USB Communication Module Implementation

The module for the USB communication was implemented to work with the same input data as the serial communication module. For a shorter development cycle it was decided to build the module based upon an existing example module out of the DE2 demonstration sources from Altera. This module did transmit inputs from a PS2 keyboard via USB to a connected host-PC on the JTAG USB interface. This is usually applied to flash a hardware design file on the FPGA. In the modified version the BLOB detection and tracking results are transmitted to the connected host-PC with one Byte per message. The communication works controller based and requires a driver software on the host-PC to receive the system results. This driver software has been provided by the Computer Vision research group. It was integrated into the software for receiving and visualizing the BLOB detection results on the host-PC.

6 Tools & Equipment

Hardware design with FPGAs works very different from the common design principles and perspective of software development. Although the development environments look somehow similar, their functionality is much more powerful than anyone can grasp at first sight. In this Section the main aspects of the applied software tools will be introduced. This is intended to give an idea of the process flow from the source code to the hardware design file that can be flashed and executed on an FPGA. In addition, this Section presents the applied hardware equipment of this work. This should give an overview of the requirements for hardware design with FPGAs.

6.1 VHDL and Verilog

The application of a hardware description language (HDL) is the most common form a hardware design for an FPGA is realized. Using a high level design allows one to apply specialized compiler programs to optimize the source code for the target hardware, by reducing logic overhead, detecting critical paths and matching functional logic to available resources.

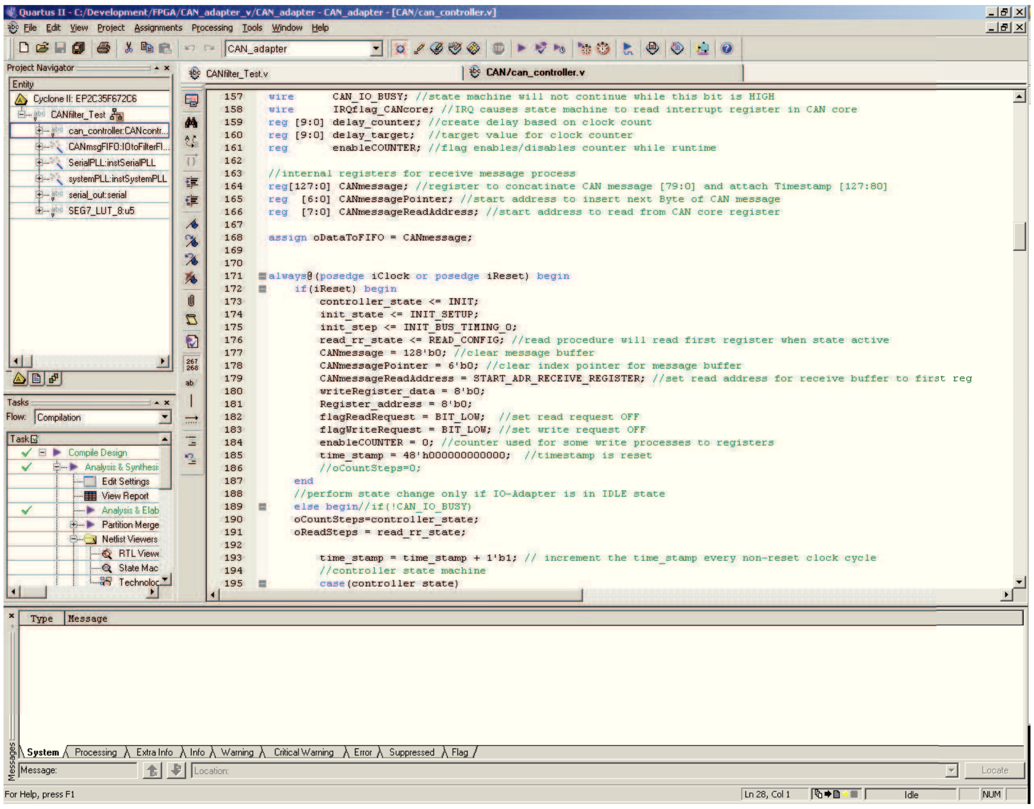
For implementation with HDLs the two most famous languages are Verilog [6, 40] and Very High Speed Integrated Circuit Hardware Description Language (VHDL). VHDL has been invented in the 1980s and should reduce the documentation that was required for hardware designs at that point. The first standard for VHDL has been published by IEEE in 1987. Today's latest version of the standard was published in 2008 [33] and is also known under the name VHDL 4.0. Using VHDL, the developer can choose between three different types to describe the hardware design. Those are referred to as behavioral, structural and Register Transfer Level (RTL) data flow descriptions. VHDL also supports a combination of those three concepts, known as mixed description.

The behavioral description is used to create a process based description of the hardware design. By reducing the modules functionality into tasks with fixed input and output signals, each task will execute if its input signals are changing. This concept creates the system in the form of a set of consecutive or parallel processes. The structural description specifies the internal and external configuration of design entities. The RTL description is used to create the hardware design as the system's data-flow. Each processing task in an RTL design is created as a concurrent block of execution. The order of execution is defined by the connections between blocks.

Verilog has been introduced in 1984 to create hardware designs that support simulation. The Verilog syntax is intentionally related to C, to allow an easy entrance for engineers and developers in the hardware and software area. Verilog's first standard 1364-1995 [30] has been published by the IEEE in 1995. The extension of the standard in 2001 [31], allows one to use Verilog to verify the hardware design. This concept is called hardware verification language (HVL) and is widely used for verification of electronic circuits. With HVL ability, the Verilog design can be simulated using real world constraints and functional test coverage assistance. This allows the developer to verify even timing constraints in the hardware design without the physical target platform. Today's common development environments for hardware design allow the combination of different HDLs in a single design. This gives the developer the choice to pick the HDL which best fits the given implementation task [10].

6.2 Quartus II

For the implementation of the hardware design the development environment Quartus II (Figure 30) from Altera has been used. It allows several different methods to realize a hardware design, starting from state machine design or block diagrams to hardware descriptions with AHDL, VHDL, Verilog and SystemVerilog. The tool of course allows the combination of several different design methods in one system.



```
wire      CAN_IO_BUSY; //state machine will not continue while this bit is HIGH
wire      IRQflag_CANcore; //IRQ causes state machine to read interrupt register in CAN core
reg [9:0] delay_counter; //create delay based on clock count
reg [9:0] delay_target;  //target value for clock counter
reg       enableCOUNTER; //flag enables/disables counter while runtime

//internal registers for receive message process
reg[127:0] CANmessage; //register to concatinate CAN message [79:0] and attach Timestamp [127:80]
reg  [6:0] CANmessagePointer; //start address to insert next Byte of CAN message
reg  [7:0] CANmessageReadAddress; //start address to read from CAN core register

assign oDataToFIFO = CANmessage;


always@(posedge iClock or posedge iReset) begin
    if(iReset) begin
        controller_state <= INIT;
        init_state <= INIT_SETUP;
        init_step <= INIT_BUS_TIMING_0;
        read_rr_state <= READ_CONFIG; //read procedure will read first register when state active
        CANmessage = 128'b0; //clear message buffer
        CANmessagePointer = 6'b0; //clear index pointer for message buffer
        CANmessageReadAddress = START_ADR_RECEIVE_REGISTER; //set read address for receive buffer to first reg
        writeRegister_data = 8'b0;
        Register_address = 8'b0;
        flagReadRequest = BIT_LOW;  //set read request OFF
        flagWriteRequest = BIT_LOW; //set write request OFF
        enableCOUNTER = 0; //counter used for some write processes to registers
        time_stamp = 48'h000000000000;  //timestamp is reset
        //oCountSteps=0;
    end
    //perform state change only if IO-Adapter is in IDLE state
    else begin//if(!CAN_IO_BUSY)
    oCountSteps=controller_state;
    oReadSteps = read_rr_state;

        time_stamp = time_stamp + 1'b1; // increment the time_stamp every non-reset clock cycle
        //controller state machine
        case(controller state)
```

Figure 30. Quartus II Development Environment from Altera.

Once the system design is implemented the tool chain in Quartus II runs through several different steps to create the hardware netlist file, which specifies the logical description of the system. First, step in the compilation process is the ”Analysis & Synthesis”, which performs a transformation from the hardware description into logic elements. At the same time the synthesizer optimizes the design by reducing the logic elements. In order to create a mapping of the program logic onto the FPGA the ”Fitter” tool does place and route according to the resources on the target device. During this procedure the tool is searching for the shortest interconnection paths in between the Logic Elements (LE) to reduce signal runtime. The maximum timing requirements for the place and route operation are given by the ”Analysis & Synthesis” procedure.

In the next step the tool chain performs a timing analysis of the hardware netlist file. This determines the performance parameters for the hardware design, such as maximum clock speed and signal runtime. The Quartus II environment allows the developer to choose between the "Classic Timing Analyzer" and the "TimeQuest Analyzer". The results of the first one are sufficient to locate parts of the hardware design, which cause problems for the system's functionality. It will identify interconnection paths that are too long with the configured clock rate. The "Classic Timing Analyzer" is a proprietary tool for which Altera has stopped further development. The results of the "TimeQuest Analyzer" give more detailed information, since it will locate critical paths of the design for a minimum and a maximum clock rate. The intent for the invention of the "TimeQuest Analyzer" was to support the industry standard for design constraints and timing assignments, known as "Synopsis and Design Constraints format" [3].

6.3 IP cores library

Code reusability is a major criteria in software development. This concept has found its way into the hardware design area as well. Several growing FPGA and ASIC communitys provide designs as open source [46]. These projects usually run under a free software license, such as GPL or LGPL which makes their usage for commercial products complicated for companys. Besides, the vendors of hardware design platforms started to include ready-to-use module packages that fulfill standard operations. The Quartus II environment contains a Plug-In Manager, called "MegaWizard" that can be used to create and configure such intellectual property (IP) cores. The application of IP cores in a hardware design might be restricted, dependent on the purpose of the approach. While the usage for research projects is usually free, the integration into commercial products requires a license fee.

6.4 Simulation Tools

The localization of errors during runtime in a hardware design can be very time consuming. In comparison to debugging software applications, it is not possible to perform step-wise code execution in hardware. For validating the design before putting it into hardware, the developer can use simulation tools to run the design in the development environment. Altera provides a simulator that is able to perform functional and timing simulation of the design. The special advantage of the timing simulation is that it can simulate the signals like they would behave in hardware. This includes the characteristics of set-up and hold times of the signals, which is not covered in a functional simulation. The downside is that the simulator works based on waveform files (Figure 31), which are cumbersome to create. In addition, the waveform simulator does not allow a step-wise execution of the source code.

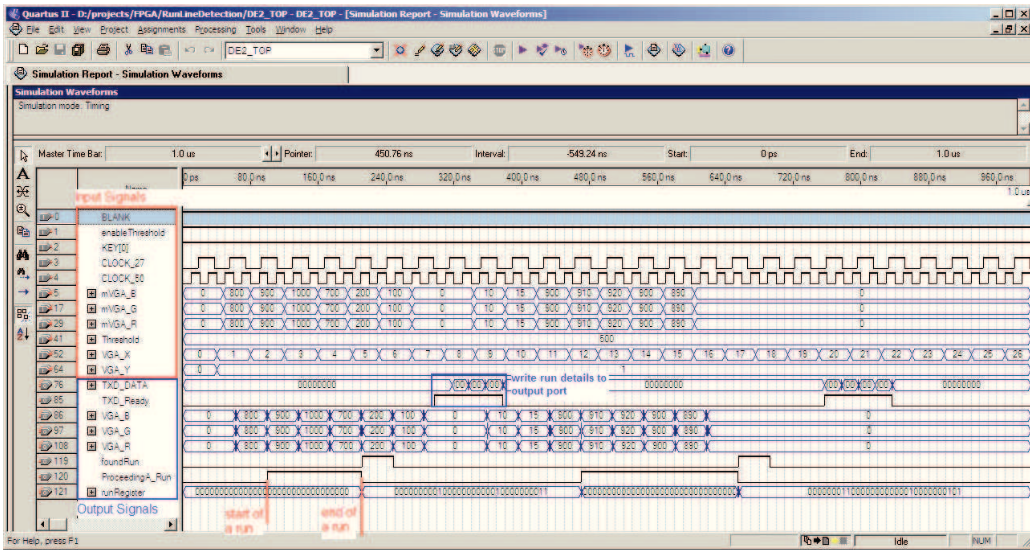
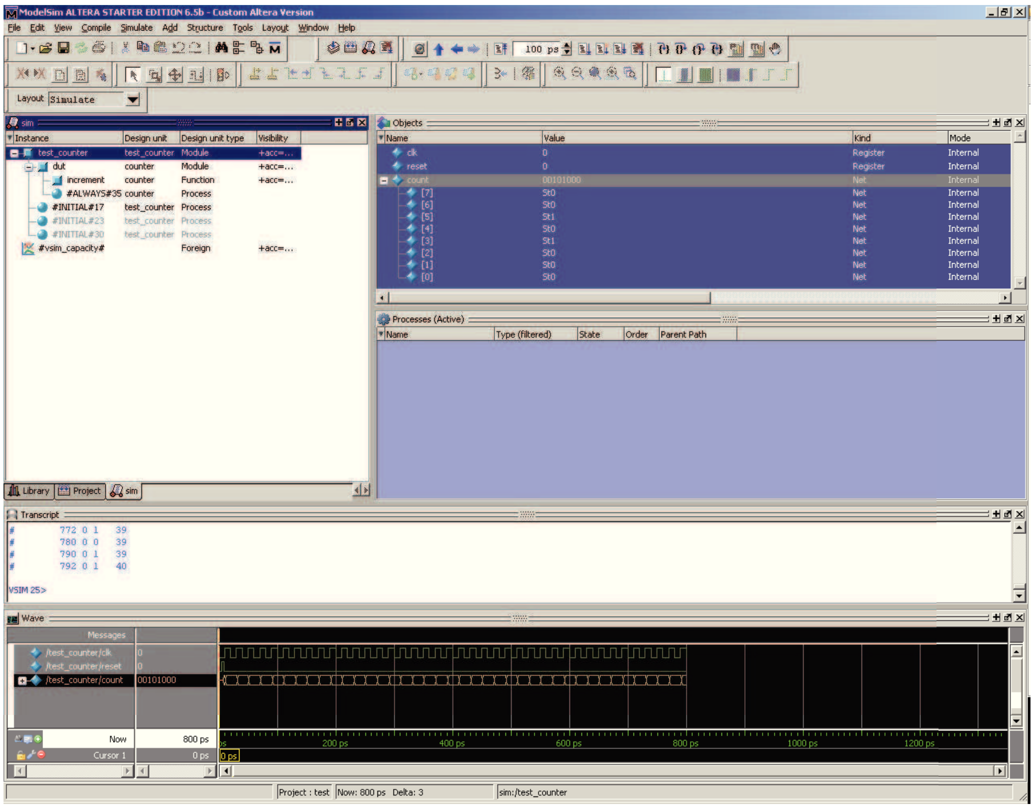


Figure 31. Altera Waveform Simulator used for Validation of RLE Pre-Processing.

In order to perform a step-wise debugging of the hardware design, Mentor Graphics provides a tool called "Model-Sim" (Figure 32). It permits only functional simulation of the design and cannot simulate parallel execution. If a design consists of several modules, their execution is performed in sequential. The simulator requires a test bench file, that instantiates the hardware design and defines the input signals. If the hardware design contains IP cores from the Altera library, the simulator comes with a set of librarys that can run those as black boxes. This allows testing the IP core modules instantiated without the ability to observe their internal composition. In addition to print-statements, which can be used inside the source code to create outputs on the console of "Model-Sim", the internal registers can be observed and changed during simulation. It also creates a log-file of the signals over time, which is visualized as a waveform file.


ModelSim ALTERA STARTER EDITION 6.5b - Custom Altera Version
File Edit View Compile Simulate Add Structure Tools Layout Window Help
100 ps
Layout Simulate
sim
Objects
Processes (Active)
Library Project sim
Transcript
Wave
Messages
Now 800 ps
Cursor 1 0 ps
200 ps 400 ps 600 ps 800 ps 1000 ps 1200 ps
Project : test Now: 800 ps Delta: 3 sim:/test_counter

Instance	Design unit	Design unit type	Visibility
test_counter	test_counter	Module	+acc=...
dut	counter	Module	+acc=...
increment	counter	Function	+acc=...
#ALWAYS#35	counter	Process	
#INITIAL#17	test_counter	Process	
#INITIAL#23	test_counter	Process	
#INITIAL#30	test_counter	Process	
#vsim_capacity#		Foreign	+acc=...

Name	Value	Kind	Mode
clk	0	Register	Internal
reset	0	Register	Internal
count	00101000	Net	Internal
[7]	St0	Net	Internal
[6]	St0	Net	Internal
[5]	St1	Net	Internal
[4]	St0	Net	Internal
[3]	St1	Net	Internal
[2]	St0	Net	Internal
[1]	St0	Net	Internal
[0]	St0	Net	Internal

Name	Type (filtered)	State	Order	Parent Path

```
#          772 0 1    39
#          780 0 0    39
#          790 0 1    39
#          792 0 1    40

VSIM 25>
```

Messages	
/test_counter/clk	0
/test_counter/reset	0
/test_counter/count	00101000

Figure 32. Mentor Graphics Model-Sim Simulator for Functional Simulation

6.5 DE2 and DE2-70

The Altera DE2 Development and Education board [50] is a low-cost target platform for digital logic and hardware design (Figure 33). It uses a Cyclone II FPGA with 33,216 Logic Elements. The board is equipped with a large set of interface technologies and internal memory that allow the design of embedded systems in the complete range from pure logic to Hardware/Software Co-Design. The board has been used for the R&D1 and R&D2 project. For this thesis work the target platform has been changed to the DE2-70 board [52]. Most of the hardware on the board is similar to the DE2 board. The DE2-70 uses a FPGA with twice the number of LEs. It has larger external memory modules and a faster RS232 controller. The wiring of the DE2-70 board is different for the connection of the USB-JTAG interface. It does not allow a direct application of the interface for data transmission in the hardware design.

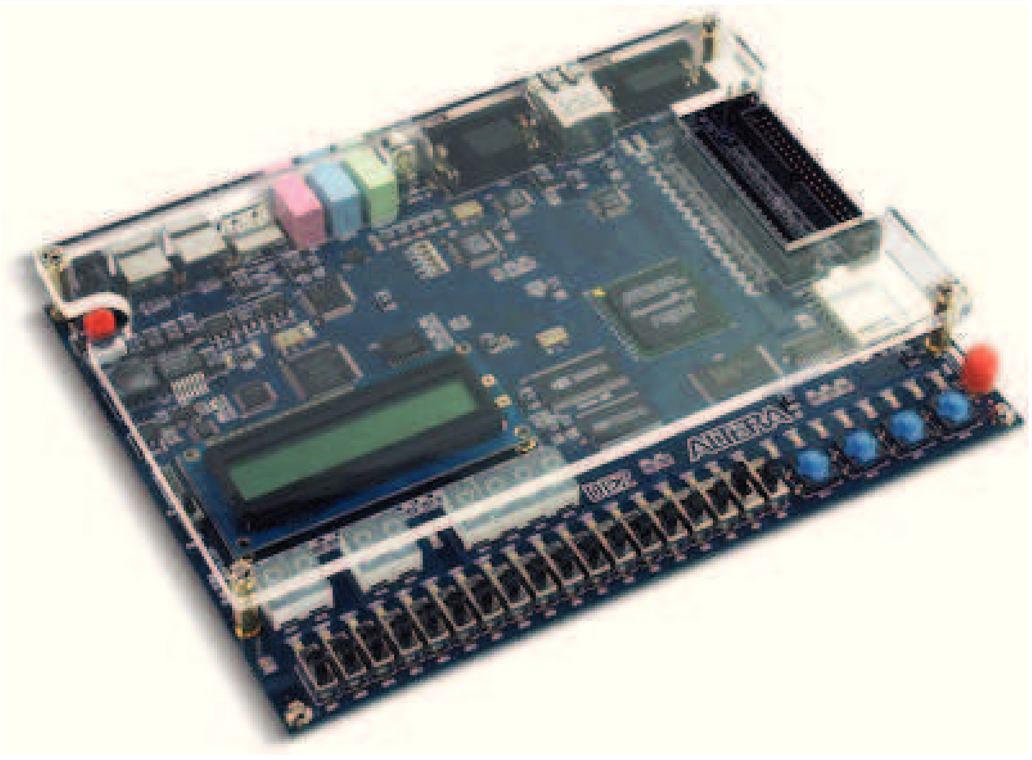


Figure 33. Altera's DE2 Development and Education Board.

6.6 mvBlueCOUGAR-X

The mvBlueCOUGAR-X [38] is a industrial Gigabit Ethernet camera from Matrix Vision. The camera supports the GigE Vision [5] network protocol to allow high frame rates and the GenICam interface [22] to configure the camera device. It is available with CCD or CMOS sensor technology as Gray scale or RGB Bayer mosaic [37]. For this work the model with the CMOS Grey scale sensor was used.

7 Verification and Validation

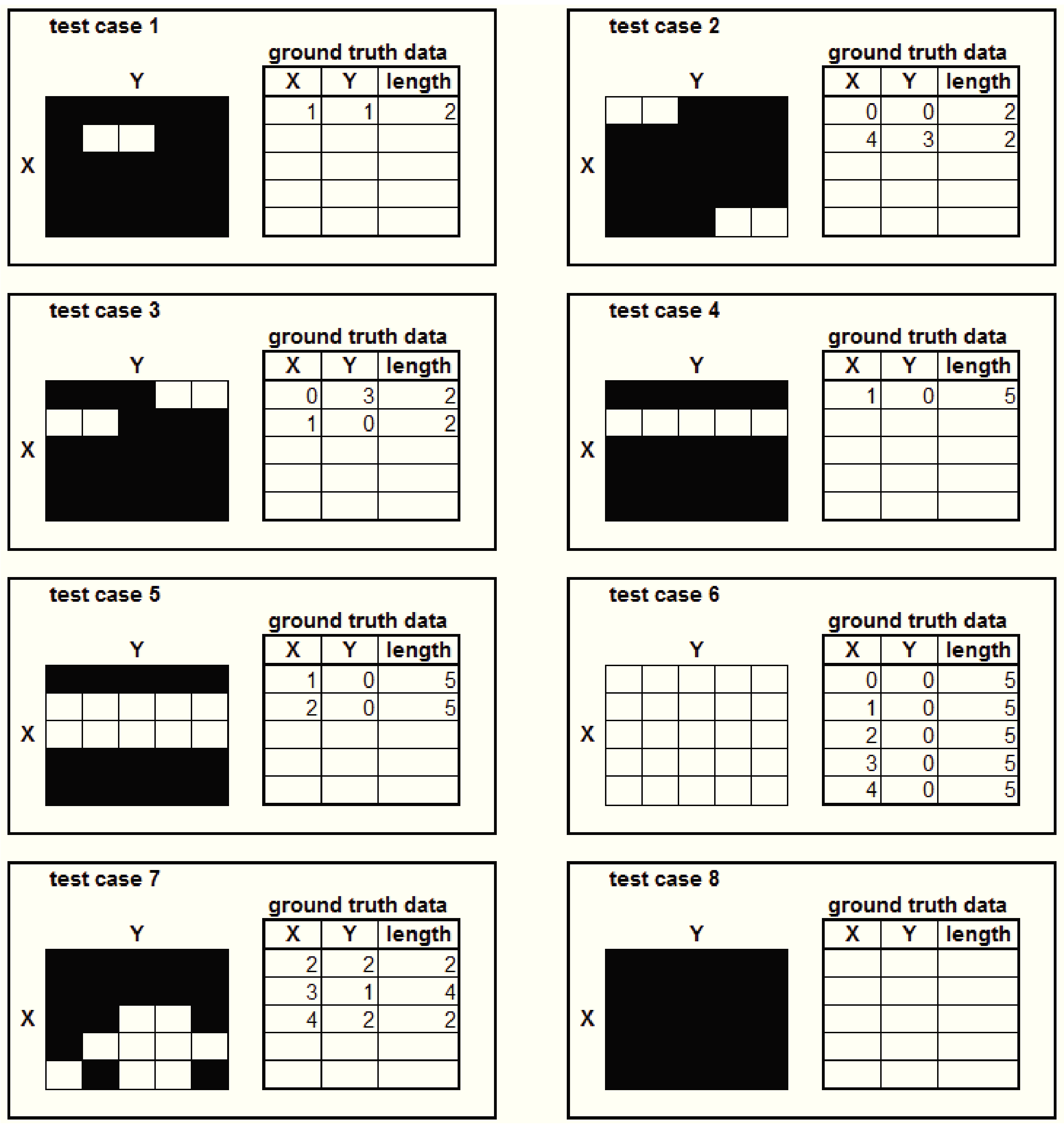
The verification and validation of the versatile implementation work required different technologies and tools (Section 6.4). For evaluating the functional correctness of the hardware design, each single module has been simulated using the given development environment. The benchmarking of precision and performance has been covered by specialized test bench files for each module. For the evaluation of the software approach, the hardware benchmark test files could be reused to provide the input data. The results for the tracking approaches in hardware and in software have been stored and visualized on the computer containing the development environment. The following sections discuss the verification and validation procedures of each module and their results.

7.1 Pre-Processing Run-Length-Encoding

The written test bench files for the functional simulation of the RLE based pre-processing contained several different cases to cover the described conditions from Section 4.4. Figure 34 shows the representation of the testcase data that has been used for the verification. The image data was encoded as integer values, describing the position of the pixel in the frame and its brightness value. The characteristics of the detected runs had been printed on the console. For the verification of the correctness, the output has been compared to manually estimated results by hand.

The results of the functional simulation matched with the expected values. This did prove the logical correctness of the design. In the next step the execution of the hardware design on the target platform has been tested.

For the verification of the correctness of the RLE pre-processing on the DE2 development board, the test data has been reused. By integrating the test data into a separate module of the hardware design, this module provided data which allowed the evaluation of the system's output. Having the test data in the hardware design



test case 1

ground truth data

X	Y	length
1	1	2

test case 2

ground truth data

X	Y	length
0	0	2
4	3	2

test case 3

ground truth data

X	Y	length
0	3	2
1	0	2

test case 4

ground truth data

X	Y	length
1	0	5

test case 5

ground truth data

X	Y	length
1	0	5
2	0	5

test case 6

ground truth data

X	Y	length
0	0	5
1	0	5
2	0	5
3	0	5
4	0	5

test case 7

ground truth data

X	Y	length
2	2	2
3	1	4
4	2	2

test case 8

ground truth data

X	Y	length

Figure 34. Testcases for Verification of RLE Pre-Processing Module.

has the disadvantage that the system requires more resources on the FPGA. Having knowledge about the expected results allows a better evaluation of the system's precision.

The hardware design with the integrated image data has been tested with an increasing clock rate to verify its maximum performance. The result data has been transmitted on a serial interface to a connected host-PC and stored in a text file for manual evaluation. The result data contains the starting position of a run, its length and an averaged value for the brightness of all pixels in that run. It could be verified that the result data matched the expected values for the integrated test data. For the RLE pre-processing module a maximum clock rate of 145 MHz could be measured. This would allow the system to process an input image material of 640x480 pixels with up to 117 frames per second.

For a second test procedure with real-time image data, the D5M camera has been used as the input device. The DE2 board has been placed in the test model that is shown in Figure 35.

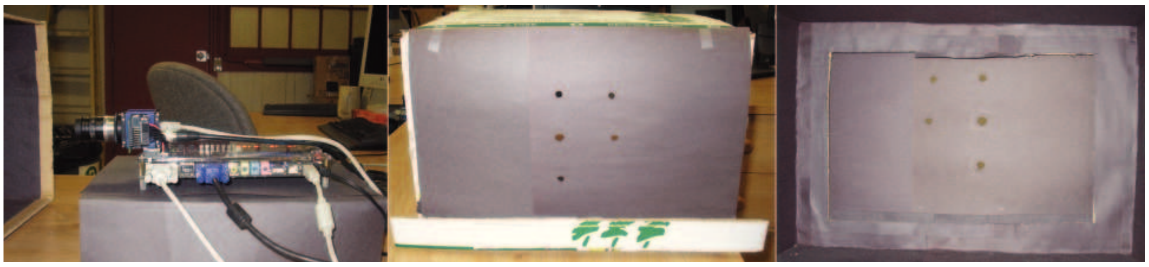


Figure 35. Testsetup for Threshold Adjustment Validation.

The existing application on the host-PC to receive and store the result data has been extended in functionality to visualize the runs. It will draw the runs in its original size in a window with the same resolution as the processed image material. During the tests, the serial interface became a bottleneck for the transmission of the RLE encoded data. A single BLOB in the test model contained between 30 and 35 runs and the model was built to create five BLOBs in the image material. With a frame rate of 50 fps. the system would require a bandwidth of 34 Kilobytes per second. The shortage of bandwidth caused an overflow of the FIFO buffer in between the RLE pre-processing module and the serial interface module. This lead to a loss of run data and an incorrect visualization on the host-PC. With the test setup it took around two seconds before the overflow occurred. This allowed the system to show a proper visualization for a short duration after startup. The RLE encoding of the image data was considered to work correct and could be applied in the BLOB detection approach.

7.2 Variable Threshold Adjustment

The module for the variable threshold adjustment is the only exception in terms of the applied verification concept. The proper execution of the module has been tested on the hardware platform without using specialized test bench modules to create input data. The input data has been acquired with the D5M camera which is attached to the target board. The manipulation of the configurable values and the automated adjustment of the threshold values are shown on the seven segment display of the DE2 board. With a simplified model for the expected image data of the proposed application environment the changing of the BLOB's size could be simulated to the camera. This allowed a rather complete validation of the module's correctness by hand.

Figure 36 depicts how the input data is used to set the reference values for the BLOB's size. The diameter of all BLOB's on the X- and Y-axis is used to estimate an average size for all BLOBs. This reference value will remain fixed once the system is set to "running" mode. The threshold value for the BLOB detection will be incremented or decremented by one according to the change of the BLOB size.

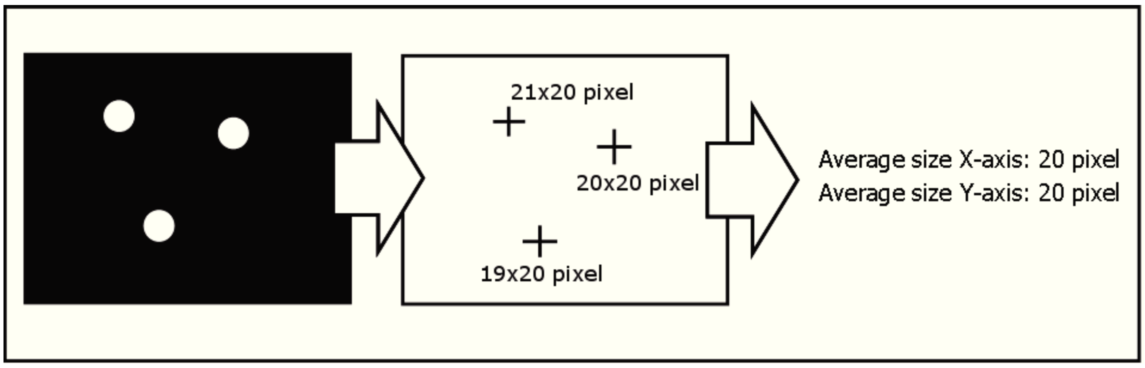


Figure 36. BLOB Size Averaging Process for Threshold Adjustment.

The "ThresholdAdjustment" module receives the information about the average

BLOB size from the "BLOB Detection" module. In the current approach both modules run with the same clock speed of 50 MHz. That guarantees an update of the threshold value every 50 μs.

For validation, the system is configured during runtime while the camera remained in a fixed distance to the model. Figure 35 gives an idea of the setup which has been used. Once the configuration was complete, the system was set into "running" mode to observe the threshold adjustment.

The validation of the threshold adjustment has been performed with different initial configurations. It could be verified that the threshold value changed accordingly to the change of the BLOB size. The threshold value remained in the defined minimum and maximum values.

7.3 Tracking in Software

One requirement for the comparison of the tracking approaches in hardware and software was that both solutions work with similar input data. The defined format for the result data of the BLOB detection is a 49 bit or 57 bit record for a single BLOB. This depends on the activation of the subpixel precision for the computation of the BLOB center points in the BLOB detection module. The resulting BLOB data contains the ID, the (X,Y) position and the diameter of the BLOB on the X and Y axis.

For the first evaluation of the tracking approach in software, the results of the BLOB detection on the DE2 board have been integrated into a hardware design file and reused for testing. The expected results for the tracking procedure have been estimated by hand. This test setup was used for the measurement of the maximum performance of the tracking in software and its precision. It could be verified that the tracking approach worked correct for the applied test data. The BLOBs could be tracked successfully as long as the conditions for the maximum distance between one BLOB in two consecutive frames was not injured. This required that the BLOB moved with a distance smaller than the distance between two different BLOBs in the same image. For tracking in software, with visualization, the best performance for image data with up to five BLOBs was 50 frames per second. This was impacted by the required operations for creating a graphical output with OpenCV. The value was measured during the testing.

When the visualization was disabled, the tracking application could process the incoming data faster than the DE2 board could provide it. This was caused by the performance of the serial communication module on the DE2 board. Having a bandwidth of 57600 bit per second and the number of BLOBs restricted to five, the tracking would work for up to 167 frames per second if subpixel precision is disabled. For subpixel precision enabled the tracking would perform up to 144 frames per second. These values are a theoretical maximum which has not been verified yet.

In a second evaluation procedure the D5M camera has been used as the input device. For the creation of the image material, the test model from Figure 35 has been used. The tracking in software worked correctly for 45 frames per second. That is the maximum speed for the D5M camera for an image resolution of 640x480 pixels. The validation of the correctness was done by hand, using the visualization feature of the tracking application.

The performance is a result of the thread based implementation of the tracking approach using the Intel TBB library. The implementation is optimized for an Intel dual-core platform using one thread for reading the serial port and another for the tracking computation and visualization. For the check that no data was lost due to buffer overflow, the size of the serial input buffer and the data buffer in between the threads has been observed during the tests. If one of the threads would have been to slow, the buffers would show an increase of their size.

7.4 Tracking in Hardware

The verification of the correct functionality was performed by simulating the tracking module in Altera's Model-Sim (Section 6.4). Test bench files have been created to provide the input data and evaluate the results. The applied values are the same as for the verification of the tracking in software from Section 7.3. The results of the simulation showed that the implementation of the tracking procedure is correct. All values matched the expected result data that was defined by the expected BLOB ID after the tracking procedure had finished to process the latest frame data.

# of BLOBs	Center point computation	Tracking	Subpixel precision	Logic Elements	Memory Bits	DE2 board	DE2-70 board	fmax detection (MHz)	fmax tracking (MHz)
5	BB	No	No	7,768	79,360	OK	OK	53.26	—
5	CoM	No	No	22,145	128,512	OK	OK	53.26	—
5	BB	No	Yes	7,881	83,456	OK	OK	51.57	—
5	CoM	No	Yes	22,744	132,608	OK	OK	51.57	—
6	BB	No	No	9,541	79,360	OK	OK	47.23	—
6	CoM	No	No	27,308	128,512	OK	OK	47.23	—
6	BB	No	Yes	9,898	83,456	OK	OK	46.43	—
6	CoM	No	Yes	28,824	132,608	OK	OK	46.43	—
7	BB	No	No	11,776	79,360	OK	OK	46.13	—
7	CoM	No	No	32,925	128,512	OK	OK	46.13	—
7	BB	No	Yes	12,010	83,456	OK	OK	45.62	—
7	CoM	No	Yes	34,919	132,608	OK	OK	45.62	—
8	BB	No	No	14,620	79,360	OK	OK	47.29	—
8	CoM	No	No	39,240	128,512	—	OK	47.29	—
8	BB	No	Yes	14,666	83,456	OK	OK	45.83	—
8	CoM	No	Yes	41,175	132,608	—	OK	45.83	—
5	BB	Yes	No	23,458	106,088	OK	OK	53.26	151.93
5	CoM	Yes	No	37,961	155,240	—	OK	53.26	151.93
5	BB	Yes	Yes	27,477	114,100	OK	OK	51.57	135.67
5	CoM	Yes	Yes	42,469	163,252	—	OK	51.57	135.67
6	BB	Yes	No	25,213	106,416	OK	OK	47.23	139.18
6	CoM	Yes	No	43,340	155,568	—	OK	47.23	139.18
6	BB	Yes	Yes	31,994	114,392	OK	OK	46.43	120.77
6	CoM	Yes	Yes	48,257	163,544	—	OK	46.43	120.77
7	BB	Yes	No	31,780	106,744	OK	OK	46.13	135.46
7	CoM	Yes	No	52,486	155,896	—	OK	46.13	135.46
7	BB	Yes	Yes	39,586	114,684	—	OK	45.62	117.36
7	CoM	Yes	Yes	58,731	163,836	—	OK	45.62	117.36
8	BB	Yes	No	46,364	107,072	—	OK	47.29	125.33
8	CoM	Yes	No	71,115	156,224	—	—	—	—
8	BB	Yes	Yes	54,955	114,976	—	OK	45.83	109.67
8	CoM	Yes	Yes	82,736	164,128	—	—	—	—
System was tested with 45-50 fps. for all configurations									

Table 4. Resource Allocation and Performance Characteristics for 5 to 8 BLOBs.

In the next step the tracking module had to be tested after integration into the DE2 target platform. For the verification the input data was implemented into a separate module in the hardware design. This allowed the comparison of the result data with the expected outcome. The result data had been recorded and displayed on a connected host-PC. The tracking module has been tested with 50 MHz, 80 MHz and a 100 MHz input clock. For all three clock rates it provided correct results until the buffer of the serial interface module showed an overflow. This was caused by the slower performance of the serial interface module.

The design of the tracking module supports a maximum clock rate of 150 MHz. With the state machine design the module requires "*85 clock ticks * # of BLOBs*" on average to process the result data of a single frame with up to five BLOBs. In theory this would allow the system to process 250000 frames per second. The verification of this performance is not possible with the serial communication interface to transmit the result data. It is also very unlikely that the system will perform with that speed without showing some errors caused by signal runtime in the hardware design. The number of clock ticks will increase with the number of BLOBs that can be tracked. Table 4 shows the resource and performance results for the BLOB detection and tracking design for both applied target platforms.

In another test procedure the D5M camera has been applied to perform the tracking approach with live image data. Using the model from Figure 35 the results of the tracking have been evaluated by hand during runtime. The visualization of the tracking results on the connected host-PC allowed a reasonable validation by hand for an input speed of 45 frames per second. This was the limitation of the D5M camera for the applied resolution of 640x480 pixels. In another test scenario the CMOS camera from Matrix Vision has been applied. The camera supports a faster frame rate but the serial communication module was not able to transmit the results fast enough. It could be observed that the visualization on the host-PC showed a delay that did increase over time until the status LEDs of the DE2 board signaled an overflow of the input FIFO for the serial communication module.

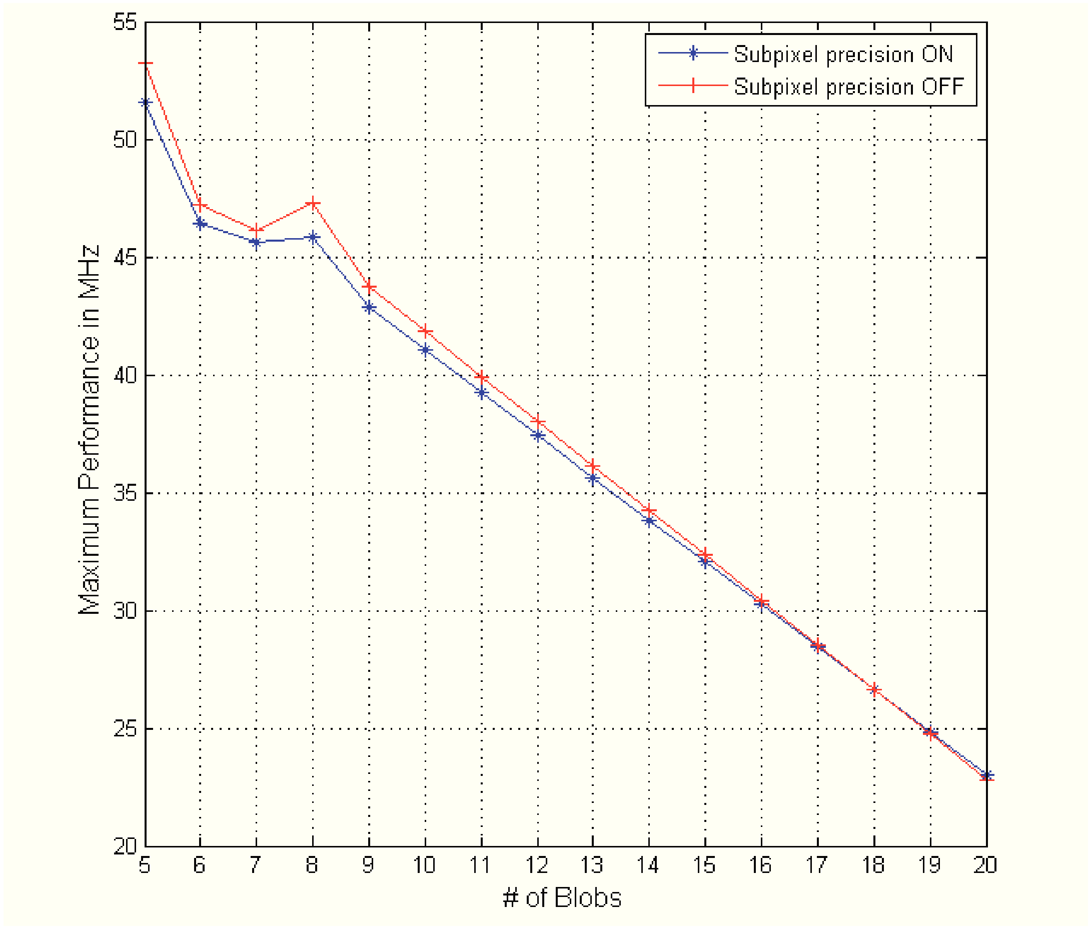


Figure 37. Performance for BLOB Detection with and without Subpixel Precision.

The configuration of the BLOB detection and tracking solution is centralized in one global settings file. Table 4 shows the number of available configurations for the system. The column "# of BLOBs" describes the maximum number of BLOBs that can be detected and tracked by the system. In the column "Center point computation" the method used in the detection module to estimate the BLOB's center points is specified, either Bounding Box or Center-of-Mass. The analysis of the resource requirements gives an idea of the applicability of the detection and tracking on the DE2 and DE2-70 board (Section 6.5). As can be seen in the columns for the maximum clock rate of the detection and tracking functionality, the detection module has a lower performance in general. However, with an increasing number of BLOBs the maximum performance of the tracking module decreases much faster than the detection module. Figures 37 and 38 are showing the relation between the amount of BLOBs that can be processed by the system and the maximum performance in MHz. They also show the impact of the subpixel precision on the system.

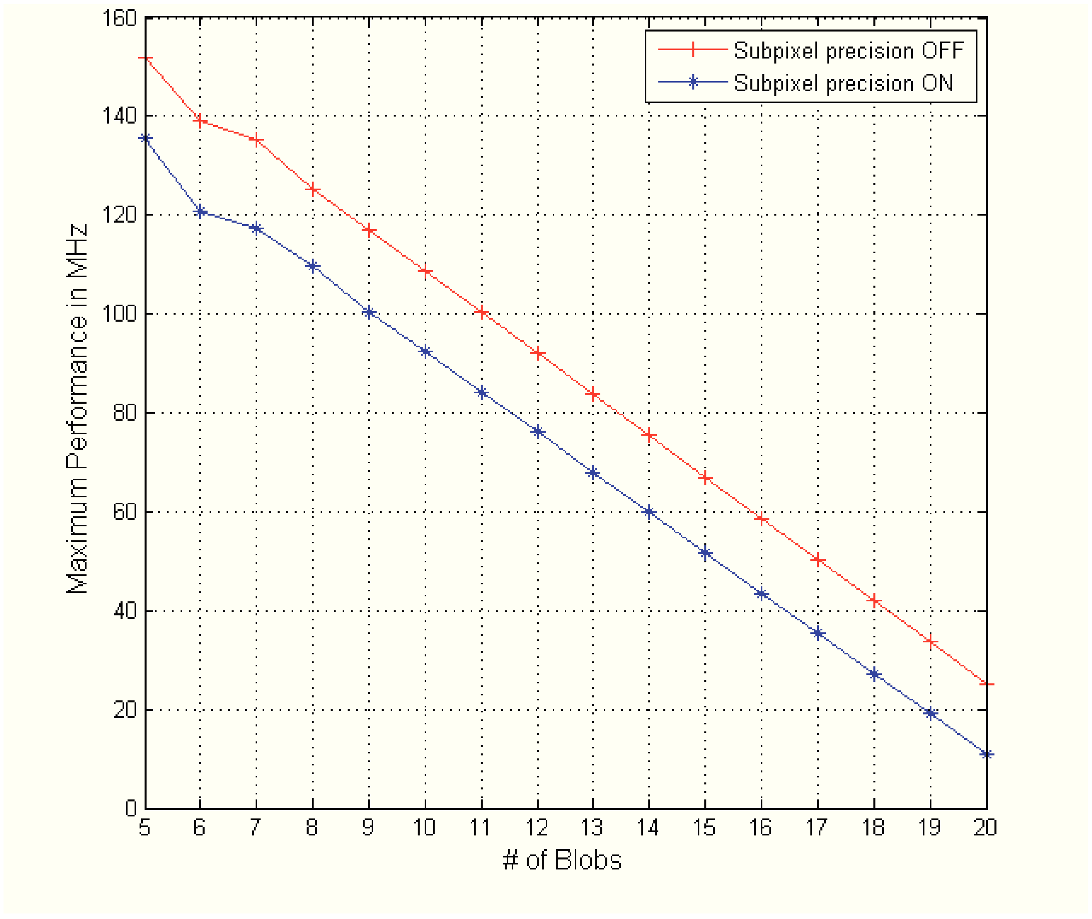


Figure 38. Performance for BLOB Tracking with and without Subpixel Precision.

The difference on the maximum performance for the BLOB detection with and without subpixel precision is relatively small compared to the difference for the tracking module. However, in both cases the clock rate is still high enough to process the frame rates that can be provided by the given input devices.

The bigger problem of the hardware design is the evolution of the resource requirements for the detection and tracking modules with an increasing number of BLOBs. In Figure 39 it can be seen that the resource requirements for the BLOB detection, using Center-of-Mass for the center point estimation, is already too high

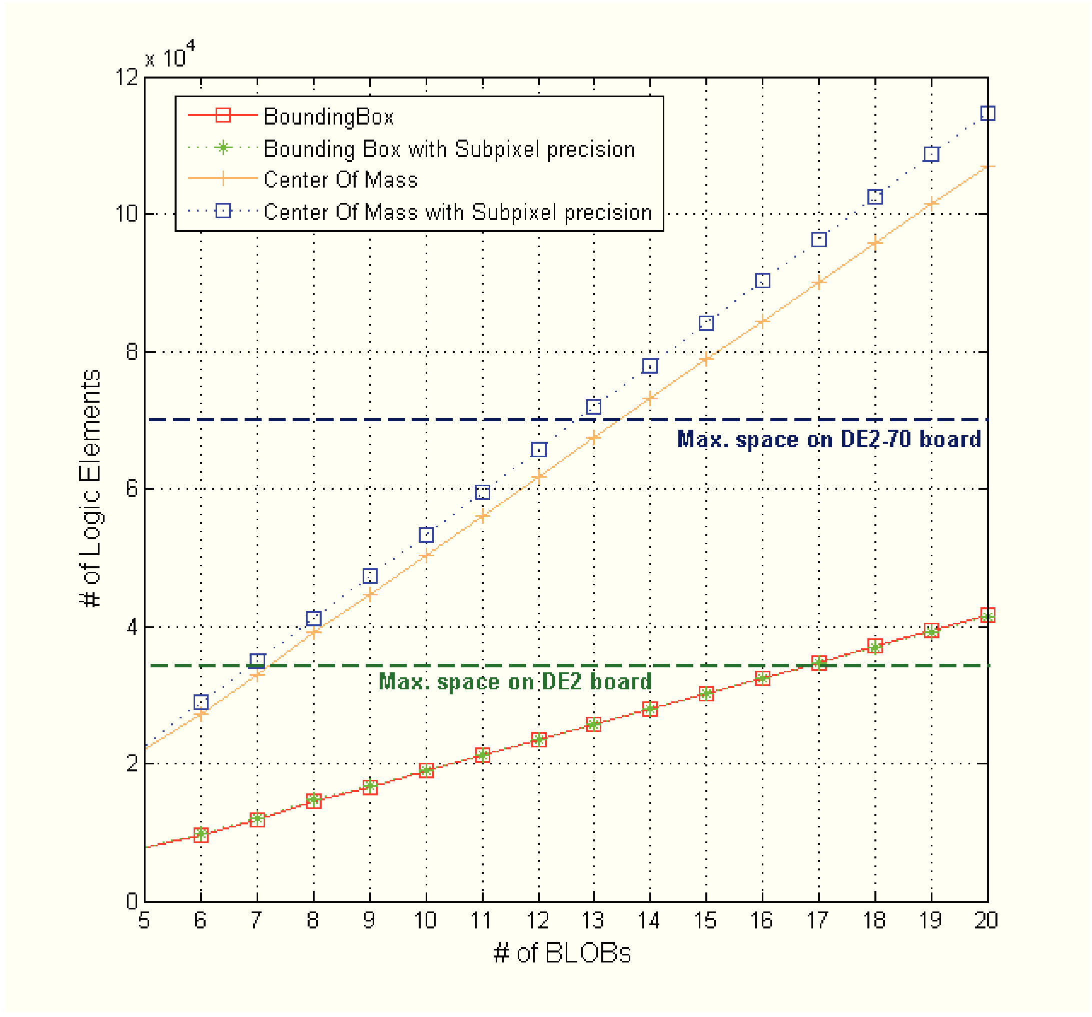


Figure 39. Resource Allocation for BLOB Detection with Increasing Number of BLOBs.

for the available target platforms with fourteen BLOBs. But even with the Bounding Box based approach the maximum number of BLOBs for the DE2-70 board is approximately thirty.

For the tracking module, the shortage of resource requirements becomes even worse, since the tracking module cannot be used without having the detection module included. The tracking module has to support as many BLOBs as the detection module. In Figure 40, it can be seen that the design is running out of space for both target platforms with eight BLOBs for Center-of-Mass and twelve for Bounding Box.

For comparison of the tracking in hardware and in software the test bench files

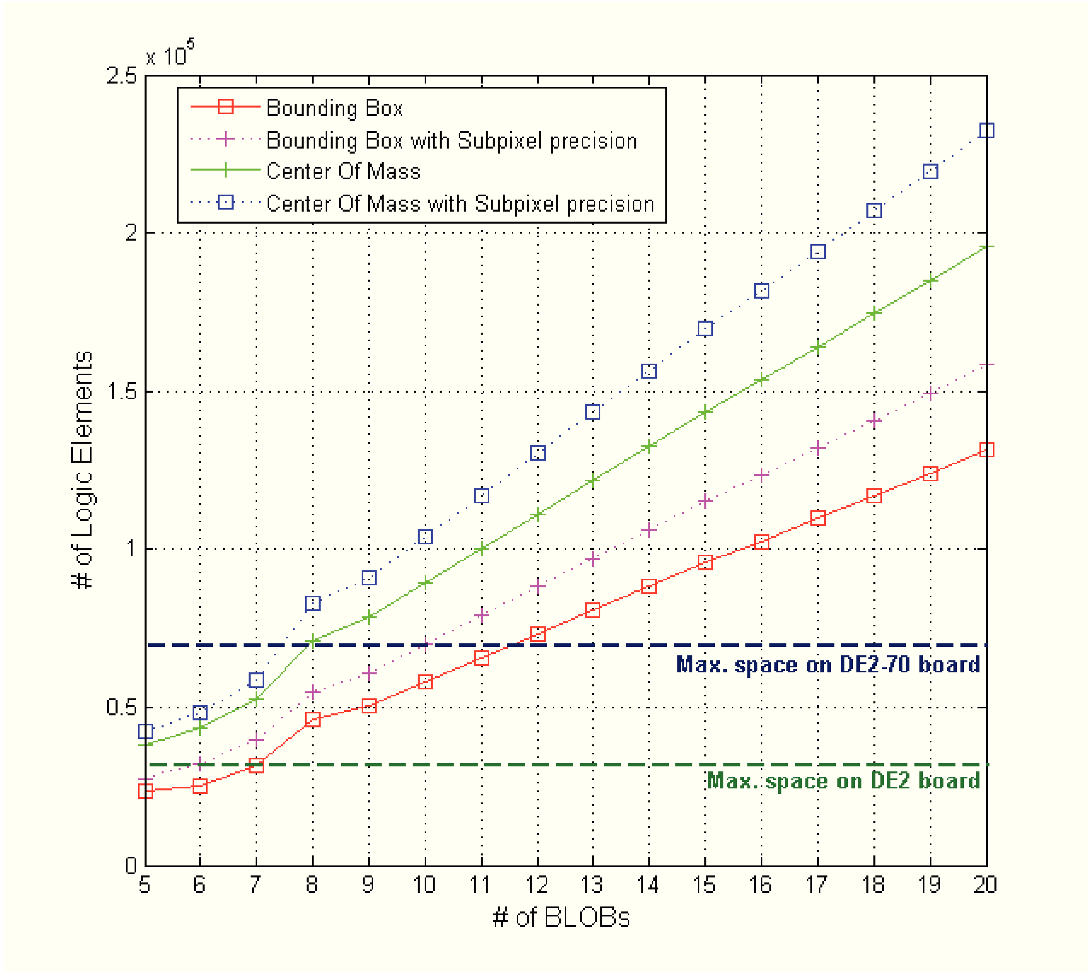


Figure 40. Resource Allocation for BLOB Tracking with Increasing Number of BLOBs.

have been used which provide defined input values from a separate hardware module. This permits checks to ensures both approaches work with similar precision. The recorded results from the tracking procedures have been evaluated by hand. For the test benches both approaches showed the same correct results.

Because of the resource requirements for test bench files in hardware, the number of test samples had been kept minimal. Every additional test data that is implemented in hardware is affecting the design and with it the performance values. Therefore the test bench files have not been used for a comprehensive performance evaluation. The performance for the tracking in hardware has been averaged, based on the number of clock ticks that the module processes for one BLOB result.

For the validation of the accuracy of the center-point results the hardware approach has been compared with a similar software implementation. The software solution utilizes the OpenCV [1] library for the BLOB detection and center-point computation process. In this verification step, the results of the OpenCV based approach are interpreted as the ground-truth. The precision of the hardware solution is measured by the error between its results and the results of the software implementation. The precision error has been estimated over one hundred samples. The image material showed BLOBs with perfect circular shape and BLOBs with blur effect. The image material had a resolution of 640x480 pixels. The offset in the center-point result are shown in Figures 41 and 42.

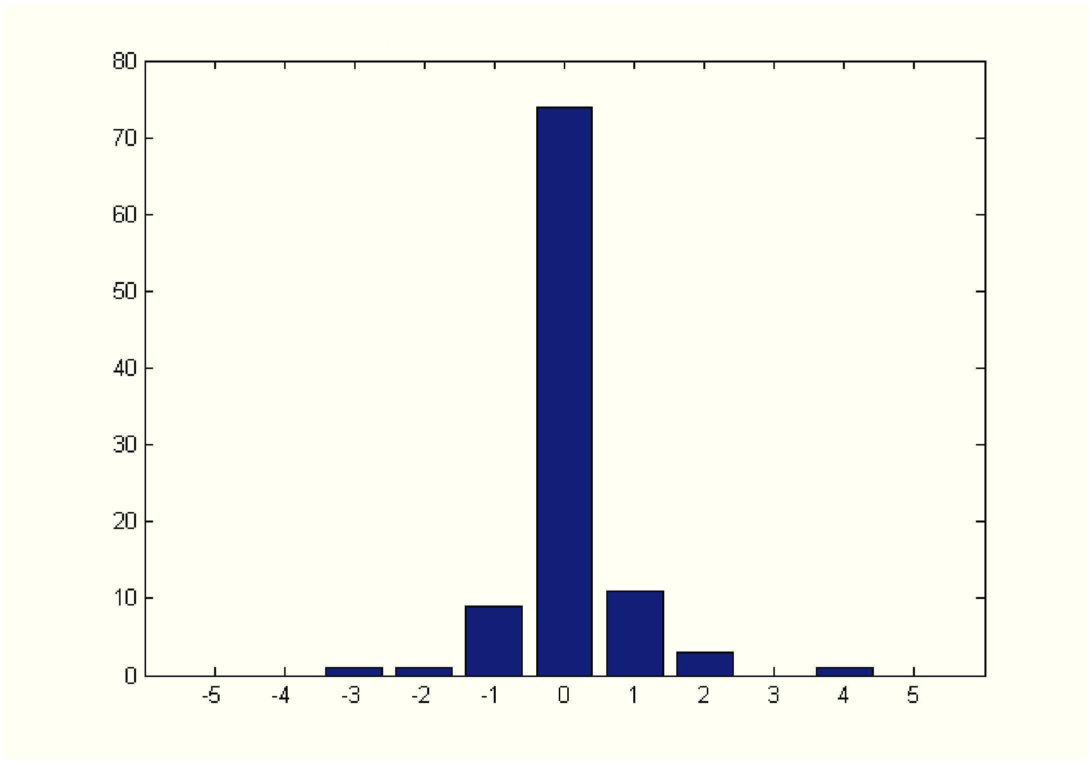


Figure 41. Comparison of BLOB Center-Points on X-Axis between Hardware and Software Approaches.

In Figure 41 the offset in pixels for the estimated center point of the BLOB on the X axis in the applied image material can be seen. An offset of 0 means that the

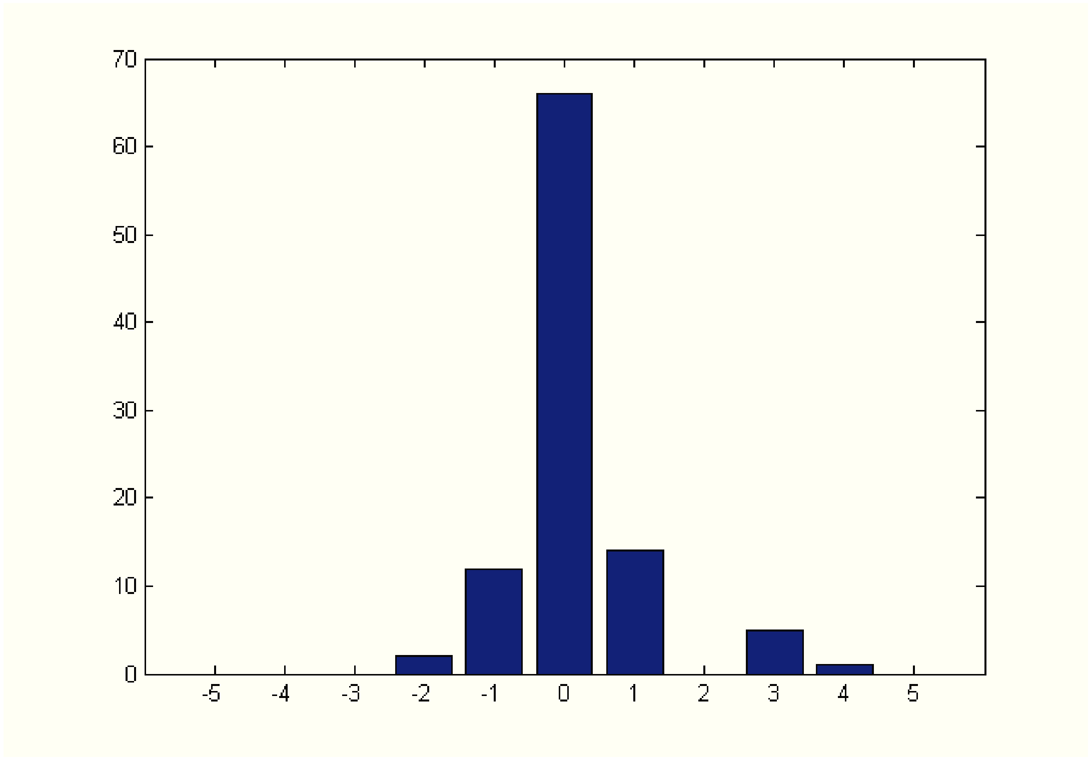


Figure 42. Comparison of BLOB Center-Points on Y-Axis between Hardware and Software Approaches.

computed pixel position by the hardware solution is the same as the result of the software implementation. Figure 42 is showing the offset in pixel for the Y axis in the applied image material. For the applied test data the hardware approach showed a precision error on the X axis of 0,0547 %. The error on the Y axis is 0,1021 %. The overall precision error in the center point computation module of the system is 0,078 % with the applied image resolution of 640x480 pixels. It has to be taken into account that the software approach is computing with integer values. The ground-truth results can show a rounding error of one pixel. That explains most of the error values in the given Figures above.

7.5 Subpixel precision

The verification of the subpixel precision module has been done after integrating the fixed-point arithmetic into the design. A decrease of the jitter effect of the center points was observable in the visualization for the BLOB detection results on the host-PC. The results were compared for the previous existing integer values with the four digit fixed-point values. A further increase of the number of decimal places does not seem reasonable, instead should the application of input devices with a higher resolution be taken into account.

7.6 USB Communication Module

The correct functionality of the USB communication module has been verified with a test bench module that provides artificial result data to the input FIFO. The USB module did work correct. But in the performance analysis with real test data did the USB module turned out to be a bottleneck. This was caused by the transmission of the BLOB detection results in message sizes of one Byte. The additional overhead of the controller-device communication on the bus did cause a much higher delay than expected. This lead to an overflow of the internal FIFO that provided the input data for the USB module. It would have been a solution to change the message sizes to transmit complete results about a whole frame in a single message. But as an additional problem the USB communication module could not be integrated into the hardware design for the DE2-70 board. The problem here is the different way how the USB JTAG interface is hooked up to the FPGA on the DE2-70 board. This JTAG interface is not longer accessible directly on the board. Because of those two problems is the USB module not useful for the current approach.

8 Conclusion

The pre-processing stage to compress the relevant image data by run-length-encoding helped to bring more flexibility into the whole system design. It is also a major factor for the high performance ability that has not been completely exhausted thus far. The only critical point about the RLE is the definition of the encoding criteria. The minimum length that a run needs to have was defined to be at least two. This value should be changed, according to the application environment and the quality of the equipment. With a low-cost camera the noise in the image material might increase and lead to the detection of false positive results. But, the detection of single phantom runs will still not lead to wrong BLOB detection results, since a BLOB has to contain several runs that are adjacent. The only issue might be the decrease of the result precision for real BLOBs, if the false positive run is adjacent to the BLOB.

With respect to the expected image material, the identification of relevant pixels for BLOB detection is based on the brightness value of the pixels. All pixels that belong to a BLOB should have a significant higher value, compared to the pixels which are not part of a BLOB. The application of a fixed threshold value was not sufficient, since the variation of the BLOB's shape and intensity can be expected in the proposed application environment. This was the motivation for the implementation of the automated threshold adjustment. With a reference value for the size of the BLOB that is configured at startup of the system, the threshold value is updated by the system after every frame. The adjustment has been successfully used in the test model and worked as expected. The advantages of this functionality will pay off once the system is integrated into the target application environment to perform usability tests.

The integration of the USB communication module into the design did not work as expected. According to the specification for the DE2 board and the USB 2.0

standard a bandwidth between 1.5 and 625 Megabyte per second should be possible [50]. The design of the USB transmitter that has been reused for the USB module could not provide such performance. The transmission only worked in data packages of one byte at a time. Together with the communication overhead that comes with USB, the module performed slower than the serial interface module. In addition, there were some changes in the hardware architecture between the DE2 and the DE2-70 board. Those changes did not allow the integration of the USB module into the hardware design for the DE2-70 board.

The integration of the four digit subpixel precision had an observable impact on the detection and tracking results. Having this additional refinement allowed to reduce the rounding error for the integer values to decimal values in steps of 0,0625. The benefit was observable in the visualization of the results for the BLOB detection and tracking designs. The output showed a highly visible jitter for the BLOB center points. This jitter was reduced with the subpixel precision.

All settings for the hardware design have been integrated into one global configuration file. This had a major impact on the flexibility of the hardware design. It does not only allow the user to switch between existing functionality but also simplifies the integration of new equipment, such as different input cameras, into the design.

The hardware design has been implemented to allow detection and tracking from five up to eight BLOBs. These configurations have been tested successfully on the target boards DE2 and DE2-70 for the configurations that could fit. The decrease of the maximum performance was still sufficient for the input speeds of the applied cameras. In Section 7.4 the increase of resource requirements for the design with additional BLOBs has been analyzed. For the given target platforms a maximum number of thirty BLOBs detection might be possible, using the Bounding Box method for the center point computation. An additional performance evaluation would be required if the detection of thirty BLOBs is attempted. For the given concepts of BLOB detection and tracking, it is not recommendable to increase the number of possible BLOBs much further. Target platforms with larger FPGAs are available on the market but the price for them is ten to one-hundred times higher. This would not fit into the idea of the Immersion Square project to create immersive environments with low cost.

The main goal of this thesis was to compare two tracking solutions for the proposed BLOB detection approach. The tracking functionality has been implemented in hardware and in software. The important aspects for the validation were focused precision, performance and implementation cost. Both approaches worked reliably based on the same input data. The software application could be verified to work correctly for 45 fps using the D5M camera mounted onto the DE2 board providing the BLOBs center points. For the tracking in hardware the maximum performance was estimated by its maximum clock rate, which depends on the number of BLOBs that can be processed. Comparing only the performance value one would say that the tracking in hardware is the preferable solution. However, taking into account that the software approach does not only use euclidean distance to track the BLOBs, but also uses the list of neighbours to match new BLOBs to old ones, the software solution is more powerful. The high-level implementation allows a flexible variation for the number of BLOBs while the hardware implementation requires an extension of the design. This would again require a target platform with a bigger FPGA. This results in the tracking in software much more attractive than the tracking in hardware. In addition, the implementation in software took less than half the implementation time that was required for realizing the design in hardware. If high-performance is a major requirement the BLOB tracking in hardware would be a possible way to solve that problem, as has been shown in this work. Using a concept that combines a soft core processor with a hardware design might be a way to overcome the downsides of the existing tracking approach.

The overall precision error of 0,078 % for the center point computation was estimated with respect to the applied image resolution of 640x480 pixels. It has to be taken into account that the assumed ground-truth data of the software based approach might not compute the exact center point. The results of the software implementation is computed in integer values and might show a rounding error of one pixel in either direction on the image plane. For verifying this statement the application of image material with higher resolution as input for the software approach is a possible way. This would allow a more precise comparison of the software implementation and the subpixel precision of the hardware approach.

9 Future Work

The validation of the RLE encoding in Section 7.1 shows that the serial interface is too slow to transmit the entire image data in real-time. This is not an issue for the transmission of the detection and tracking results so far. But, it might become an issue if faster frame rates can be achieved or more information about the BLOBs are to be included in the result data. A USB transmitter module could be integrated into the design of the DE2 board. The performance validation showed that the design of the USB transmitter was not suitable for the application purpose. The restricted communication that could only send packages of one byte did not perform as required. With the additional overhead for host-device communication in USB, the module had slower performance than the serial communication module. The implementation of a complete module to handle the USB interface and the protocol processing is an open point that is suitable to address in future work.

For a better validation of the overall system it would be useful to have the ability to feed synthetic test data into the system. This would permit test data with ground truth values that can be used for a more precise validation. Having such an interface would solve the problem of changing validation results and resource requirements that are caused by integrating test benches into the hardware design. Available technologies are either the serial interface, the USB interface or the Ethernet interface. As discussed above, the serial interface is not sufficient to transmit a whole image in real-time. For the application of the USB or the Ethernet interface, the design requires several cooperating modules. These interfaces require modules which handle the low level communication with the particular chip on the target platform and the high level protocol handling.

An open problem of the existing approach for the BLOB detection and tracking is the required implementation work to extend the design to support more BLOBs. This results in an increase of resources that are occupied by the hardware design.

This is caused by the technique in which the detected runs are merged into BLOBs. The adjacency checks have to be performed for each detected run with every BLOB container that has been allocated in the registers of the FPGA. It would give the design more flexibility if the registers could be replaced by memory modules where the number of detected BLOBs can be changed during runtime. One possible way would be the implementation of a memory management unit, that dynamically changes the number of containers during the detection or tracking process. The adjacency check for all existing containers would then depend on their number. The more BLOBs present in the image material, the longer the system would take to perform the detection or tracking process. Another opportunity is the integration of a soft core processor that can execute implementations in high-level languages. The search and match procedures for the adjacency check and the tracking steps could be programmed more flexibly in software. This would change the limitation for the maximum number of BLOBs that can be processed, because it depends on the amount of memory that is available and not on the number of logic elements in the FPGA. The performance of the hardware design would remain constant with an increasing number of BLOBs. An evaluation of the resource requirements for an appropriate soft core processor would be required in advance.

The current results regarding the detected and tracked BLOBs are very condensed. By integrating more features for the analysis of the BLOBs, the system could be used to estimate more detailed information about the user behavior. Such as, direction of movement or point of interest. It might also allow one to predict the next step of the user by classifying repetitive input patterns. One possible extension might be the identification of the BLOB's head and tail to predict the direction of movement. This could be solved by creating a histogram of the distribution for the brightness value of the BLOB. The head of a BLOB can be expected at the brightest spot of the BLOB. Different kinds of motion with the light emitting device by the user in the Immersion

Square will create different kinds of BLOB shapes. The rotation of the light emitting device will show different BLOBs than the movement in one particular direction. By classifying the different kinds of BLOB shapes the application of a pattern recognition procedure could be used to extract more information about the user behavior and intention. With the current tracking solution the system is transmitting the results for all BLOBs for each frame where the matching was successful or where a new BLOB occurred. The system does not check if the position of the BLOB has changed. It would be possible to reduce the required bandwidth for the transmission of the tracking results to the connected host-PC by applying a static target cancellation. Only the results for the tracked BLOBs that have changed and the information of new BLOBs would be transmitted. In addition the loss of BLOBs in the tracking results would be required to eliminate outdated BLOBs. The BLOB detection and tracking solution has been evaluated and verified, using a simplified test model. For the proposed target application it would be useful to gain more information about the usability and feasibility of the system in the Immersion Square environment [28]. The integration of the system into the application environment would allow a comprehensive analysis of those criteria. This will be a main project target in the ongoing research work of the related MI6 project.

References

[1] Opencv 2.1 c++ reference. BSD license, 2010.

[2] F. Alsaqre and Y. Baozong. Multiple moving objects tracking for video surveillance systems. In *Signal Processing, 2004. Proceedings. ICSP '04. 2004 7th International Conference on*, volume 2, pages 1301 – 1305 vol.2, 31 2004.

[3] Altera. *Quartus II Handbook Version 10.0.* Altera, 101 Innovation Drive, San Jose, CA 95134, 1 edition, July 2010.

[4] Altera. *SCFIFO and SCFIFO Megafunctions User Guide.* Altera Cooperation, 101 Innovation Drive, San Jose, CA 95134, January 2010.

[5] ASA. Gige vision standard, 2010.

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

[7] R. T. Azuma. A survey of augmented reality. *In Presence: Teleoperators and Virtual Environments 6*, 6:355–385, August 1997.

[8] D. G. Bariamis, D. K. Iakovidis, and D. E. Maroulis. An fpga-based architecture for real time image feature extraction. In *in: Proceedings of the ICPR International Conference on Pattern Recognition*, pages 801–804, 2004.

[9] 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.

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

[11] A. J. Bernstein. Analysis of programs for parallel processing. *Electronic Computers, IEEE Transactions on*, EC-15(5):757 –763, oct. 1966.

[12] N. D. Binh. A robust framework for visual object tracking. In *Computing and Communication Technologies, 2009. RIVF '09. International Conference on*, pages 1 –8, 13-17 2009.

[13] A. Bochem, R. Herpers, and K. Kent. Hardware acceleration of blob detection for image processing. In *Advances in Circuits, Electronics and Micro-Electronics (CENICS), 2010 Third International Conference on*, pages 28 –33. CENICS, Juli 2010.

[14] A. Bochem, R. Herpers, and K. B. Kent. Acceleration of blob detection within images in hardware. Technical Report TR 09-197, University of New Brunswick, Faculty of Computer Science, Fredericton, NB, E3B 5A3, December 2009.

[15] A. Bochem, K. Kent, and R. Herpers. Acceleration of blob detection in a video stream using hardware. Technical Report TR 10-201, University of New Brunswick, Faculty of Computer Science, Fredericton, NB, E3B 5A3, April 2010.

[16] S. D. Brown and Z. Vranesic. *Fundamentals of Digital Logic with Verilog Design*, volume 1. McGraw-Hill Higher Education, 2003.

[17] H. Chen-Huei, H. Yue-Wei, and W. Menq-Jiun. Automatic concentricity measurement by image processing. pages 853 –857 vol.2, may. 1994.

[18] J. U. Cho, S. H. Jin, X. D. Pham, and J. W. Jeon. Object tracking circuit using particle filter with multiple features. In *SICE-ICASE, 2006. International Joint Conference*, pages 1431 –1436, 18-21 2006.

[19] C. Cruz-Neira, D. J. Sandin, and T. A. DeFanti. Surround-screen projection-based virtual reality: the design and implementation of the cave. In *SIGGRAPH '93: Proceedings of the 20th annual conference on Computer graphics and interactive techniques*, pages 135–142, New York, NY, USA, 1993. ACM.

[20] D. Culler, J. Singh, and A. Gupta. *Parallel Computer Architecture: A Hardware/Software Approach.* Number ISBN-13: 978-1558603431. Morgan Kauufmann, 1 edition, August 1998.

[21] E. Davis. *Techgnosis: myth, magic and mysticism in the information age*, volume 2. Three Rivers Press, November 16 1999.

[22] EMVA. Generic interface for cameras (genicam) standard, 2006.

[23] T. Frank, M. Haag, H. Kollnig, and H.-H. Nagel. Tracking of occluded vehicles in traffic scenes. In *ECCV '96: Proceedings of the 4th European Conference on Computer Vision-Volume II*, pages 485–494, London, UK, 1996. Springer-Verlag.

[24] O. Grau. *Virtual Art - From Illusion to Immersion.* Number ISBN-13: 978-0262072410. The MIT Press, January 17 2003.

[25] 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.

[26] M. Henning and M. Spruiell. Distributed programming with ice standard, June 2010.

[27] R. Herpers, W. Heiden, M. Kutz, D. Scherfgen, U. Hartmann, J. Bongartz, and O. Schulzyk. Fivis - bicycle simulator in the immersive visualisation environment "immersion square". 2009.

[28] R. Herpers, F. Hetmann, A. Hau, and W. Heiden. Immersion square - a mobile platform for immersive visualisations. University of Applied Sciences Bonn-Rhein-Sieg, 2005.

[29] S. Hirai, M. Zakoji, A. Masubuchi, and T. Tsuboi. Fpga-based realtime vision system. Journal of Robotics and Mechatronics, 2005. Vol.17, No.4, pp.401-409.

[30] IEEE. Ieee standard hardware description language based on the verilog(r) hardware description language, Oct 1996.

[31] IEEE. Ieee standard verilog hardware description language, 2001.

[32] IEEE. Standard for systemverilog - unified hardware design, specification, and verification language, 2007.

[33] IEEE. Ieee standard vhdl language reference manual, Jan 2009.

[34] Intel. *Intel(R) Threading Building Blocks - Getting Started Guide.* Intel, 004 edition, December 2009.

[35] Intel. *Intel(R) Threading Building Blocks - Reference Manual.* Intel, 1.18 edition, October 2009.

[36] Intel. *Intel(R) Threading Building Blocks - Tutorial.* Intel, 1.16 edition, November 2009.

[37] U. Lansche. *Matrix Vision - mvBlueCOUGAR-X Documentation.* Matrix Vision, Talstrasse 16, 2009.

[38] U. Lansche. *mvBlueCOUGAR-X.* Matrix Vision GmbH, Talstrasse 16, DE - 71570 Oppenweiler, 1.0 edition, April 2010.

[39] 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.

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

[41] 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.

[42] G. D. Micheli, R. K. Gupta, Rajesh, and K. Gupta. Hardware/software co-design. *IEEE Micro*, 85:349–365, 1997.

[43] P. Milgram and F. Kishino. A taxonomy of mixed reality visual displays. *IEICE Transactions on Information Systems*, E77-D(12), December 1994.

[44] MPI. Mpi: A message-passing interface standard, September 2009.

[45] OMG. Common object request broker architecture (corba/iiop) standard, 2008.

[46] opencores.org, http://www.opencores.org. *Online Community for Open Source Projects in ASIC and FPGA design.*, October 2010.

[47] OpenMP. Openmp application programming interface - the openmp api specification for parallel programming standard, May 2008.

[48] J. Reinders. *Intel Threading Building Blocks: Outfitting C++ for Multi-Core Processor Parallelism.* Number ISBN-13: 978-0596514808. O'Reilly Media, 1 edition, July 2007.

[49] S. W. Smith. *The Scientist & Engineer's Guide to Digital Signal Processing.* Number ISBN 978-0966017632. California Technical Pub., 1st edition, 1997.

[50] Terasic. *DE2 Development and Education Board - User Manual.* Altera, 101 Innovation Drive, San Jose, CA 95134, 1.41 edition, 2007.

[51] Terasic. *Terasic TRDB-D5M Hardware Specification.* Terasic, June 2008.

[52] Terasic. *DE2-70 Development and Education Board - User Manual.* Altera, 101 Innovation Drive, San Jose, CA 95134, 1.08 edition, 2009.

[53] Texas-Instruments. *MAX232, MAX232I DUAL EIA-232 DRIVERS/RECEIVERS*. Texas Instruments, Post Office Box 655303, Dallas, Texas 75265, March 2004.

[54] 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.

[55] F. Vogt, J. Wong, S. Fels, and D. Cavens. Tracking multiple laser pointers for large screen interaction. In *Ext. Abstracts UIST*, pages 95–96, 2003.

[56] C. Wienss, I. Nikitin, G. Goebbels, K. Troche, M. Gőbel, L. Nikitina, and S. Műller. Sceptre - an infrared laser tracking system for virtual environments. volume isbn 1-59593-321-2, pages pp. 45–50. Proceedings of the ACM symposium on Virtual Reality software and technology VRST 2006, 2006.

[57] B. Wilkinson and P. Calder. Augmented reality for the real world. In *Computer Graphics, Imaging and Visualisation, 2006 International Conference on*, pages 452 –457, jul. 2006.

[58] L.-Q. Xu, J. L. Landabaso, and B. Lei. Segmentation and tracking of multiple moving objects for intelligent video analysis. *BT Technology Journal*, 22(3):140–150, 2004.

[59] M. Yaki and M. Youssef. Tnrac: a system for tracking multiple moving non-rigid objects using an active camera. *Signal, Image and Video Processing*, 3(2):145–155, June 2009.

Vita

Candidate's full name:	Alexander Bochem
University attended:	B.Sc. Computer Science 2008
	Bonn-Rhein-Sieg University of Applied Sciences
	Sankt Augustin, Nordrhein-Westfalen, Germany

Publications:

A. Bochem, R. Herpers, and K. B. Kent. Acceleration of blob detection within images in hardware. Technical Report TR 09-197, University of New Brunswick, Faculty of Computer Science, Fredericton, NB, E3B 5A3, December 2009.

A. Bochem, R. Herpers, and K. Kent. Hardware acceleration of blob detection for image processing. In Advances in Circuits, Electronics and Micro-Electronics (CENICS), 2010 Third International Conference on, pages 28 33. CENICS, Juli 2010.

A. Bochem, K. Kent, and R. Herpers. Acceleration of blob detection in a video stream using hardware. Technical Report TR 10-201, University of New Brunswick, Faculty of Computer Science, Fredericton, NB, E3B 5A3, April 2010.