ASTERICS - An Open Toolbox for Sophisticated FPGA-Based Image Processing

Michael Schaeferling*, Markus Bihler*, Matthias Pohl†, Gundolf Kiefer*

*University of Applied Sciences Augsburg
Dept. of Computer Science
An der Hochschule 1
86161 Augsburg, Germany
Email: {gundolf.kiefer, michael.schaeferling }
@hs-augsburg.de

†Heidelberger Druckmaschinen AG
Dept.: Control Systems and Electronic Components
Kurfürstenanlage 52-60
69115 Heidelberg, Germany
Email: matthias.pohl@heidelberg.com

***Abstract*—Image processing on embedded platforms is still a challenging task, especially when implementing extensive computer vision applications. Field-programmable gate arrays (FPGAs) offer a suitable technology to accelerate image processing by customized hardware. Most available image processing frameworks for FPGAs concentrate on pixel-based modules for simple preprocessing tasks. This paper presents a framework which also aims to cover the integration of higher-level algorithms. Therefore, it offers modules and interfaces to perform window-oriented filter operations and incorporate software-defined operations. Several complex, higher-level algorithms, such as undistortion and rectification, natural feature description, edge detection and Hough transform have been adopted and integrated into the frameworks image processing flow. This paper describes the framework, its interfaces and some of the existing modules. Finally, some applications which were already implemented using this framework are presented.**

Keywords: real-time image processing, computer vision, system on chip, programmable logic, image rectification, corner detection, Hough transform, edge detection, natural features

I. Introduction

Computer vision is a growing area of research and development with numerous applications in the automotive, medical systems, and consumer electronics industries. The involved algorithms range from simple image preprocessing to sophisticated interpretation tasks. Especially in the field of embedded systems, available processing power, memory bandwidth and power resources are very limited, all of which would be needed to perform real-time image processing. Using dedicated hardware is a well-known way to accelerate signal processing. Field Programmable Gate Arrays (FPGAs) provide the required flexibility to create hardware structures to optimally fit a given algorithm and to efficiently exploit parallelism. Additionally, growing capacities make FPGAs a seminal platform with the possibility to integrate even more complex image processing tasks in the future.

Computer vision systems usually contain a chain of more or less common operations, starting with simple image filters and ending with complex interpretation algorithms to extract meaningful information such as ”a traffic sign X is visible at location Y in the image”. In general, the operations can be divided into four classes, namely a) point-based, b) window-based, c) semi-global and d) global operations, the characteristics of which are summarized in Figure 1 [1][2].

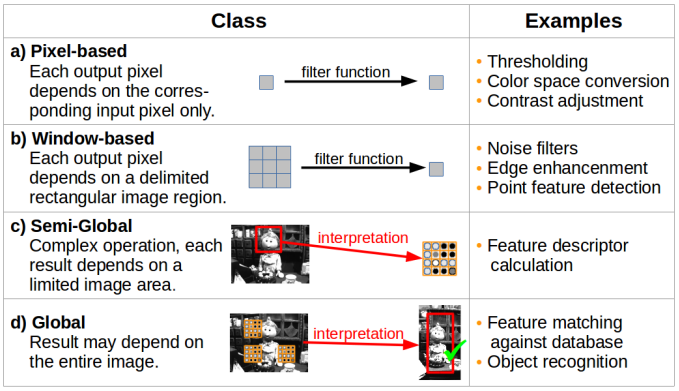


Fig. 1: Operation classes in image processing

In a point-based operation, each output pixel depends on the corresponding pixel of the input image only. Examples are brightness adjustment or thresholding. In a window-based operation, each output pixel depends on a set of pixels located around the corresponding pixel of the input image. Examples are noise filters or simple edge filters. Both global and semi-global operations are typically complex tasks and defined in software. Unlike global operations, semi-global (or patch-based) operations only use a defined area of the image to gather high-level information, for example to perform feature descriptor calculation in object recognition.

While current commercial frameworks mainly address pixel-based operations, the proposed *ASTERICS* (”*Augsburg Sophisticated Toolbox for Embedded Real-time Image Crunching Systems*”) framework aims to become an open toolbox for all of the four classes of image processing operations. Using defined, simple interfaces and a fine-grained module structure, the *ASTERICS* framework also aims to simplify and accelerate the development process. Thus, it contains a library of hardware modules, connected via unified interfaces

for point and window operations. Window-based operations are implemented by an efficient, semi-automatically generated 2D pipeline structure. Semi-global and global operations are handled by a hardware-software-codesign, where hardware structures seamlessly extract and provide data for software-based processing. So-called *Patch Processing Units* (*PPUs*), implemented as small standard CPUs or specialized hardware, perform semi-global operations. Global operations are processed by one or more soft- or hard-core CPUs. All components together form an image processing SoC, which may be implemented in an FPGA device or an ASIC.

So far, several demonstration applications and complex image processing chains have successfully been implemented using the framework. These include a real-time implementation of the *SURF* ("*Speeded Up Robust Features*") feature detector, the Canny edge detector, a configurable Hough transform module suitable for mid-size FPGAs, and a module to perform real-time image distortion removal and stereo rectification.

An overview on related work towards image processing frameworks for reconfigurable hardware is given in Section II. The basic concepts and interfaces of the *ASTERICS* framework are described in detail in Section III. Section IV gives an overview on selected modules. Some applications which were implemented using the *ASTERICS* framework are described in Section V. Section VI concludes the work and gives some future prospects.

II. Related Work

There are several image processing frameworks proposed in the field of programmable logic. A framework to preprocess image data within distinct processing modules to finally estimate depth information is presented by Geisen et. al. [3]. Kasik and Peterek present another video processing toolbox, correcting and applying effects on image data [4]. Toolkits for image processing are offered amongst other IP cores for image (de)coding for example by Xilinx and Altera [5][6]. By design, these architectures only cover pixel-based and/or global image operations as they typically only support basic pixel-based modules. Structures to efficiently perform window-based and semi-global operations are not specifically addressed.

SoC-Frameworks can be used to build SoPC architectures with a main system bus to connect one or several CPUs, memories and peripheral components. Xilinx therefore provides the *Embedded Development Kit* and the *Vivado Design Suite*, where both soft-core (*MicroBlaze*) and hard-core (ARM) processors can be integrated into the system. With the *SoPC Builder* and the *SoC Embedded Design Suite*, Altera also provides CAD software to build SoPCs incorporating their *Nios II* soft-core processor or an ARM hard-core processor.

Various bus standards for SoC architectures have been established, such as the PLB (CoreConnect, IBM) or the Avalon (Altera) bus, traditionally used within Xilinx and Altera SoPCs, respectively. The AXI (AMBA, ARM) bus nowadays is used when the SoPC incorporates an ARM processor, with Xilinx using its streaming capabilities in their image processing framework. As an open-source alternative, the Whishbone standard offers a relatively simple interface to connect master and slave components. It can be implemented in different topologies, for example as shared bus or crossbar switch, but also a dataflow topology allowing to chain up multiple processing elements connected pairwise by streaming-like interfaces.

III. ASTERICS: Concepts and Interfaces

The *ASTERICS* framework is designed as a modular building set to perform various real-time image processing tasks. Figure 2 depicts a typical layout of an image processing system supported by the framework. It consists of several image processing modules, each performing a dedicated processing task. The framework features different interfaces to connect the incorporated modules and to pass image data within the pipeline from the data source to a data sink. Especially the kernel-based interface, featuring a novel *2D Window Pipeline* module, and the software interface for semi-global image processing enable the framework to realize image processing pipelines covering all four classes listed in Figure 1.

A. Pixel-based Interface

In many applications, the image data flow starts with pixel-based processing, which is performed within the framework by a set of consecutive *Filter Modules*. First, image data needs to be acquired, for example from an image sensor or from system memory. The subsequent *Filter Modules* perform basic filter operations, such as cropping or contrast adjustment, in order to preprocess the image for the application at hand. An overview on selected modules is given in Section IV-A. Filter modules are connected by a streaming interface named *Real-Time Pixel Bus (RTPB)*. An *RTPB* transports pixel-related data along with meta information, such as data valid and frame synchronization signals. Furthermore, the interface supports backpressure as a method to slow down or stall data traffic. Thus, a module can generate a stall signal which is passed to the preceding module, indicating that the predecessor should not pass any data from the next cycle on, until the signal is revoked.

B. Kernel-based Interface

Kernel- or window-based processing, as, for example, local filters or other locally narrow operations, can efficiently be performed using a sliding window buffer [1][7]. The principle of the sliding window technique is depicted in Figure 3. The window buffer stores a number of consecutive image lines, with a rectangular part (the so-called window) being implemented by registers, so that they are freely accessible. With each pixel shifted into the image data buffer, the window virtually slides forward within the image. A *Window Module* implements a specific window-based operation using multiple pixels. It is connected to the image data buffer to access all required image data. Thus, the *Window Module* continuously calculates a new result for each image position.

ASTERICS introduces the concept of a configurable *2D Window Pipeline Module*, which incorporates an enhanced

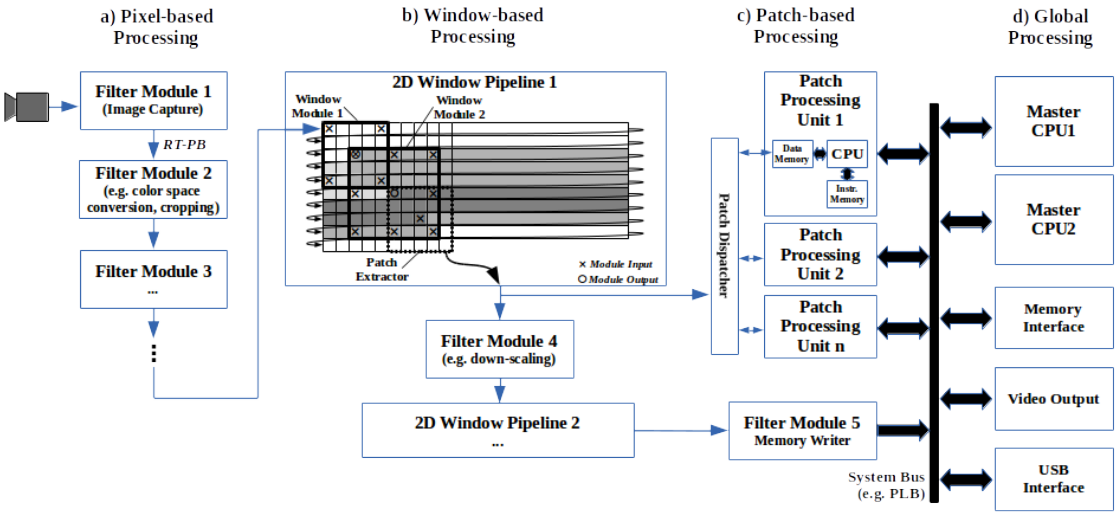


Fig. 2: Exemplary structure of an image processing system

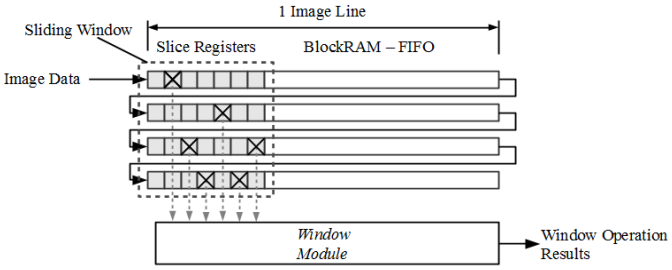


Fig. 3: Principle of the sliding window technique

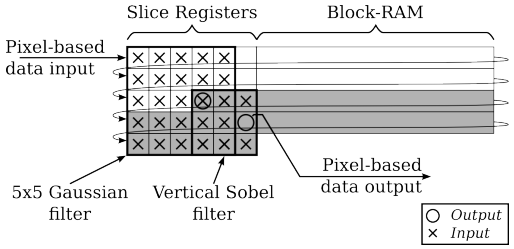


Fig. 4: Exemplary *2D Window Pipeline*, implementing Gaussian filtering and the Sobel operator

sliding window buffer and an arbitrary number of *Window Modules*. The output of a *Window Module* is either written back to the pipeline to be further processed by other *Window Modules* or passed to a subsequent *Filter Module*.

Figure 4 depicts an exemplary *2D Window Pipeline* with two *Window Modules*. The first 5x5 filter applies Gaussian filtering to the incoming pixel data. Its result is written back into the pipeline at the position marked by ⊗. From that point, the 2D pipeline (additionally) transports the filtered data (marked grey). The second *Window Module* (3x3) applies the vertical Sobel operator, where the top and bottom rows are weighted, to detect vertical edges. As a whole, the *2D Window Pipeline* uses the pixel-based interface to connect to preceding and following *Filter Modules*.

In order to support the system designer at this point, the framework provides a tool to semi-automatically generate and optimize the data pipeline structure, offering a trade-off between latency and resource utilization. First, a graphical editor is used to place the *Window Modules* within the pipeline. Second, the tool generates the pipeline structure as a combination of slice registers and Block-RAM FIFOs, as slice registers are needed to access all required image data simultaneously and Block-RAM is used to buffer data. Detailed information on the kernel-based interface can also be found in [2].

C. Software Interface

Semi-global and global operations are usually defined by complex algorithms and may need to be implemented in software. The *ASTERICS* framework offers methods to handle both classes efficiently.

Semi-global operations are applied to selected regions-of-interest or patches of the original image. For this, the framework provides *Patch Extractor Modules* to identify and extract relevant image patches from the data stream of the *2D Window Pipeline*. The extracted data is passed to an array of small, dedicated *Patch Processing Units* (*PPUs*). These *PPUs* are capable to execute software algorithms and can operate fully parallel without interfering with each other. The assignment of subtasks to *PPUs* is coordinated by the *Patch Dispatcher*. Each *PPU* contains a fast local memory which is also accessible by the *Patch Dispatcher* to allow fast data transfers to the *PPU*. The operation's results, which often do not represent image pixels but rather some kind of interpretation of the actual image data, are passed to the memory of the main system.

Finally, high-level algorithms, where global operations are also defined in software, may be applied to results gained so far. The processing chain as yet described can be incorporated into a common System-on-Chip structure with one or more Master CPUs to execute these algorithms. Further details on

Pixel-based modules
Input and Output modules
Capture from image sensors
Memory Reader
Memory Writer
Adapters and format converters
Crop
Scaler
Synchronization
Stream-Selector
Collect / Disperse
Join / Split
Standard Filters
Brightness adjustment
Contrast adjustment
Difference image calculation
Histogram
Window-based modules
FIR modules (Gaussian filter, ...)
Bayer-RGB-Converter
Advanced processing
Non-Linear Image Transformations
Natural Feature Detector
Canny Edge Detector
Hough Transform
System-level modules
Video Output
USB Interface

TABLE I: Provided modules

the software interface are described in [2].

IV. ASTERICS: MODULES

The *ASTERICS* framework offers a catalogue of modules for (mostly) real-time image processing, ranging from simple preprocessing tasks to complex image transformation and analysis, and is open to be enriched by future developments. Table I gives an overview on some existing modules. The next subsection gives a brief overview on the catalogue and its standard modules. After that, some of the more sophisticated modules are described in detail.

A. Standard Modules for Pixel- and Window-based Operations

For image acquisition, the framework provides interfaces to capture data from image sensors of various vendors. Frames can either be captured in continuous mode or triggered in a single shot mode, which may be needed for stereo vision applications. Data may also be acquired by reading from the systems main memory using the *Memory Reader* module. Conversely, results can be stored there using the *Memory Writer* interface module. Current implementations support PLB or AXI bus interfaces, but can easily be extended towards other bus standards.

A set of adapters and format converters help to optimally adjust the image data pipeline, which is not fixed to a set of predefined image data types. Image dimensions can be changed by cropping or scaling image data. Synchronization of image data streams, for example needed for stereo vision applications, is also supported. Data flow management is performed by another set of modules. The stream selector module can be used to switch between different data processing stages. Also, data streams may be joined or split, for example to accompany corresponding pixels of stereo images into a single data stream. Furthermore, consecutive pixels of the image data stream can be collected and packed into a single data word. This can be used, for example, for parallel pixel processing or to efficiently transfer four 8-bit grayscale pixels using a 32-bit memory writer module. Conversely, another module is capable to disperse such packed data words to obtain sequenced pixel values again.

A set of filters for standard operations is implemented, such as inverting image color data or adjusting brightness and contrast by addition or multiplication of pixel data with a fixed value. For image data analysis, the framework offers a module to calculate the image histogram, which may be used as a basis for the afore mentioned image enhancement tasks.

The framework provides a method to efficiently integrate window-based operations into the processing pipeline, as for example FIR filters. Existing modules for this class of image operations perform Gaussian blurring or apply the Sobel operator to image data. When dealing with color image sensors, image data is often provided in a Bayer pattern format. As many image processing algorithms are designed to handle RGB image data, the Bayer-to-RGB converter module can be used for conversion.

Data visualization can be performed on system level, using the comprised video output module. It outputs image data to a monitor, where the module supports both analog and digital data output. Augmentation of the displayed image data is supported by the modules software library to draw elements to an overlay, for example lines, circles or text. The framework also provides a USB interface module, for example to connect an *ASTERICS* enabled FPGA-based camera with a host system. A flexible software interface supports the system developer to link the host with the *ASTERICS* system.

Besides the afore mentioned standard modules, the *ASTERICS* framework also incorporates more complex modules for image analysis and more extensive transformations, as described in the following sections.

B. Non-Linear Image Transformations

Many computer vision applications use stereo images, for example, to gain depth information of the observed scene. Therefore it is important to remove lens distortion and to rectify the stereo image pair, as depicted in Figure 5.

Figure 6 shows another exemplary transformation. The camera sensor captures a whiteboard from a displaced point of view, which would preserve an observer to properly perceive the content. Image data is now undistorted and transformed in a way, so that the resulting output image shows the information as if the camera was located directly in front of it.

For all these non-linear image transformations, the framework features the *NITRA (Non-linear Image TRAnsformation)* module. The *NITRA* module comprises an optimized architecture (depicted in Figure 7) to apply these transformations in

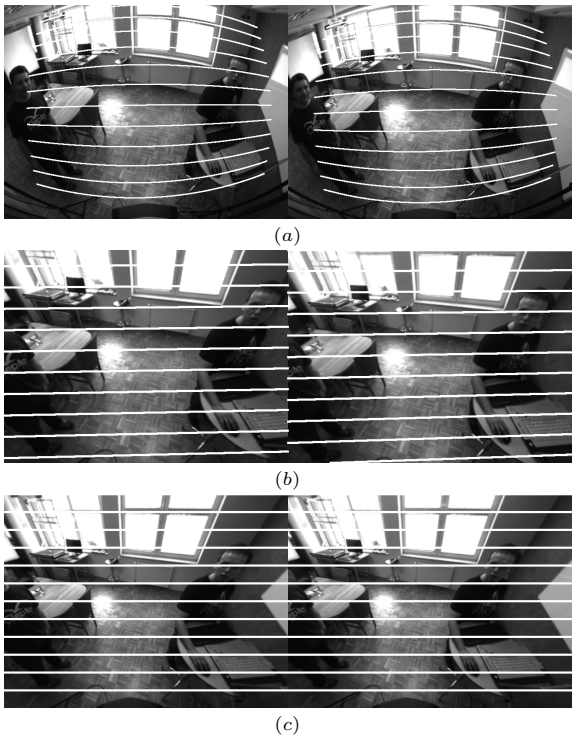


Fig. 5: Example images showing raw (a), undistorted (b) and rectified (c) image pairs of a stereo camera setup

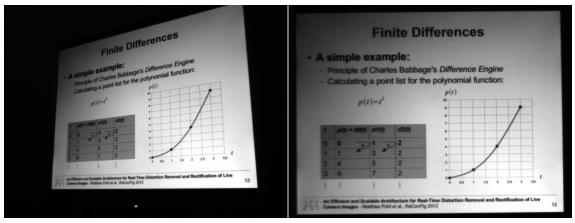


Fig. 6: Exemplary image transformation: Whiteboard remapping

real-time and allows a trade-off between accuracy and resource utilization.

Unlike other approaches, the *NITRA* module does not need expensive look-up tables for translating pixel coordinates. The *Coordinate Generator* calculates exact sampling coordinates based on higher-degree polynomial approximations. Incoming pixel data is stored in a local line buffer. The samples are read from the buffer, interpolated and written to the main memory. Due to the pipeline structure of the module, it is able to process a new pixel in each clock cycle. Thus, it features real-time capability and is suitable for high-resolution cameras with high pixel clock rates. More detailed information on this module is presented in [8]. An application for the *NITRA* module is presented in Section V-A.

C. Natural Feature Detection

Feature detection is a fundamental processing step for object recognition or optical tracking tasks, as often utilized in augmented reality applications. The prominent *SURF* (”*Speeded Up Robust Features*”) feature detector [9] has been implemented as a real-time capable module of the *ASTERICS* framework, making extensive use of the *2D Window Pipeline* concept introduced in Section III-B. Figure 8 depicts the *Window Modules*, and their arrangement in the *2D Window Pipeline*. In the underlying algorithm, approximations of Gaussian filters are applied to the original grayscale image on different scale levels (*Hessian Determinant Calculation, HDC*). Non-maximum-suppression is used to determine a sole result for each image coordinate. In the module, four *HDC* filter responses are calculated by the incorporated *Window Modules* for four scale levels in parallel, while a *Non-Maximum-Suppression Window Module* directly calculates the final result. This structure allows to process a new pixel in one clock cycle, which makes this module real-time capable even for higher resolution (e.g. HD) images. It represents one of the fastest SURF detector implementations published so far [2]. More information on the implementation is provided in [2], while Section V-B gives an example of a full object recognition application on a smart camera using this module.

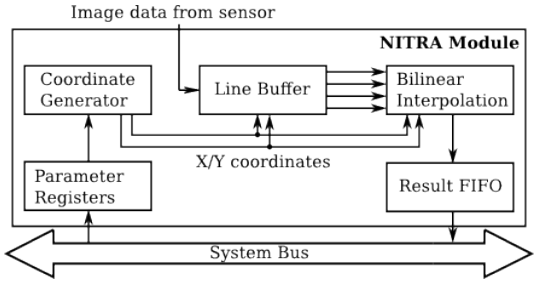


Fig. 7: Structure of the *NITRA* module for real-time non-linear image transformations

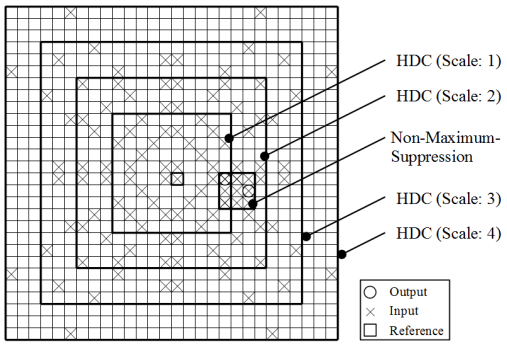


Fig. 8: Structure of the *SURF* detector module, used for real-time natural feature detection

D. Canny Edge Detector

Edge detection is an important preprocessing step for computer vision tasks like line detection or model-based object recognition. The *ASTERICS* framework provides an implementation of the Canny edge detection algorithm [10]. It consists of the four steps (a) Gaussian blurring, (b) calculation of a

gradient image with the Sobel operator, (c) non-maximum-suppression and (d) thresholding. These steps are implemented by pixel- and window-based filter modules as depicted in Figure 9. Originally, step (d) of the Canny edge detection algorithm applies a thresholding algorithm with hysteresis on the result of step (c). That would cause a latency of one frame, as it is required to step over the image data forwards and backwards. In the current implementation, step (d) is optimized for a small latency of 3 image rows and 11 pixels. The hysteresis is processed in one direction only. A detailed description of the hardware implementation of the Canny edge detection algorithm can be found in [11]. In Section V-C, a demonstrator system for the Hough transform using this module is described.

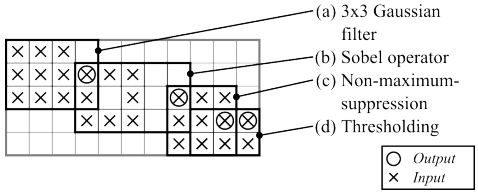


Fig. 9: Structure of the Canny edge detection module

E. Hough Transform

Detecting straight lines in an image is a demanding task in computer vision. One approach is the Hough transform [12], which can be used to detect lines, circles or any other arbitrary shapes in an edge image, by mapping the global detection problem to a more easily solved local peak detection problem in a parameter space [13]. The *ASTERICS* implementation calculates the parameter space of straight lines with the θ-ρ parametrization, where θ represents the angle of a line's normal and ρ represents the algebraic distance of a straight line to the coordinate system's origin [14]. Each edge pixel in the input image is regarded as an intersection point of a family of straight lines and the parameters of every straight line are calculated. Each assumed straight line causes one vote in the parameter space. After processing each edge pixel, local maxima in the parameter space image represent straight lines in the input image.

Figure 10 shows the structure of the *Hough Module*. The input is a binary edge image, as produced by the *Canny Module*, described in Section IV-D. The *Coordinate Counter* keeps track of the input image's pixel coordinates, which are required for the Hough transform. Then, the coordinates of the edge pixels are written into the *Edge Buffer*. The calculation of the parameter space image is implemented in the *Accumulator Array*, storing it in Block-RAM. Using parameters, the resolution of the parameter space is adjusted. The *Controller* coordinates the calculation and the read out process of the parameter space image.

As the *Hough Module* is able to buffer the edge pixels and additionally only a fraction of an edge image's pixels actually represent edges, the *Accumulator Array* does not have to process one pixel per clock cycle. It consists of a configurable

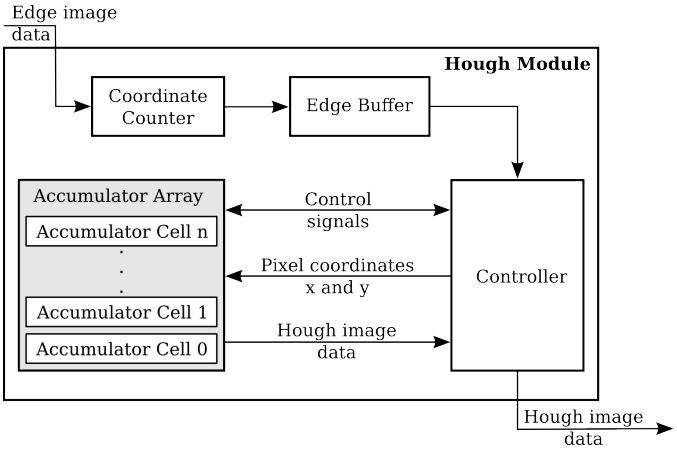


Fig. 10: Structure of the *Hough Module*

number of *Accumulator Cells*, splitting the parameter space image into independent sections, which are calculated in parallel. Within each *Accumulator Cell*, a configurable number of parameter space image rows are processed and stored. The *Accumulator Cells* are able to process one vote for one parameter space image row per clock cycle. Thereby, the level of parallelization of the *Hough Module* is configurable by the number of parameter space image rows processed per *Accumulator Cell*. The more *Accumulator Cells* are used, the more votes are calculated in parallel.

V. Applications

The following subsections give an overview on several complete image processing applications implemented using the *ASTERICS* framework on FPGA-based platforms.

A. Image Distortion Removal and Stereo Rectification

An image processing system was developed in association with *FORTecH Software GmbH* to gain undistorted and rectified images from captured raw image data, as depicted in Figure 5. These ideal image pairs are passed to a host system which performs gesture recognition and gaze analysis. The involved algorithms are based on analyzing stereo image pairs to gain gesture and gaze information, as depicted in Figure 11.

In this system, two stereo cameras (Aptina MT9V034) are triggered synchronously and image data is transferred directly into a Spartan 3E 1600 FPGA. Within the FPGA, both image data streams are processed with a resolution of 640x480 pixels in parallel by performing image transformation tasks within two instances of the *NITRA* module. The undistorted and rectified images are transferred via the *ASTERICS* USB interface to the host system, where the stereo images are used to perform further image processing tasks.

In this application, the incorporated *NITRA* modules were configured to guarantee a maximum total error of 0.3 pixels. Using this configuration, one *NITRA* module occupies just 12% of available Spartan-3 slices (1810 slices), 31% of Block-RAM (160 kBit) and 18% of the provided multiplier units (3x

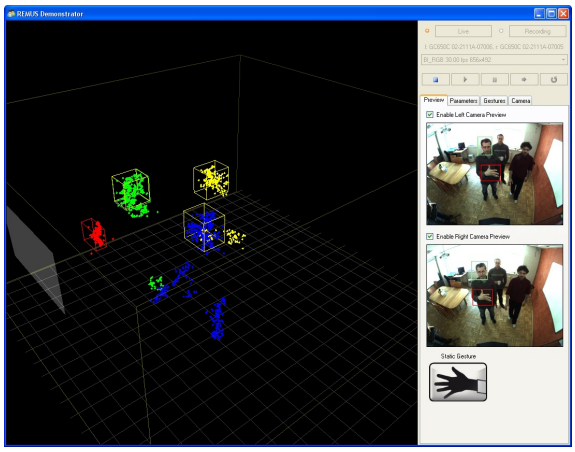


Fig. 11: Gesture recognition and gaze analysis, based on stereo images

MULT18) within the cameras' low-cost Spartan 3E FPGA. Further details on the application and implementation details can be found in [8].

B. Object Recognition using SURF Feature Extraction

A complete object recognition system, based on the well-known *SURF* algorithm which performs natural feature detection and description, was implemented in a single mid-sized FPGA using the *ASTERICS* framework. A mobile demonstrator platform has been developed for the detection and identification of objects in a live image stream which, for example, can be used as a mobile museum guide as depicted in Figure 12.

Fig. 12: SURF-based mobile museum guide

For this system, an FPGA-based camera was chosen for implementation. It features a Spartan-6 LX150 FPGA and an image sensor (Aptina MT9V034) for image capture while a VGA display directly connects to the SoPCs video output module to augment the original camera image with object information.

The structure of the system is depicted in Figure 13. It shows how all the image processing steps are covered by the framework. Image capture and preprocessing is realized within pixel-based processing modules. Following, computationally extensive real-time critical operations are handled within the frameworks structures for window- and patch-based processing. The *SURF* detector stage is realized using the *2D Window Pipeline* structure (see Section IV-C) while the software-based descriptor stage benefits from the concept of parallel semi-global operations using an array of independent *PPUs*. Based on the information gained so far, the SoPCs Master CPU executes the matching stage to finally identify and highlight found objects in the output image. More details on this application and implementation details are described in [2].

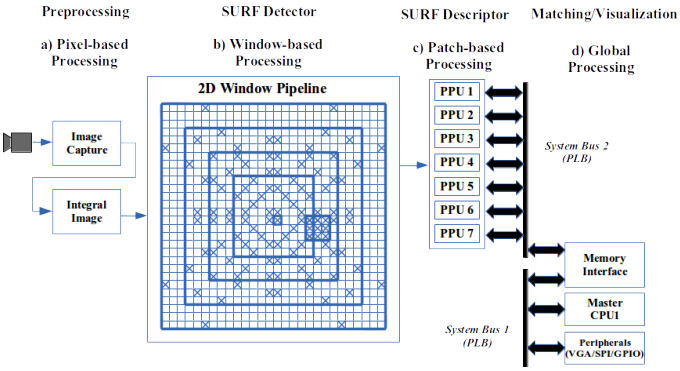


Fig. 13: Structure of the object recognition system, fully integrated in a single Spartan-6 LX150 FPGA

C. Line Extraction using Hough Transform

To show the capabilities of the *Canny Module* and the *Hough Module*, a demonstrator system has been developed. It is able to visualize the input image, the edge image of the *Canny Module* and the parameter space image, calculated by the *Hough Module*. Figure 14 shows the structure of the demonstrator system. The *Output Multiplexor* is configured via software drivers to choose a pixel stream of one of the three modules and forwards it to the *Memory Writer* memory. The *Video Output* module reads the output image data from the system memory and visualizes it by means of a VGA interface. Figure 15 shows three output images of the demonstrator system.

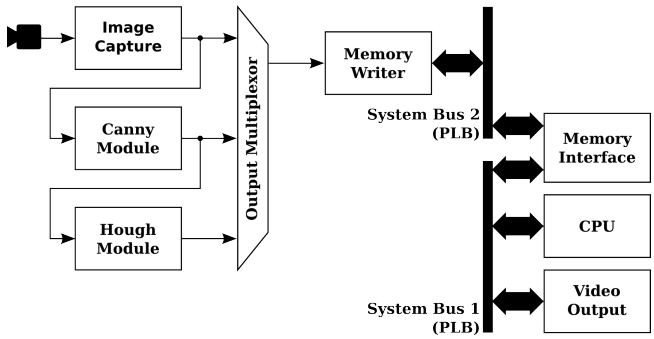
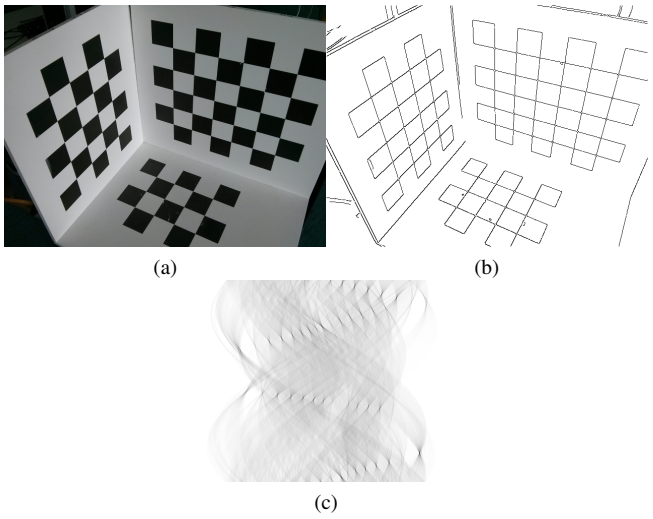


Fig. 14: Structure of the demonstrator system for the *Canny Module* and the *Hough Module*



(a) (b) (c)

Fig. 15: Exemplary output images of the demonstrator system; (a) input image (640x480 pixels); (b) edge image (640x480 pixels); (c) parameter space (512x320 pixels)

Several versions with different configurations of the *Hough Module* where implemented on a Xilinx Spartan-6 LX150 FPGA, equipped with an Aptina MT9V034 image sensor. The input image and the edge image calculated by the *Canny Module* have a resolution of 640x480 pixels. Less than 1% (189 slices) of the available Spartan-6 slices and 1.5% of Block-RAM (64 kBit) are used by the *Canny Module*. The resource utilization of the *Hough Module* and the resolution of the parameter space image depend on the *Hough Module's* configuration. The highest possible resolution, implementing the *Hough Module* in a Spartan-6 LX150 is 512x848 pixels with a depth of 9 bits and an *Accumulator Array* consisting of 53 *Accumulator Cells*. 10% of available Spartan-6 slices (2334 slices), 90% of Block-RAM (3832 kBit) and 59% of the multiplier units (106x MULT18) are occupied. The smallest implemented configuration has a parameter space image resolution of 128x128 pixels with a depth of 9 bit divided into 4 *Accumulator Cells*. Therefore, the *Hough Module* uses 1% of available Spartan-6 slices (234 slices), 5% of Block-RAM (176 kBit) and 4% of the multiplier units (8x MULT18). These results show the flexibility of the *Hough Module*. Depending on the requirements, it allows a trade-off between a high Hough space resolution and a high level of parallelization or a low hardware utilization.

VI. Conclusion

This paper introduced the *ASTERICS* framework, a flexible and efficient framework for computer vision and related image processing tasks. It covers simple pixel-oriented operations and window operations, but also supports complex semi-global and global operations. The underlying module library enables the system designer to efficiently build systems for sophisticated image processing tasks. Various applications implemented so far demonstrate the capabilities of the framework. Future work deals with rapid prototyping support and extensions of the module library towards shape-oriented computer vision tasks and advanced industrial inspections.

Acknowledgement

Parts of this work have been supported by the German Federal Ministry of Education and Research (BMBF), grant number 17N3709.

The authors would like to thank *FORTecH Software GmbH*, Rostock, for cooperation and valuable discussions towards the implementation of non-linear image transformations.

We are also grateful for the cooperation with the *Augsburger Puppenkiste*, which provided a real-world object recognition scenario.

References

[1] C. T. Johnston, K. T. Gribbon, and D. G. Bailey, "Implementing Image Processing Algorithms on FPGAs," in *Proceedings of the Eleventh Electronics New Zealand Conference (ENZCon)*, 2004, pp. 118–123.

[2] M. Pohl, M. Schaeferling, and G. Kiefer, "An efficient FPGA-based hardware framework for natural feature extraction and related Computer Vision tasks," in *24th International Conference on Field Programmable Logic and Applications (FPL)*, Sept 2014, pp. 1–8.

[3] P. Greisen, S. Heinzle, M. Gross, and A. Burg, "An FPGA-based processing pipeline for high-definition stereo video," *EURASIP Journal on Image and Video Processing*, vol. 2011, no. 1, p. 18, 2011. [Online]. Available: http://jivp.eurasipjournals.com/content/2011/1/18

[4] V. Kasik and T. Peterek, "Video Processing Toolbox for FPGA Powered Hardware," in *International Conference on Software and Computer Applications (IPCSIT)*, 2011, pp. 242–246.

[5] Xilinx Inc., "Video and Image Processing Pack," 2014. [Online]. Available: http://www.xilinx.com/products/intellectual-property/EF-DI-VID-IMG-IP-PACK.htm

[6] Altera Corp., "Video and Image Processing Suite MegaCore Functions," 2014. [Online]. Available: http://www.altera.com/products/ip/dsp/image_video_processing/m-alt-vipsuite.html

[7] M. Schmidt, M. Reichenbach, and D. Fey, "A Generic VHDL Template for 2D Stencil Code Applications on FPGAs," in *15th IEEE International Symposium on Object/Component/Service-Oriented Real-Time Distributed Computing Workshops (ISORCW)*, April 2012, pp. 180–187.

[8] M. Pohl, M. Schaeferling, G. Kiefer, P. Petrow, E. Woitzel, and F. Papenfuß, "Leveraging polynomial approximation for non-linear image transformations in real time," *Computers & Electrical Engineering*, vol. 40, no. 4, pp. 1146 – 1157, 2014. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0045790613003273

[9] H. Bay, A. Ess, T. Tuytelaars, and L. V. Gool, "SURF: Speeded Up Robust Features," *Computer Vision and Image Understanding (CVIU)*, vol. 110, no. 3, pp. 346–359, 2008.

[10] J. Canny, "A computational approach to edge detection," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. PAMI-8, no. 6, pp. 679–698, 1986.

[11] M. Bihler and G. Kiefer, "Implementation of an Edge Detector Using a Framework for Image Processing Tasks on FPGAs," in *Applied Research Conference (ARC)*, July 2014, pp. 505–511.

[12] Paul V. C. Hough, "Method and means for recognizing complex patterns," Patent US000 003 069 654A, 1962.

[13] J. Illingworth and J. Kittler, "A survey of the Hough transform," *Computer Vision, Graphics, and Image Processing*, vol. 44, no. 1, pp. 87–116, 1988. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0734189X88800331

[14] R. O. Duda and P. E. Hart, "Use of the Hough transformation to detect lines and curves in pictures," *Communications of the ACM*, vol. 15, no. 1, pp. 11–15, 1972. [Online]. Available: http://dl.acm.org/citation.cfm?id=361242