MULTI-FOVEA ARCHITECTURE AND ALGORITHMS BASED ON CELLULAR MANY-CORE PROCESSOR ARRAYS

Ph.D. dissertation

Balázs Gergely Soós

Scientific adviser:
Csaba Rekeczky, Ph.D.

fides et ratio

Péter Pázmány Catholic University
Faculty of Information Technology
Multidisciplinary Technical Sciences Doctoral School

Budapest 2010

It would appear that we have reached the limits of what it is possible to achieve with computer technology, although one should be careful with such statements, as they tend to sound pretty silly in 5 years.

-- John Von Neumann (1949)

Acknowledgment

First of all I would like to thank my supervisor Csaba Rekeczky for introducing me to scientific work and for his constant support and belief in my research activity. Working with him gave me the opportunity to join several international projects and also to work at Berkeley, USA.

I thank Tamás Roska, head of the Cellular Sensory Wave Computing Laboratory and head of the doctoral school, for the opportunity to be a Ph.D. student and for directing my work with great empathy. Thanks to the Faculty of Information Technology (ITK) at Pázmány Péter Catholic University, and the Computer and Automation Research Institute (SZTAKI) of the Hungarian Academy of Sciences for providing the intellectual and material background for my research.

Special thanks to my host professors at the University Notre Dame (Indiana, USA) and at Johann Wolfgang Goethe Universität (Frankfurt, Germany): Wolfgang Porod and Ronald Tetzlaff.

I would like to express particular thanks to my colleagues at SZTAKI who influenced this work with their attitude to scientific research and with the atmosphere of a cutting edge research institute: Ákos Zarándy, Péter Földesy, László Kék, László Orzó, István Szatmári, András Radványi, and Péter Szolgay.

The cooperation of my closest colleagues György Cserey, Ádám Rák, József Veres, Zsolt Szálka, Dániel Hillier, Gunter Geis and Alpár Sándor is greatly acknowledged. The work with them greatly contributed to my scientific results.

By working with Tamás Roska and Csaba Rekeczky, I was given the possibility to take part in the education at ITK. Thanks for the help related to my classes to Zsuzsa Vágó and Ágnes Bércesné Novák. I appreciate the work of Vilmos Szabó and András Gelencsér former M.Sc. students, who helped me with many ideas and arguments.

The colleagues and students of the Robotics Laboratory at ITK provided constant technical help during this research. Thanks to György Cserey for the overwhelming support to bring our ideas into reality.

I am thankful to my colleagues at SearchLab Ltd., particularly Zoltán Hornák, and Balázs Kiss. Their comments helped me to augment the quality of the dissertation.

I am especially graceful to my class-mates and other graduate students for their professional and non professional help here and abroad: Éva Bankó, Csaba Benedek, Barnabás

Hegyi, Ferenc Lombai, Tamás Harczos, Zsolt Szálka, Gergely Gyimesi, Béla Weiss, Tamás Zeffer, Mária Ercsey-Ravasz, Giovanni Pazienza, Norbert Bérczi, Gergely Feldhoffer, Ákos Tar, Gábor Vásárhelyi, Kristóf Iván, Kristóf Karacs, András Kiss, Frank Gollas and Martin Eichler.

I am grateful to Ildikó Csurgay and Árpád Csurgay who supported me at Notre Dame and to the other Hungarian students: Gergő Szakmány, Edit Varga and Kristóf Tahy.

I appreciate the help of the colleagues at Analogic Ltd. and Eutecus Inc. who guided me in hardware related topics, particularly István Horváth, Tibor Gyimesi and Gábor Erdősi.

I acknowledge the kind help of Anna Csókási, Livia Adorján, Judit Tihanyi, Katinka Vida and the rest of administrative and financial personnel.

The financial support of Hungarian and USA R&D founds is acknowledged: Hungarian Scientific Research Fund (OTKA) NI61101; National R&D Program GVOP 3.1.1-2004-05-0388/3.0; National R&D Program NKFP ALFA-2/046/04 and TELESENSE-035/02/2001; Multidisciplinary University Research Initiative (MURI) of the Office of Naval Research (ONR, USA) and the VISCUBE 07-09 project, Eutecus Inc. (ONR).

Thanks to all my colleagues at ITK and at SZTAKI.

The loving support of all my family and my girlfriend Zsófi helped me through the hardest moments of this period.

Multi-fovea Architecture and Algorithms Based on Cellular Many-Core Processor Arrays

by Balázs Gergely Soós

Abstract

Intel produced the 8080 processor chip in 1974, which consisted of approximately 4500 transistors. The revolutionary development of silicon-based manufacturing technology led to the possibility to produce integrated circuits (IC) using several billion transistors in a single chip [8]. Such a large amount of basic elements opens the way to commercialize chips with thousands of connected – but independently controlled – parallel computation components, called processor cores. From the aspect of computer science it raises a new challenge to formulate the theory of many-core computing: the structure of the communication network to support maximal data transfer rate between computing nodes to keep them busy with new inputs to process and to transfer outputs to their destination. New algorithms are needed to maximally exploit these novel hardware capabilities, and new methods are required to split problems into parts that can be evaluated in parallel. The optimization of data-communication time enforces the precedence of locality and the utilization of cellular structures [9].

The obvious efficacy of biological vision systems may motivate engineers to mimic two major characteristics (parallelism and foveal processing) during the design of artificial solutions that are embedded into real world-environments. When many salient regions arise after preprocessing, all of these should be better explored. This is the multi-fovea model. Successful works (for example [10]) highlighted the importance and efficiency of this design concept that called for a unified algorithm description.

In my work I primarily focused on data-parallel interpretation and processing of topological problems, in particular image processing tasks, by using cellular processing structures within the Multi-Fovea Architecture and Algorithmic Framework. The research question from the theoretical point of view was to create a unified software model that supports the selection of parallel architecture and also covers hardware specific details for a given device. The particular practical application inspiring the theoretical work was

to analyze algorithms for visual navigation systems applicable in mini / micro unmanned air vehicles (UAVs).

The elaborated virtual platform consists of separate processor arrays specialized for parallel execution of preprocessing and foveal computation. The proposed heterogeneous structure can fit the special characteristic of various operators processing the highly parallel data input. I gave a design guideline for Multi-Fovea Architecture and presented it by the comparison of 2D registration methods for ground object motion detection from mini unmanned aerial vehicles. After giving an analytic comparison of registration methods, I proposed a novel method exploring the proposed architecture by running a larger percent of the task in parallel and in cellular structures.

TABLE OF CONTENTS

Chapter One

INTRODUCTION

1.1. Preface

The rapid development in material science and semiconductor technology and processor enhancements lead to the greatly increased performance, falling prices and widespread applications. Digital cameras with 10 Mega Pixels have become commodity. High-end video cameras are capable of capturing sequences of up to many thousand frames per second and sensors operating in the infrared domain can form images in darkness. Medical diagnostics deeply rely on imaging, for example x-ray or ultrasound, whereas physics boosted by computer science offers further methods like magnetic resonance imaging (MRI) or positron emission tomography (PET). The acquired data set has implicit data parallelism in 2D or 3D topology.

Most common everyday computer programs like word processors or internet applications do not take advantage of multi-core processing since their tasks cannot be easily split into parallel segments. Processor vendors have so far been competing in increasing the operation frequency of their serial-execution systems. Lately, however, the spread of digital multimedia (pictures, videos and music) introduced new kind of data and new kind of processing task to personal computers. The remarkable degree of structure within this type of data opened up the necessity to split up the data for parallel processing using multiple arithmetic cores and later produce the output after joining the partial results. This commercial motivation prompted large investments in multi- and many-core technology.

The need for scientific modeling of parallel processing is obvious concerning the huge variable space dimension and data size. Numerous important phenomena can be modeled via joint dynamic systems [11], which in turn raises the need to extend the classical algorithm definition originally formulated for integer numbers. The underlying cellular structure of broad classes of problems calls for the design of topologically connected many-core processor arrays and the algorithmic approaches could rely on the results of the cellular wave computing theory [12][13].

1.2. Research Goals and Results

The aim of my research was to create a virtual hardware model (Multi-Fovea Architecture) to support comparison of image processing algorithms that apply the multi-fovea model. The elaborated platform consists of separate processor arrays specialized for parallel execution of preprocessing and foveal computation. They are synchronized by a conventional serial processor via a proposed intelligent memory management unit. This heterogeneous structure can fit the special characteristic of various operators processing the highly parallel data input, like video flow.

I gave a design guideline for Multi-Fovea Architecture and presented it by the comparison of 2D registration methods for ground object motion detection from mini unmanned aerial vehicles (UAV).

After giving an analytic comparison of registration methods, I proposed a novel method exploring the proposed architecture by running a larger percent of the task in parallel and in cellular structures.

1.3. Methods Used in the Experiments

My research was motivated by the most recent neurobiological results in retina-modeling and other neuromorphic engineering solutions [14][15] along with psychophysical experiments.

Backgrounds for the proposed architecture are collected from projective geometry, image processing, topologic cellular operators and algorithms [16], parallel computing and graph theory.

The preference of local communication (cellular structures) both at the virtual and physical level is a fundamental part of the model [13].

The notation used for describing the algorithms of the thesis within the mathematical framework is a directed acyclic graph (DAG), which is widely used for scheduling problems. This description may be considered as a generalization of the Universal Machine on Flows (UMF) diagram [17] that was specially designed for cellular neural networks (CNN) algorithms executed on SIMD/MIMD type many-core processors built in highly regular topology.

After setting up the theoretical framework for complexity evaluation, I modeled relevant state of the art algorithms. To test quantitative quality of the algorithms I made a software framework in PC environment. Furthermore, I made measurements using many-core hardware configurations such as the ACE16K chip [18], the EyeRis chip [19] and the Nvidia GeForce 8800 platform [20], using their specific development tools and programming languages.

Within the framework of the ALFA project [9] I participated in the field experimental series, using a small UAV that was flying above the airport of Gödöllő (small town near Budapest), thus I could also use real video sequences in the algorithm development and testing beside the ones rendered via 3D simulation.

The comparative analysis was performed in the Matlab / Simulink programming environments [9][21]. In addition, some modules were implemented in the C/C++ language, and some reference implementations were also used from third party sources.

1.4. Framework of the Dissertation

After this introductory Chapter the reader may find the definitions of concepts used in the dissertation in Chapter two.

In Chapter three the Multi-Fovea Architecture is described in detail together with the first thesis.

State-of-the-art global registration-based algorithms are presented in Chapter four to underline the capabilities of the architecture. After the description of the algorithms, evaluation results for four video sequences are presented. The results of this comparative analysis give the basis of the second thesis group. Based on the analysis, a new algorithm is proposed called the Elastic Grid Multi-Fovea Detector or Elastic Grid Algorithm (ELG) in short. For easier comparison, the algorithm is presented together with the base algorithms. The connected thesis is formulated in Chapter five.

Summarization of the main findings concludes the dissertation.

Chapter Two

COMPUTATION MODEL

2.1. Basic Definitions

The definitions introduced below are from the fields of *computer science* and *computer architectures*. They are widely used and well known; still it is important to clarify the concepts used in the dissertation. I formulated the definitions leaning on the textbook of Vipin Kumar [22].

2.1.1. Modeling Logical Data and Computational Elements

Programming is the way of instructing *machines* to do something meaningful. The *computers* are machines dealing with numerical data. *Embedded computers* are special purpose devices directly interacting with the physical environment, acquiring data through *sensors* and give output response via *actuators*.

The world – or some phenomenon – is modeled and represented using some numerical data. Pieces of data are called *entities*. Obviously numerous meaningful groupings may exist. In most cases, the key for successful modeling is to follow natural structures of objects in a real environment. The programs and the data together form the *software* aspects of computing.

Some basic definitions can be found in APPENDIX B.

In computer science the formal definition and theorems for algorithm complexity are based on the Turing machine model with its limited operator set and unlimited memory. In practice an upper limit should be specified for the memory in use and far more complex operators are needed. In the dissertation a slightly modified definition for algorithm will be used, than the original Turing model.

Definition 1 **Operator, Operand**

An operator is a mapping defined on some single values or on a tuple of some entities of extended type resulting in a single entity. For given input dimensions the output dimension should be the same for any input values. *Unary operator*s take a single entity whereas *binary operator*s take a pair of entities as input.

A unary operator applied to a list (compatible with the base type of the list) results in a list of all elements processed by the operator.

An operand is one input tuple for the given operator.

Number of operands (i.e. the number of input tuples) may be different depending of the operand type. For example *addition* and *multiplication* can work on arbitrary number of operands, while others, like comparison assumes two operands.

Definition 2 **Algorithm, Program**

An algorithm is a recursive series of operators applied on *inputs* resulting in *outputs*. The program is the encoded version of the algorithm implementing operators via a hierarchy of instructions available on the actual machine.

- Time-independent inputs and outputs are values of extended type.
- In case of any time-dependent input, output is also time-dependent. All algorithms are *causal* – depends only on current and past input elements – in a limited time span: *final memory model*.

Definition 3 **Iteration of an algorithm**

The calculation of all output values for a given time index – using the older results in the time span stored in memory – is an iteration of the algorithm.

2.1.2. Modeling Physical Components

After introduction of the software terms and definitions we can turn to physical components realizing the computation, generally referred to as the *hardware*. In this subsection an abstract definition is given for a general architecture.

Nowadays most of the computers are electrical devices, using a binary number representation, implemented in silicon *integrated circuit chips (IC)*, mounted and connected via *printed circuit boards* (PCB). The ICs are complex reusable functional blocks. Main component of computers is the Central Processing Unit (CPU). A CPU can be implemented in a separate IC, or built onto one chip with other components. CPUs are built with Very Large Scale Integration (VLSI) technology.

General purpose CPU instructions are designed to deal with a fixed number of bits at a time. This number is the word-size – one of the most characteristic attribute of the computer. Nowadays the typical value is 32 bit, allowing operations with integers in the range of $0..2^{32}-1$ (or $-2^{16}..2^{16}-1$) while CPUs with 64 bit word-size are emerging.

Data has some internal representation inside the computer.

Definition 4 **Variable**

A basic variable is a scalar number represented by internal coding. Attributes of other types can be represented by a group of variables.

Definition 5 **Instruction, Code block**

Machine level instructions are (digitally) coded commands for elementary operator executions and data transfers. There are also instructions for altering the sequence of execution. Instructions may be grouped into code blocks.

From a top-down aspect the computer has two main building-blocks: *processor* (CPU) and *memory*, the former for computing, the latter for storing data. In most cases they are separate chips on the PCB with non-negligible transfer latency and limited transfer speed.

Definition 6 **Memory**

Memory is a temporal storage holding a large number of variables and instructions.

The physical realization of the memory unit can be different; some structures are designed for fast access, others for capacity. Some circuits are placed into the processor itself, others are separate chips. In most cases, not bits but instead larger groups are accessed at once. The most common addressing mechanism is indexed using bytes (8 bits) or words (matching the internal structure of integer representation) as units. Some clustering may exist on elementary units for effective mass data transfer.

Definition 7 **Long term memory**

Long term memory is the memory needed to store results in the time span to minimize calculation for the next iteration.

Definition 8 **Short term memory**

Memory needed to store intermediate results during a given iteration is called short term memory.

Registers are distinguished memory circuits practically without access latency inside the processor.

Definition 9 **Register**

A register is a temporal storage holding variables to be used as operands in a calculation (in the near future).

The number of registers is in the magnitude of one or ten in typical cases, thus they are dedicated for calculation rather than storage. In most architecture, operations can only be executed by referencing registers, some allow operands to be taken from memory directly. Some registers can be distinguished for some instructions as being the preferred or obligatory operands.

Some operators are realized directly in hardware structures, some others are implemented as mini programs.

CPUs consist of some main building blocks: Processing Element and Instruction Unit.

Definition 10 **Arithmetic and Logical Unit**

The Arithmetic and Logical Unit (ALU) is a controllable circuit executing various elementary calculations.

Definition 11 **Processing Element, Local Memory**

A processing element is the smallest compact hardware unit for calculation. It consists of some registers and the Arithmetic and Logical Unit, and some sub-unit for internal control and to access memory. Optional low-latency local memory can be placed inside the processing element that is also called register-file.

Definition 12 **Instruction Unit, program counter, micro-stages**

An instruction unit first *fetches* the next instruction from a special program memory region storing a code block. As a second step it *decodes* the instruction, and by using internal control mechanisms it executes the command on all connected processing elements, probably in multiple *execution steps*. The steps are called micro-stages. Different architectures may use different micro-stages, even a non-uniform sequence of them for different instructions. Typical further micro-stages are *operand fetch* and result *write-back*. The next instruction is selected by the program counter, which is a distinguished register. In most cases the counter is incremented as the last step of the execution of the instruction, or it can be updated to implement branches in the algorithm, conditional jumps or function calls.

Definition 13 **Execution Unit (processor core), Local memory, Global memory**

The execution unit (processor core) is the hardware unit for program execution with some processing elements. It is standalone by having an instruction unit. A complex program can be deployed to several collaborating execution units. Local memory is a low-latency memory situated inside the unit. Larger external memory – global memory and input/output peripherals are also accessible with higher latency. Global memory is probably accessible for more than one execution units whereas local memory is dedicated to the enclosing unit.

Definition 14 **Address space, memory management unit**

Data is accessed via I/O instructions using addressing. Instruction level addresses are resolved by a memory management unit or some control logic to access the corresponding memory module. The possible address values span the address space. In most cases, continuous address intervals are used and mapped mainly to the global memory and certain ranges may be mapped to local memory as well.

Data must be transferred from one memory unit to others. Speed critical transfers between units within the same PCB are done via bus systems. The transfer may be parallel using dedicated wire connection for each bits (e.g. PCI bus), or serial using time multiplexing transfer (e.g. PCIe link). In some cases the address is sent through a dedicated line called address bus. Since the memory management unit hides the details of physical implementation key features may be summarized in three properties:

Definition 15 **Bus width, Memory bandwidth, Latency**

Bus width is the amount of data that can be accessed in one step, and bandwidth is the maximal data transfer rate (for example measured in Mega Bytes per Second). In case of multiple accesses issued by the same unit the delay between first data request and respond is the latency. Some memory systems have preference for block addressing, supported by processor independent mechanisms (e.g. Direct Memory Access, DMA).

Computation deals with neighboring data elements frequently. This property is called *locality*. Locality is also characteristic for instructions. A sequence of commands can be aligned in memory with a small number of branching instructions. The off-chip communication to load new data has massive latency – typically two or three order of magnitude higher compared to numerical calculation – thus *caching* into on-chip memory is essential to exploit locality.

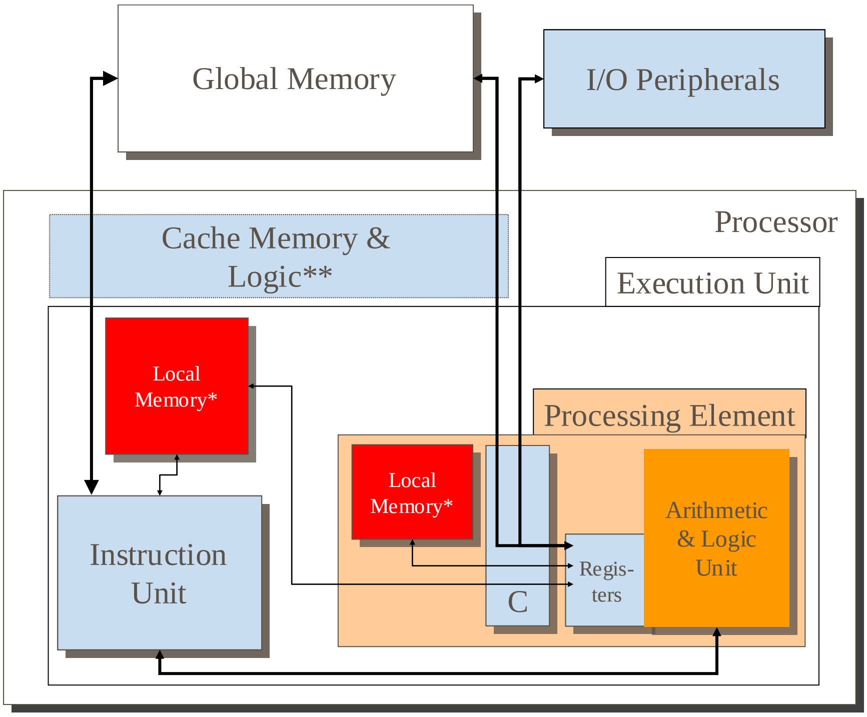


Figure 2.1. Main processing elements of a single-core computing hardware configuration. The Processor and the Global Memory are two different chips mounted to a printed circuit board. Instructions and variables are stored in the memory circuits that are accessed using addressing. On-chip Local Memory is accessed with small latency whereas Global Memory has considerably high response time. The Instruction Unit is fetching the instructions and controlling the Arithmetic and Logic Unit (ALU) to perform computations. Registers are special memory circuits for holding operands for the upcoming calculation. A Processing Element consists of an ALU, the connected registers and the control logic (denoted as C). It may also contain some Local Memory that is also called register-file. Nowadays a processor may contain multiple Execution Units (multi-core architectures). The Cache Memory is an automatic element holding a local copy of variables to decrease the effect of memory access latency. The original von Neumann architecture did not contain any Local Memory or a Cache unit.

<u>Definition 16</u> **Caching, Cache memory**

To exploit locality and multiple access of a single entity, a local copy of values can be stored to eliminate multiple data-fetches and to fetch a larger quantity of data in one packet with a single communication latency penalty. These strategies are called caching, and *pre-fetching* or *buffering*. Cache memory is a special on-chip memory. It stores indices to real addresses and variable values from that location. It cannot be accessed directly since it is handled by the memory manager hardware. In case of multiple processing elements, great effort is needed to maintain their consistency. Local memory is handled programmatically (explicitly), whereas caching is an automatic (implicit) mechanism.

Definition 17 **Processor, Array-processor**

Processors that are containing more than one processing element are called multi-cored. If the cores are organized in some topology, the unit is called array-processor. A processor in most cases is a standalone chip itself.

2.2. From Serial Execution to Parallelism

2.2.1. Serial Execution on Single Processor Architecture

Control and computational problems may be formulated into programs and decomposed into smaller units using many design concepts and paradigms. The conventional way is to separate problems into a sequence of tasks: sub-problems to be solved step-by-step to achieve the final goal. Conditional execution of some tasks and iterations are necessary to solve complex problems.

Definition 18 **Task**

A program can be decomposed into logical sections called tasks. All instructions in a task are executed serially by a selected execution unit.

For a given decomposition a flowchart diagram may be given. The description may use hierarchy.

Definition 19 **Flowchart diagram**

A flowchart diagram is a graph-based description of the process realized by the system. In our case the given computation or control algorithm defined on time-independent variables or evaluation of one time-instant for time-dependent variables. Flowcharts contain pictograms for tasks as vertices connected by directed edges representing constraints for ordering. In a flowchart description tasks share a common state space. It has special vertices for ‘START’ and ‘STOP’ as beginning and terminating points of the whole program. Special pictograms exist for conditional branching and I/O. Flowchart graphs may contain circles for representing iterations.

The expression that tasks share a common state space means that if something is altered by a given task the successor tasks a flowchart may access the updated value. The description allows *split* and *merge* in control flow. The flowchart diagram was originally designed for von Neumann architectures with a single program counter, thus a total serialization was used during implementation, although some tasks may be executed in parallel.

Definition 20 **Code block of a task**

From a given hierarchy level, the descriptions of tasks turn from abstract operators into a set of programming language instructions. After compilation the corresponding sets form a *code block* for the given task. The *entry point* is a special point of the code block for starting the execution, and the *exit point* is for termination.

Let us suppose that the solution for the given problem is somehow formulated with a corresponding data-set and an algorithm. A traditional algorithm is a single serial sequence of instructions with a single flow of execution designed to be executed by a serial processor. It can be achieved using a complete ordering of the tasks.

Complex problems may be decomposed into smaller logical groups of elementary instructions, usually by using some level of hierarchy. This is the imperative way of programming. It means commanding the system to make calculations on input values and assign results to variables. Again, the execution is purely sequential. An instruction is evaluated, and then the next is processed, although the execution may branch. Branches are possible via reference labels to instructions.

Typical control flow commands are:

- Continuation at a different instruction (unconditional jump – GOTO)
- Continuation at a different instruction and later returning (subroutine call)
- Executing a sequence of instruction only if some condition is met (conditional execution via conditional jump)
- Executing a sequence of instructions iteratively while some condition is met (while loop)
- Stopping the execution (termination)

Using unstructured GOTO statements made programs hard to read, so it was an important theoretical result that any Turing complete algorithms can be formulated with conditional execution and while loops [23]. On the other hand, extended flow control statements – like multiple choice continuation and other loop constructs – turned out to be practical.

Non-local extensions of the control flow – such as exception handling – give extra convenience to programming, although they are not covered in this Chapter since they make program parallelizing extremely hard, and can be avoided with structured coding practice.

For a given decomposition and a selected valid serialization a control flow graph may be given. This description may also use hierarchy via function calls.

Definition 21 **Control flow graph, Basic block**

The control flow graph is a representation of all possible control paths during program execution. Vertices in the graph are *basic blocks*. They are sequences of instructions (function calls and assignments) without any jumps or jump target points. Directed edges point to possible continuation points, witch may be targets of an unconditional jump, a conditional branch, or simply the next instruction of a program. The entry and exit blocks for a function are special ones encapsulating all incoming and outgoing possibilities. Control flow graphs may contain circles for representing iterations.

At a low (instruction) level, possible variables referred by machine code operations are basically scalars. To handle composed data structures, indirect references are used. Operators are defined on references to give access to all elements of an indexed structure relative to the base element. Each variable has its scope. Some of them are global, belonging to the main scope of all tasks; others may be local for a given function.

2.2.2. Concurrency

Each problem has an upper limit for economical calculation time. Nobody would wait hours for the answer to a simple train-schedule query. Concurrency is the possibility of doing some work steps in parallel without altering the result.

Independent workloads may be pipelined. This is similar to the assembly line of a factory. Let us assume that ten hours is the production time of a car. If this process can be split into ten identically long serial steps, and each of them can be assigned to a stage of the assembly line, with this method the overall throughput of the factory is a car in every hour. This simple pipelining is only possible if work pieces are independent of each other.

In case of program execution, independence of two instructions holds true if their attributes are not altered during the execution of the other.

Definition 22 **Read-set, Write-set**

The read-set of an instruction is the set of operand variables; the write-set is a set of variables that the new values are assigned to.

Definition 23 **Data dependency**

Instruction i_1 depends on instruction i_2 if the read-set of i_1 intersects with the write-set of i_2.

Since branching is common, some prediction of the more probable control flow is necessary to fill the pipeline. If the prediction does not turn out to be true, the remaining instructions must be discarded, and the instructions from the alternative flow must be processed. Referring to the assembly line example, a branch prediction miss is analogous to a sudden and high priority request to switch production to sedan cars from coupés. The already started coupés must be abandoned and the production of the sedan is started. It means not only the loss off half-ready cars but also latency before the first coupé is ready.

As it was described above, locality is an important attribute for memory access patterns. In general, neighboring instructions reuse data produced in the near past. Moreover, variables that are logically connected and probably referenced together can be organized in memory to be close. Caching, as it was introduced before (Definition 16) is a good automatic mechanism to reduce the memory access penalty.

Besides the memory, computer systems also have some limited resources, for example keyboard, printer, files that may be accessed only by a single processing element at a given time, in a well defined order. Their scheduling, the handling of resource and I/O conflict can be handled similarly to memory management (in some systems I/O is mapped to memory space directly), although the resource access times are normally in two or three orders of magnitude higher than memory transactions.

Definition 24 **Instruction level parallelism**

Considering control flow at finest detail, instruction level parallelism is exploited if more than one instruction of an algorithm is executed concurrently. In the von Neumann execution model with a single instruction pointer, neighboring instructions may overlap at a micro-stage level.

Definition 25 **Out-of-order execution**

If some operand for the next calculation is not in the cache, the time for fetching may be utilized by execution of some other instruction that is not dependent. This technique is called out-of-order execution.

These methods (Definition 24, Definition 25) need intensive hardware support: prefetching of instructions, branch prediction and caching. All of these are implemented in CPUs of modern PCs to increase throughput. Since hardware capabilities are limited, outstanding compilers are needed to order instructions in a way to ensure that this implicit parallelism becomes creditable.

2.2.3. Explicit Parallelism

A higher dimension of parallelism is to use multiple lines of execution. Explicit parallelism, defined by the programmer makes it clear what to run in parallel, and how to organize communication. Handling the extreme workload of numerous complex problems is not feasible using even the best available serial processor to be beneficial at all. For these cases it is important to identify tasks of the problem that can be executed in parallel without altering the final result, and distribute them to multiple processing units.

The flowchart of a program may contain visibly parallel tasks. The existence of global variables makes it necessary to define a complete ordering that ensures the correct behavior of the program. Some tasks are truly independent of each other; since they do not have any data dependencies, their execution order is interchangeable.

Definition 26 **Parallel task, task parallelism**

Parallel tasks are special tasks that can be run in any serialized order to get the same result. The correctness offers the possibility to execute them in parallel.

Definition 27 **Thread, context switching**

A thread is the execution context of a code block on a processing element consisting of the data, a code block itself, and the actual position of the instruction pointer.

With hardware support, all states of a thread can be encoded, saved to memory, and restored later. This mechanism is called *context switching*. It is an important feature for a large number of threads executed in parallel on a smaller number of processing elements.

Definition 28 **Data Parallelism**

Multiple threads may be useful to complete the workload of a single task, in case of processing elements of a non-scalar structured data set. This form of parallelism is data parallelism.

For special parallel tasks with the same code block, program counters can be run simultaneously and they can be mapped to processing elements sharing a common instruction unit. This is the Single Instruction Multiple Data stream hardware configuration.

Data dependency is not an exclusive fact against running multiple threads. Using synchronization methods, mutual exclusion for write set elements and necessary ordering, valid operation can be ensured.

Definition 29 **Level of parallelism, granularity**

The average number of instructions between synchronization steps is the granularity of the parallelization. *Fine grain* solutions utilize a large number of threads and frequent synchronization, whereas *course grain* solutions process more work per thread using a smaller number of execution contexts.

The common state space concept of flowchart representation can be supported by architectures following the *shared memory model*. Most important software libraries in use are *OpenMP* and *Posix thread*, offering different thread creation and synchronization methods.

Definition 30 **Shared Memory Model**

In case of a shared memory model of parallel computing, the address space of each processing unit contains an overlapping global memory region. This can be accessed by all processing units, and is protected by synchronization mechanisms.

For a large number of processing units, synchronization hardware to protect a large address space effectively can become complex. Furthermore, disjoining memory spaces and using local memories offers higher bandwidth if the granularity is large enough that the threads can work efficiently without the need of frequent data exchange.

The distributed memory model eliminates the global region from address space completely.

Definition 31 **Distributed Memory Model**

An alternative way of connecting parallel processing unit is message passing. The address space of all processing units is separated. They may exchange data and synchronize via messages. This is the Distributed Memory Model of parallel computing.

The dataflow diagram offers an explicit notation for parallelism and an execution model. Optimizing compilers also generate low-level dataflow graphs beside the control flow graphs to find independent instructions. This graph contains a node for every assignment. The data-driven execution model originates back to the ’70s [24][25]. In the next three definitions concepts collected by Johnson et al. will be used [26]. In contrast to control flow diagrams, this notation focuses on data paths rather than control paths.

Definition 32 **Dataflow diagram**

The dataflow diagram is a graph-based description of the algorithm. It describes the evaluation of problems defined on time independent variables or evaluation of one time-instant for time dependent variables. In this model, tasks represent connected black-box systems, each having a separate variable space. Data exchange is explicit and implemented by messages. In dataflow diagrams, tasks are represented by vertices and communication links are modeled as directed edges. For every parameter passed, a distinct edge is added. Diagrams may have special vertices for environmental input – *Sources* and output – *Sinks*. Special pictograms exist for conditional *enabling*, *merging* and *splitting* of *data paths*. A dataflow diagram does not contain explicit ordering for task execution. Every task may be evaluated when all input messages have arrived then after calculation output messages have been sent. This is the data driven execution model. Dataflow graphs may contain circles for representing iterations.

Serial ordering is not needed in the case of dataflow programming, since variable spaces of tasks are independent. All data exchange is explicit. Variables may have direct representation as nodes; although they are never overwritten by direct loops, outputs of calculations are fed to new versions. The execution is fully parallel in an asynchronous or synchronous way. Inner loops may have a higher update frequency than the global iteration cycle correlating to the update of time-dependent inputs.

The dataflow model is excellent for describing explicit parallelism. Software environments – for example LabView from National Instruments, Simulink from Matlab and System Generator from Xilinx – are designed for hardware in the loop measurements. They all follow dataflow programming methods.

The message passing programming concept can be used in several programming languages through the quasi standard Message Passing API.
In the dissertation the synchronous evaluation is used, which is more compatible to the time series concept introduced before. A further restriction is used: only well-behaved dataflow models are considered.

Definition 33 **Stream model**

In the case of well-behaved dataflow models, exactly one set of output messages is generated for a set of input messages, and all messages have predefined dimensions. Elements of a non-scalar structured data set are processed one-by-one. Merging of data paths must be defined uniquely, by using well formed network of conditional enabling or priority levels. This model is called stream model, as input data is streaming through the transforming steps of the system.

The stream model was popularized through the Stream and Stretch processors around the Millennium by a Stanford University group and later a spinoff company [27].
Using the stream model, the graph of an algorithm can be evaluated via breadth-first traversal started from the input sources (data-driven model) triggering the evaluation of all successors, or from the output sinks using depth-first traversal (demand-driven model). In case of time-dependent input signals, the push model is more natural.
The streaming model allows sub-results belonging to inputs at different time instances in the graph at different processing stages, and to split complex values into a series of elements and process one-by-one.

Definition 34 **Pipelined dataflow execution**

For dataflow graphs following the stream model, messages belonging to a given time instant may wave through the system and data from different input sets may be present at different stages of the evaluation. This is the pipelined evaluation model.

For an algorithm that stops in a Turing sense, the number of all iterations should be finite in function of the input size, moreover for a limited input size this iteration limit can be majorated by a constant. It means that iterations may be unrolled to sequences that are probably shortened if some condition is true (break from the loop).

Dataflow diagrams may be created in hierarchy. Each task can be described with dataflow diagrams containing operators at the lower level of abstraction. At the most detailed level, only operators matching the instruction set of the current (virtual) machine are allowed.

Using a pure dataflow model offers high utilization of parallelism and eliminates the need for global shared memory. In contrary, it needs fine grain communication compared to calculation done in each stage.

States of a black-box represented by a task may change in the function of the input values. Some tasks may be assigned to the same processor core or cores accessing the same memory; the data exchange is still modeled as message passing, although in this case communication overhead is small.

A hybrid model describing fine details with a sequence of instructions sharing a common variable space offers coarse grain communication. Moreover, state-variables may be stored in local memory. These facts underline the definition of task (Definition 18)

For communication-effective implementation of an algorithm, the goal is to move all steps involved in a loop into a common task to formulate loop-less top level description. If it is feasible, the graph is a *directed acyclic graph* (DAG). It may contain delayed versions of the previous calculations (stored in long term memory).

In the frame of this dissertation my main goal was to deal with video processing algorithms that can be modeled with a DAG at the top level of hierarchy.

Chapter Three

MULTI-FOVEA ARCHITECTURE

3.1. Image Flow Processing

The main targets of the Multi-Fovea Architecture are the image processing algorithms. Basic concepts are summarized shortly in this section.

Images are results from 2D sensors sampling (light) intensity with finite spatial resolution. Color images are handled as multiple 2D *channels*. For practical reasons Images will be defined using a more general structure than a matrix:

<u>Definition 35</u> Image, pixel

Image is a type qualifier for entities described by a 2D to 1D function:

$$I:[x1..x2]\times[y1..y2]\rightarrow P, P\subset\mathbb{R},$$

Values are defined for integer locations by the image matrix. In the function ‘x’, ‘y’ components of a left-handed Cartesian coordinate system is used, following the image processing terminology. Elements of the image matrix are called pixels. They are indexed in row, column order. For sub-pixel locations image values are defined via an interpolation function:

$$I\ \ u,v\ \ = \mathbf{I}_{v,u}; u\in\ \ 1,2,..n\ \ ; v\in\ \ 1,2,..m\ \ ; n,m\in\mathbb{Z}$$

$$I\ \ \mathbf{x}\ \ = Interpolate\ \ \mathbf{I},\mathbf{x}$$

The size of an image is equal to the size of its image matrix.

$$\left\| I\ \ \mathbf{x}\ \ \right\| = \left\|\mathbf{I}\right\|$$

If the range of the function is binary, the image is called a *mask*, highlighting that some property is true or false at a given location. In other cases the image is called a *map* describing some spatial attribute.

Processing of single images is the study of *image processing*. In case of snapshots taken continuously it is possible to extract temporal information beside spatial information from the environment. Video processing algorithms are dealing with flows generated by imagers.

Definition 36 **Image-to-image operators**

Operators taking images as attributes and resulting in images are image-to-image operators. The output image must be specified by defining all elements of its image matrix. *Spatial transformations* alter the domain of the image, while r*ange transformations* alter only the values of the functions. Spatial transformations may refer to (u,v) values that are outside the domain of the input image. In this case a default value is used.

Definition 37 **2D Operator**

A 2D operator is an image-to-image operator that is defined on the elements of the input image matrix.

Definition 38 **Image flow, Frame, Frame rate**

Image flow or video is a list of images of the same size (I_k) taken at regular intervals. The index represents time using the constant time step.

Individual images are also called frames. Frame rate is the number of image matrices generated by a source in a second.

Definition 39 **Image flow/video processing algorithm**

Video processing is an algorithm using an image flow as its main input. The discretized time unit is aligned to the frame time (reciprocal of the frame rate) of the device supporting the input flow.

Neighboring pixels of the image matrix are closely connected in a logical sense. In most cases they are projections from the same real-world object. The topologic connectivity of the pixels must be respected during processing.

The high level description of the algorithm consists of operators defined at the image level, not at the pixel level. This abstraction is useful since usually preprocessing (noise reduction, spatial filtering for feature extraction) is highly uniform at the pixel level. A wide range of operators either combine two images point by point – for example, image subtraction and addition – or when defined for a single input image, process a small neighborhood of pixels (typically $3\times 3, 5\times 5$) – for example, convolution.

3.2. Introduction of the Multi-Fovea Approach

Processing the entire data captured by an image sensor at full resolution is computationally expensive, and in most cases unnecessary. Even in the human visual system, data convergence can be observed: the amount of data processed and transferred from photoreceptors in the retina to cortical structures via the optic nerve significantly decreases, whereas the abstraction of the information extracted increases. Light intensity is captured by roughly 130 million sensory cells and is transferred by only 1 million ganglion cells.

Similarly to biological vision mechanisms, in an artificial visual system a decision can be made at an early stage of the *image flow processing algorithm* to locate interesting regions. Thus, the computational effort can be focused on critical areas, and an efficient processing scheme can be formulated with moderate data transfer between modules.

I have designed a virtual heterogeneous many-core architecture for image processing algorithms that are convergent, starting from direct sensory input and can be described by acyclic dataflow graphs. Convergence is referring to the extraction of compact information from inputs represented with topological maps with smaller resolution, image parts, or scalar values. This property calls for heterogeneous processor structures. In applications, where the high frame-rate is important (e.g. 10.000 fps), sensor pixels can be built in the processing topology to eliminate the need of wide and/or ultrafast cross chip communication circuits. If the program can be transformed into a representation containing iterations and recursions only at operator level an acyclic dataflow graph can be created. In this case, program execution can be mapped to many cooperative processors requiring clean and pre-calculated synchronizations.

Hardware realization can be designed to solve data parallel tasks in each region, or existing vision processors can be utilized. The concept is to define a virtual architecture that can work as a common abstraction level. It offers high level algorithm design and analysis opportunity hiding the underlying hardware.

Selected regions are called *foveal windows* analog to the fovea of the mammalian retina. They are rectangular regions covering a part of the original input frame depending on the *scale factor*.

Fovea-based video processing algorithms use image processing operators on original or scaled images or at extracted interesting regions. The basics of this model were also described in [28].

One of the key topics of the dissertation is to summarize design considerations utilizing this concept. Some restrictions are used to achieve maximal parallel computation. Algorithmic capabilities are presented in the field of independent motion detection. Four different classes of state-of-the-art algorithms were observed. Common high-level elements of the algorithms under consideration can be separated into three major categories: at first interesting regions are selected by using mainly *topological 2D operators* (Class 1), then the regions are processed using local adaptation in each region (Class 2) and some numerical descriptors are extracted. Finally, depending on the topology of the windows and the extracted values, a global decision is made (Class 3) for aligning consecutive frames. Their flow-chart can be found in APPENDIX C.

Processing steps in the upper three classes are highly different in terms of the required operator set. In the next section a *unified virtual architecture* is proposed for optimal computation with three different types of processors: the *Frontend Processor Array (FPA)*, the *Foveal Processor Array (FVA)* and the *Backend Processor (BP)*. They communicate via an intelligent *Memory Manager Unit*. The virtual architecture can be realized on various hardware components, offering a common platform for the algorithms.

Based on the analysis, a new algorithm is proposed called the Elastic Grid Multi-Fovea Detector or Elastic Grid Algorithm (ELG) in short. It is characterized by moderate hardware complexity while maintaining competitive detection quality.

Algorithm design using the Multi-Fovea approach can be formulated as follows. To describe a general *video processing algorithm*, a *dataflow diagram* is used (*modeling*). Then, all *processing blocks* are mapped to a *virtual processor* depending on the required operator set and transfer times (*partitioning*). For a given underlying hardware platform the individual blocks are implemented, code segments and parameters could also be optimized (*implementation*).

3.3. Virtual Hardware Model

Computationally extensive applications need special balance between hardware and software components to achieve performance in an economical way. Low level programming for a given hardware can result in very efficient code although the current hardware capabilities limit the thinking of the programmer, and the code is not reusable for new software versions and especially not for next generation of the hardware. Moreover, programs may not be portable at all. Software development also requires deep knowledge of the current hardware. On the other hand, high level conceptual programming is comfortable for the programmers and offers good effort tuning possibilities; however, the effort to write the necessary compilers is out of scope of smaller hardware companies.

Defining common software programming interfaces (Application Programming Interfaces - API) and writing hardware specific drivers is a working solution. For high definition computing it is necessary to specify communication and synchronization schemes, memory hierarchies and efficient high-level operators beside the specification of the basic instruction set.

Virtualization may offer a solution and split the needed effort between the players in business. A programming method for the virtual architecture to be used by developers may be defined. The common abstraction layer may be defined over several different hardware platforms. Vendors can still find their competitive edge and customers by implementing some aspects faster while offering all services at an adequate level. In that case, the effort to write the compiler to the virtual machine can be shared and the machine-specific implementation effort is probably acceptable by manufacturers. The biggest advantage is that vendors can persuade software companies to use emerging technologies, since the gap between the conceptual programming and the hardware level is hidden by the compiler and the running environment. Moreover, virtual architectures may offer larger-scale services than are actually implemented in some emulated way, extending technical limitations.

The most prominent virtual architecture nowadays is probably the JAVA virtual machine and .NET platform, used by millions of programmers on several different platforms, offering enormous scalability spanning from heavy-duty server applications to games in handheld devices.

In our case – since the multi-fovea image processing needs different operators in preprocessing, foveal processing and classification stages – multiple virtual units are defined. While analyzing the different algorithms, the minimal set of operations may be collected that must be supported by the units.

3.4. Multi-Fovea Framework

As it was declared in (Section 2.2.2), the stream model and dataflow description is used for describing the image flow processing algorithms. This high level description is close to conceptual level, still may be implemented in a straightforward way. In the frame of the Multi Fovea Approach, two important extensions to the basic stream model are required. In the basic stream model, data flows of composite types are allowed with predefined dimensions. This means continuous traversing of all elements and homogenous operator application is valid along with merging of all results in the same predefined serial order.

In case of video processing (especially for preprocessing) high level operators are defined as image-image transformations. Elements for stream processing may be a rectangular region of pixels. Since neighboring pixels are needed for topological operators, neighboring packets must be accessible or temporarily stored in memory. This is the first extension that allows referring neighboring packets in 2D topology. Furthermore, extracting the interesting portion of the stream is also added to minimize data transfer and calculation. A hybrid architecture is needed to support these features. This is described in this section in detail.

In [4] I proposed the virtual hardware architecture called *Multi-Fovea Framework* comprising of three different types of processors for ideal computation of each image processing step, which communicate via a complex *Memory Manager Unit* (Figure 3.1).

The first processing unit is called *Frontend Processor Array* (FPA) used for preprocessing, and it also contains the sensor (or interface) for image capturing.

3.4.1. Frontend Processor Array

The data-parallel structure of the problem allows the usage of a large number of independent *threads*, each processing small, possibly overlapping partitions of the input image maps. Since the data and operators rely on 2D pixel topology, it is practical to identify the threads with 2D ID-s. If the threads are branchless, *Processing Elements* may share a common *Instruction Unit*.

The underlying implementation of the FPA can be a strong single threaded processor or a pixel-pipeline. Alternatively, a real array of cores may be designed executing many data parallel threads with distributed local memory. In the later case, communication links are required to neighbors for sharing overlapping data either in a coarse grain or a fine grain configuration.

As a result of preprocessing, the fixed sequence of operators produces some filtered versions of the input frame combined with some images from the past. Combination of grayscale maps should produce at least one feature mask indicating interesting locations.

Preprocessing should run in real time keeping up with the frame rate of the input source. This unit must have enough local memory to store all intermediate data in the processing step of a given input frame – *Short Term Local Memory (STLM)*, and even some extra memory to store results from previous time instant(s) – *Long Term Local Memory (LTLM)*.

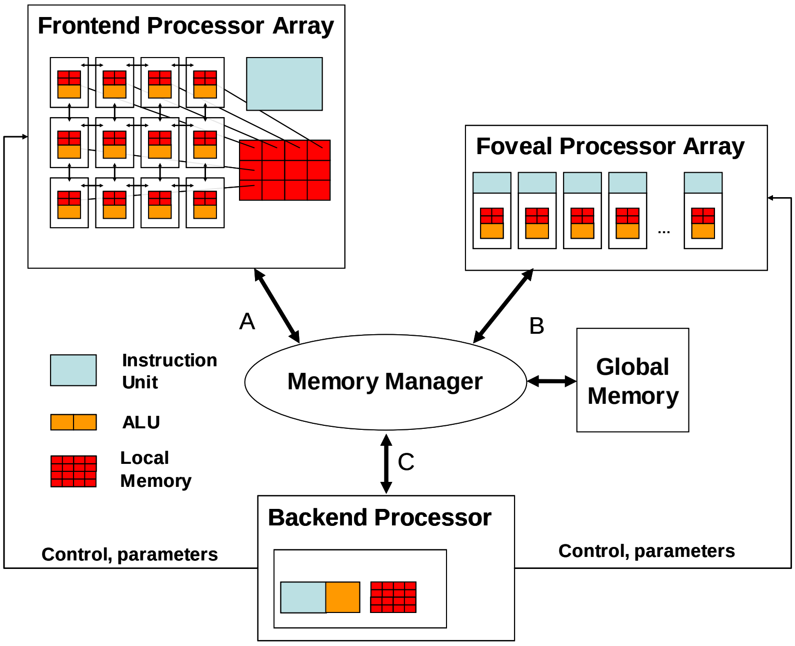


Figure 3.1. Main processing elements of the virtual hardware architecture: the Frontend Processor Array (FPA) for data-parallel steps with Processing Elements in 2D topology; the Foveal Processor Array (FVA) for task-parallel steps, and the Backend Processor (BP) responsible for control, organization, and classification.

A Processing Element (PE) consists of some registers and a logic and arithmetic unit (ALU) and optionally some local memory. An instruction unit can support multiple PEs. Images can be stored in a distributed way in the FPA to grant fast access to mapped image parts if communication link for neighbors are present for sharing overlapping data.

Processors interact via an intelligent Memory Manager and some direct control lines. List of 2D vectors, scalars and images of full or scaled size are transferred through the communication channel "A". Channel "B" is for images of foveal size and scalars (or vectors). Images of arbitrary size and scalars (or vectors) are transferred through channel "C".

The basic operators used in preprocessing are enumerated as follows.

The resolution of the sensor array is A_0 (m rows and n columns). In some cases a smaller resolution is enough for describing the scene. Support for down-sampling to create images with scale factor (2,4,8..) pixels is highly desirable.

Definition 40 **Image scaling operator, scale factor**

An image scaling operator is a unary image to image operator resulting in a smaller image matrix. The resolution of the sensor array is $A_0 = [M, N]$ (M rows and N columns). The image downscaled with factor $s \in \mathbb{Z}$ has the resolution $A^{\frac{1}{s}} = \left(\frac{1}{s^2}\right) A_0$.

The mapping for image matrix elements is implementation specific.

For data parallel execution it is important to split images into smaller pieces and implement image-image operators using those parts.

Definition 41 **Image window (window)**

An image window (or window, in short) w is a pair $\ \mathbf{c}, \mathbf{r}\ $ defining a rectangular region of an image. $\mathbf{c}$ is the center vector, and $\mathbf{r}$ is the radius vector (half size along both dimensions).

Definition 42 **Extracting operator (CutOut)**

An extracting operator is an operator resulting in a part of the image defined by a window. The extracting operator may have a boundary condition for image parts not covered by the input image matrix in case of an overextended window.

$$E(\mathbf{I}_1\ \ \mathbf{x}\ \ , w(\mathbf{c},[r_1, r_2])) : \mathbf{I}_1\ \ \mathbf{x}\ \ \rightarrow \mathbf{I}_2\ \ \mathbf{x}$$

$$\left\| \mathbf{I}_2\ \ \mathbf{x}\ \right\| = r_1 * r_2$$

The Frontend Processor Array is a virtual regular 2D topology of processing elements. The image must be split into tiles to allow one-to-one mapping between processing element and image windows. A regular topology can be described by simple constraints: the common radius values for windows, and a grid describing center points. The memory manager can be used to position the grid with some offset. The radius for extraction can be set to cover all image pixels, or some pixels may be excluded. Overlapped windows are not preferred, since processing elements can access the needed data from their neighbors through communication mechanisms. The processor – window mapping is rigid one-to-one association that must be defined in the program. The defined grid dimensions mapped to the physical processor structure through virtualization. If the physical dimension is larger, some processing elements are in idle state.

Definition 43 $m \times n$ **Grid**

An $m \times n$ grid is a 2D list of vectors $\{\mathbf{g}_{v,u}\} : \mathbf{g}_{v,u} = [u,v]^T ; v \in \ 1,2,..m \ ; u = \ 1,2,..n$, or scaled and translated: $\mathbf{g}_{v,u}* = \lambda \mathbf{g}_{v,u} + \mathbf{t}$.

Definition 44 **Tiles of an image**

Tiles of an image compose a 2D list of images extracted from an image by using all the vectors of a grid with the same radius, covering all pixels once.

$$T \ \mathbf{I}, \mathbf{g}_{v,u}, r \ = E \ \mathbf{I}, w(\mathbf{g}_{v,u}, r) \ = \ \mathbf{I}_{T \ v,u}$$

Definition 45 **Concatenation operator (Merge)**

Images with proper sized image matrices can be concatenated (horizontally or vertically) by concatenating their image matrices.

$$C_h \ I_1 \ \mathbf{x} \ , I_2 \ \mathbf{x} \ = I_3 \ \mathbf{x} \ ; \mathbf{I}_3 = \ \mathbf{I}_1 \ \ \mathbf{I}_2$$

$$C_v \ I_1 \ \mathbf{x} \ , I_2 \ \mathbf{x} \ = I_4 \ \mathbf{x} \ ; \mathbf{I}_4 = \begin{bmatrix} \mathbf{I}_1 \\ \mathbf{I}_2 \end{bmatrix}$$

The concatenation of an 1D image list is a single image, defined by the concatenation of image matrices in the order of indexing. For a 2D image list along the first dimension, horizontal concatenation is done for all images in a given row, resulting in a 1D image list, then a vertical concatenation is applied resulting in a final image.

The concatenation of non-image typed arguments is a list.

Image processing operators following a pure stream model may be calculated over tiles of images. This splitting allows the use of many processing threads. *Pointvise operators*, for example addition can be calculated by applying a core operator (*Add* in the example) to each pair of tiles and then by concatenating partial results.

$$I \ \mathbf{x} \ + I \ \mathbf{x} \ = \mathrm{int}\, erpolate \ \mathbf{I} + \mathbf{J}, \mathbf{x}$$

$$\mathbf{I} + \mathbf{J} = C \ \ Add \ \ T \ \mathbf{I}, \mathbf{g}_{v,u}, r \ \ , T \ \mathbf{I}, \mathbf{g}_{v,u}, r$$

Definition 46 **Pointvise operators**

The operator *F is a pointvise binary image operator* if the image matrices of the operands are defined to generate results equal in size to the inputs, and all elements of the output matrix are dependent only on the corresponding input element.

$$F : \mathbf{F}_{v,u} = f \;\; \mathbf{I}_{v,u}, \mathbf{J}_{v,u}$$

A special class of operator, neighborhood operators or 2D topologic operators are definable with a grid and a core operator, thus they are good targets for parallelization. A core operator refers a range of pixels in a given radius as read set, and updates a smaller range of pixels. In that case the write sets are disjoint and cover the full image, although the read sets overlap. The most widely used neighborhood operator is convolution. In finest granularity, convolution is defined for separate output pixels.

To implement such operators, *overlapped tiling* (*T**) is needed. The union of all tiles is always larger than the original image matrix.

$$\mathbf{A} = \mathbf{I} \otimes \mathbf{J}_{3x3} : \mathbf{A}_{v,u} = \sum_{i=v-1..v+1} \sum_{j=u-1..u+1} \mathbf{I}_{v,u} \cdot \mathbf{J}_{i,j}$$

$$\mathbf{A} = C \;\; Conv \;\; T* \;\; \mathbf{I}, \mathbf{g}_{v,u}, 3 \;\; , \mathbf{J}$$

To avoid overlapping tiles, a possible solution is to give read access to the core operators to neighboring tiles. This can be solved by communication between threads or buffered pipeline processing.

Details on efficiently implementing topological operators using various hardware configurations can be found in [29].

The main purpose of preprocessing is the localization of interesting regions, this is the final important functionality connected to the Frontend Processor Array is the location extraction function. It results in a vector of positions that can be used to define foveal windows.

Definition 47 **Location extraction operator (Locate)**

L is the unary location operator that results in a list of vectors that covers all true elements of the input mask:

$$L \;\; \mathbf{M} \;\; = L_{\mathbf{M}} = \;\; [u,v] \,|\, \mathbf{M}_{v,u} = true \;\; ,$$

$$\forall [u,v] : \mathbf{M}_{v,u} = true \rightarrow [u,v] \in L_{\mathbf{M}}$$

3.4.2. Foveal Processors

After preprocessing, interesting regions with a resolution of $mw \times nw$ (Aw size) are selected and stored in a list. Individual windows are referred to as w_i, whereas the coordinate of the corresponding center is referred to as $\mathbf{w}_i$. Foveal windows are extracted from some preprocessed image by the Memory Manager Unit, and sent to a processing element of the Foveal Processor Array.

<u>Definition 48</u> **Foveal image**

$\mathbf{I}_{\mathbf{F}\ i}\ \mathbf{x}$ is a foveal image, the result of an extraction operator applied to a given image.

$$w_i \ : \ \mathbf{I}_{\mathbf{F}\ i}\ \mathbf{x} = E\ \mathbf{I}, w_i$$

Since the windows are not covering the full image, this operator is called the CutOut operator. CutOut may take multiple images. In that case it is a group of operators resulting in multiple lists of foveas taken from the same regions of each image (see Figure 3.2).

Foveal processors (cores inside FVA) are fed by the Memory Manager Unit. This unit maps the corresponding windows of the filtered images to the memory space of a processing unit. Improved analysis needs more sophisticated algorithms with branching; therefore, these steps are task-parallel rather than data-parallel. It means the possibility for pipelined processing is limited.

The foveal windows can be distributed in various configurations and their overlapping is small, thus topological thread – processing element mapping is not reasonable.

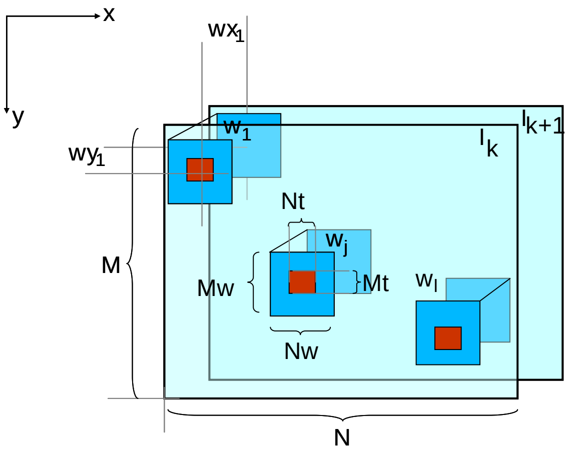


Figure 3.2. On the preprocessed maps interesting locations are marked. Since feature points can be distributed in various configurations and they are not linked, 2D fovea indexing is not necessary. Foveal regions are squared windows extracted from the maps. A window is referred to as w_i *,whereas the coordinate of the corresponding center is referred to as* $\mathbf{w}_i$*. Foveas cover* $mw \times nw$ *sized regions. Operations executed inside foveas can be sophisticated, characterized by large neighborhood (*$mt \times nt$ *sized) and possibly branching in their control flow.*

To describe a feature by a support region, some fixed number of pixels is required to have enough variance. In most cases, this means that window size in pixels does not depend on the scale of the given map. Instead the size of the window is fixed; therefore, the coordinates of the centers are scaled before merging.

Operations inside foveas usually use large neighborhood ($mt \times nt$), and refer only some locations inside the foveal window, and not all possible placements. This function may be supported by local CutOut or more efficiently by indirect addressing. This operator is called Index to distinguish it from the one working in the memory manager.

<u>Definition 49</u> **Local extraction operator (Index)**

Index is an extracting operator working locally inside a Foveal Processor Array resulting in a part of the image, defined by an offset vector and a radius. The index operator may have a boundary condition for image parts not covered by the input image matrix in case of an overextended region definition.

$$E(\mathbf{I}_t, \mathbf{d}, [Nt, Mt]^t):\ \mathbf{I}_t, \mathbf{d} \ \rightarrow \mathbf{I}_2$$

$$\|\mathbf{I}_2\| = [Mt, Nt]^t$$

Accumulation of partial results through commutative operators –addition, for example – is common. Support for so called scan primitives is a useful additional feature in foveas.

Since the number of the foveas may be much larger than the number of the processing elements, long term local memory is not required at this level. All results needed for the next iteration need to be saved. The frame is processed when all foveas are ready. Foveal processors may have more sophisticated programs with branches and limited iterations as well, optionally supported with high level data-parallel instructions implemented by further threads. In this case templates may have large radius and may be executed only at given locations, not for all possible placements inside the foveal windows.

Output of a fovea may be an image part backprojected by the memory manager unit to a global image using the position of the fovea in the original frame, or some scalars collected to a list.

3.4.3. Backend Processor

The Backend Processor is a serial processor that can access any global memory space and conducts all the serial calculation. BP is capable of setting up the window configuration for foveas and loading the program for both foveas and frontend units.

Algorithmic steps should be analyzed, and depending on their properties, different mappings could be applied. Considering data transfers, computational steps should be assigned to the appropriate virtual hardware module. This is the *partitioning* step of the algorithm design.

To sum up the requirements about the virtual architecture, the following definitions are given:

Definition 50 **Frontend Processor Array (FPA)**

The Frontend Processor Array is one array processor of the virtual hardware model for processing full-sized images in cellular data parallel fashion through 2D indexed threads, with efficient global synchronization. The main operator family consists of fine grained 2D topological operators with small neighborhood, defined by branchless core operators described on finest granularity. It should support numerous Short Term Local Memories, and some Long Term Local Memories in full image size. CutOut and Merge should be implemented (or may be implicit via distributed memory space). Read-only access for data space of neighboring threads should be granted.

It should provide an interface to the image sensor for full image upload. and download to the Memory Manager Unit, and should implement Locate. As an additional feature, it may implement scaling operators, and it may offer an interface to the Memory Manager Unit to CutOut foveas directly.

Definition 51 **Foveal Processor Array (FVA)**

The Foveal Processor Array is one array processor of the virtual hardware model for processing foveal regions extracted from images in task parallel fashion, with efficient local synchronization for threads processing a given fovea. It should handle 2D topological operators with larger neighborhood. Foveas are evaluated independently, control flows may diverge. It should support efficient synchronization between threads of a given fovea, and some mechanism between all foveas. Access to the data space of threads of a given fovea should be granted. It should support numerous Short Term Local Memories in foveal image size. It should provide an interface to the Memory Manager unit to upload and download foveal sized images. An indexing operator (indirect addressing) is a required feature in accessing local memory space. As additional features, it may implement Long Term Local Memory and communication between foveas, and direct accumulation.

Definition 52 **Backend Processor (BP)**

The Backend Processor is the serial processor of the virtual hardware model, for processing non-image data, or image data not fitting the capabilities of the other two execution units. It is responsible for coordinating the other modules.

Definition 53 **Memory Manager Unit**

The Memory Manager Unit provides data from the global memory for the other processors. Communication can be synchronized by the Backend Processor. This unit implements the CutOut and Merge operators for the Foveal Processor Array.

3.5. Implementation

The virtual architecture is feasible if a large portion can be realized in real hardware. With the emerging new devices the Multi-Fovea Architecture becomes more and more feasible and important to offer abstraction for different solutions.

3.5.1. Cellular Architectures

Along with the spreading of the Cellular Neural Network [30] paradigm, hardware implementations were introduced. The operators in the CNN paradigm may cover morphologies, convolution-like templates and kernel-based wave functions. The latter are implemented via direct feedback or iteration. The algorithmic capabilities of the CNN-UM [31] (CNN universal machine) were inspiring the definition of the Frontend Processor Array with its cellular distributed structure.

Implementations may be categorized into three major groups. The first and usually the most powerful is the mixed-mode (analog and digital) VLSI implementation like the Ace16K [18], the SCAMP chip [32], and the eye-RIS chip [19]. They are also referred to as focal plane processors since they have direct optical input via sensors integrated into processing elements. These architectures employ one-to-one mapping between pixels and processors. Vision Systems are created based on them like the Bi-i system [33] or the eye-RIS system designed for industrial purposes.

The second class is the emulated digital class that splits into two subcategories. The first is the pipelined version – such as the FALCON [34] architectures implemented on DSPs or FPGAs – while the other is the coarse grained processor array, for example Xenon [35], where n pixels are mapped to m processors in sparse-grain configuration.

The third category consists of optical implementations like POAC [36] which takes the advantage of optically feasible filters (e.g. lens).

These implementations give researchers a handful of tools for processing two or three dimensional sensory data flows which are received usually from visual or tactile sources. Universal Machine on Flows (UMF) [17] flow-chart is the basic algorithmic notation.

Systems built using these cellular arrays were always extended with a strong serial processor (or the CPU of the host system was used) to do serial classification tasks.

The foveal concept was introduced in the third version of the Bi-i system, through the Cellular Multi-core Video Analytics (CMVA) engine and in the highly configurable Xenon architecture. Those extensions are important for efficiently processing inhomogenous operators.

The latest development of the Eutecus Ltd. called VISCUBE [37] – designed in a joint effort with the researchers of Anafocus Ltd., and the Cellular Sensory Wave Computing Laboratory – is a true multi-foveal architecture.

3.5.2. Compute Unified Device Architecture

In the PC environment the multi-core tendency had a high impact on the video card market. Nowadays Graphics Processing Units (GPUs) with 128 processor cores are widely available. Owens et al. show in their survey [38] that more than five times greater computational power can be achieved compared to nowadays' CPU used worldwide in personal computers. The new architectures using general purpose cores in multiple single instruction multiple data (SIMD) groups offer data-parallel and also task-parallel parallelism. Nvidia discovered the extreme need for general purpose programming environments and released the Compute Unified Device Architecture (CUDA) environment [39] for their general purpose (GP-GPU) cards (Figure 3.3). Threads may be lunched in 2D or 3D topology with user defined granularity and core function. The contexts running on the same multiprocessor may synchronize. This setup is isomorphic to the Foveal Processor Array structure. In [1] we have described a mapping to realize functionalities of the Frontend Processor Array.

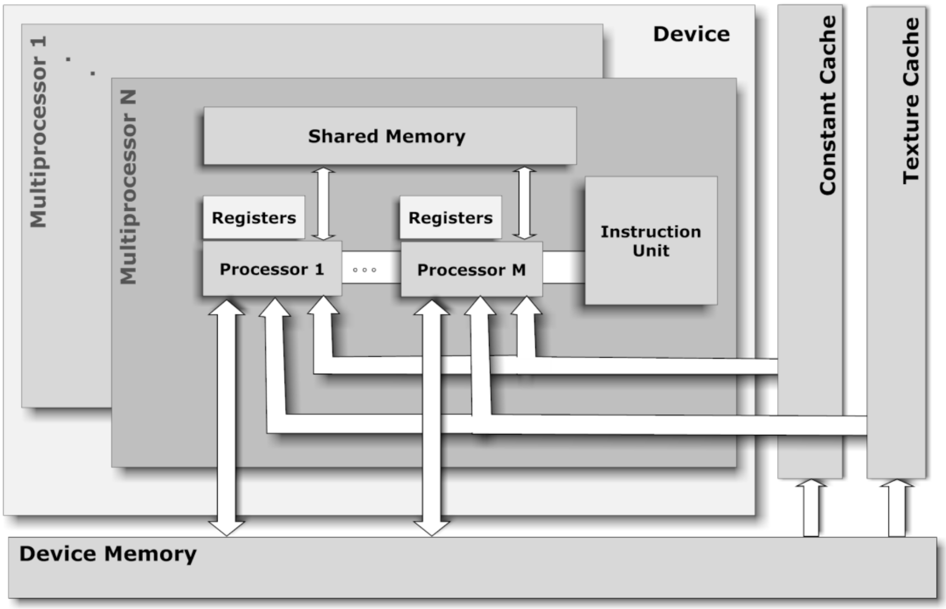


Figure 3.3. Hardware model of the Nvidia GeForce 8 series cards [20]. It is a new, unified hardware architecture with multiprocessors (MPs) that have dedicated shared memory accessed by a few scalar-based processors. Processors (processing units) work with their own registers and are driven by a common Instruction Unit forming a single instruction multiple data architecture. Algorithms can run on multiple MPs, although communication between MPs via the Device Memory is relatively slow. Data can be loaded from the read-only Constant and Texture Cache as well.

3.6. Conclusion

In this Chapter, the Virtual Multi-Fovea Architecture and computational framework has been proposed, which decomposes a broad class of image processing algorithms into topologically different parallel processor arrays. The description of the virtual architecture was also given to support the optimal hardware oriented decomposition from the initial description of an algorithm. It was shown that this concept is more effective for converging algorithms than the popular streaming model.

Thesis I.: Algorithms dealing with direct topographic sensory inputs may contain large number of steps suitable for data-parallel execution due to the natural structure of the data. Based on this observation, I have worked out a novel virtual hardware architecture model (Multi-Fovea Architecture) enabling communication-effective decomposition of those family of image processing algorithms, that are convergent, starting from direct sensory input and can be described by acyclic data flow graphs. The proposed structure effectively evaluates algorithms consisting of operators with different radii of coupling and topology, and nonhomogeneous spatial coverage by using three specific processor arrays. This heterogeneous structure fits the family of algorithms better than the general homogeneous parallel structures without losing general programmability.

Chapter Four

MULTI-FOVEA ALGORITHMS

4.1. Overview

4.1.1. Unmanned Aerial Vehicles

Unmanned aerial vehicles offer economical solutions for vegetation classification, flood and fire defense and large area surveillance. Today unmanned airplanes are capable of flying over the operation zone following a predefined path using an intelligent navigation system based on GPS and motion sensors. During the flight, they can gather information and transmit to a ground station via radio connections. Recorded video shots can be analyzed after landing in offline mode: consequently, thorough analysis is feasible either by human experts or using machine intelligence. The flight path can be modified when interesting events are detected in order to collect more detailed information. The aim of the ALFA project was to devise an optimal architecture for an onboard visual system capable of making these decisions. The Multi-Fovea Architecture had been designed to be universal for a wide range of video processing tasks. It is reviewed and analyzed focusing on the specific application area of independent ground motion detection.

4.1.2. Airborne Motion Detection

In large field airborne surveillance applications [40] the detection of moving ground objects is a key issue. After detection of these objects, they can be followed by the airplane and with enough information they can be identified as well. A good review for tracking can be found in [41]. Besides military applications, another application field is traffic monitoring [42].

For medium altitude video flows (100-300m) main streams in detection are optical flow [43][44][45] and registration-based methods using background subtraction. For low altitude videos, a real 3D analysis of the scene is required [46][47][48][49][50]. However, in large field surveillance tasks medium altitude is more common.

For a good review on general optical flow methods and registration methods, refer to [51] and [52], respectively.

In this Chapter, feature-based registration methods for background subtraction are reviewed and compared to highlight the capability of the framework. This approach for independent motion detection is popular among researchers [53][54][55][56]. Creating panoramic images from frames captured by a rotating camera is also an active research field. This problem covers similar registration tasks but may use offline algorithms with much larger computational needs [57][58][59][60][61].

Mikolajczyk and Schmid recently compared local image feature descriptors [62]. They highlighted the efficiency of the popular Scale Invariant Feature Transform [63]. I will compare the SIFT-based algorithm and the Kanade-Lucas Tracker [64][65] with traditional Block Matching [66] and Harris corner [67] -based Corner Pairing Algorithms. On the basis of the overall analysis, I propose a new algorithm called the Elastic Grid Multi-Fovea Detector, which is characterized by moderate hardware complexity while maintaining competitive detection quality.

4.2. Independent Motion Analysis

4.2.1. Images and Video Frames

Let us assume that the airplane flying over the inspection area carries a camera that faces the ground. The camera captures frames on regular time instances. Frames $\mathbf{I}_t\ \mathbf{x}$ ($t \in\ 1,2,..K$) are sampled light intensities that are projected to the image plane (sensor array) collected into a list for all time instances.

Homogenous representation of points on the image plane is a column vector $\mathbf{x}_{3_H}\ x1,x2,x3\ =\ x1,x2,x3\ ^T \mathbf{,} x1,x2,x3 \in \mathbb{R}$, where the corresponding point in Cartesian coordinates is $\mathbf{x}_2\ x1',x2'\ \mathbf{,}\ x1' = x1/x3; x2' = x2/x3$. Scene points (points in the 3D world) are represented by Cartesian coordinates in most cases $\mathbf{x}_3\ x1,x2,x3\ \mathbf{,}$ $x1,x2,x3 \in \mathbb{R}$. Homogenous representation will be denoted by the symbol “H” in subscript over the dimension.

Images are described by functions, and defined and stored using matrices. In practice, video sensors have finite resolution; therefore, intensity values in frames are defined at integer coordinated pixels only – m rows $\times\, n$ columns by the image matrix $\mathbf{I}_k$, in horizontal and vertical order, respectively ($u=1,..n$, $v=1,..m$), $\mathbf{I}_k\;\;u,v\;:=\;\mathbf{I}_{k\;\;v,u}$. For noninteger points, values can be interpolated – $\mathbf{I}_k\;\;\mathbf{x}_2\;\;,\mathbf{I}_k\;\;\mathbf{x}_{3^H}$.

The camera projects scene points to image points:

$$\mathbf{x}_{3^H}=P\;\;\mathbf{x}_{4^H} \tag{4.1}$$

P is defined more precisely in the 4.2.3 subsection. It assigns a ray of 3D points to an image point.

In a simplified capturing model we have light sources and reflecting surfaces. Pixel value in a frame is the total intensity coming from the specific ray; therefore, we are interested in the point $\mathbf{x}_3$ where the ray intersects a surface element of the scene. We consider surfaces that exhibit Lambertian reflectance (diffuse reflection). Therefore intensity for an image point depends on the incoming intensity and emission at the corresponding 3D location but not on the relative orientation of the surface element and the camera since the surface causes omnidirectional reflection.

$$\mathbf{I}_k\;\;\mathbf{x}_2\;\;=\mathbf{I}\;\;\mathbf{x}_3 \tag{4.2}$$

Detailed description of epipolar geometry and camera models can be found in [68] and [69].

4.2.2. Background and Objects

The scenes considered – namely large open-field areas or highways – with region of interest constraints may be regarded as flat surfaces, since the variation in height of the ground is small compared to the distance from the camera. Thus, we can model the ground as a plane with a texture map $\mathbf{B}\;\;\mathbf{x}_2$. This texture is the *background* image, describing the intensity values of the static empty screen. In some cases a small part of the sky is also visible in the frames. The bounded volumes of the 3D scene having non-negligible height or changing their position are objects. Objects in frames can be described by their *shapes* and *appearances*. The silhouette of an object is the region where it covers the background. The shape is the description of the silhouette, and the appearance is the model how it alters the background. Both properties are time-dependent because of the camera motion. By definition, areas where shadow is cast also belong to the specific object.

4.2.3. Global Image Motion Model

Using a homogenous vector representation of image coordinates $\mathbf{x}_{3^H}$ and world points $\mathbf{x}_{4^H}$, camera mapping (4.1) may be directly described as a 3×4 linear projection:

$$\mathbf{x}_{3^H:k}(x',y',1) = P(\mathbf{x}_{4^H}) = H^k{}_{3\times 4} \;\; x,y,z,1 \;^T \tag{4.3}$$

This representation may be used for pinhole or orthographic camera models representing camera pose-dependent external parameters and internal parameters as well. This linear model is a good approximation if the lens distortion is compensated or negligible as in our case. The world coordinate system may be defined as the ground plane lying in the ”X–Y” plane. The camera at time instant k is located at $\mathbf{c}_{3:k}$ and has a specific orientation. During the frame-by-frame time the camera center is moved and its orientation is changed. Points from the surface are projected to image planes, forming video frames $\mathbf{I}_k \;\; \mathbf{x}$ and $\mathbf{I}_{k+1} \;\; \mathbf{x}$. Since for all background points the z coordinate component is zero, mapping can be simplified. The plane-to-plane transformation for the actual image can be described by a 3×3 linear assignment.

$$\begin{aligned} \mathbf{x}_{3^H} \;\; {}_k &= H^k{}_{3\times 3} \;\; x,y,1 \;^T \\ \mathbf{x}_{3^H} \;\; {}_{k+1} &= H^{k+1}{}_{3\times 3} \;\; x,y,1 \;^T \end{aligned} \tag{4.4}$$

Or a direct relation may be expressed between points in images k and $k+1$:

$$\mathbf{x}_{3^H} \;\; {}_{k+1} = H^{k+1}{}_{3\times 3} \left[H^k{}_{3\times 3} \right]^{-1} \;\; \mathbf{x}_{3^H} \;\; {}_k \tag{4.5}$$

$$\mathbf{x}_{3^H} \;\; {}_{k+1} = H^{k+1,k}{}_{3\times 3} \;\; \mathbf{x}_{3^H} \;\; {}_k \tag{4.6}$$

This transformation maps points from the coordinate system of the k th frame to the coordinate system of $k+1$th frame. The geometrical transformation may be calculated for all image points of $\mathbf{I}_k$:

$$\begin{aligned} &\mathbf{I}_k \to \mathbf{J}_k \;\; H^{k+1,k}{}_{3\times 3}[u,v,1]^T \;\; = \mathbf{I}_k \;\; [u,v,1]^T \\ &u \in \{1,2,..n\}; v \in \{1,2,..m\} \end{aligned} \tag{4.7}$$

This means that frames containing common parts from the background can be aligned by a linear transformation matrix by using homogenous representation. In the most general case this can be a projective transformation. This is the global model for image motion (*global motion model*) describing the effect of the camera motion in consecutive frames.

To calculate a smooth transformation, integer coordinates are used in the target coordinate frame and interpolation is applied in the source frame (inverse mapping):

$$\begin{aligned} &\mathbf{J}_k\ [u',v',1]^T \ = \mathbf{I}_k\ \left[H^{k+1,k}{}_{3\times 3}\right]^{-1}[u',v',1]^T \\ &u' \in \{1,2,..n\};v' \in \{1,2,..m\} \end{aligned} \tag{4.8}$$

4.2.4. Motion Detection

Using a global motion model, more frames can be aligned to a common coordinate frame. A large *mosaic* image can be created from aligned images combining image matrices where they overlap (*blending*), and fill uncovered regions with a default value.

In most cases the airplane flies above an unknown field, which means the background image is unknown; if the background image is known, then the pose of the airplane is unknown. Indeed for the InputFrame ($I_{k+1}\ \ \mathbf{x}$), the previous image BaseFrame may be used as a reference after estimating the proper global motion and AlignedFrame can be calculated ($J_k\ \ \mathbf{x}$) from $I_k\ \ \mathbf{x}$. They both cover parts of the background and different snapshots of the moving objects. The detection is the process of creating DetectionMask with “1” elements for locations that are recognized to be part of an object silhouette in the frame of $I_{k+1}\ \ \mathbf{x}$. The clusters in DetectionMask are listed in separate masks $\mathbf{O}_j\ \ \mathbf{x}$ (ObjectMasks).

The first task is to calculate frame-to-frame alignment. If it is reliable for a sequence of consecutive frames, a local background mosaic can be constructed from them. It is a robust estimate for a part of the background image, more reliable than using only a single frame from the past. For slowly moving objects or objects with special motion vectors, a small projected motion vector arises, resulting in small changes for shapes in consecutive frames. For a steady camera, the solution is to decrement the frame rate, but for a moving observer, a large overlap is also needed for efficient frame-to-frame registration. Small errors in frame-to-frame registration do not limit the detection capability. However, the time span of reliable *local background mosaics* is limited, since the error accumulates.

Building a reliable global mosaic for estimating the background image and to track the full path of the airplane (Simultaneous Localization and Mapping) is a difficult problem and it is not covered in this work. The main objective was to solve the detection task.

4.3. Algorithms

As it was described in Section 4.2.1, the series of input image frames are considered as the main input to the system. They are projections of the scene at different camera locations and orientations since the airplane is moving.

In most cases, objects alter the background image in a special way, thus separate images can be analyzed for spatial features (e.g. colorful cars on the gray street). If the size of the object is known, even a filter tuned for a certain spatial frequency can be used. Since the background may also be textured and it is difficult to link features to form contours, it is more tempting to extract primitive spatial features and evaluate the change of their position in time. This means spatio-temporal analysis of the flow.

First, feature pairs are (i) extracted and (ii) matched. Using this point-to-point correspondence, (iii) a *global motion model* can be estimated. Finally, (iv) this transformation can be calculated for all pixel points in a frame using interpolation. The first four steps (Figure 4.2 a-d) of the process are called registration [52]. Since numerous feature pairs can be part of an object, a robust technique is necessary.

An *error measure* can be defined on the intersecting frame regions, and outstanding regions can be detected. Since background regions must fit with small error, extracted regions are objects. This concept works only if the objects cover a small portion of the frame. For a basic solution the necessary steps are summarized in Figure 4.2, representative stages in Figure 4.3.

The first step (a) is FeatureSelection. Feature points are selected from the new frame captured by the sensor (called InputFrame, or $\mathbf{I}_{k+1}$). At first a list of feature point locations is created, $\mathbf{fp}$ containing l_1 elements. For the extraction either some foveas are used or the full image is processed. Some feature locations are robust so they are selected for tracking: BasePoints $\mathbf{bp}$. BasePoints used at a given step are derived from $\mathbf{I}_k$.

The second step (b) is FeatureMatching. On the basis of image parts extracted from $\mathbf{I}_k$ from the vicinity of BasePoint locations and on FeaturePoints, a list of vectors is created, called InputPoints. For all elements in BasePoint a location is assigned with a *similarity measure* value (mu). If a point is lost, $mu\{i\}$ will be zero; if matching is robust, then $mu\{i\}$ will be equal to one. Matching is done by using l_2 numbers of foveal windows. Typically, this is the length of $\mathbf{bp}$ list. The signed difference between $\mathbf{ip}\{i\}$ and $\mathbf{bp}\{i\}$ is the ith displacement vector, $\mathbf{h}\{i\}$.

Using a threshold on mu values, reliable point correspondences are selected. The number of matched point pairs is l_3. Illustration for steps (a) and (b) is given in Figure 4.1.

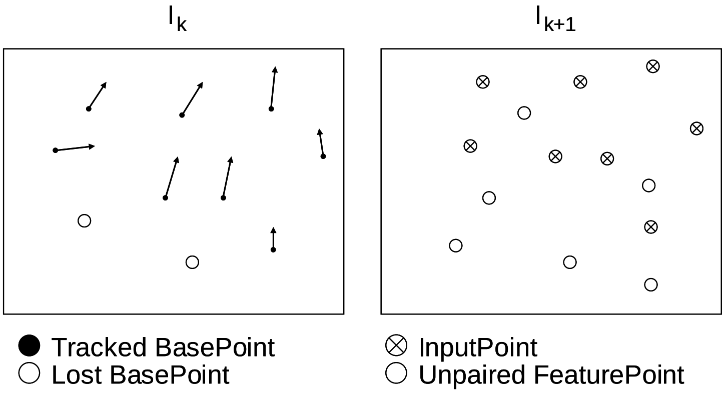


Figure 4.1. BasePoints are feature points selected for tracking on BaseFrame. New FeaturePoints are extracted from the incoming frame InputFrame – $\mathbf{I}_{k+1}\ \mathbf{x}$ *and matched, or search is applied in the vicinity of the BasePoint locations. During correspondence search / match InputPoints{i} is assigned to BasePoints{i}. InputPoint{i} is NULL if BasePoint{i} is lost.*

Steps (a) and (b) can be done simultaneously (Block Matching Algorithms - BMA). The regions around point pairs can be matched. There exists a transformation that maps one region to its corresponding pair in the consecutive frame regarding the chosen error measure. For short time intervals, even pure displacement can be used as a *local motion model*.

After extracting point features and forming pairs, based on (6), a transformation matrix can be linearly estimated using four point-to-point pairs. This is the third step of the algorithm (c). Since points are located with moderate precision in frames, some error arises even for background pairs. If the matrix is used for registering the full image afterward, it is crucial to use more correspondences with some robust fitting technique, for example *RANdom SAmple Consensus* (*RANSAC*) [70] or *Least Median Square*. Outliers after the fitting indicate moving objects with high probability.

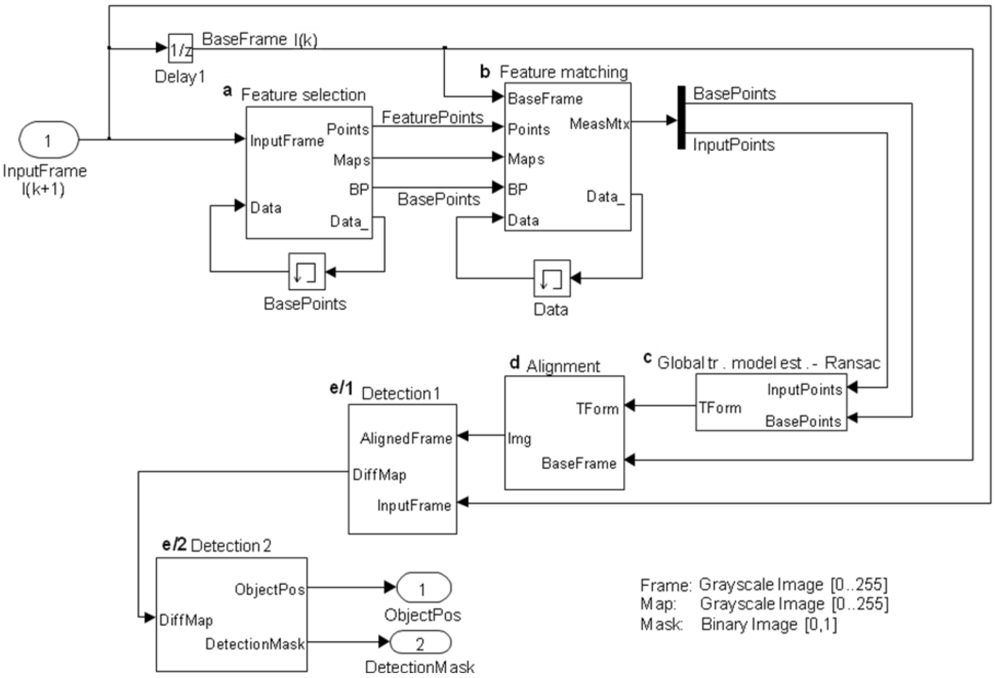


Figure 4.2. Global Registration Based Algorithm Family

a) Feature/template selection: locates robust feature point locations on the incoming frame, InputFrame $\mathbf{I}_{k+1}\ \mathbf{x}$ *.*

- Some grayscale maps are extracted along with the vector of robust feature point locations, FeaturePoints $\mathbf{fp}_j$ *.*

- BasePoints or $\mathbf{bp}_i$ *is a list of feature points selected for tracking in frame* $\mathbf{I}_k\ \mathbf{x}$

b) Feature/template matching: matches feature pairs, finds corresponding InputPoints $\mathbf{ip}_i$ *on* $\mathbf{I}_{k+1}\ \mathbf{x}$ *(or select form* $\mathbf{fp}_j$ *) for all BasePoints.*

- $\mathbf{ip}\ i$ *=NULL if* $\mathbf{bp}\ i$ *is lost.*

- mu_i *is also defined holding similarity measure for matching pairs.*

- Certain maps are stored for the next frame to support localization.

c) Global transformation model estimation: estimates transformation on point correspondences.

- A robust transformation matrix $\mathbf{H}^{k,k+1}{}_{3\times3}$ *is calculated to map points in* $\mathbf{bp}\ i$ *to* $\mathbf{ip}\ i$

d) Alignment: calculates transformation for $\mathbf{I}_k\ \mathbf{x}$ *and interpolates it.*

- The full image $\mathbf{I}_k\ \mathbf{x}$ *is transformed to the coordinate system of* $\mathbf{I}_{k+1}\ \mathbf{x}$

- The resulting image, AlignedFrame $\mathbf{J}_k\ \mathbf{x}$ *should be defined for all pixel coordinates, thus inverse mapping is applied with interpolation in the frame of* $\mathbf{I}_k\ \mathbf{x}$

$$\mathbf{J}_k\ \ u,v,1^T = \mathbf{I}_k\ \ \mathbf{H}^{k,k+1}{}_{3\times3}\ u,v,1^T$$

e) Detection:

e/1; calculates error map - DiffMap(AlignMap).

- DiffMap($\mathbf{E}\ \mathbf{x}$ *) is a grayscale image highlighting possible objects. Global registration based algorithms use AlignMap(* $\mathbf{E}_A\ \mathbf{x}$ *).*

$\mathbf{E}_A\ \mathbf{x} = absdiff\ \mathbf{I}_{k+1}\ \mathbf{x} - \mathbf{J}_k\ \mathbf{x}$

e/2; performs segmentation to create DetectionMask.

The BaseFrame can be aligned using the estimated transformation (d). DiffMap is a grayscale description with high pixel vales for suspected object regions. Global registration-based methods calculate an error measure; AlignMap takes the absolute-difference of the InputFrame and the aligned version of the previous frame. For this group of algorithms, DiffMap is defined to be equal to AlignMap.

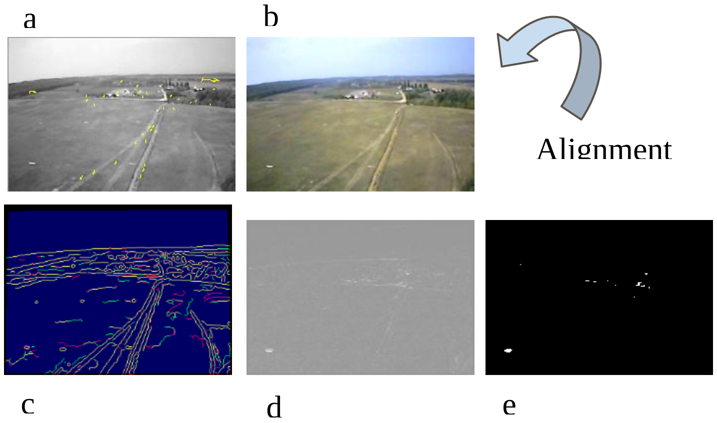


Figure 4.3. Intermediate Results for the Global Registration Based Ground –Motion Detection Algorithms: matched feature point pairs displayed over the previous frame (a); InputFrame (b); grayscale detection result (d); binary result with white blobs for moving objects (e). The overlayed edge-images (c) demonstrate the quality of the alignment.

Since frames have finite resolution, fine features – textures and region boundaries – are mapped to discrete pixels, the exact location depending on the interpolation strategy. This one pixel ambiguity can lead to high registration error around edges. Another reason for possibly high error values is when the underlying assumption of the flat-world model is violated. In those cases when an object changes its position between frames high error values also arise around present and previous silhouette locations. Thus, the analysis of the error map can highlight objects, especially moving ones. This method can identify object boundaries and non-overlapping object parts but not the exact object shape. Therefore, this process is called *moving object detection* as opposed to *object extraction* where the goal is to recover the exact object shape. However, this detection framework is considered to give a focusing mechanism for shape extraction. Foveas can be directed to these regions and further analysis is required to extract the object shape in a more computationally effective way.

If an object is detected in more frames, a tracker can be initialized to describe the motion of the object and possibly to build up a better object shape. Later on the track can be classified as belonging to a moving or a static object.

In the next four subsections four different methods will be shortly described. All of them utilize the basic algorithmic concept but focus different amounts of computational effort on specific stages of the estimation-detection procedure.

4.3.1. Corner Pairing Algorithm

One of the most widely used point feature extractors is the Harris Corner Detector [67]. It uses autocorrelation-function to extract locations with a small support region that robustly differ from their neighborhood, that is, have large intensity changes in both x and y directions inside their surrounding regions. These feature points are likely to be present in the next frame as well. Corners are extracted from the incoming frame and stored for matching in the next time step. If the support region of a corner in the Base-Frame is similar to a support region in the InputFrame, they are considered as projections form the same 3D region and paired. Feature extraction and matching routines were taken from Torr’s toolbox [71], which uses the sum of the absolute differences (SAD) as similarity measure for matching.

To extract interesting locations, the following image operators are needed:

Pointvise: **gain** (multiplication with constant), **product** (binary pv multiplication), **add**ition

Cellular (with $r=1$): derivation in x and y direction, Gaussian low-pass filter,

Locate: **max**imum location based locate in 3x3 neighborhoods

For constructing correspondence, there exist more sophisticated methods, for example graph cut [72]. As an alternative, a simple search for pairs may also be applied with gating based on Manhattan distance to keep complexity low (e.g., the three closest corners in $k+1^{\text{th}}$ frame are considered for each BasePoint). For this algorithm the second approach is used.

Since the feature extraction can be done with small neighborhood, it is tempting to do this step on the frontend processor array. Then, for each location in frame k the support window is extracted and matched with 3 windows from frame $k+1$. This step is within the capabilities of a foveal processor (AbsDiff and Accumulation operators are needed). If one matching is stronger than the others and also larger than a predefined threshold level, the pairing is considered to be successful. Since there is no search (possible locations are predefined), the window size can be equal to the template size. In the comparisons, this algorithm will be referred to as *Corner Pairing Algorithm* (*CPA*).

4.3.2. Block Matching Algorithms

If there is no hardware to support efficient array calculation to estimate autocorrelation for all pixels, larger regions can be handled together. One possibility is to define FeaturePoints statically as points of a sparse grid without locating feature points. After filtering uninteresting locations with negligible variation in pixel values, displacements are estimated for support regions centered at grid points. These techniques are called *Block Matching Algorithms* (*BMA*) or pattern matching algorithms.

The feature selection in this case can be done in the Foveal Processor Array with similar operators to the ones applied in CPA, except that Gaussian filtering may be exchanged to accumulation for the full window. The result from each fovea is a variation descriptor value that can be used to exclude bad windows from further processing.

A rectangular pattern, a *template*, is extracted from $\mathbf{I}_k \;\; \mathbf{x}$ around BasePoint locations and matched against displaced image parts of the same size as $\mathbf{I}_{k+1} \;\; \mathbf{x}$. Since there are no previously determined possible locations a search is performed in a given range. These search locations are displacements of integer values. They can be characterized by a similarity map centered around zero displacement. The basic operator of the search is the calculation of the similarity measure between the template and the corresponding image part of a given displacement at every try. It is called the *match operator*. In most cases this measure is the sum of the absolute differences or the sum of the squared differences. If the search radius is large, the Brute-force or Full Search method (*BMA-FS*) with exhaustive search can be outperformed by suboptimal or adaptive methods and solutions such as the Spiral Search, which focuses on smaller displacements at the beginning. They make an effort to keep count of already processed locations when selecting the next one. On the contrary, they calculate less elements of the similarity map than the Brute-force search. Block matching algorithms are widely used in video encoding for motion-compensation [MPEG1, MPEG2].

The search logic and the match operator may be implemented in the Foveal Processor Array. Since evaluation of different foveas may diverge, this is a real task-parallel problem. In each iterations of a given fovea the calculated matching for the current displacement vector is accumulated locally, and finally the best batch is sent for merging.

Diamond Search (*BMA-DS*) is one of the preferred adaptive methods. Diamond Search uses two diamond shaped *search pattern*s: a *Large Diamond Search Pattern* (LDSP; 5×5) and a *Small Diamond Search Pattern* (SDSP; 3×3). The similarity measure is calculated at every displacement grid point masked by the actual pattern and registered, thus overlapping possibilities are calculated only once. However, all of them are considered when optimum is chosen for the current step. Search starts with LDSP step which is repeated until the actual optimum is at the center of the mask, at which point a final SDSP is applied to find the exact solution.

In Figure 4.4 an example is presented. Optimal displacement is $4,1$ which is found using 21 search operators as opposed to 81 operators in the case of Full Search.

Since BasePoints are fixed, there is no guarantee that the support region has significant intensity variance to support matching, thus a relatively large template size is necessary and autocorrelation-based prefiltering can be useful to eliminate BasePoints in homogeneous image regions.

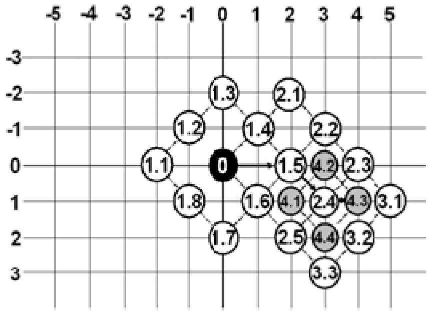


Figure 4.4. Diamond Search Steps: 1. apply LDSP around (0,0) grid point and step East. 2. apply LDSP around (2,0) and step South-East, 3. apply LDSP around (3,1) . Since best choice is (3,1) which is the center element, apply an SDSP around (3,1). The final result is (4,1). Altogether 9+5+3+4=21 match operators were executed, whereas for a Full Search with radius=4 81operators are needed. The example is from [81].

4.3.3. Kanade-Lucas Tracker Based Algorithm

The Kanade-Lucas Tracker (*KLT) algorithm* is a well-known solution for tracking feature points in a video flow. The basic concept of Lucas-Kanade optical flow calculation was presented in 1981 [64] and later extended to track feature points [65].

Point features are extracted exploiting the properties of the selected local matching model. In the basic realization a pure displacement model is used for consecutive frames, although an extension for affine changes also exists.

The template is extracted from the BaseFrame and matched in the new InputFrame. The similarity measure is the (weighted) sum of squared differences for all the pixels of the template. The matching is done with subpixel accuracy; therefore, interpolation is needed.

$$\text{E}=\sum_T \left[\mathbf{I}_{k+1}\;\;\mathbf{x}+\mathbf{h}\;\;-\mathbf{I}_k\;\;\mathbf{x}\;\;\right]^2 \tag{4.9}$$

The optimization for the minimal similarity measure is done using a zero constraint for the gradient.

$$\begin{aligned} 0 &= \frac{\partial E}{\partial \mathbf{h}} \\ 0 &= \frac{\partial \sum_T \left[\mathbf{I}_{k+1}\;\;\mathbf{x}+\mathbf{h}\;\;-\mathbf{I}_k\;\;\mathbf{x}\;\;\right]^2}{\partial \mathbf{h}} \end{aligned} \tag{4.10}$$

If *h* is small, $\mathbf{I}_{k+1}\;\;\mathbf{x}+\mathbf{h}$ may be estimated by its Taylor polynomial.

$$0=\frac{\partial}{\partial \mathbf{h}}\sum_T \left[\mathbf{I}_{k+1}\;\;\mathbf{x}_i\;\;+\begin{bmatrix}\dfrac{\partial \mathbf{I}_{k+1}}{\partial x}\;\;\mathbf{x}_i & \dfrac{\partial \mathbf{I}_{k+1}}{\partial y}\;\;\mathbf{x}_i\end{bmatrix}\mathbf{h}-\mathbf{I}_k\;\;\mathbf{x}_i\;\;\right]^2 \tag{4.11}$$

$$0=\sum_T 2\begin{bmatrix}\dfrac{\partial I_{k+1}}{\partial x}\;\;\mathbf{x}_i \\ \dfrac{\partial I_{k+1}}{\partial y}\;\;\mathbf{x}_i\end{bmatrix}\left[I_{k+1}\;\;\mathbf{x}_i\;\;-I_k\;\;\mathbf{x}_i\;\;+\begin{bmatrix}\dfrac{\partial I_{k+1}}{\partial x}\;\;\mathbf{x}_i & \dfrac{\partial I_{k+1}}{\partial y}\;\;\mathbf{x}_i\end{bmatrix}\mathbf{h}\right] \tag{4.12}$$

$\mathbf{x}_i$ elements are taken from a rectangular area; therefore, $\mathbf{I}_k\;\;\mathbf{x}_i$ and $\mathbf{I}_{k+1}\;\;\mathbf{x}$ values can be collected after interpolation to $\mathbf{F}$ and $\mathbf{G}$ matrices, respectively. Using subscript notations "*x*" and "*y*" for spatial derivatives and $\bullet$ for element wise product (12) translates to:

$$\begin{bmatrix} \sum \mathbf{G}_x \bullet \mathbf{G}_x & \sum \mathbf{G}_x \bullet \mathbf{G}_y \\ \sum \mathbf{G}_y \bullet \mathbf{G}_x & \sum \mathbf{G}_y \bullet \mathbf{G}_y \end{bmatrix} \begin{bmatrix} \mathbf{h}_1 \\ \mathbf{h}_2 \end{bmatrix} = \begin{bmatrix} \sum (\mathbf{F}-\mathbf{G}) \bullet \mathbf{G}_x \\ \sum (\mathbf{F}-\mathbf{G}) \bullet \mathbf{G}_y \end{bmatrix} \tag{4.13}$$

$$\mathbf{Z}_{2\times 2}\mathbf{h}_2 = \mathbf{e}_2 \tag{4.14}$$

This linear equation system can be solved, thus the local optimum can be found for the displacement vector. In order to calculate $\mathbf{h}$, the $\mathbf{Z}$ matrix must be invertible. This holds true if both eigenvalues are large positive numbers. This property is used for selecting good features to track. This feature selection is analogue to Harris corner extraction. The linearization error is moderate only for small displacement values; therefore, an image pyramid is created to support coarse-to-fine processing. Furthermore, an iterative search is applied at all levels to handle large displacements.

The pyramid creation can be supported by the Frontend Processor Array. The operators required for corner extraction were enlisted in the section of the Corner Pairing Algorithm. The displacement estimation fits to the Foveal Processor Array. Displacements are represented with 1/8 pixel accuracy with fix-point rational numbers.

The Index&Interpolate function can be implemented using linear weights on the corresponding parts of the image matrix. This function is not the implemented by the Memory Manager Unit as the CutOut operator, but executed directly in the short term local memory inside the Foveal Processor Array.

To implement division special arithmetical unit is needed inside the Foveal Processor Array.

4.3.4. Scale Invariant Feature Transform Based Algorithm

Scale Invariant Feature Transform (*SIFT*) [63] is a state-of-the-art solution for Key Point matching with two algorithmic steps. It extends the local displacement model with rotation and scale. The first phase extracts a scale invariant point set from Gaussian scale-space, whereas the second phase creates a distinctive descriptor vector that enables highly reliable feature point correspondence matching. This description is quasi invariant to affine transformations and illumination changes. The major drawback of the method is the numerical complexity, thus it cannot be realized exclusively on serial processors.

First, a Gaussian *scale-space pyramid* is generated using a series of convolutions of the input image and a Gaussian kernel $\mathbf{G}(x, y, \sigma)$.

$$G(x, y, \sigma) = \frac{1}{2\pi\sigma^2} e^{-\frac{(x^2+y^2)}{2\sigma^2}} \tag{4.15}$$

Parameter σ describes scaling. For consecutive *octaves* σ of the Gaussian convolution the kernel doubles, whereas the effective resolution of the image decreases by half. By resampling every second pixel, a starting image for the next octave is generated. The σ values are selected to span *O* octaves, with *ns* subdivisions in each. When the pyramid is ready, filtered images with consecutive scales are subtracted from each other to produce the *difference of Gaussian scale-space* (approximation of the Laplacian of Gaussian operator). (Figure 4.5).

$$L(x, y, \sigma) = G(x, y, \sigma) * I(x, y) \tag{4.16}$$

The feature points (key points) are selected from this three-dimensional image stack. A point is selected if it is a local maximum or minimum – depending on whether the luminance of the object was light or dark – of the neighboring ($3 \times 3 \times 3 = 27$) pixel values.

In his original work Lowe proposed to start scale-space generation from an interpolated double resolution image (up-scaling) and to calculate three intermediate scales for each octave.

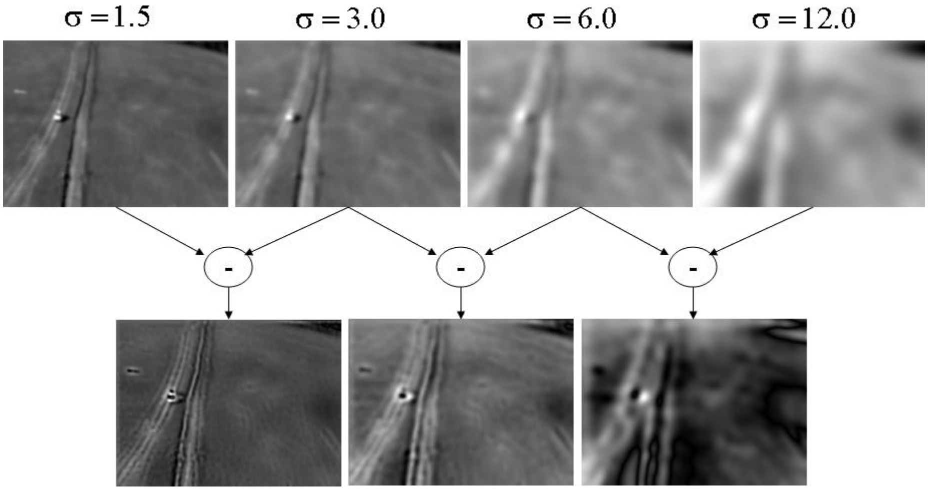


Figure 4.5. Top row: The input image is convolved with the Gaussian function (for example four monotone increasing sigma values: 1.5, 3.0, 6.0, 12.0), Bottom row: The nearest filtered images are subtracted from each other, creating the Difference of Gaussian (DoG) scale-space. Local extremes are extracted from the middle image using 3x3x3 connectivity as feature locations.

The size of the objects will shrink according to octaves, and small zooming effects may be cancelled due to the subdivisions in scale-space.

For locating the interesting points, several Gaussian filters are needed. They can be implemented by a series of 3x3 convolutions. 3x3x3 maximum location over consecutive differences may be implemented with a series of 3x3 maximums and 3x3 logical functions, thus the feature selection may be assigned to the Frontend Processor Array.

The SIFT descriptor is extracted from the vicinity of the key point (template region) in the corresponding scale-map. First, the gradient vectors for all pixels indexed by their magnitude and orientation are calculated, and an orientation histogram with 36 bins is created. To achieve rotation invariance, a transformed template is calculated for all regions by rotating the templates. The amount of rotation is determined by the maximum peak of the weighted histogram to align most edges in vertical direction. Multiple descriptors are created if several significant peaks exist, which increases the robustness. Second, the updated templates are divided into 4×4 subregions, and a 8-bin histogram is calculated from the gradient vectors for each subregion in the same fashion as in the first step, resulting in a 128-long vector descriptor for all key points.

Creation of the descriptors would push the definition for Foveal Processor Array to be extended. It requires among topological steps the creation of histograms and localization of maximum positions. They are hard to implement in parallel. The efficient implementation of arctangent and square-root functions needs lookup tables that can only accessed serially. The imager rotation is also definitely a serial step. Altogether only a small percentage of a fovea's workload can be executed in parallel.

Descriptor vectors can be matched with gating on proximity using the scalar product as a similarity measure by the serial Backend Processor.

4.3.5. Global Registration-Based Detection

InputPoints and BasePoints can be filtered to remove unreliable elements: $\mathbf{bp} \rightarrow \mathbf{bpf}$, $\mathbf{ip} \rightarrow \mathbf{ipf}$. l_3 denotes the number of point pairs. After point correspondences are extracted, alignment can be done by searching for optimal transformation. The global motion model is a projection that is estimated by the *Direct Linear Transform* (*DLT*) method [68]. To make this review self-contained, a brief summary is given.

Equation (4.6) may be rewritten using filtered points

$$\forall k : \mathbf{ipf}_{3^H} \;\; k \;\; \overset{=}{\longleftrightarrow} H^{k+1,k}{}_{3\times 3}\mathbf{bpf}_{3^H} \;\; k \tag{4.17}$$

This mapping is defined on homogenous coordinates, which means that the vectors are not equal but parallel, differing by a non-zero scale factor. It is better to emphasize that they are collinear by using cross product.

$$\mathbf{ipf}_{3^H} \;\; k \;\; \times H^{k+1,k}{}_{3\times 3}\mathbf{bpf}_{3^H} \;\; k \;\; = 0_3 \tag{4.18}$$

We can also use $\mathbf{h}_3^{1T}, \mathbf{h}_3^{1T}, \mathbf{h}_3^{1T}$ notation for rows in $H^{k+1,k}{}_{3\times 3}$, $\mathbf{ipf} \;\; _i$ for component of $\mathbf{ipf}_{3^H} \;\; k$, and $\mathbf{bpf}_3$ for $\mathbf{bpf}_{3^H}$

$$\begin{bmatrix} \mathbf{ipf} \;\; _2\mathbf{h}_3^{3T}\mathbf{bpf}_3 - \mathbf{ipf} \;\; _3\mathbf{h}_3^{2T}\mathbf{bpf}_3 \\ \mathbf{ipf} \;\; _3\mathbf{h}_3^{1T}\mathbf{bpf}_3 - \mathbf{ipf} \;\; _1\mathbf{h}_3^{3T}\mathbf{bpf}_3 \\ \mathbf{ipf} \;\; _1\mathbf{h}_3^{2T}\mathbf{bpf}_3 - \mathbf{ipf} \;\; _2\mathbf{h}_3^{1T}\mathbf{bpf}_3 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} \tag{4.19}$$

Furthermore, $\mathbf{h}_3^{1T}\mathbf{bpf}_3 = \mathbf{bpf}_3{}^T\mathbf{h}_3^1$

$$\begin{bmatrix} 0_3^T & -\mathbf{ipf} \;\; _3\mathbf{bpf}_3{}^T & \mathbf{ipf} \;\; _2\mathbf{bpf}_3{}^T \\ \mathbf{ipf} \;\; _3\mathbf{bpf}_3{}^T & 0_3^T & -\mathbf{ipf} \;\; _1\mathbf{bpf}_3{}^T \\ -\mathbf{ipf} \;\; _2\mathbf{bpf}_3{}^T & \mathbf{ipf} \;\; _1\mathbf{bpf}_3{}^T & 0_3^T \end{bmatrix} \begin{bmatrix} \mathbf{h}_3^1 \\ \mathbf{h}_3^2 \\ \mathbf{h}_3^3 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} \tag{4.20}$$

This gives equations for all corresponding feature pairs. Since the equations are corresponding to homogenous vectors, they are not independent. To solve the system, at least four point pairs are needed. The resulting over-determined linear system can be solved by using singular value decomposition (SVD). The singular values comprise $\mathbf{H}$.

To make the optimization robust against outliers, the RANSAC method can be applied. Its concept is to use a minimal set of points selected randomly to determine a transformation and then calculate a score for this selection. The score depends on the number of *inlier*s consistent with the model of this transformation. For inliers *distance measure* is smaller than a threshold limit. In this case four point pairs are selected. Degenerate point sets with collinear points should be avoided: before running SVD, a test should be performed. The transformation with the largest number of inliers (l_4) is selected among many tries.

If the probability of any point pair belonging to the background is q, the probability that any of the four selected point pairs is part of the foreground can be estimated as:

$$1-q^4 \tag{4.21}$$

To be sure to have selected only inliers at least once with – for example – a 99% probability, more trials should be evaluated (N).

$$1-\ 1-q^{4}\ {}^{N} > 0.99 \tag{4.22}$$

After estimating the transformation and having a l_{in} number of point pairs consistent with the actual best try, we can estimate q using the relative frequency:

$$\tilde{q} = \frac{l_3}{l_{in}} \tag{4.23}$$

Then, it is possible to evaluate (4.22) using the estimate $\tilde{q}$ and decide whether to generate further random sets. In addition, a hard limit for N can be defined to limit the number of iterations.

The best transformation candidate defines the final inlier set. As a last step, a DLT routine can be applied to all of the reliable pairs using the first two independent lines of (4.20) to yield the final estimate.

The complexity of the small SVD for all tries is:

$$9*12^2 + 12^3 \ = \ 3024 \tag{4.24}$$

whereas for the final DLT step complexity is in the magnitude of

$$9*\ 2*l4\ {}^{2} +\ 2*l4\ {}^{3} \tag{4.25}$$

Since this is cubic in the number of used pairs, the l_4 number is limited to 20.

For the implementation, the toolbox by Kovesi was used [73].

4.4. Elastic Grid Method

The calculation of the projective transformation of the global motion model is rather time consuming since a global spatial transformation with interpolation is required. The algorithm described in this section gives an alternative solution by estimating the global transformation with tiles and local displacements. It performs a joint optimization process through coupling of the local displacement estimations utilizing the multi-fovea concept and the possibility of using foveal windows for efficient calculation.

Even projective transformation conserves collinearity: if a point lies on a line defined by others, the points will still be collinear after the transformation. This property can be used to define an adaptive iterative search mechanism.

Elastic contours are popular tools for image processing applications, for example segmenting noisy images. The contour is built up from segments defined by control points. These points are iteratively moved in the image by a task-specific external force towards an exact segmentation result, whereas internal force balances this effect to keep contours pleasant (e.g. having low curvature). The elastic contour concept may be extended to an *elastic grid*, which could also be viewed as an extension and generalization of the Block Matching family. In this case $\mathbf{bp}$ points are not located feature points, but fixed points placed along a regular sparse grid. Since they are placed in a 2D topology, they can be naturally indexed with {row, column} indices, $\mathbf{bp}\{k,l\}$.

The algorithm starts with calculating the similarity measure for the template and corresponding region with integer displacements in a given range using normalized Sum of Absolute Difference (SAD). SAD values may be collected into a potential map for all {*i*,*j*} locations. Searching starts with $0,0\ ^{T}$ displacement. During the search, a 3×3 box search pattern is used. It follows that nine elements of the similarity measure surface are calculated.

In all iterations of the elastic grid evolution, for all windows the missing values are computed from the potential maps selected by the 3×3 search mask centered at the current $\mathbf{ip}\{k,l\}$ locations, and the smallest among them is selected to compute the corresponding external force ($\mathbf{F}_{ext}$). The amplitude is the difference between potential values of the current center and that of the selected location pointing in its direction.

By construction, all $\mathbf{bp}\{k,l\}$ form collinear points with their neighbors and the same must stand for corresponding $\mathbf{ip}\{k,l\}$ points. An elastic grid can be defined in Input-Points as control points, with the internal forces having $(2+2)$-neighbor connectivity (Figure 4.6). The collinearity constraint translates to the grid being pleasant, if connecting line segments are almost parallel or displacement vectors are close to the average of their neighbors. For calculating *x* and *y* components of internal forces ($\mathbf{F}_{\text{int}\ \ x}$, $\mathbf{F}_{\text{int}\ \ y}$) only data from neighbors in the west and east or in the north and south are used. Components of internal forces are defined as the difference from the sum of the corresponding displacement vector components weighted with their similarity measure:

$$\mathbf{F}_{\text{int}\ \ x}\ \ i,j\ = mu\ \ i,j\ \left[\mathbf{h}\ \ i,j\ \right]_x - \left(\sum_{k=-1,1} mu\ \ i+k,j\ \left[\mathbf{h}\ \ i+k,j\ \right]_x \right) \tag{5.1}$$

$$\mathbf{F}_{\text{int}\ y}\ \ i,j\ = mu\ \ i,j\ \left[\mathbf{h}\ \ i,j\ \right]_y - \left(\sum_{k=-1,1} mu\ \ i,j+k\ \left[\mathbf{h}\ \ i,j+k\ \right]_y\right) \tag{5.2}$$

Depending on the sum of internal and external forces, one neighboring element of the displacement grid is selected for all locations.

The search moves all control points toward smaller error values, but when the distortion of the grid grows, it is lowered by climbing to a slightly worse location of its potential field.

This joint optimization method can find a good solution for untextured windows with flat potential maps and can find global optima without the need for exhaustive search.

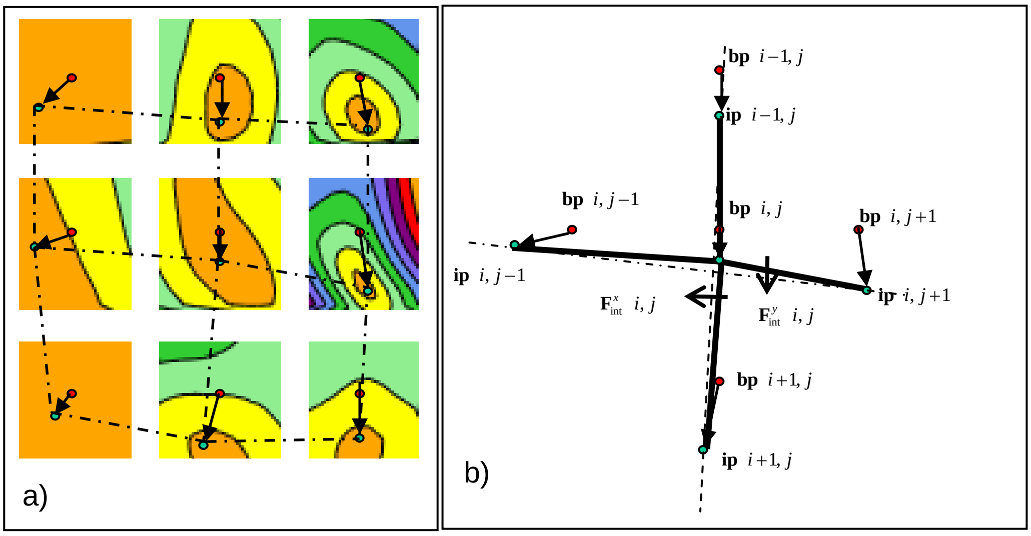


Figure 4.6. Elastic Grid Multi-Fovea Detector. BasePoints are not located but placed on a predefined 2D topology. They are indexed with 2D indices. Templates are extracted around BPs from the BaseFrame and matched against image parts from the InputFrame using the sum of absolute differences as a similarity measure. An elastic grid is defined on InputPoints. Grid starts from zero displacements and converges towards optimal displacement values. SAD values are arranged to form potential maps for external force calculation (a). Internal forces are calculated using 2+2 connectivity for x and y components (b).

In the Elastic Grid algorithm the global motion model is not estimated, and no global projective image alignment is done. Instead, the calculated displacements are applied to all corresponding regions.

A window containing an object with independent motion component would deform the grid that is mainly formed by the background features. Therefore, after a few iterations locations with high-amplitude internal force highlight possible object regions.

The multiple displacement model gives a tiled-alignment used for DiffMap calculation that can be analyzed in the same way as the first four algorithms. Alternatively, only the highlighted regions can be selected for analysis.

Since it cannot be presumed that enough cores exist in the Foveal Processor Array to cover all foveas at the same time, external handling of Long Term Local Memories are required. On the other hand, since the access can be scheduled together with the foveas, efficient transfer is possible.

4.5. Performance of Methods

4.5.1. Metrics for Quality

The quality of the algorithmic output can be assessed and compared both at the registration and the detection level. The overall metrics are defined to take both aspects into consideration.

Registration is described by:

the edge coverage defined on high-pass-filtered versions of images

$$(e_{Edge} = 1 - \frac{\left\| I_{edge} \cap J_{edge} \right\|}{\min(\left\| I_{edge} \right\|, \left\| J_{edge} \right\|)})$$

If the global transformation estimation is successful, homogenous regions are perfectly overlapping and a high percent of the boundaries (edges) are covered. A large percent of feature points should be part of the background, thus during optimization they should turn out to be inliers leading to a small global symmetric distance.

The ground-truth reference was created manually for all frames marking all objects with an independent blob (R_i). DetectionMap is labeled to result in a set of detection blobs (O_i). An object is detected if any detection blob intersects the corresponding reference. The set H contains objects that are detected. P_1 is the set of blobs that overlaps with any reference markings, whereas P_2 is the set of false positive detection patches. (Figure 4.7)

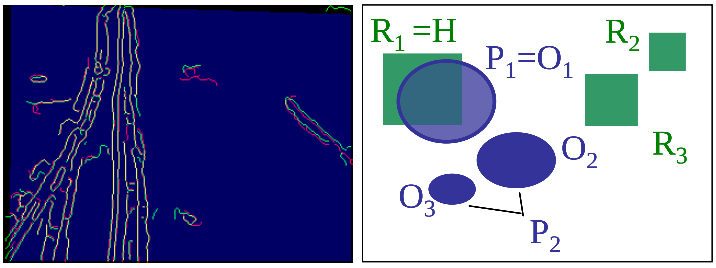


Figure 4.7. *Two levels of the performance comparision - Left: registration level. Right: detection level.* Edges of InputFrame are displayed in green. AlignedFrame is valid inside a region marked in blue. Edges of AlignedFrame are displayed in red. Alignment is perfect at regions where edges overlap, marked in yellow.

The ground-truth reference is created manually for all frames. Reference objects (R_i) *are displayed in green, whereas detection patches* (O_j) *are shown in blue. The set H contains detected objects.* P_1 *is the set of blobs that overlaps with any reference marking, whereas* P_2 *is the set of false positive detection patches.*

4.5.2. Time Complexity of Algorithms

The time complexity of each algorithm is presented using larger units. The steps defined in previous sub-sections are refined to functions and functions to elementary building blocks. The dataflow of an algorithm represents the elementary blocks and their connections. Complexity is given for all functions in the corresponding tables. Blocks within a given function are mapped to a common processor. The necessary data is fetched from the global memory and the results are written back in case they are needed for a function that is mapped to another processor or in case they do not fit to local memory.

Topological steps assigned to the Frontend Processor Array or to the Foveal Processor Array can be realized in serial, pipelined or array hardware components. For more details on efficient implementation of topological operators see [74].

In case of a fully serial solution for any calculations, all operands should be read to registers from local memory and subsequently all results should be written back. Transfers and operators are considered to consume 1 unit time per pixel. Enough registers should exist to hold intermediate data and constant values during an elementary operation. Indirect memory addressing may be used for processing a full matrix pixel-by-pixel. For this at least 3 indirect references are needed. Incrementing the address does not give extra time overhead. A core with a small number of registers can process all pixels and all the blocks of the dataflow in a serialized order. To store intermediate matrices Local Short Term Memory is needed.

It is tempting to use overlapping read, calculation, and write instructions since data can be processed in a well-defined serialized order. For overlapping neighborhoods, it is inefficient to fetch data multiple times. Instead, it is better to use an internal buffer from registers and pump data through it. For each time tick, one element from all input matrices is pushed in and one element of the output is produced after some delay. The computation is not characterized by execution time but by pixel-delay. If all blocks are realized with independent cores and connected with extra smoothing buffers to equalize uneven delays, the full function can be realized with a pipeline.

In case of fully parallel array processors, images are stored in a distributed manner. All cores have a small portion of multiple images in their registers. Point-by-point arithmetic can be done in one step, whereas it takes a few extra communication steps to calculate a neighborhood operator. To evaluate a function, intermediate images must be stored locally and building blocks are processed in a serialized order.

During comparison, array processor implementation for FPA with one-to-one pixel processor mapping was considered, with enough local memory, supporting point-point arithmetic, gauss filtering, downscaling, and logic operations. The complexity of foveal calculation was multiplied by the number of foveas. Since these parts are fully task parallel, the scale factor for these functions’ execution time is roughly inversely proportional to the number of physical execution units.

4.6. Comparison

To evaluate the capabilities and performance of the algorithms, output results for four video recordings have been compared. All videos had 240x320 pixel resolution. The first sequence is a rendered artificial 3D model – Artificial Sequence. Three sequences Godollo_1, Godollo_2, Godollo_3 were captured during the ALFA project by a micro UAV above the airport of Gödöllő (a city in Hungary). In the next subsections, measurements for quality and the time complexity are presented for all the four sequences with different parameters.

The most robust Full Search method is capable of giving reliable frame-to-frame registrations for long time spans. This was used as reference to show the correctness of the used plane-model for the sequences.

After analyzing a large number of frame-to-frame results, and accumulated results, I can state that image pairs with E_{Edge} - the error measure for full image alignment in high spatial frequency - less than 0.55 can be used to build local background mosaics and to track objects in the ground-based coordinate frame without significant accumulated registration error. In the case of larger error a new mosaic should be started. If E_{Edge} is smaller than 0.7, AlignedFrame can be used for detection without yielding large false positive error.

In the following sections, quality and computational complexity is analyzed and presented for sequences with different parameters.

For hardware (computational) complexity, the analyzed parameters are the template width and the prescribed number of feature points for region-based methods, whereas for the SIFT, it is the number of octaves (*O*) and intermediate scales (*ns*).

For detection quality, outputs with template width equal to 8 and 80 point features are compared to the case when SIFT was running on 2 octaves and 2 sub-scales (Table 1.).

The results presented in next subsections will also contain the values of the Elastic-Grid Multi-Fovea Detector.

	Artificial 135/130	Godollo_1 120/79	Godollo_2 300/230	Godollo_3 35/31	Complexity (operation per pixel)
SIFT	130	52	200	29	~1100
KLT	130	52	217	29	~450
BMA	128	61	208	29	~100
CPA	125	55	183	27	~75
ELD	92	52	194	28	~50

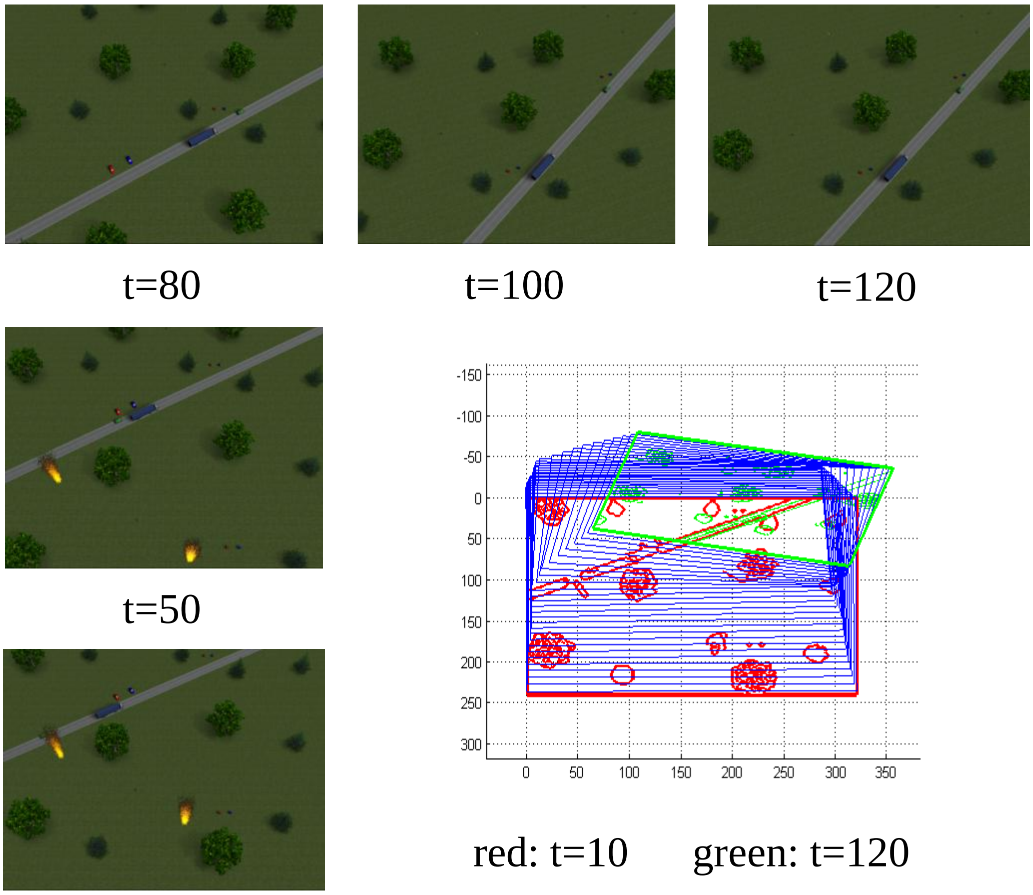
Table 1. True-positive detection results and computational complexity of the algorithms

Total number of frames in a sequence / the frame number on which target '1' is visible is given for each sequence in the header. Rows contain the number of true-positive detection results based on hand-made references together with the average computational steps normalized with the pixel count of the frames.

4.6.1. Artificial Sequence

Table 2contains some representative frames from the Artificial Sequence. This is a generated sequence of 135 frames. The airplane is flying above a green field with some brushes and trees. The ground plane is flat with a grass-like texture. There are also two fire spots. Two vehicles are moving on a long straight road of concrete. One of them is a fast moving car, the other one is a truck. The sequence is aligned to the 10-th frame and montage for displaying the global alignment is also displayed in the table.

The airplane is flying ahead for about 80 frames and when it is roughly above the road, it turns right. The size of the car is between 47 and 61 pixels, 56 in average. The size of the truck is between 277 and 301 pixels, 284 in average.



t=80 t=100 t=120

t=50

red: t=10 green: t=120

t=10

Table 2. Representative frames of the Artificial Sequence
In the figure reference transformations are used to show the views in a common frame. The edge image of the starting (red) and last frames (green) and boundaries of intermediate ones (blue) are displayed.
As it is presented, global alignment to a starting frame is possible using the composition of frame-to-frame transformations. The figure highlights that the flat-world model is adequate, giving large registration error for tall objects (brushes), and moving objects.

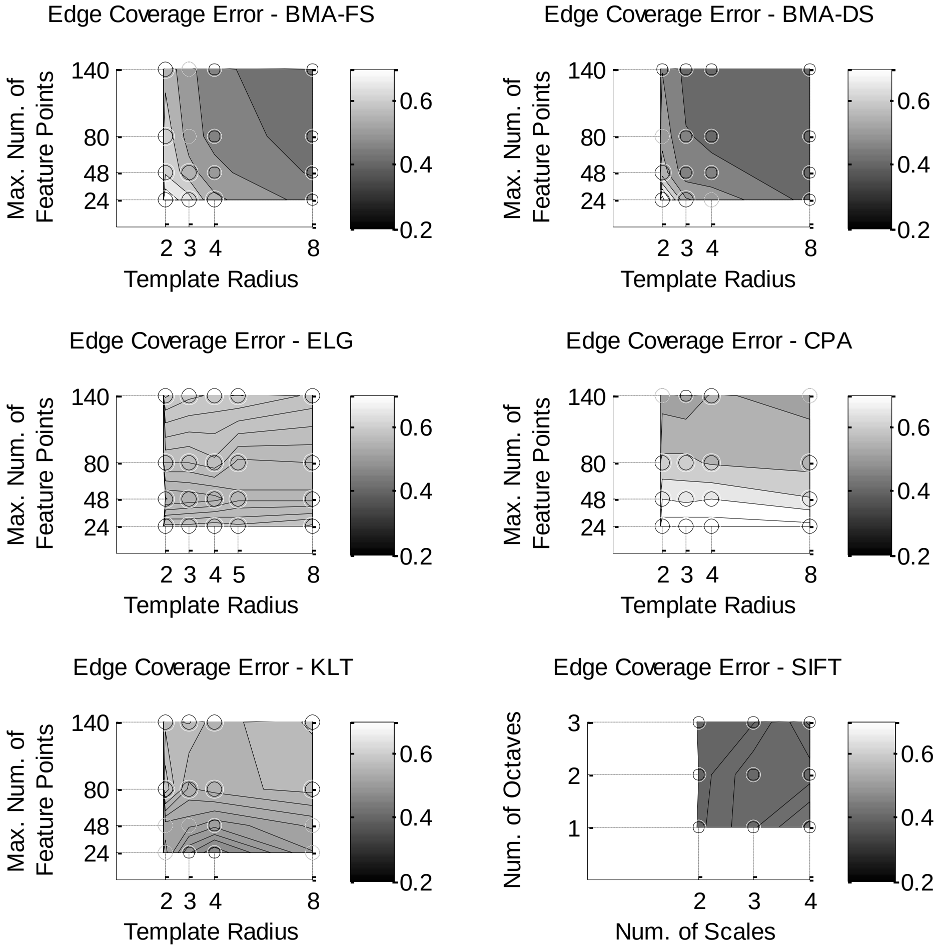


Table 3. Registration error evaluation for the Artificial Sequence

General considerations

Registration results are presented for the algorithms (Block Matching-Diamond Search, Block Matching-Full Search, Elastic Grid, Kanade-Lucas Tracker, Scale Invariant Feature Transform, Corner Pairing Algorithm). The Edge Coverage Error describes the registration error in the high spatial frequency domain. This metrics is more sensitive than the widely used L1 or L2 norm. Values are given as function of the two major complexity influencing parameters. For the first five algorithms these parameters are (i) maximum number of feature points/foveal windows (y axis) and (ii) radius of the feature matching template (x axis).

The SIFT algorithm processes the input image in many scales generated by Gaussian filters. Applying a filter with $\sigma = 2.0$ reduces the effective image resolution degenerate by a factor of two. This step in scale-space is called an octave. After that the image can be downsampled with factor of 2. The computational complexity of the SIFT algorithm is dominantly parameterized by the (i) number of octaves in use (Num. octaves), and the intermediate scales calculated for each octave (Num. scales).

Comparative analysis

-Block matching algorithms use autocorrelation prefiltering to skip flat matching templates. Full search achieves better results than adaptive diamond search, since it evaluates all possible displacement vectors in the given range. With a larger support region, the matching and prefiltering performance increases. Since displacements are estimated using integer precision, larger number of point pairs gives additional transformation estimation precision.

-ELG calculates estimates for all templates along its grid even for flat regions, although model based regularization compensates for missing information. The diagram of ELG shows that r=2 is enough using the robustness of joined search of all foveas. It is necessary to use more foveas since displacements are estimated using integer precision.

-Since CPA extracts corners, r=2 is reliable for matching (templates have extreme variance). Since it uses direct pairing, and not search, it must extract a large number of individual features to have the pair of each from the previous frame.

-KLT finds a reliable pair to almost any starting feature, since it locates reliable locations first. To estimate the transformation minimally, 8 reliable pairs are needed. Since KLT gives subpixel results, having 48 initial features is robust. KLT uses more octaves for searching. Due to downscaling, it is beneficial to have a larger support radius.

-SIFT: frame-to-frame zooming is small intermediate scales gives a small benefit to robust matching. For this sequence enough feature pairs can be found in the first octave.

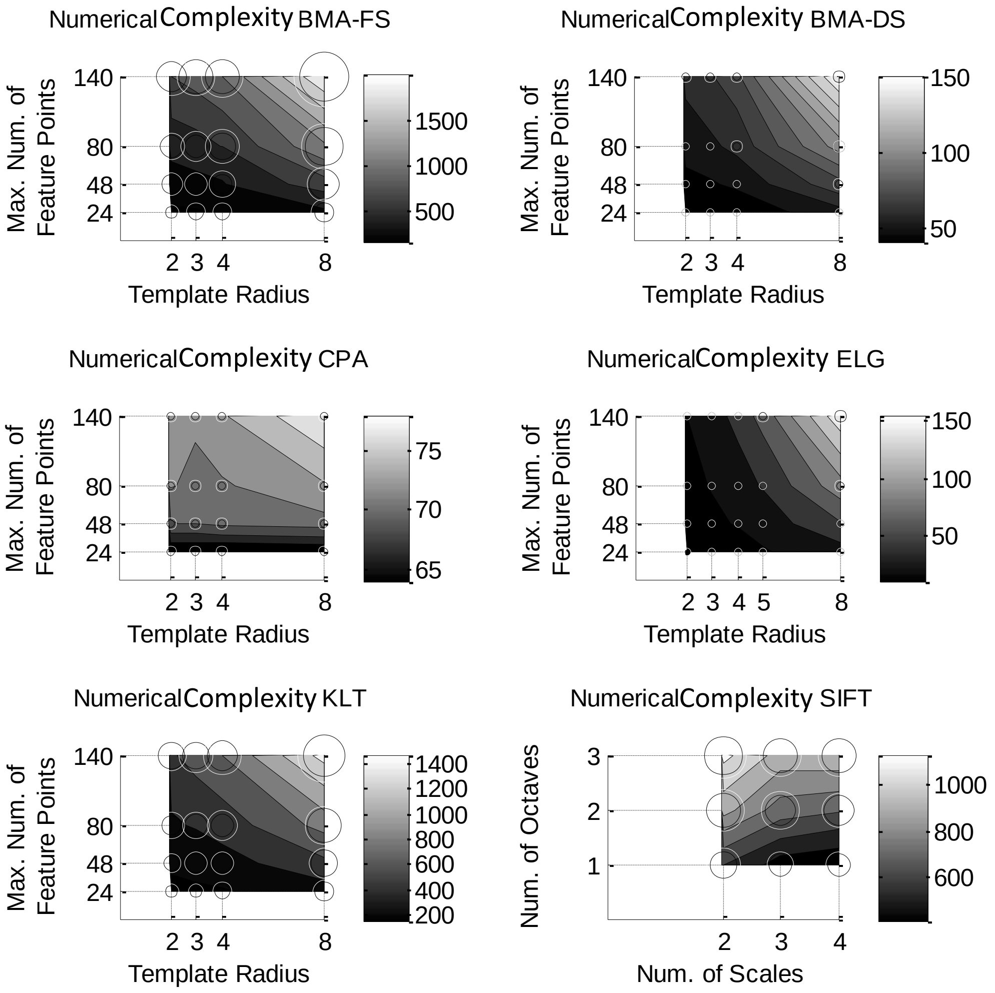


Table 4. Numerical complexity evaluation for the Artificial Sequence

General considerations

Registration results are presented for the algorithms (Block Matching-Diamond Search, Block Matching-Full Search, Elastic Grid, Kanade-Lucas Tracker, Scale Invariant Feature Transform, Corner Pairing Algorithm). The Edge Coverage Error describes the registration error in the high spatial frequency domain. This metrics is more sensitive than the widely used L1 or L2 norms. Values are given as function of the two major complexity influencing parameters. For the first five algorithms these parameters are (i) maximum number of feature points/foveal windows (y axis) and (ii) radius of the feature matching template (x axis).

The SIFT algorithm processes the input image in many scales generated by Gaussian filters. Applying a filter with $\sigma = 2.0$ reduces the effective image resolution degenerate to the half. This step in scale-space is called an octave. After that the image can be downsampled with factor of 2. The computational complexity of the SIFT algorithm is dominantly parameterized by the (i) number of octaves in use (Num. octaves), and the intermediate scales calculated for each octave (Num. scales).

Comparative analysis

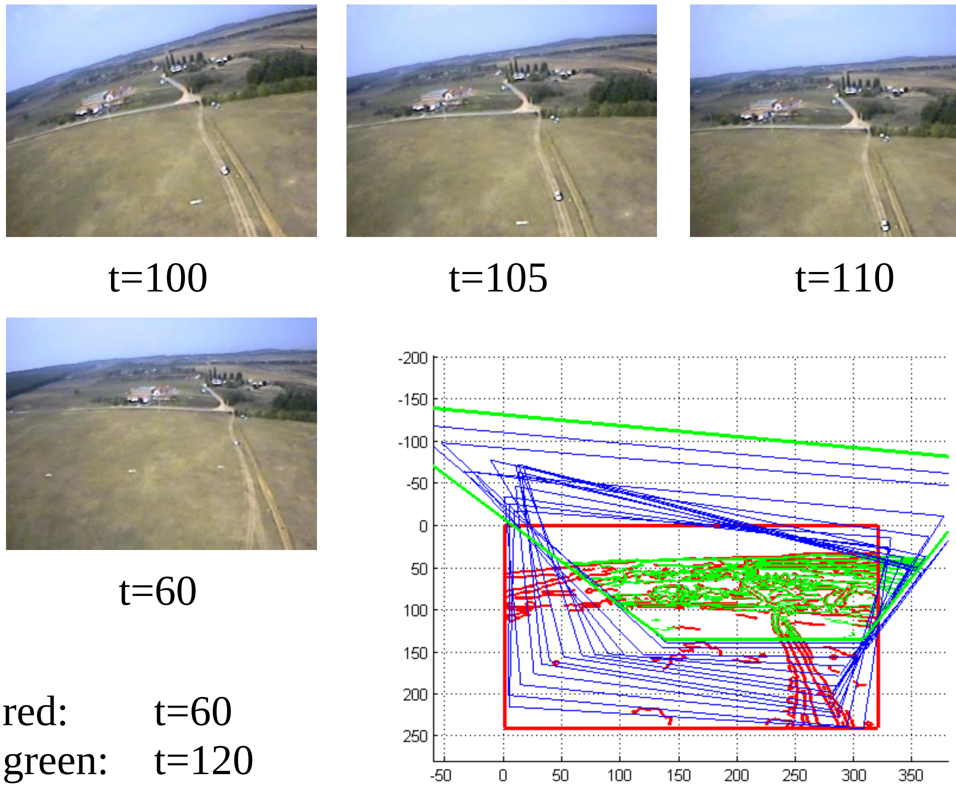
Both SIFT and KLT pay complexity to localize reliable features in full-size image processing. They both do sophisticated calculation for matching in foveas. SIFT also needs serial computation. For this sequence, there is an extremely large number of fine features due to sharp textures without motion blur effect. Those are traceable for KLT; however, SIFT is tuned for slightly larger features to deal with real videos. Matching complexity for SIFT is less in this case but KLT performs better.

CPA uses KLT-like corner localization placed in the frontend, and needs primitive foveal processing but a large number of windows.

The ELG and BMA methods do not need a frontend processor. Since Full Search can be implemented without any adaptation, it needs the most primitive hardware structure. Diamond Search uses only local adaptation, and can be realized with small number of stateless foveal processors with context switching. ELG offers good registration results with moderate complexity in the case of enough foveal processors arranged in 2D topology.

4.6.2. Godollo_1 Sequence

This sequence was captured above Gödöllő (120 frames). The airplane was flying above a dirty road on which a car was approaching. The ground of the airport is roughly flat. A crossing concrete road with a line of trees limits the meadow. Beyond the road there are some buildings as well. The path of the airplane was almost parallel to the ground (at roughly 70m altitude), thus a part of the sky is also visible in all frames. The size of the car varies between 30 and 110 pixels with a mean of 36. Representative frames are displayed in Table 5



t=100 t=105 t=110

t=60

red: t=60
green: t=120

Table 5. Representative frames of Godollo_1 Sequence
In the figures reference transformations are used to show the views in a few typical frames. The edge images of the first (red) and last frames (green) and boundaries of intermediate ones (blue) are displayed. During the flight the airplane travels a long path resulting in a large change inside the fore-scene and zoom to the previously back-scene parts. Two montages are needed to show the full path. The large distortion of the rectangular frame highlights the need for a perspective motion model.

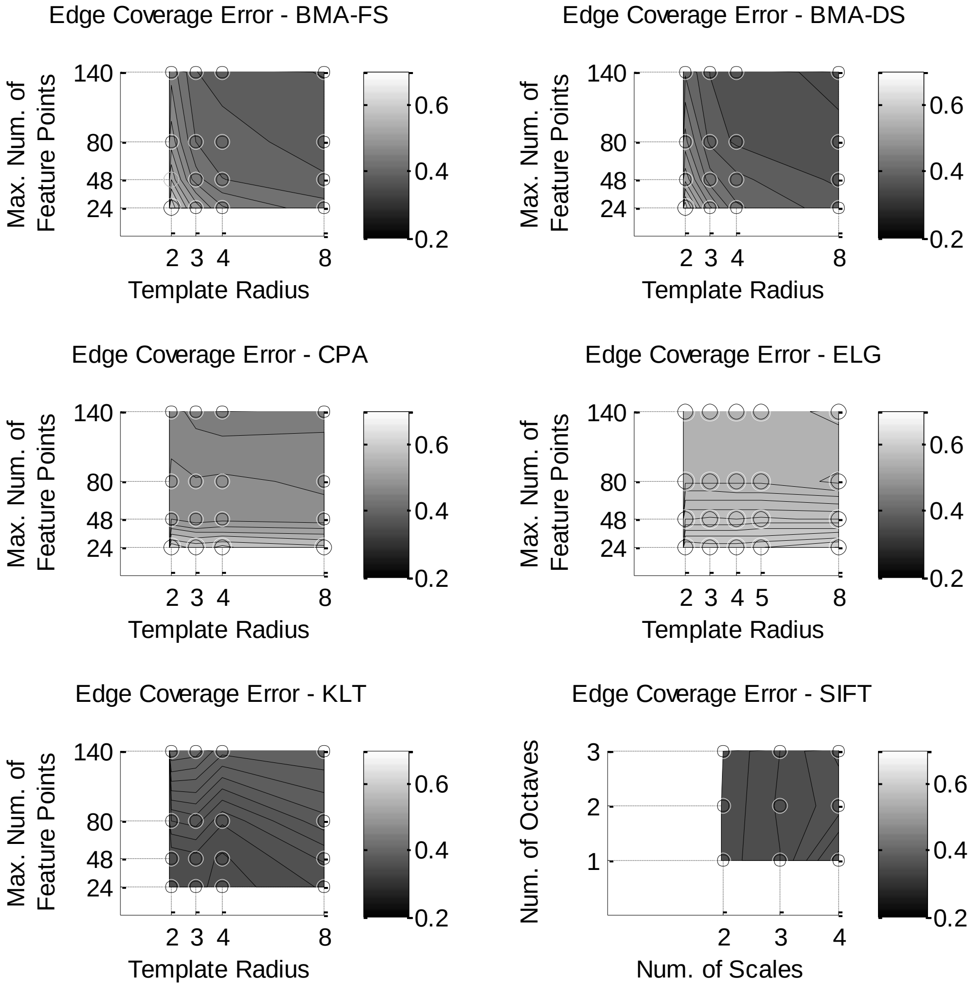


Table 6. Registration error evaluation for the Godollo_1 Sequence

General considerations

Registration results are presented for the algorithms (Block Matching-Diamond Search, Block Matching-Full Search, Elastic Grid, Kanade-Lucas Tracker, Scale Invariant Feature Transform, Corner Pairing Algorithm). The Edge Coverage Error describes the registration error in the high spatial frequency domain. This metrics is more sensitive than the widely used L1 or L2 norms. Values are given as function of the two major complexity influencing parameters. For the first five algorithms these parameters are (i) maximum number of feature points/foveal windows (y axis) and (ii) radius of the feature matching template (x axis).

The SIFT algorithm processes the input image in many scales generated by Gaussian filters. Applying a filter with $\sigma = 2.0$ reduces the effective image resolution degenerate to the half. This step in scale-space is called an octave. After that the image can be downsampled with factor of 2. The computational complexity of the SIFT algorithm is dominantly parameterized by the (i) number of octaves in use (Num. octaves), and the intermediate scales calculated for each octave (Num. scales).

Comparative analysis

Best competitors regarding quality: both SIFT and KLT pay complexity to localize reliable features in full-size image processing. They both do sophisticated calculation for matching in foveas. SIFT also needs serial computation.

CPA uses KLT-like corner localization placed in the frontend, and needs primitive foveal processing but a large number of windows.

ELG and BMA methods do not need a frontend processor. Since Full Search can be implemented without any adaptation it needs the most primitive hardware structure. Diamond Search uses only local adaptation, and can be realized with small number of stateless foveal processors with context switching. ELG offers good registration results with moderate complexity in the case of enough foveal processors arranged in 2D topology.

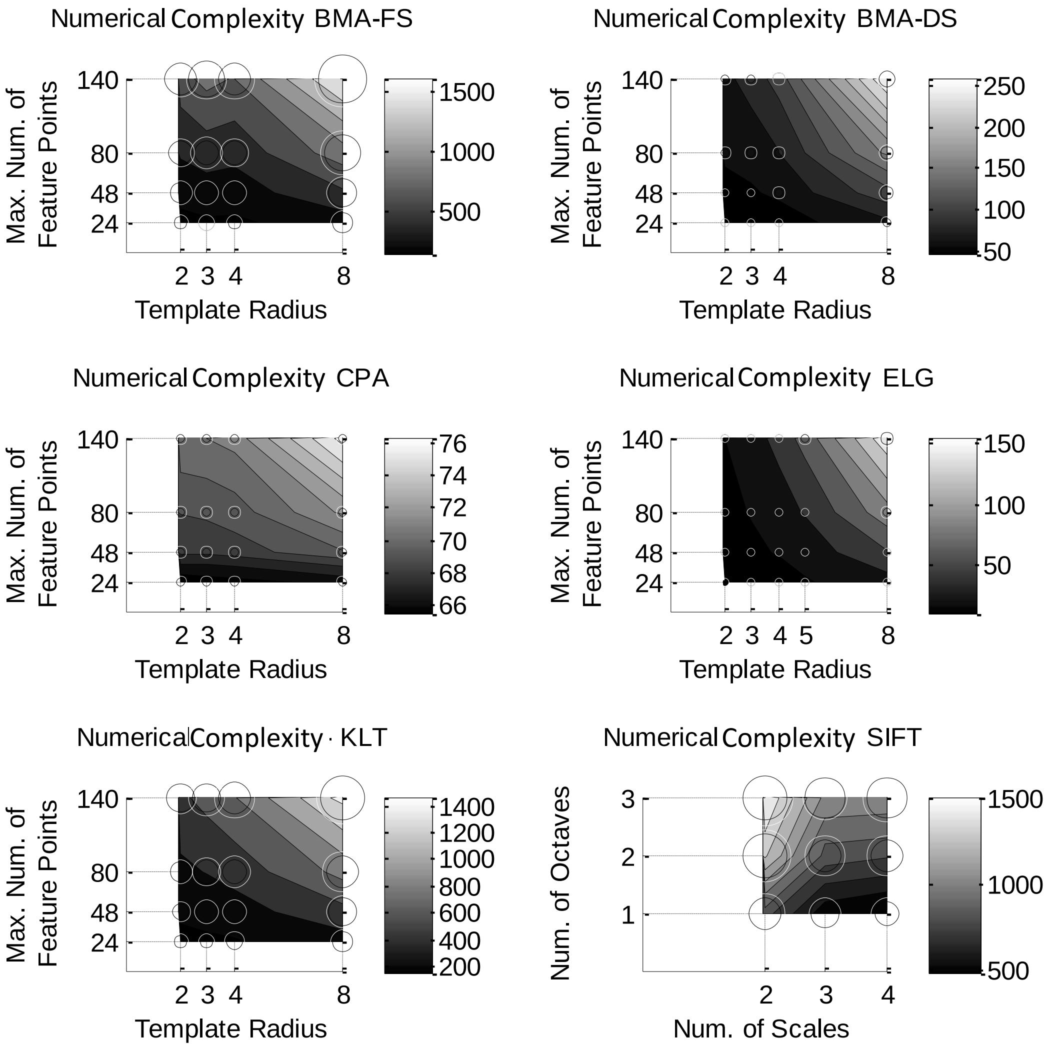


Table 7. Numerical complexity evaluation for the Godollo_1 Sequence

General considerations

Registration results are presented for the algorithms (Block Matching-Diamond Search, Block Matching-Full Search, Elastic Grid, Kanade-Lucas Tracker, Scale Invariant Feature Transform, Corner Pairing Algorithm). The Edge Coverage Error describes the registration error in the high spatial frequency domain. This metrics is more sensitive than the widely used L1 or L2 norm. Values are given an function of the two major complexity influencing parameters. For the first five algorithms these parameters are (i) maximum number of feature points/foveal windows (y axis) and (ii) radius of the feature matching template (x axis).

The SIFT algorithm processes the input image in many scales generated by Gaussian filters. Applying a filter with $\sigma = 2.0$ reduces the effective image resolution degenerate to the half. This step in scale-space is called an octave. After that the image can be downsampled with factor of 2. The computational complexity of the SIFT algorithm is dominantly parameterized by the (i) number of octaves in use (Num. octaves), and the intermediate scales calculated for each octave (Num. scales).

Comparative analysis

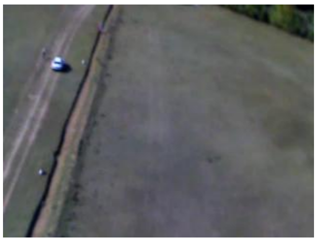
Both SIFT and KLT pays complexity to localize reliable features in full-size image processing. They both do sophisticated calculation for matching in foveas. SIFT also needs serial computation.

CPA uses KLT-like corner localization placed in the frontend, and needs primitive foveal processing but a large number of windows.

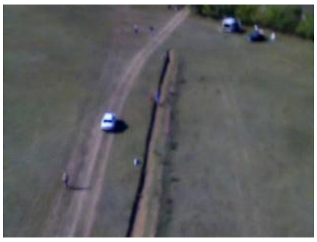
The ELG and BMA methods do not need a frontend processor. Since Full Search can be implemented without any adaptation, it needs the most primitive hardware structure. Diamond Search uses only local adaptation, and can be realized with small number of stateless foveal processors with context switching. ELG offers good registration results with moderate complexity in the case of enough foveal processors arranged in 2D topology.

4.6.3. Godollo_2 Sequence

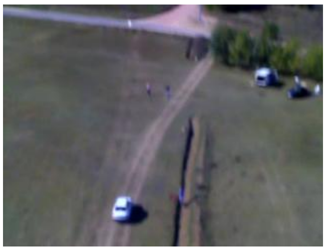
The Godollo_2 Sequence (See Table 8) was also captured above Gödöllő (300 frames) on a different day with a different color balance setup. The airplane was flying above the same dirty road on a similar path as in the Godollo_1 Sequence. There were some people walking on the ground and a car was moving on the road with parallel direction to the airplane. In the first 100 frames the altitude of the UAV is largely decreasing; after this time, the path of the airplane is almost horizontal to frame 200 when it starts climbing. The size of the car varies between 68 and 209 pixels with a mean of 185.



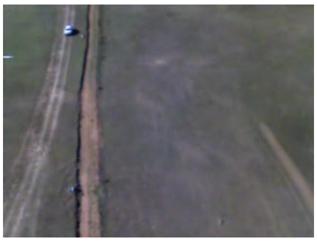
t=200



t=230

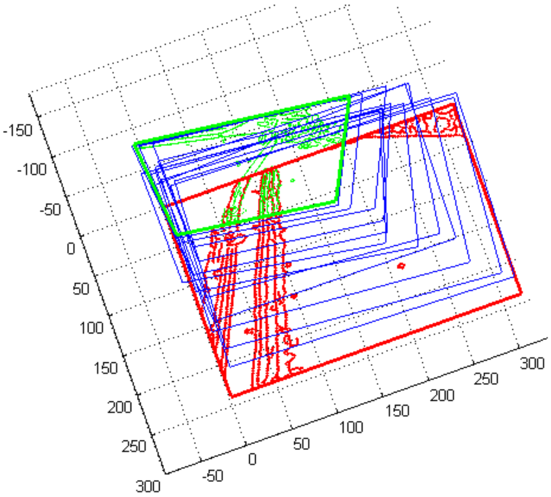


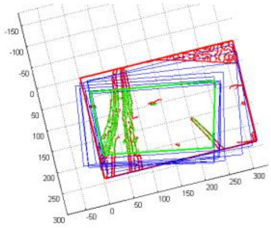
t=250



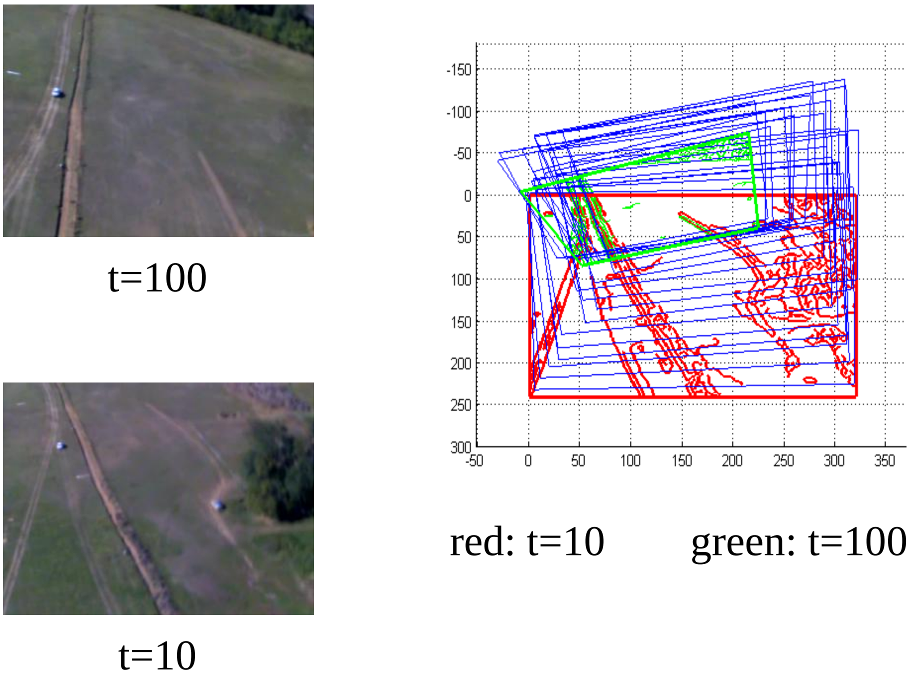
t=140

red: t=200 green: t=250





red: t=140
green: t=100



t=100

t=10

red: t=10 green: t=100

Table 8. Representative frames of Godollo_2 Sequence

In the figures reference transformations are used to show the views in a few typical frames. The edge images of the first (red) and last frames (green) and boundaries of intermediate ones (blue) are displayed. During the flight the airplane travels a long path resulting in a large change inside the fore-scene and zoom to the previously back-scene parts. Three montages are needed to show the full path. The large distortion of the rectangular frame highlights the need for a perspective motion model.

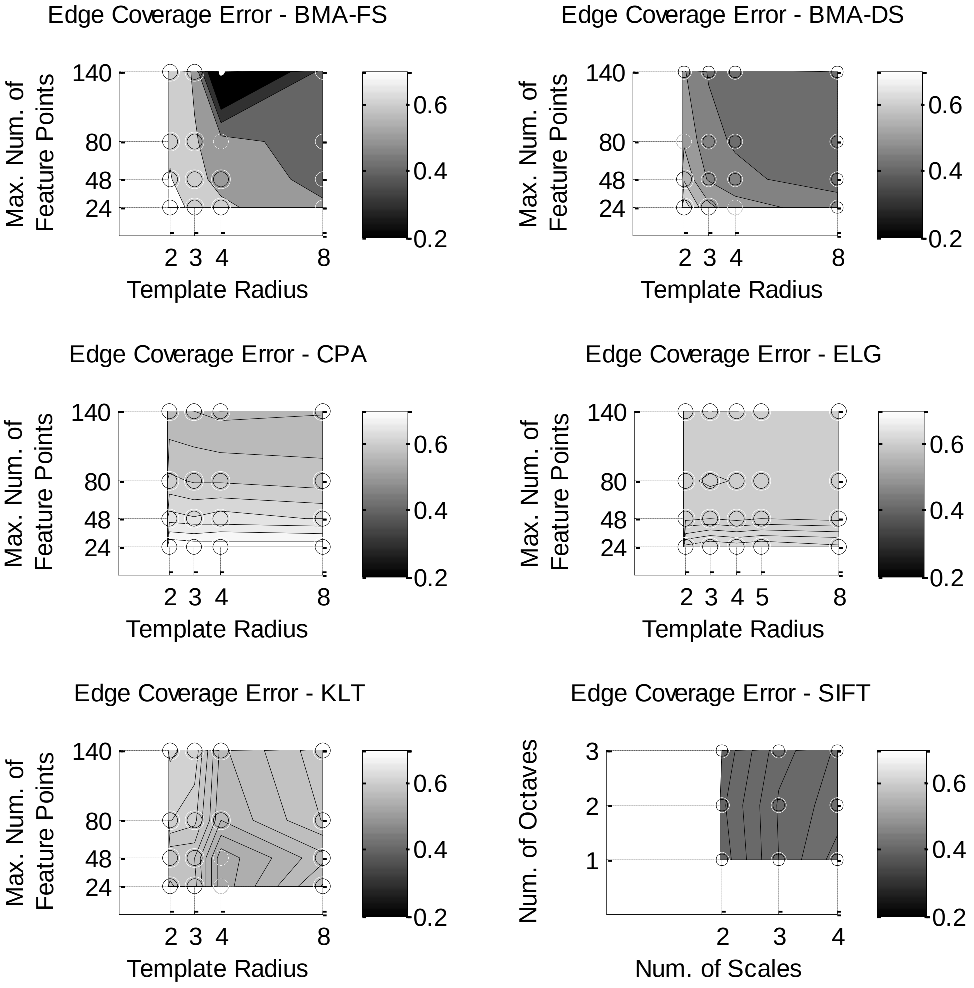


Table 9. Registration error evaluation for the Godollo_2 Sequence

General considerations

Registration results are presented for the algorithms (Block Matching-Diamond Search, Block Matching-Full Search, Elastic Grid, Kanade-Lucas Tracker, Scale Invariant Feature Transform, Corner Pairing Algorithm). The Edge Coverage Error describes the registration error in the high spatial frequency domain. This metrics is more sensitive than the widely used L1 or L2 norms. Values are given as function of the two major complexity influencing parameters. For the first five algorithms these parameters are (i) maximum number of feature points/foveal windows (y axis) and (ii) radius of the feature matching template (x axis).

The SIFT algorithm processes the input image in many scales generated by Gaussian filters. Applying a filter with $\sigma = 2.0$ reduces the effective image resolution degenerate to the half. This step in scale-space is called an octave. After that the image can be downsampled with factor of 2. The computational complexity of the SIFT algorithm is dominantly parameterized by the (i) number of octaves in use (Num. octaves), and the intermediate scales calculated for each octave (Num. scales).

Comparative analysis

This sequence spans along low and high altitudes that can be considered during operation. SHIFT can take advantage of its capability to find features in more scales. It performs the best on average.

Both SIFT and KLT pay complexity to localize reliable features in full-size image processing. They both do sophisticated calculation for matching in foveas. SIFT also needs serial computation.

CPA uses KLT-like corner localization placed in the frontend, and needs primitive foveal processing but a large number of windows.

ELG and BMA methods do not need a frontend processor. Since Full Search can be implemented without any adaptation it needs the most primitive hardware structure. Diamond Search uses only local adaptation, and can be realized with small number of stateless foveal processors with context switching. ELG offers good registration results with moderate complexity in the case of enough foveal processors arranged in 2D topology.

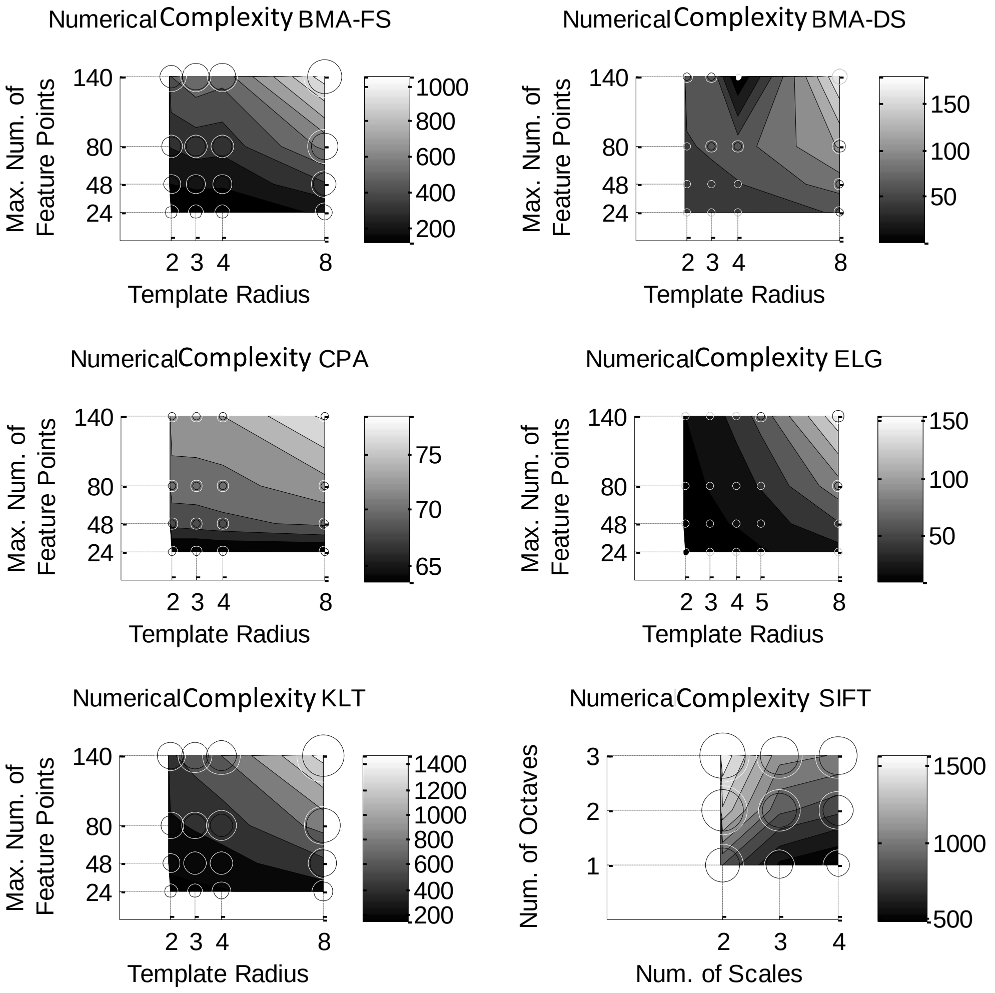


Table 10. Numerical complexity evaluation for the Godollo_2 Sequence

General considerations

Registration results are presented for the algorithms (Block Matching-Diamond Search, Block Matching-Full Search, Elastic Grid, Kanade-Lucas Tracker, Scale Invariant Feature Transform, Corner Pairing Algorithm). The Edge Coverage Error describes the registration error in the high spatial frequency domain. This metrics is more sensitive than the widely used L1 or L2 norms. Values are given as function of the two major complexity influencing parameters. For the first five algorithms these parameters are (i) maximum number of feature points/foveal windows (y axis) and (ii) radius of the feature matching template (x axis).

The SIFT algorithm processes the input image in many scales generated by Gaussian filters. Applying a filter with $\sigma = 2.0$ reduces the effective image resolution degenerate to the half. This step in scale-space is called an octave. After that the image can be downsampled with factor of 2. The computational complexity for SIFT algorithm is dominantly parameterized by the (i) number of octaves in use (Num. octaves), and the intermediate scales calculated for each octave (Num. scales).

Comparative analysis

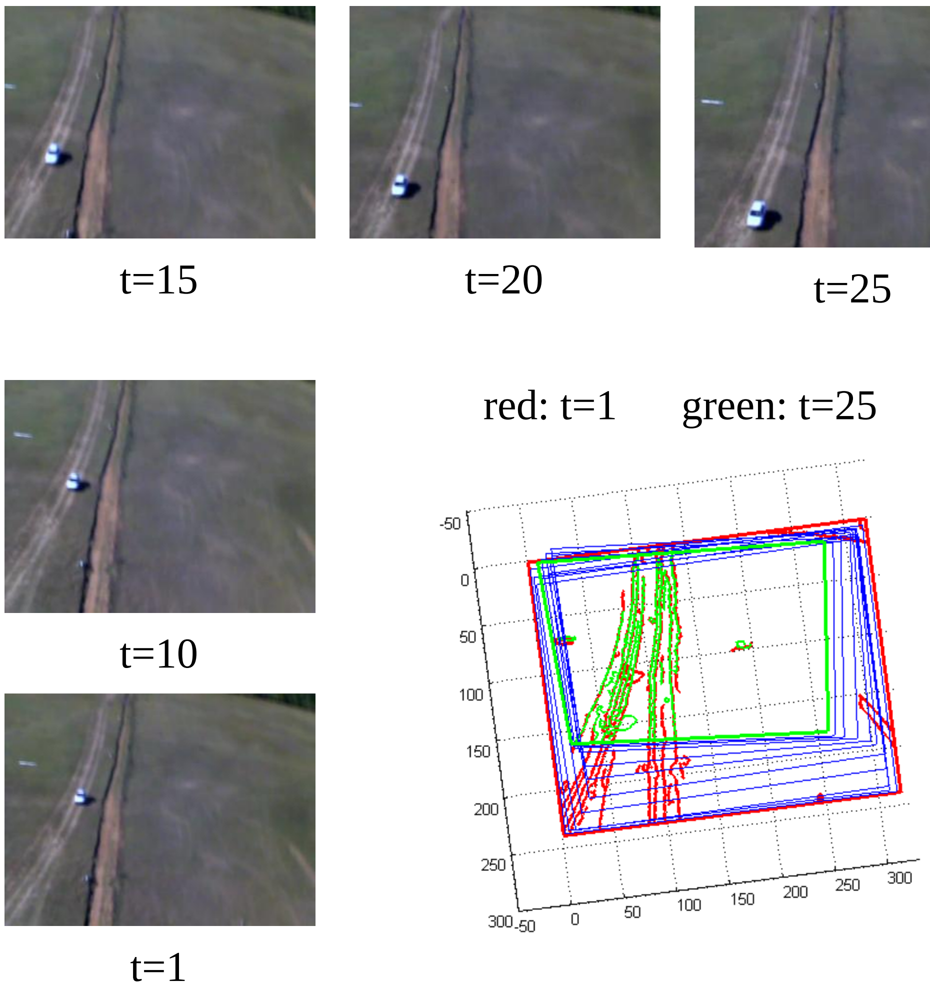
Both SIFT and KLT pays complexity to localize reliable features in full-size image processing. They both do sophisticated calculation for matching in foveas. SIFT also needs serial computation.

CPA uses KLT-like corner localization placed in the frontend, and needs primitive foveal processing but a large number of windows.

The ELG and BMA methods do not need a frontend processor. Since Full Search can be implemented without any adaptation it needs the most primitive hardware structure. Diamond Search uses only local adaptation, and can be realized with small number of stateless foveal processors with context switching. ELG offers good registration results with moderate complexity in the case of enough foveal processors arranged in 2D topology.

4.6.4. Godollo_3 Sequence

The Godollo_3 Sequence (See Table 11) was originated from the same flight as the previous one. In this sequence the car was approaching towards the airplane. This is a 35 frame long video. The size of the car varies between 280 and 960 pixels with a mean of 526.


t=15

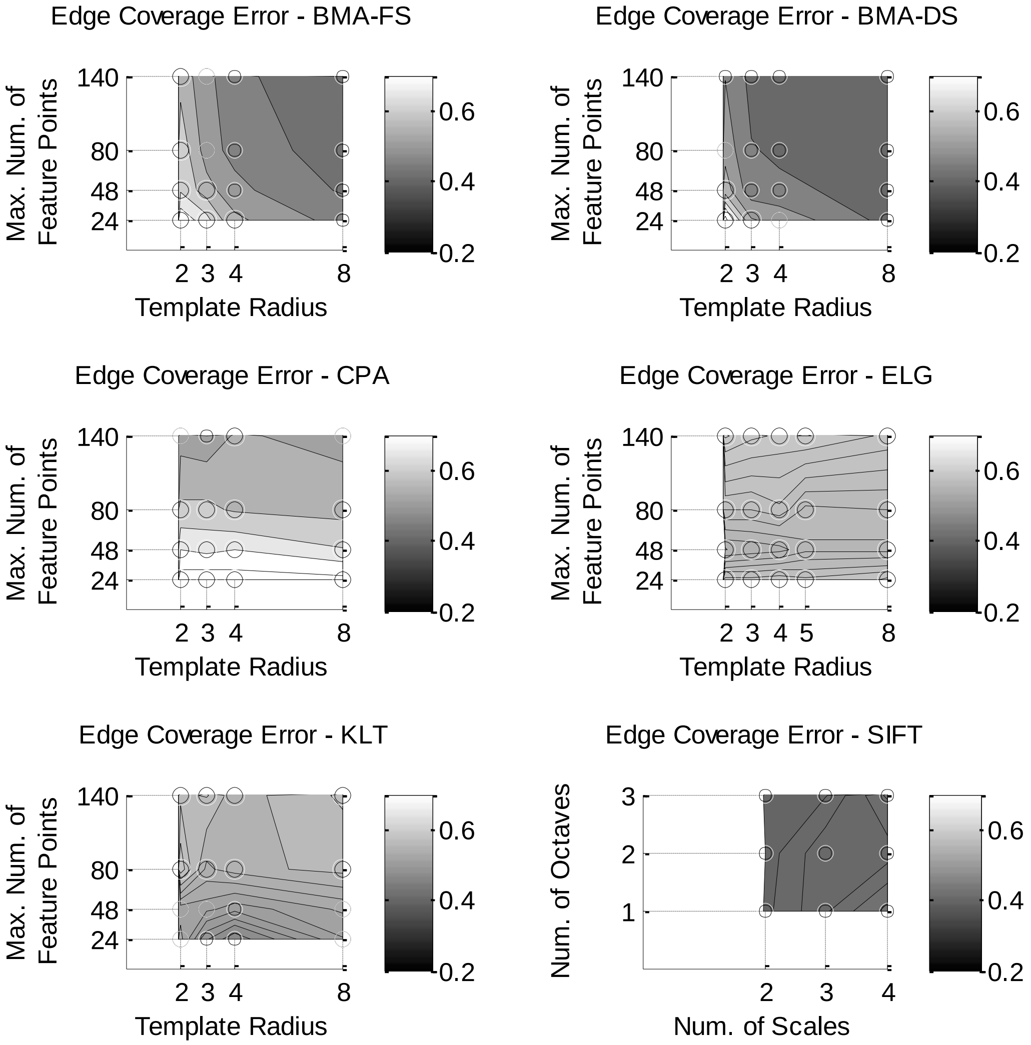
t=20

t=25

t=10

t=1

Table 11. Representative frames of Godollo_3 Sequence
This video sequence is captured from the lowest altitude. The feature blobs are large, and corner-like features are mostly around the road structures. This uneven distribution makes model fitting a hard task. On the other hand, the car object covers a large number of pixels, thus it can be detected with slightly worse alignment and morphological post-processing.



Edge Coverage Error - BMA-FS

Edge Coverage Error - BMA-DS

Edge Coverage Error - CPA

Edge Coverage Error - ELG

Edge Coverage Error - KLT

Edge Coverage Error - SIFT

Table 12. Registration error evaluation for the Godollo_3 Sequence

General considerations

Registration results are presented for the algorithms (Block Matching-Diamond Search, Block Matching-Full Search, Elastic Grid, Kanade-Lucas Tracker, Scale Invariant Feature Transform, Corner Pairing Algorithm). The Edge Coverage Error describes the registration error in the high spatial frequency domain. This metrics is more sensitive than the widely used L1 or L2 norms. Values are given as function of the two major complexity influencing parameters. For the first five algorithms these parameters are (i) maximum number of feature points/foveal windows (y axis) and (ii) radius of the feature matching template (x axis).

The SIFT algorithm processes the input image in many scales generated by Gaussian filters. Applying a filter with $\sigma = 2.0$ reduces the effective image resolution degenerate to the half. This step in scale-space is called an octave. After that the image can be downsampled with factor of 2. The computational complexity of the SIFT algorithm is dominantly parameterized by the (i) number of octaves in use (Num. octaves), and the intermediate scales calculated for each octave (Num. scales).

Comparative analysis

This video sequence is captured from the lowest altitude. The feature blobs are large, and corner-like features are mostly around the road structures. This uneven distribution makes model fitting a hard task.

Both SIFT and KLT pays complexity to localize reliable features in full-size image processing. They both do sophisticated calculation for matching in foveas. SIFT also needs serial computation.

CPA uses KLT-like corner localization placed in the frontend, and needs primitive foveal processing but a large number of windows.

ELG and BMA methods do not need a frontend processor. Since Full Search can be implemented without any adaptation it needs the most primitive hardware structure. Diamond Search uses only local adaptation, and can be realized with small number of stateless foveal processors with context switching. ELG offers good registration results with moderate complexity in the case of enough foveal processors arranged in 2D topology.

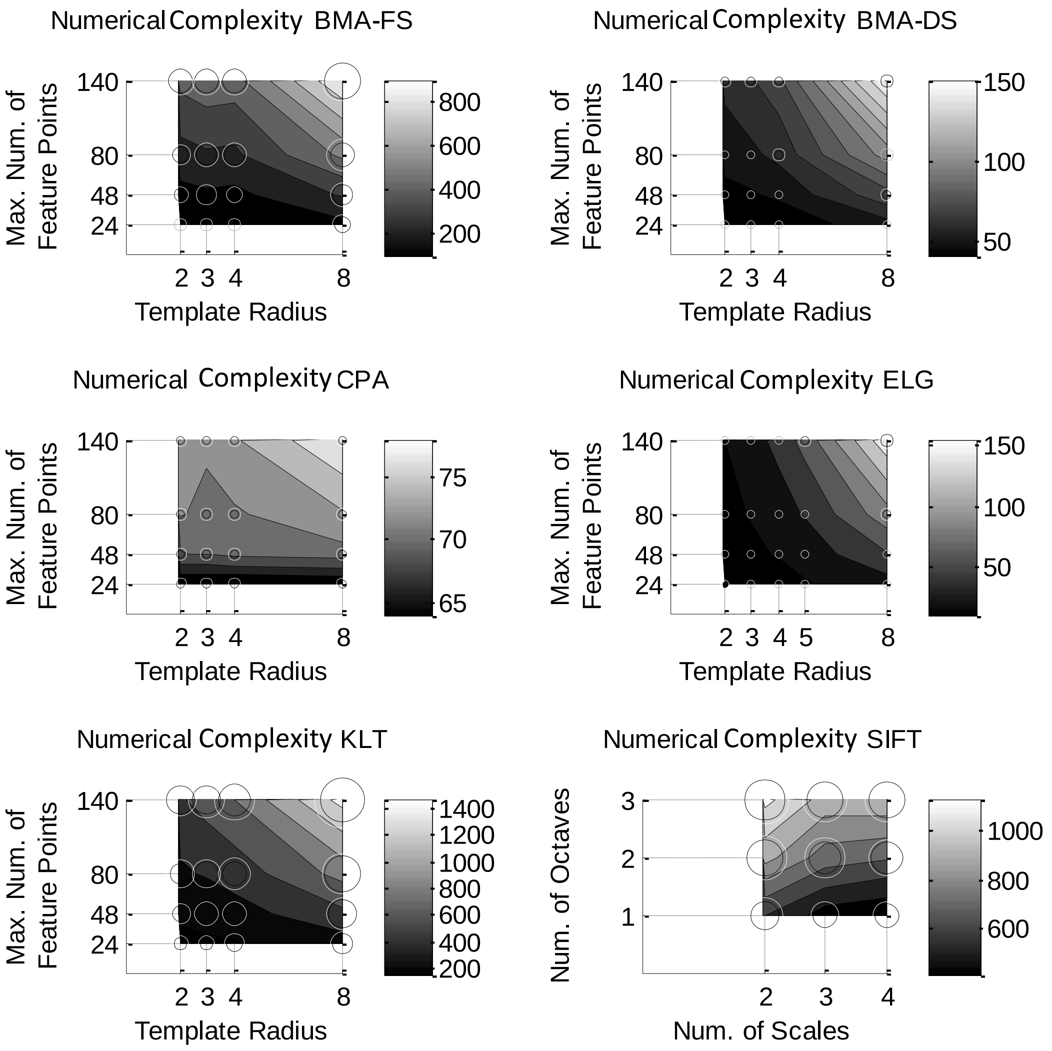


Table 13. Numerical complexity evaluation for the Godollo_3 Sequence

General considerations

Registration results are presented for the algorithms (Block Matching-Diamond Search, Block Matching-Full Search, Elastic Grid, Kanade-Lucas Tracker, Scale Invariant Feature Transform, Corner Pairing Algorithm). The Edge Coverage Error describes the registration error in the high spatial frequency domain. This metrics is more sensitive than the widely used L1 or L2 norms. Values are given as function of the two major complexity influencing parameters. For the first five algorithms these parameters are (i) maximum number of feature points/foveal windows (y axis) and (ii) radius of the feature matching template (x axis).

The SIFT algorithm processes the input image in many scales generated by Gaussian filters. Applying a filter with $\sigma = 2.0$ reduces the effective image resolution degenerate to the half. This step in scale-space is called an octave. After that the image can be downsampled with factor of 2. The computational complexity for SIFT algorithm is dominantly parameterized by the (i) number of octaves in use (Num. octaves), and the intermediate scales calculated for each octave (Num. scales).

Comparative analysis

Both SIFT and KLT pays complexity to localize reliable features in full-size image processing. They both do sophisticated calculation for matching in foveas. SIFT also needs serial computation. For this sequence, there are more extremely large features that are out of the scope of SIFT, so it tracks less points than in average.

CPA uses KLT-like corner localization placed in the frontend, and needs primitive foveal processing but large number of windows.

The ELG and BMA methods do not need a frontend processor. Since Full Search can be implemented without any adaptation it needs the most primitive hardware structure. Diamond Search uses only local adaptation, and can be realized with small number of stateless foveal processors with context switching. ELG offers good registration results with moderate complexity in the case of enough foveal processors arranged in 2D topology.

4.7. Conclusion

In this Chapter the description power of the virtual framework was presented through modeling, partitioning, and implementing five algorithms in the field of moving platform surveillance.

Among other important findings the most important results are the following for average flight paths:

- using the best result from all algorithms for each consecutive image pairs, a global mosaic can be created. Using this as reference, the flat 2D model was validated;
- the Gaussian filter based feature extraction gives enough feature points for matching even with limited effort;
- it is not important to maintain rotation invariance at local motion estimation level to achieve good registration results;
- the fine sub-pixel based matching part of the KLT algorithm is reliable even in case of a small number of point pairs;
- Block Matching algorithms offer an efficient alternative if the Frontend Processor Array is not supported by the hardware.

As conclusion the following thesis points can be stated:

Thesis II.: I have shown that the proposed virtual architecture can uniformly cover important problems in sensory image processing. As a proof of concept, 2D registration based ground motion detection methods for mini Unmanned Aerial Vehicles (UAV) were presented. I have given models for the state-of-the-art solutions by creating the data-flow diagrams, I have mapped the operators to processing structures and I have compared the algorithms in a unified simulation environment. I have given performance analysis concerning the computational complexity, registration quality, detection robustness, and parameter sensibility of the algorithms. Based on the detailed analysis of the algorithms I have proposed modifications to achieve better complexities, which I summarize in two sub-theses.

A: I have shown that up-scaling is unnecessary for scale-space based point feature detection in the sequences captured by a mini UAV flying at medium altitude. This means that processing does not need to be more fine-grained than the original sen-

sor resolution. Furthermore, I have experimentally validated that the visible zoom factor due to altitude changes between consecutive frames is small (less than one percent) and as a result, evaluating more than two inter-octave scales does not give any benefit in alignment capability.

B: I have shown that the KLT and SIFT algorithms can be merged effectively, in case hardware support for diffusion and extrema localizing operator working in a 3x3 neighborhood exist in the cellular processor array.

Chapter Five

ELASTIC GRID MULTI-FOVEA DETECTOR

5.1. Elastic Grid Method

A novel algorithm (Elastic Grid Multi-Fovea Detector, or shortly Elastic Grid Method) was proposed to utilize the advantages of the Virtual Multi-Fovea Architecture. This algorithm relies on topologically connected foveal processors to create a “locally interacting” motion map of the observed field. It was experimentally shown that the multiple displacement motion model used is appropriate for detecting objects moving on the ground from a mini UAV. The proposed algorithm was compared with state-of-the art methods highlighting its good output quality and moderate computational complexity.

Thesis III: I have proposed a novel independent motion detection algorithm (Elastic Grid Multi-Fovea Detector), that exploits the parallel and cellular capabilities and communication model of the proposed architecture. I have experimentally shown that the model based connected multiple displacement method is adequate for the detection of ground objects moving on an open field from a mini Unmanned Air Vehicle (UAV). I have proven by using complex metrics that the proposed algorithm offers a better trade-off than previous solutions using the same scene assumptions.

Chapter Six

SUMMARY

6.1. Main Findings and Results

I have created a virtual hardware model (Virtual Multi-Fovea Architecture) to support comparison of image processing algorithms that can fit in the multi-fovea model. By defining this virtual layer, existing hardware platforms and programming methods may be interfaced. The platform consists of separate processor arrays specialized to parallel execution of preprocessing and foveal computation. They are synchronized by a conventional serial processor via a proposed intelligent memory management unit. This heterogeneous structure can fit the special characteristics of various operators processing the input video flow.

I have given a design guideline for the Virtual Multi-Fovea Architecture and presented it by the comparison of 2D registration methods for ground object motion detection from mini UAVs.

After giving an analytical comparison of registration methods I proposed a novel method exploring the proposed architecture by running a larger percentage of the tasks in parallel and in cellular structures.

6.2. New Scientific Results

Algorithms dealing with direct topographic sensory inputs may contain large percent of steps suitable for data-parallel execution due to the natural structure of the data. Based on this observation I have worked out a novel virtual hardware architecture model (the Virtual Multi-Fovea Architecture) enabling communication-effective decomposition of those family of image processing algorithms, that are convergent starting from direct sensory input and can be described by acyclic data flow graphs. The proposed structure effectively evaluates algorithms consisting of operators with different radius of coupling and topology, and nonhomogoneous spatial coverage by using three specific processor arrays. This heterogeneous structure fits the family of algorithms better than the general homogenous parallel structures without losing general programmability.

Published in [1][5]

Processors built to deal with integers or floating point numbers are calculating operators on a small set of inputs (two in most cases). They incorporate a small amount of dedicated memory – the so-called registers – for storing only a few numbers that can be used as operands. The great majority of the data is stored in an external memory chip. The programs consist of calculation and data transfer instructions. The evaluation of the code may branch depending on some results or it can run in iterations (loops).

The computation deals with neighboring data elements frequently; this property is called locality. Locality is also characteristic for instructions: commands of a subroutine can be aligned in memory resulting sequences and a minimal number of branching instructions. The off chip communication to load new data has massive latency – typically two or three order of magnitude higher compared to numerical calculation – thus caching into on-chip memory is essential to exploit locality.

The principal fraction of computer problems is formulated in serial form, although the intermediate results are not used immediately and the operand of a binary operator can be evaluated in an arbitrary order (instruction level parallelism). To permit the optimizing, the pre-fetching of instructions is necessary. Processors in modern PCs contain more execution lines; moreover, some independent instructions may be reordered and processed while waiting for memory access (out-of-order execution). In the case of handling branches in the program, all paths should be analyzed or some heuristic method is needed to make a decision. The sweep of vector instruction sets and wide registers for handling structured data have further increased the complexity of the already extremely complex processor structure.

Handling the extreme workload of numerous problems on a single processor is not fast enough to be beneficial at all. Nobody would wait hours for the answer to a simple train-schedule query. For these cases it is important to identify tasks of the problem that can be executed in parallel without altering the final result, and distribute them to multiple processors.

All devices should be supplied with task and data and in addition, a communication network should be built to connect them for enabling exchange of partial results and synchronization. Parallel computing raises new challenges: multiple accesses to common resources (most important ones are the memory and the communication network) should be granted while maintaining (cache) coherency. The further increase of the operation frequency is limited by physical constraints. On the other hand, the placement of many cores on a chip is feasible, thus exploiting parallelism efficiently is essential.

Imperative programming languages handle large data stacks by processing elements in loops. The multimedia inputs offer a great level of data parallelism. Each element should be handled the same way; in most cases, calculations depend only on neighboring structures. These functions are called cellular operators. Loops implementing such operators may be evaluated using many execution contexts. Due to the known locality of the given operators, the large amount of automatically handled cache memory can be replaced with a small amount of local memory circuits dedicated to a given arithmetic unit loaded with direct instructions. The most extensively used imperative language is C/C++. OpenMP [75] is an extension for shared memory multi-threading that can create parallel code for loops to be executed on multi-core processors using the explicit locality constraints given by the programmer.

Parallel data container structures without constraints for traversal but having explicit notion of locality offer great potential for compile-time optimization (RapidMind [76], Intel Threading Building Blocks [77]). In the case of serial description of a program, the dependencies between operators should be scouted by the compiler to enable parallelization. It is worth to ask the programmer to explicitly express producer-consumer locality by drawing dataflow diagrams.

Custom application-specific integrated circuits (ASIC) that were directly designed to boost performance of specific tasks can exploit all parallelism in a given algorithm, in exchange for completely losing software programmability. For example, encoder chips for multimedia compression are common in handheld devices. Homogenous structures of general purpose processors offer a programmable and more scalable solution. However, it is worthwhile to design hybrid hardware structure to fit specific algorithmic classes.

I have designed a virtual heterogeneous many-core architecture (Figure 3.1), for image processing algorithms that are convergent, starting from direct sensory input and can be described by acyclic dataflow graphs. Convergence is referring to the extraction of compact information from inputs represented with topological maps with smaller resolution, image parts, or scalar values. This property calls for heterogeneous processor structures. In applications, where the high frame-rate is important (e.g. 10.000 fps), sensor pixels can be built in the processing topology to eliminate the need of wide and/or ultrafast cross chip communication circuits. If the program can be transformed into a representation containing iterations and recursions only at operator level an acyclic dataflow graph can be created. In this case, program execution can be mapped to many cooperative processors requiring clean and pre-calculated synchronizations. The virtual architecture hides the details of the physical hardware from the software engineering point of view, but models the scheduling and communication latencies.

The proposed virtual architecture contains a dedicated cellular processor array to take advantage of cellular locality and data parallelism which is characteristic of the family of cellular image processing operators [78]. This is designed to handle full images through space invariant operators without data-dependent branching, thus it may be controlled by a single instruction unit in SIMD fashion.

For the majority of image processing algorithms the final goal is to highlight specific regions (segmentation and classification), or to detect and possibly identify objects and/or events. This means that after some topological steps they focus on selected portions of the input image flow. The thorough analysis of chosen windows is supported by another dedicated processor array, the Foveal Processor Array. The arithmetic units (or small groups) in this array are controlled by independent instruction units to support data dependent branching. The communication and synchronization between the foveas is indirect. The generated results are either window sized topological data or descriptor vectors.

The non-parallelizable tasks and the evaluation of descriptor vectors are executed in a serial processor, the Backend Processor. It is also responsible for synchronizing the other arrays. The data transfer (sensor sized or scaled images, windows, and scalars) is supported by an intelligent multi-port Memory Management unit.

The Virtual Multi-Fovea Architecture is designed to hide a specific physical implementation. Execution time can be assigned to data transfers and global image processing operators, thus different algorithms for the same problem can be compared. The most important issues from the hardware implementation aspect are (i) used global operators (with a minimal instruction set for the core operators, explicit locality and topology description), (ii) the separation of program code segments with and without branching (iii) continuous data block definitions for data transfer.

One of the most powerful universal parallel technologies in PC environment is the General Purpose Graphic Processing Unit (GP-GPU) based Compute Unified Device Architecture (CUDA) from NVIDIA [79]. AMD-ATI also offers GP-GPU computing, although their software support is more focused on graphical applications at the present stage.

The fovea-based parallel array and the serial processor are common elements in both, thus the uniqueness of the latter is the dedicated cellular unit for evaluating 2D topologic operators that are working on image parts in a small connection radius. In the preprocessing stage most of the image processing algorithms need such operators (for example, convolution and image morphology) transforming the whole input image with each of them working on a neighborhood of a few pixels with full-grain connectivity, thus the parallel evolution requires local communication on a massive scale. I have shown that the GP-GPU array may implement the functions of the Frontend Preprocessor Array, thus this architecture may also covered by the virtual architecture.

I have shown that the proposed virtual architecture can uniformly cover important problems in sensory image processing. As a proof of concept 2D registration based ground motion detection methods for mini Unmanned Aerial Vehicles (UAV) were presented. I have given models for the state-of-the-art solutions by creating the data-flow diagrams, I have mapped the operators to processing structures and I have compared the algorithms in a unified simulation environment. I have given performance analysis concerning the computational complexity, registration quality, detection robustness, and parameter sensibility of the algorithms. Based on the detailed analysis of the algorithms I have proposed modifications to achieve better complexities, which I summarize in two sub-theses.

Published in: [4]

2D image registration techniques can be applied to data from Mini UAVs operating at the altitude of 80-100m flying over a flat inspection area to detect ground motion [53]. Consecutive projections of a flat screen captured from different locations can be aligned to a common coordinate frame. Changes may occur due to moving objects on the ground.

Salient points in the given image respecting a robust metric – for example *corner* at the intersection of edges – are called Feature points (or Point features). Point feature based alignment is one of the mainstream solutions for 2D image registration.

Besides the thoroughly discussed Harris Corner [67] (used by the Corner Pairing Algorithm, CPA) and Kanade-Lucas Tracker – KLT [64][65] the most cited point-feature extractor is the Scale Invariant Feature Transform – SIFT [63]. Video compression pushes for the improvement of block matching techniques [66] (BMA) as well.

Robust registration [68] and motion detection methods based on these representative groups of algorithms were covered in my assessment. Many surveys give comparative results for registration quality [62] [52], although none of them include hardware complexity factors in the evaluation metrics. The general flowchart of the diagram is given in Figure 4.2, and the intermediate results were presented in Figure 4.3.

I have evaluated the algorithms on synthetic and many real-world video sequences for ground object motion detection and presented the design tradeoffs. In case of large-field inspection the common flight path follows a rectangular shape with long straight edges and short rotating maneuvers. The sequences were taken from the straight portions of flight videos. The simulated sequence consists of sharp still images rendered from different calculated locations; on the contrary, the real shots are slightly distorted by the motion-blur effect and video compression artifacts. All sequences use 320x240 pixel resolution and were taken at 20 frames per second. The maximal measured displacement was 12 pixels in any direction. Higher sampling rate with high-sensitivity sensors would be desired to keep the average displacement in the 1-2 pixel range.

All algorithms used conceptually identical registration and detection steps.

The complexity of the CPA, BMA and KLT algorithms may be parameterized using two factors: (i) the number of foveal regions and (ii) the size of template images used for matching. The preprocessing phase is non-tunable. On the other hand, the SIFT algorithms can be described via parameters of preprocessing.

The detection robustness of the algorithms was compared after selecting the optimal parameters. Table 1 shows that their capabilities are similar even though they consume computational complexity in a quite different range. If the mission of the UAV is not only to detect the presence of moving object, but also to localize and identify them, the considerably high complexity of KLT and SIFT are justified since the results may be efficiently reused in further processing.

The registration capabilities for frame-to-frame alignment were compared in the high spatial frequency domain via the overlapping ratio of the binary edge images. In the case of high precision estimation, the consecutive transformation matrices may be accumulated and longer series of images can be transformed to a common coordinate frame. This advanced capability can be used to detect objects at relatively low motion speeds compared to the image sampling frequency.

The different algorithms are optimal for different hardware setups. SIFT is outstanding in quality, but it requires a very complex preprocessor and foveal arrays to be effective. On the other hand, the calculated description vectors may also be used for object identification. KLT involves fovea intensive calculations using branches and fractional number representation; although the preprocessing computation is less intensive compared to the one in the case of SIFT. Both methods are characterized with high registration quality.

Block Matching methods do not need a preprocessing array (although their performance could be enhanced by preprocessing). The different strategies of Diamond Search and Full Search can balance complexity and registration quality. The latter incorporates high number of searching steps (n) and finds global optima. It does not need branching, thus it can be implemented in less complex hardware; on the other hand, the first method converges to a local optimum rapidly ($\sqrt{n}$).

The Corner Pairing Algorithm offers lower registration quality; its small computation complexity, however, is remarkable for detection purposes.

Based on the detailed analysis of the algorithms, I have proposed specific modifications leading to significant improvements which will be summarized in the corresponding three sub-theses.

I.1. I have shown that up-scaling is unnecessary for scale-space based point feature detection in the sequences captured by a mini UAV flying in medium altitude. This means no processing is needed in finer grid than the original sensor resolution. Furthermore, I have experimentally validated that the visible zoom factor due to altitude changes between consecutive frames is small (less then one percent) and as a result, evaluating more than two inter-octave scale does not give any benefit in alignment capability.

The SIFT algorithm applies a series of Gaussian filters (low-pass characteristic in space, smoothing the image) and calculates differences between filtered images. These intermediate maps can be used to robustly localize blob-like features with different sizes in the original image. The Gaussian filter with σ=2 parameter gives an output with an effective resolution of half of the original, meaning an octave in scale-space. Lowe in [63] proposed to start scale-space generation from an interpolated double resolution image (up-scaling) and to calculate three intermediate scales for each octave.

Gaussian filtering can be effectively calculated in a dedicated full-grain cellular processor array [18]. Creating a double resolution array increases complexity at least by a factor of four, and using more intermediate scales implies additional components.

During the evaluation of registration quality I have experimentally shown that adequate feature pairing can be calculated in case of two intermediate scales, and that the number of feature point pairs is large enough to robustly estimate alignment transformation.

The simplified algorithm does not differ from the original regarding its average detection robustness.

I.2. I have shown the KLT and SIFT algorithms can be merged effectively, in case hardware support for diffusion and extrema localizing operator working in 3x3 neighborhood exist in the cellular processor array

The extrema localizing operator working in a 3x3x3 pixel neighborhood is needed for feature point localization in the case of the scale-space approach. This can be calculated using a subroutine in current hardware configurations while direct realization is also possible. Using the simplifications that were described in the previous section, the running time of feature point localization is reduced to the millisecond range.

The most important additional results for average flight paths are the following: (i) it is not important to maintain rotation invariance in local motion estimation level to achieve good registration results; (ii) the fine sub-pixel based matching part of the KLT algorithm is reliable even in case of a small number of point pairs. Therefore, the combination of scale-space based feature extraction and KLT like feature matching offers a good solution.

I have proposed a novel independent motion detection algorithm (Elastic Grid Multi-Fovea Detector), that exploits the parallel and cellular capabilities and communication model of the proposed architecture. I have experimentally shown that the model based connected multiple displacement method is adequate for the detection of ground objects moving on open field from a mini Unmanned Air Vehicle (UAV). I have proven by using complex metrics that the proposed algorithm offers better trade-off than previous solutions using the same scene assumptions.

Published in [2] [3]

The mainstream methods estimate a single global image transformation (projection) for 2D registration, directly fitting the flat-world assumption. Their estimation process needs complex floating point operations and the transformation itself requires a non-continuous memory access pattern. However, the proposed model-driven multiple displacement estimation can deal with moderate relief variation and operates only by using the continuous coalescing windowing memory access mechanisms.

The scheme of the method is presented in Figure 4.6. The computation is focused on the foveal processor array. The local partial results converge through iterations, using the results from the neighboring foveas.

	Artificial 135/130	Godollo_1 120/79	Godollo_2 300/230	Godollo_3 35/31	Complexity (operation per pixel)
SIFT	130	52	200	29	~1100
KLT	130	52	217	29	~450
BMA	128	61	208	29	~100
CPA	125	55	183	27	~75
EGMD	92	52	194	28	~50

Table 14. True-positive detection results and computational complexity for the algorithms

Total number of frames in a sequence / the frame number on which target ‘1’ is visible are given for each sequence in the header. Rows contain the number of true-positive detection results based on hand-made references together with the average computational steps normalized with the pixel count of the frames.

I have proven the effectiveness of my algorithm using a simulated and multiple real-life sequences. In case of momentous flight maneuvers, on average the Elastic Grid Multi-Fovea Detector gives similar detection results to the more complex algorithms for usual surveillance flight paths. On the contrary, it requires far less computational effort (Table 14).

6.3. Application of the Results

The Virtual Multi-Fovea Architecture is an adequate computational model for small and compact embedded detection-classification systems. In the frame of the VISCUBE project a multi-foveal chip is under design and prototype production. My algorithmic research fundamentally influenced the decision on what hardware-implemented instruction set is to be used in the first generation of the VISCUBE chip to be manufactured in 3D silicon technology.

The Multi-Fovea Architecture can be implemented using other many-core devices like FPGA-s. The related design considerations that were discussed in the dissertation are applicable in the parallel implementation of wide range of video processing algorithms. Numerous integrated circuit and system manufacturer (for example IBM, Intel, Nokia, Apple) supports the upcoming standard called OpenCL that is extremely similar to CUDA in its concept. The predictable general spread of those platforms will grant application potential to my results.

THE AUTHOR'S PUBLICATIONS

Journal Publication, Book Chapter

[1] Soós, B. G., A. Rák, J. Veres, and G. Cserey, "GPU boosted CNN simulator library for graphical flow based programmability," *EURASIP Journal on Advances in Signal Processing*, vol. 2009, 2009, p. 11.

[2] Soós, B. G., V. Szabó, and C. Rekeczky, "Elastic Grid Based Multi-Fovea Algorithm for Real-Time Object-Motion Detection in Airborne Surveillance," *C. Bhaatar, W. Porod and T. Roska : Cellular Nanoscale Sensory Wave Computing*, Springer, 2009.

Conference papers, Technical report

[3] B.G. Soós and C. Rekeczky, "Elastic Grid Based Analysis of Motion Field for Object-Motion Detection in Airborne Video Flows," *Circuits and Systems, ISCAS 2007. IEEE International Symposium on,* 2007, pp. 617-620.

[4] B.G. Soós, V. Szabó, and C. Rekeczky, *Multi-Fovea Architecture and Algorithms for Real-Time Object-Motion Detection in Airborne Surveillance*, Budapest, Hungary: Pazmány Peter Catholic University, 2009.

[5] B.G. Soós, A. Rák, J. Veres, and G. Cserey, "GPU powered CNN simulator (SIMCNN) with graphical flow based programmability," *Cellular Neural Networks and Their Applications, CNNA 2008. 11th International Workshop on,* 2008, pp. 163-168.

[6] A. Rák, F. Gergely, B.G. Soós, and G. Cserey, "CPU-GPU hybrid compiling for general purpose: Case studies," *Cellular Neural Networks and Their Applications, CNNA 2010. 12th International Workshop on*, 2010.

Co-authored journal publication

[7] A. Rák, B.G. Soós, and G. Cserey, "Stochastic Bitstream Based CNN and its Implementation of FPGA," *International Journal Circuit Theory and Applications*, vol. published online, Nov. 2008.

BIBLIOGRAPHY

[8] B. Stackhouse, S. Bhimji, C. Bostak, D. Bradley, B. Cherkauer, J. Desai, E. Francom, M. Gowan, P. Gronowski, D. Krueger, C. Morganti, and S. Troyer, “A 65 nm 2-Billion Transistor Quad-Core Itanium Processor,” *Solid-State Circuits, IEEE Journal of,* vol. 44, 2009, pp. 18-31.

[9] ITRS, “International Technology Roadmap for Semiconductors,” *http://www.itrs.net*, 2009.

[10] C. Rekeczky, I. Szatmari, D. Balya, G. Timar, and A. Zarandy, “Cellular multiadaptive analogic architecture: a computational framework for UAV applications,” *Circuits and Systems I: Regular Papers, IEEE Transactions on [Circuits and Systems I: Fundamental Theory and Applications, IEEE Transactions on],* vol. 51, 2004, pp. 864-884.

[11] A. Turing, “The chemical basis of morphogenesis,” *Philosophical Transactions of the Royal Society (part B),* vol. 237, 1953, pp. 37-72.

[12] L.O. Chua and T. Roska, *Cellular neural networks and visual computing: foundation and applications*, Cambridge University Press, 2002.

[13] T. Roska, “Cellular Wave Computing in Nanoscale via Million Processors,” *C. Bhaatar, W. Porod and T. Roska : Cellular Nanoscale Sensory Wave Computing*, Springer, 2009.

[14] B. Roska and F. Werblin, “Vertical interactions across ten parallel, stacked representations in the mammalian retina,” *Nature,* vol. 410, 2001, pp. 583-587.

[15] D. Bálya, B. Roska, T. Roska, and F.S. Werblin, “A CNN framework for modeling parallel processing in a mammalian retina,” *International Journal of Circuit Theory and Applications,* vol. 30, 2002, pp. 363-393.

[16] L. Kek, K. Karacs, and T. Roska, *Cellular Wave Computing Library V2.1,* Budapest: Cellular Sensory Wave Computers Laboratory, Computer and Automation Research Institute, Hungarian Academy of Science, 2007.

[17] T. Roska, “Computational and computer complexity of analogic cellular wave computers,” *Cellular Neural Networks and Their Applications, 2002. (CNNA 2002). Proceedings of the 2002 7th IEEE International Workshop on*, 2002, pp. 323-338.

[18] A. Rodriguez-Vazquez, G. Linan-Cembrano, L. Carranza, E. Roca-Moreno, R. Carmona-Galan, F. Jimenez-Garrido, R. Dominguez-Castro, and S. Meana, “ACE16k the third generation of mixed-signal SIMD-CNN ACE chips toward VSoCs,” *Circuits and Systems I: Regular Papers, IEEE Transactions on,* vol. 51, 2004, pp. 851-863.

[19] AnaFocus, “eye-RIS 1.1 Leaflet of AnaFocus Ltd,” 2007.

[20] NVIDIA, *GeForce 8800 GPU Architecture Overview*, NVIDIA Corp., 2006.

[21] MathWorks, “Simulink - Simulation and Model-Based Design,” *http://www.mathworks.com/products/simulink/*, 2008.

[22] V. Kumar, *Introduction to Parallel Computing*, Addison-Wesley Longman Publishing Co., Inc., 2002.

[23] C. Böhm and G. Jacopini, “Flow diagrams, turing machines and languages with only two formation rules,” *Commun. ACM,* vol. 9, 1966, pp. 366-371.

[24] A.L. Davis, “The architecture and system method of DDM1: A recursively structured Data Driven Machine,” *Proceedings of the 5th annual symposium on Computer architecture*, ACM, 1978, pp. 210-215.

[25] A. Davis and R. Keller, “Data Flow Program Graphs,” *Computer,* vol. 15, 1982, pp. 26-41.

[26] W.M. Johnston, J.R.P. Hanna, and R.J. Millar, “Advances in dataflow programming languages,” *ACM Comput. Surv.,* vol. 36, 2004, pp. 1-34.

[27] B. Khailany, W. Dally, U. Kapasi, P. Mattson, J. Namkoong, J. Owens, B. Towles, A. Chang, and S. Rixner, “Imagine: media processing with streams,” *Micro, IEEE,* vol. 21, 2001, pp. 35-46.

[28] C. Rekeczky, I. Szatmári, D. Bálya, G. Tímár, and A. Zarándy, “Cellular multiadaptive analogic architecture: a computational framework for UAV applications,” *Circuits and Systems I: Regular Papers, IEEE Transactions on [Circuits and Systems I: Fundamental Theory and Applications, IEEE Transactions on],* vol. 51, 2004, pp. 864-884.

[29] A. Zarandy and C. Rekeczky, *Analysis of the implementation and the efficiency of 2D operators on topographic and non-topographic processor architectures*.

[30] L. Chua and L. Yang, “Cellular neural networks: Theory and Applications,” *Circuits and Systems, IEEE Transactions on,* vol. 35, 1988, pp. 1257-1272.

[31] T. Roska and L. Chua, “The CNN universal machine: an analogic array computer,” *Circuits and Systems II: Analog and Digital Signal Processing, IEEE Transactions on,* vol. 40, 1993, pp. 163-173.

[32] P. Dudek and P. Hicks, “A general-purpose processor-per-pixel analog SIMD vision chip,” *Circuits and Systems I: Regular Papers, IEEE Transactions on,* vol. 52, 2005, pp. 13-20.

[33] A. Zarandy and C. Rekeczky, “Bi-i a standalone ultra high speed cellular vision system,” *Circuits and Systems Magazine, IEEE,* vol. 5, 2005, pp. 36-45.

[34] Z. Nagy, “Implementation of emulated digital CNN-UM architecture on programale logic devices and its applications,” P.hD. thesis, Department of Image Processing and Neurocomputing University of Pannonia Veszprém, 2007.

[35] P. Foldesy, A. Zarandy, C. Rekeczky, and T. Roska, “High performance processor array for image processing,” *Circuits and Systems, 2007. ISCAS 2007. IEEE International Symposium on,* 2007, pp. 1177-1180.

[36] S. Tokes, L. Orzo, C. Rekeczky, T. Roska, and A. Zarandy, “An optical CNN implementation with stored programmability,” *Circuits and Systems, 2000. Proceedings. ISCAS 2000 Geneva. The 2000 IEEE International Symposium on,* 2000, pp. 136-139 vol.2.

[37] P. Foldesy, R. Carmona-Galan, A. Zarandy, C. Rekeczky, A. Rodriguez-Vazquez, and T. Roska, “3D multi-layer vision architecture for surveillance and reconnaissance applications,” *Circuit Theory and Design, 2009. ECCTD 2009. European Conference on,* 2009, pp. 185-188.

[38] J.D. Owens, D. Luebke, N. Govindaraju, M. Harris, J. Kruger, A.E. Lefohn, and T.J. Purcell, “A Survey of General-Purpose Computation on Graphics Hardware,” *Computer Graphics Forum,* vol. 26, 2007, pp. 80-113.

[39] NVIDIA, “CUDA Compute Unified Device Architecture - Programming Guide 1.1,” Nov. 2007.

[40] Weiming Hu, Tieniu Tan, Liang Wang, and S. Maybank, “A survey on visual surveillance of object motion and behaviors,” *Systems, Man, and Cybernetics, Part C: Applications and Reviews, IEEE Transactions on,* vol. 34, 2004, pp. 334-352.

[41] A. Yilmaz, O. Javed, and M. Shah, “Object tracking: A survey,” *ACM Comput. Surv.,* vol. 38, 2006, p. 13.

[42] M. Molinier, T. Häme, and H. Ahola, “3D-Connected Components Analysis for Traffic Monitoring in Image Sequences Acquired from a Helicopter,” *Image Analysis,* 2005, pp. 141-150.

[43] G. Adiv, “Determining Three-Dimensional Motion and Structure from Optical Flow Generated by Several Moving Objects,” *Pattern Analysis and Machine Intelligence, IEEE Transactions on,* vol. PAMI-7, 1985, pp. 384-401.

[44] A. Argyros, M. Lourakis, P. Trahanias, and S. Orphanoudakis, “Qualitative detection of 3D motion discontinuities,” 1996, pp. 1630-1637 vol.3.

[45] R. Black and A. Jepson, “Estimating multiple independent motions in segmented

images using parametric models with local deformations," 1994, pp. 220-227.

[46] H. Sawhney, Y. Guo, and R. Kumar, "Independent motion detection in 3D scenes," *Pattern Analysis and Machine Intelligence, IEEE Transactions on,* vol. 22, 2000, pp. 1191-1199.

[47] M. Irani and P. Anandan, "A unified approach to moving object detection in 2D and 3D scenes," *Pattern Analysis and Machine Intelligence, IEEE Transactions on,* vol. 20, 1998, pp. 577-589.

[48] Zhigang Zhu, Hao Tang, G. Wolberg, and J. Layne, "Content-based 3D mosaic representation for video of dynamic 3D scenes," *Applied Imagery and Pattern Recognition Workshop, 2005. Proceedings. 34th*, 2005, pp. 6 pp.-203.

[49] M.I.A. Lourakis, A.A. Argyros, and S.C. Orphanoudakis, "Independent 3D Motion Detection Using Residual Parallax Normal Flow Fields," 1998, pp. 1012-1017.

[50] S. Fejes and L.S. Davis, "Detection of Independent Motion Using Directional Motion Estimation," *Computer Vision and Image Understanding,* vol. 74, May. 1999, pp. 101-120.

[51] J. Barron, D. Fleet, S. Beauchemin, and T. Burkitt, "Performance of optical flow techniques," *Computer Vision and Pattern Recognition, 1992. Proceedings CVPR '92., 1992 IEEE Computer Society Conference on*, 1992, pp. 236-242.

[52] B. Zitova and J. Flusser, "Image registration methods: a survey," *Image and Vision Computing,* vol. 21, Oct. 2003, pp. 977-1000.

[53] R. Kumar, H. Sawhney, S. Samarasekera, S. Hsu, Hai Tao, Yanlin Guo, K. Hanna, A. Pope, R. Wildes, D.A.-.H. Hirvonen, M.A.-.H. Hansen, and P.A.-.B. Burt, "Aerial video surveillance and exploitation," *Proceedings of the IEEE,* vol. 89, 2001, pp. 1518-1539.

[54] B. Morse, D. Gerhardt, C. Engh, M. Goodrich, N. Rasmussen, D. Thornton, and D. Eggett, "Application and evaluation of spatiotemporal enhancement of live aerial video using temporally local mosaics," *Computer Vision and Pattern Recognition, 2008. CVPR 2008. IEEE Conference on*, 2008, pp. 1-8.

[55] S. Ali and M. Shah, "COCOA: tracking in aerial imagery," *Airborne Intelligence, Surveillance, Reconnaissance (ISR) Systems and Applications III,* Orlando (Kissimmee), FL, USA: SPIE, 2006, pp. 62090D-6.

[56] R. Pless, T. Brodsky, and Y. Aloimonos, "Detecting independent motion: the statistics of temporal continuity," *Pattern Analysis and Machine Intelligence, IEEE Transactions on,* vol. 22, 2000, pp. 768-773.

[57] J.W. Hsieh, "Fast stitching algorithm for moving object detection and mosaic construction," *Image and Vision Computing,* vol. 22, 2004, pp. 291-306.

[58] M. Brown and D.G. Lowe, "Recognising Panoramas," *Proceedings of the Ninth IEEE International Conference on Computer Vision - Volume 2,* IEEE Computer Society, 2003, p. 1218.

[59] R. Szeliski, "Image alignment and stitching: a tutorial," *Found. Trends. Comput. Graph. Vis.,* vol. 2, 2006, pp. 1-104.

[60] H. Sawhney and R. Kumar, "True multi-image alignment and its application to mosaicing and lens distortion correction," *Pattern Analysis and Machine Intelligence, IEEE Transactions on,* vol. 21, 1999, pp. 235-243.

[61] K. Kaaniche, B. Champion, C. Pegard, and P. Vasseur, "A Vision Algorithm for Dynamic Detection of Moving Vehicles with a UAV," 2005, pp. 1878-1883.

[62] K. Mikolajczyk and C. Schmid, "A Performance Evaluation of Local Descriptors," *IEEE Trans. Pattern Anal. Mach. Intell.,* vol. 27, 2005, pp. 1615-1630.

[63] D.G. Lowe, "Distinctive Image Features from Scale-Invariant Keypoints," *International Journal of Computer Vision,* vol. 60, Nov. 2004, pp. 91-110.

[64] B.D. Lucas and T. Kanade, "An Iterative Image Registration Technique with an Application to Stereo Vision," *International Joint Conference on Artificial Intelligence,* Vancouver: 1981, pp. 674-679.

[65] J. Shi and C. Tomasi, “Good features to track,” *Computer Vision and Pattern Recognition, 1994. Proceedings CVPR '94., 1994 IEEE Computer Society Conference on*, 1994, pp. 593-600.

[66] S. Zhu and K. Ma, “A new diamond search algorithm for fast block-matching motion estimation,” *Image Processing, IEEE Transactions on,* vol. 9, 2000, pp. 287-290.

[67] C. Harris and M. Stephens, “A combined corner and edge detector,” *Proceedings Fourth Alvey Vision Conference,* Manchester, UK: 1988, pp. 147--151.

[68] R. Hartley and A. Zisserman, *Multiple view geometry in computer vision*, Cambridge University Press, 2004.

[69] Z. Zhang, “Determining the Epipolar Geometry and its Uncertainty: A Review,” *Int. J. Comput. Vision,* vol. 27, 1998, pp. 161-195.

[70] M.A. Fischler and R.C. Bolles, “Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography,” *Commun. ACM,* vol. 24, 1981, pp. 381-395.

[71] P. Torr, *A Structure and Motion Toolkit in Matlab: Interactive adventures in S and M,* Microsoft Research, 2002.

[72] V. Kolmogorov and R. Zabih, “Multi-camera Scene Reconstruction via Graph Cuts,” *Proceedings of the 7th European Conference on Computer Vision-Part III*, Springer-Verlag, 2002, pp. 82-96.

[73] P.D. Kovesi, *MATLAB and Octave Functions for Computer Vision and Image Processing*, School of Computer Science and Software Engineering, The University of Western Australia, .

[74] A. Zarandy, C. Rekeczky, and P. Foldesy, “Analysis of 2D operators on topographic and non-topographic processor architectures,” *Cellular Neural Networks and Their Applications, 2008. CNNA 2008. 11th International Workshop on*, 2008, pp. 57-62.

[75] R. Chandra, R. Menon, L. Dagum, D. Kohr, D. Maydan, and J. McDonald, *Parallel Programming in OpenMP*, Morgan Kaufman, 2000.

[76] Rapidmind, “RAPIDMIND,” *http://www.rapidmind.net/.*

[77] “Intel Performance Libraries,” 2008.

[78] A. Zarándy and C. Rekeczky, “Many-core Processing Array Architecture Selection Guide for Topologic Problems,” *C. Bhaatar, W. Porod and T. Roska : Cellular Nanoscale Sensory Wave Computing,* Springer, 2009.

[79] J. Nickolls, I. Buck, M. Garland, and K. Skadron, “Scalable Parallel Programming with CUDA,” *Queue,* vol. 6, 2008, pp. 40-53.

[80] IEEE-Software and Systems Engineering Vocabulary, *http://pascal.computer.org/sev_display/index.action*.

[81] Xiaoquan Yi and Nam Ling, “Rapid block-matching motion estimation using modified diamond search algorithm,” *Circuits and Systems, 2005. ISCAS 2005. IEEE International Symposium on,* 2005, pp. 5489-5492 Vol. 6.

APPENDIX A

Complexities

Complexities for the algorithms are briefly described in the following tables. Functions are described in rows. They can be optimally implemented on the Frontend Processor Array (FPA), on the Foveal Processor Array (FVA) or on the serial Backend Processor (BP): one of them is marked. In case of foveal processing, the number of used foveal windows is also displayed. The Input/Output is described using notation S for scalars, and p for points (two scalars).

At first, complexity for the global registration based detection part is given (Table 15), and then all the algorithms one by one.

	Detection part of algorithms	FPA	FVA	BP	fovea	Read/ Write	Algo Step	STLM	LTLM
c	Global tr model est.								
	RANSAC: for l3 feature pairs l4 of them will turn to be inlier in N iterations			X		R-l3x2p W-l4x2p	N*15000		
	Estimate: linear estimation for inliers			X		R-l4x2p W-9S	$\left(9\times 2\times l4^{2} + 2\times l4^{3}\right)\times 5$		
d	Alignment								
	Transform: transform previous frame			X		R-1As R-9S W-1As	30As		
e	Detection								
ser	Detect: Ab-sDiff+threshold+morphology	X				R-1As W-l5xp W-1As	5As+10As	1As	1As (Frame)
arr							3As		

Table 15. Complexity of the global registration based detection part of the algorithms (steps c,d,e).

	Corner Pairing Algorithm	FPA	FVA	BP	fo-veas	Read/ Write	Algo Step	STLM	LTLM
a	Feature/template selection								
	ReadCamera: input frame from sensor	X				W-1As			1As (Frame)
ser	Extract: Harris corner extraction	X				W-l1xp	89As+ 20As	3As	
arr							12As+ 16As		
b	Feature/template matching								
	SelectA: possible pairs with gating			X		R-l1xp W-l2xp	$5\times l1^{2}$		
	Check: correlation check		X		l2	R-2At W-S	5At		
	SelectB: keep good pairs			X		R-l2xS R-l2xp W-l3x2p	3xl2x10		
c	Global tr model est.								
	RANSAC: for l3 feature pairs l4 of them will turn to be inlier in N iterations			X		R-l3x2p W-l4x2p	N*15000		
	Estimate: linear estimation for inliers			X		R-l4x2p W-9S	$\left(9\times\ 2\times l4^{2} + 2\times l4^{3}\right)\times 5$		
d	Alignment								
	Transform: transform previous frame			X		R-1As R-9S W-1As	30As		
e	Detection								
ser	Detect: AbsDiff+threshold+morphology	X				R-1As W-l5xp W-1As	5As+10As	1As	1As (Frame)
arr							3As		

Table 16. Complexity of Corner Pairing algorithm, together with global transformation registration based detection.

	Block Matching Algorithm	FPA	FVA	BP	fovea	Read/ Write	Algo Step	STLM	LTLM
a	Feature/template selection								
	ReadCamera: input frame from sensor	X				W-1As			1As (Frame)
	Prefilter: autocorrelation filtering in l1 windows on a fixed grid (l1= Mc x Nc)		X		l1	R-1At W-2S	[2*4+3*4]At+ 20	3At	
	SelectA: select good locations (l2 from l1)			X		R-l1xp W-l2xp	4l1		
b	Feature/template matching								
	CalcDisplacement: AbsDiff for templates with [full,diamond] search, q steps		X		l2	R-2Aw W-5S	$2At + q \times 6At$ $+2 \times q^2$	4At 1Aw	
	SelectB: select reliable matches (l3 from l2)			X		R-l2x5S W-l3x2p	4l2		

Table 17. Complexity of the BMA algorithms.

For full search $q=(2r)^2$, for diamond search $q = 9 + 5r$, where r is the maximal displacement of the video flow. The global transformation based detection is used in the same way as in the previous algorithms (steps c,d,e also described in Table 15)

	KLT Algorithm	FPA	FVA	BP	foveas	Read/ Write	Algo Step	STLM	LTLM
a	Feature/template selection								
	ReadCamera: input frame from sensor	X				W-1As			1As (Frame)
ser	Extract: Harris corner extraction	X				W-l1xp	89As+ 20As	3As	
arr							12As+ 16As		
b	Feature/template matching								
	CalcDisplace-ment: k KLT steps for s scale		X		sxl2	R-2Aw W-5S	$18Aw+27At+ q\times\begin{pmatrix}18Aw+\\27At+\\32At+\\25\end{pmatrix}$	9Aw	
	SelectB: keep good pairs			X		R-l2x5S W-l3x2p	4l2		

Table 18. Complexity of the KLT algorithm.

q is typically 5, s is the 2-based-logarithm of r (the maximal displacement of the video flow). The global transformation based detection is used in the same way as in the previous algorithms (steps c,d,e described in Table 15)

	SIFT Algorithm	FPA	FVA	BP	foveas	Read/Write	Algo Step	STLM	LTLM
a	Feature/template selection								
	ReadCamera: input frame from sensor	X				W-1As			1As (Frame)
ser arr	Extract: Gauss scale space, differences,3D local maxima	X				W-l1xp W-OxnsxAs	$\sum_{os=1}^{O} As^{os}\left[\begin{matrix}(ns+2)30+\\(ns+2)3+\\ns\times2\times27\end{matrix}\right.$ $O\times As\begin{bmatrix}(ns+2)2+\\(ns+2)1+\\ns\times2\times27\end{bmatrix}$	5As	1As (Frame)
	Descriptor1: create edge histograms, find peaks **Descriptor2:** rotate **Descriptor3:** create descriptors		X		l1	R-At W-128xS	$10At+36\times5+$ $20At+10At$	3At 128xS	
b	Feature/template matching								
	Match: matching descriptors			X		R-2xl1x128S W-l3x2p	$l2^{2}\times128^{2}$		

Table 19. Complexity of the SIFT algorithm.
O is the number of octaves used, ns is the number of sub-scales in each. The global transformation based detection is used in the same way as in the previous algorithms (steps c,d,e also described in Table 15)

	ELG Algorithm	FPA	FVA	BP	iterations, foveas	Read/Write	Algo Step	STLM	LTLM
a	Feature/template selection								
	ReadCamera: input frame from sensor	X				W-1As			1As (Frame)
	Prefilter: autocorrelation filtering in l1 windows on a fixed grid (l1= Mc x Nc)		X		l1	R-1At W-2S	[2*4+3*4]At+20	3At	
	SelectA: select good locations (l2 from l1)			X		R-l1xp W-l2xp	4l1		
b	Feature/template matching								
	CalcDisplacement: AbsDiff for templates with joined search, q steps with topological interaction		X		qxl2	$R-2Aw$ $R-Aw$ $R-4\times 3S$ $W-3S$	8At+40	4At	(Aw)
	“Select” l3:=l2			X		R-l2x5S W-l3x2p	4l2		
c	Global tr model est.								
d	Alignment								
	Transform: transform previous frame			X		R-1As R-l3x3S W-1As	5As		
e	Detection								
ser	Detect: AbsDiff+threshold+morphology	X				R-1As W-l5xp W-1As	5As+10As	1As	1As (Frame)
arr							3As		

Table 20. Complexity of the Elastic Grid based Multi-Fovea algorithm.
q is typically 3 times r (the maximal displacement of the video flow). Steps c,d are *different from the global registration based detection.*

APPENDIX B

Some basic software related definitions are listed based on the IEEE-Software and Systems Engineering Vocabulary [80].

Definition 54 **Entity**

An entity can be anything that can be measured (estimated, defined) and can be expressed with numerical values for modeling some portion of the environment.

The mathematical formulation of computers, *computation theory* was originally the science of handling integer numbers, although it was shortly extended to procedural handling of any rational number represented with final precision (with a limited number of fractional digits as *fixed-point*, or the more flexible *floating-point* representation). *Addition* and *multiplication* are the most important basic operators, they form a special algebraic structure called *field* with the rational numbers. In practical cases the data handled by a computer is far more than a simple number: it is modeling many diverse objects from the environment with many factors.

Definition 55 **Attribute**

Complex entities may be characterized with multiple properties (parameters). Description of a specific property of an entity is an attribute.

Some attributes of the entities may have some meaningful ordering and can be organized into 1D or 2D indexed structures. Structures with higher dimensions can be organized into multiple structures of lower dimensions.

Definition 56 **Basic type qualifiers**

Scalar – atomic numerical attribute (with *addition* and *multiplication* operators)

Vector – 1D structure of scalars with a predefined number of elements

Matrix – $\mathbf{A}, \mathbf{A}_{R\times C}$ 2D structure of scalars, indexed in row column order with a predefined number of elements.

Elements can be referred as: $A_{R\times C}\ {}_{r,c}$, $A\ {}_{r,c}, a_{r,c}$, where

$\begin{array}{l} r \in\ 1,2..R \\ c \in\ 1,2,..C \end{array}$, R and C is the number of rows and columns, respectively.

The size of a matrix is denoted by $\left\|\mathbf{A}_{R\times C}\right\|, \left\|\mathbf{A}\right\|$, and $\left\|\mathbf{A}_{R\times C}\right\| = RC$

The data may have some more secondary structures among a group of attributes for an entity or defined on entities sharing some common property. The set $\langle \mathbf{S} \rangle$ of some values without ordering and with 1D or 2D ordering is also used to describe a structure:

Definition 57 **Tuple** $a,b,c,...$ **,** $a_1,a_2,a_3,...$

Values of arbitrary type can be grouped into a tuple. Elements can be referred to by indices. For example, an entity may be described as a tuple of all attributes.

Definition 58 **List -** a_i , $a_{r,c}$

A structure containing values of the same type is called a list (formally it is a well-ordered set). 1D or 2D indices can be used for referring to any element. 2D indices are in row column order. The typography of the name reflects the type of the elements.

Elements can be referred as: $a(i), a(i,j), a_i, a_{i,j}$

The size of a list is denoted by $\| \; a_i \; \|, \| \; a_{r,c} \; \|$, where

$$i \in \; 1,2,..N$$
$$r \in \; 1,2..R$$
$$c \in \; 1,2,..C$$
$$\| \; a_i \; \| = N, \|L_{R\times C}\| = R \cdot C$$

Indices for complex structures are given in decreasing order of the hierarchy, for example $\mathbf{I}_{k\;y,x} = \; \mathbf{I}_{k} \;\; {}_{y,x}$ is used for matrix elements in a list of matrices.

A wide range of attributes can be coded into the above structures using special encodings for non-numerical data, such as the ASCII code – for characters of the Latin alphabet and some other symbols – and 1-D lists of characters for text.

Definition 59 **Compound type, extended type**

Tuple and list types are compound types. Functions may also be defined as taking compound values. Type qualifier *extended type* may refer to operands of any basic or compound types.

Some entities that are valid for a long time can be considered constant, or at least do not change their values without direct interaction with the environment. Others are time dependent, and their model should be updated by measurement or tracked using some internal model of their dynamics.

Definition 60 **Time series**

In case of time-dependent entities, a common time unit is used for time-discretization (a value that is smaller than all Nyquist-rates). After sampling, values are represented in lists sharing the common time index.

APPENDIX C

Dataflow graphs

In the next subsections dataflow diagrams of the algorithms are presented. General notations are summarized in Table 21. The main three computational structures (Frontend Processor Array (FPA), Foveal Processor Array (FVA), Backend Processor (BP)) and memory management functionalities are distinguished using the colors blue, green, white and orange, respectively.

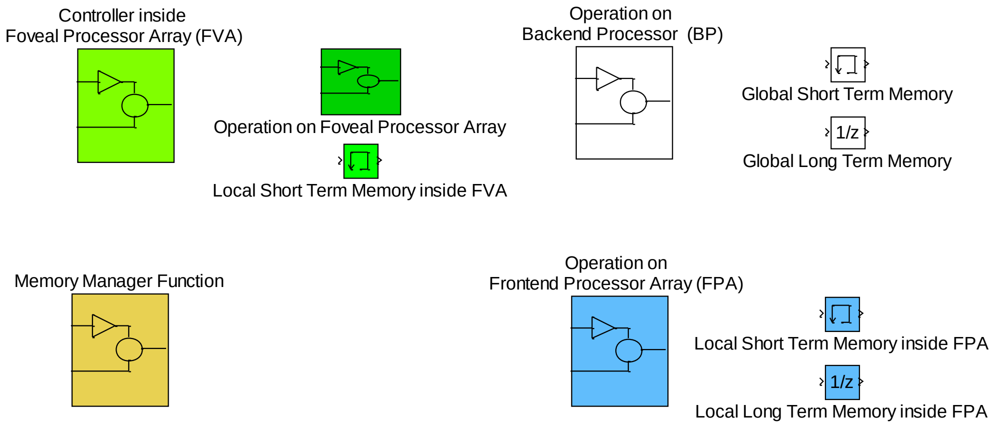
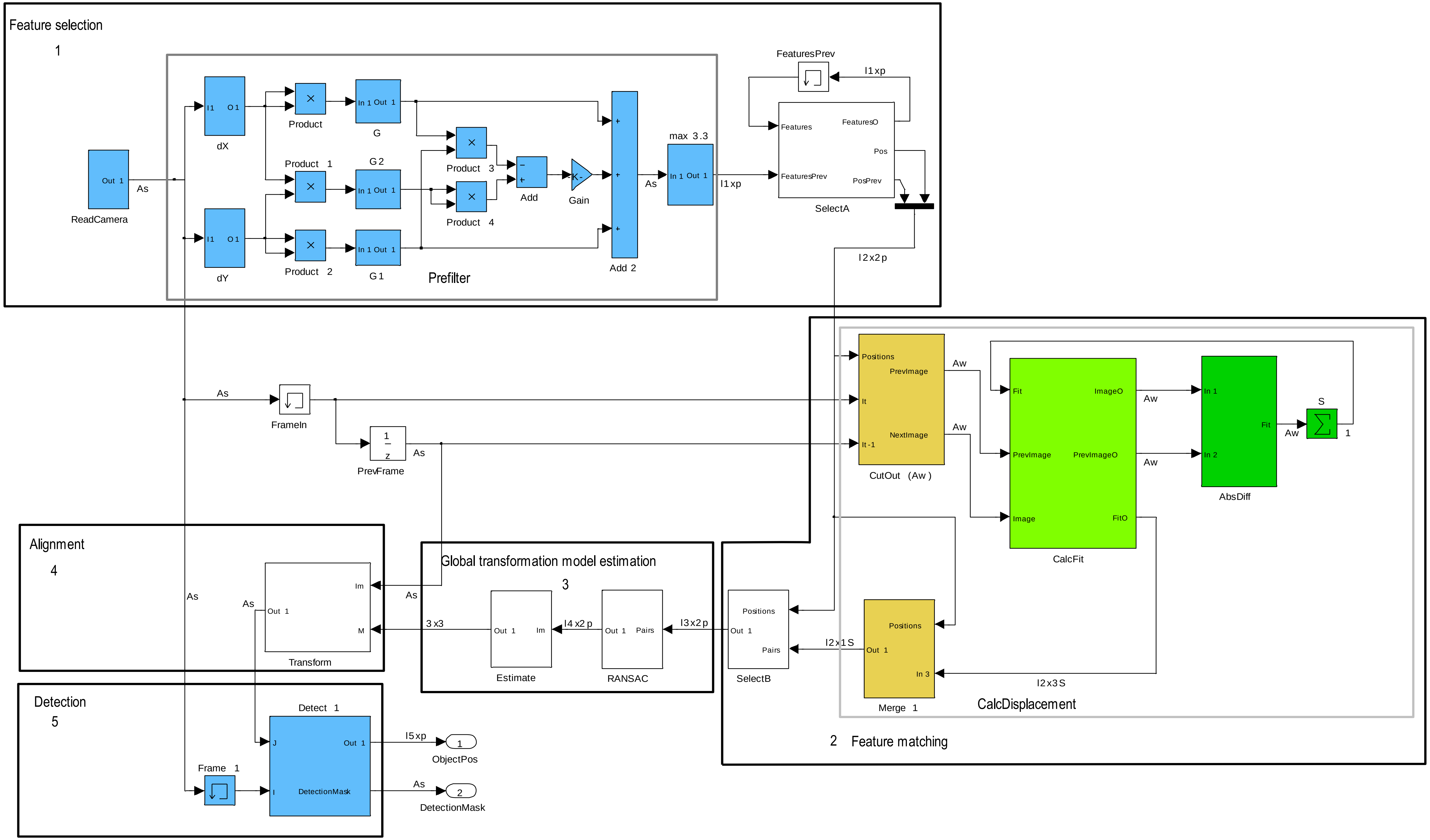
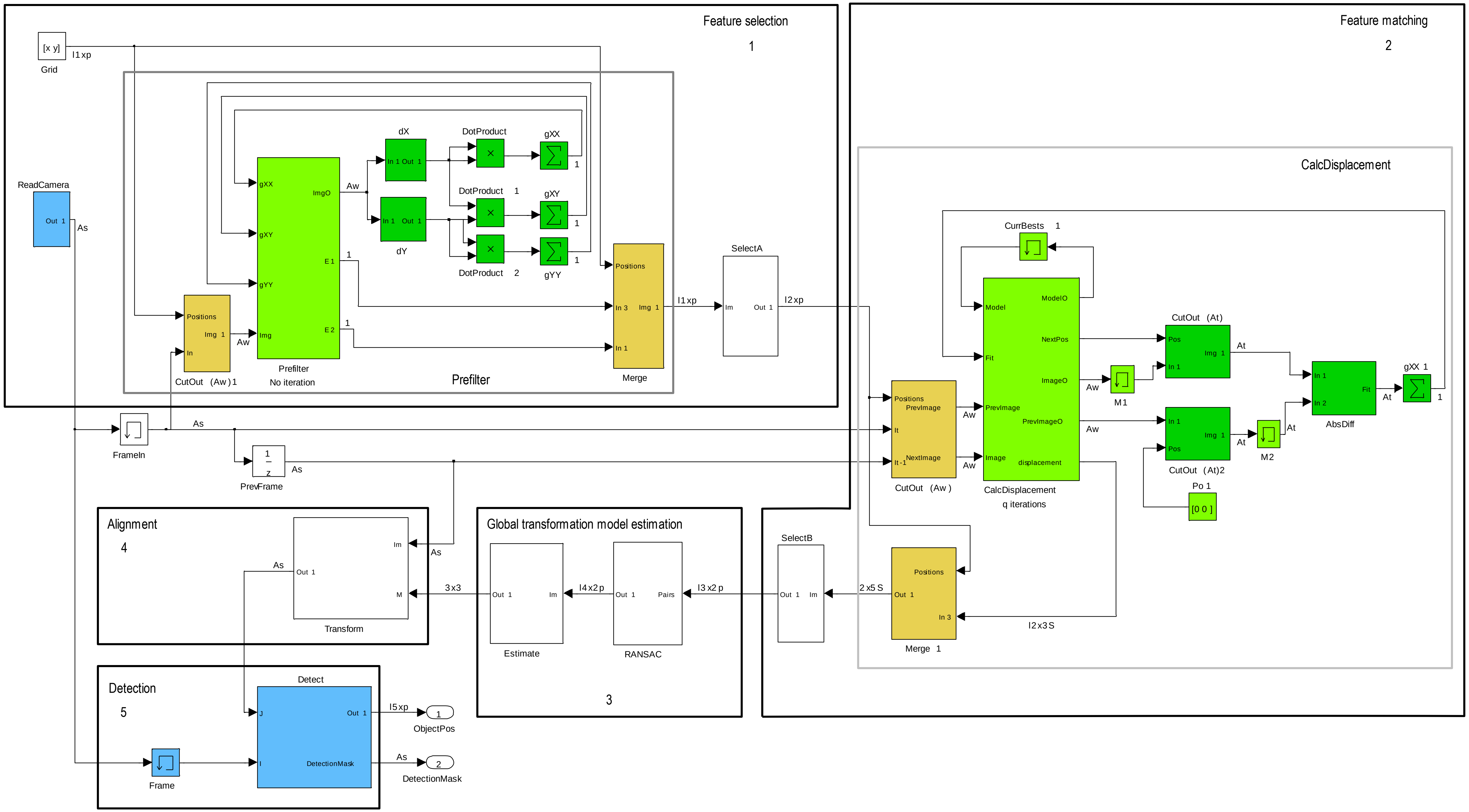

Controller inside
Foveal Processor Array (FVA)
Operation on Foveal Processor Array
Local Short Term Memory inside FVA
Operation on
Backend Processor (BP)
Global Short Term Memory
1/z
Global Long Term Memory
Memory Manager Function
Operation on
Frontend Processor Array (FPA)
Local Short Term Memory inside FPA
1/z
Local Long Term Memory inside FPA

Table 21. General notations for dataflow diagram

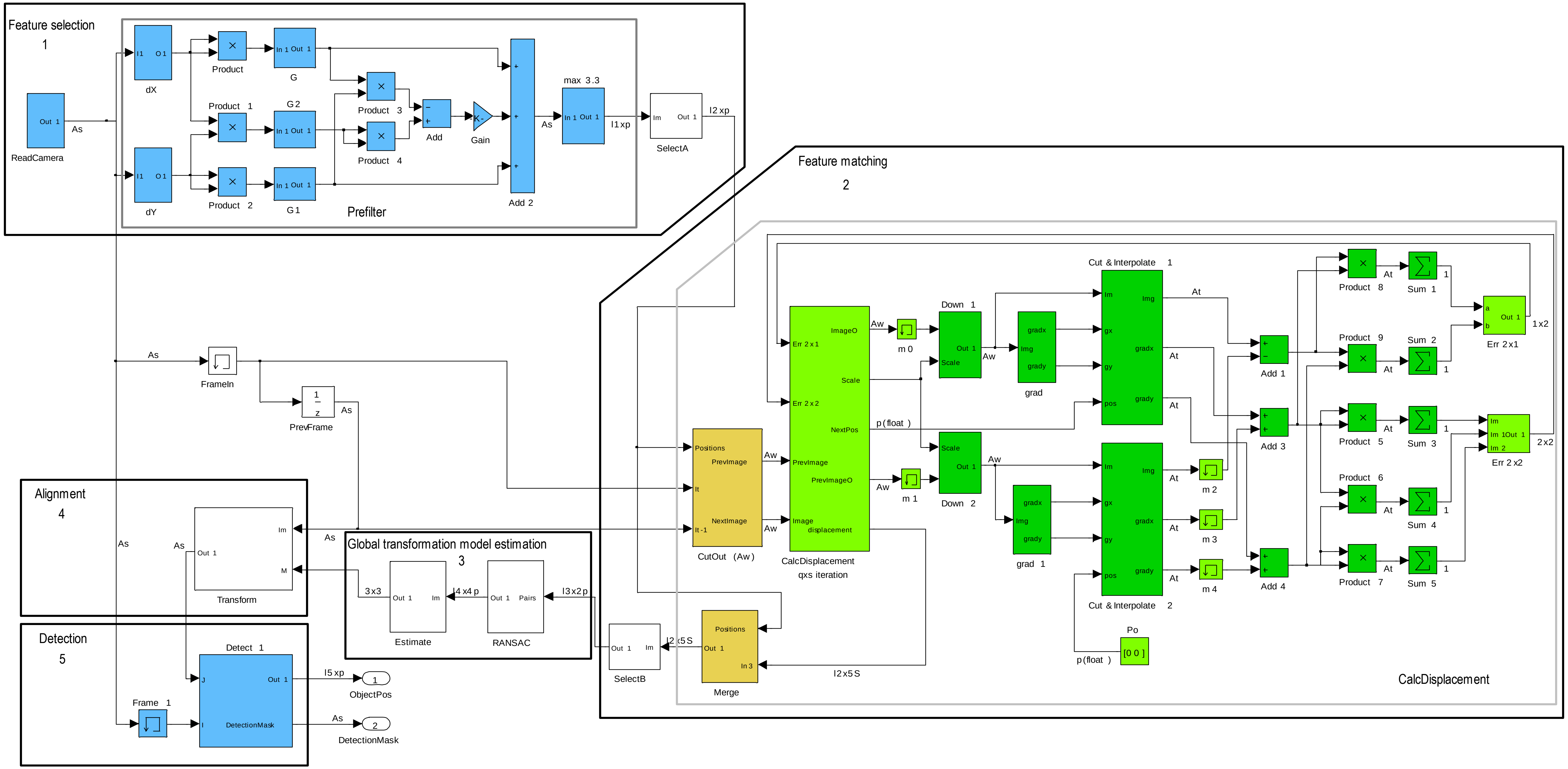
6.4. Corner Pairing Algorithm



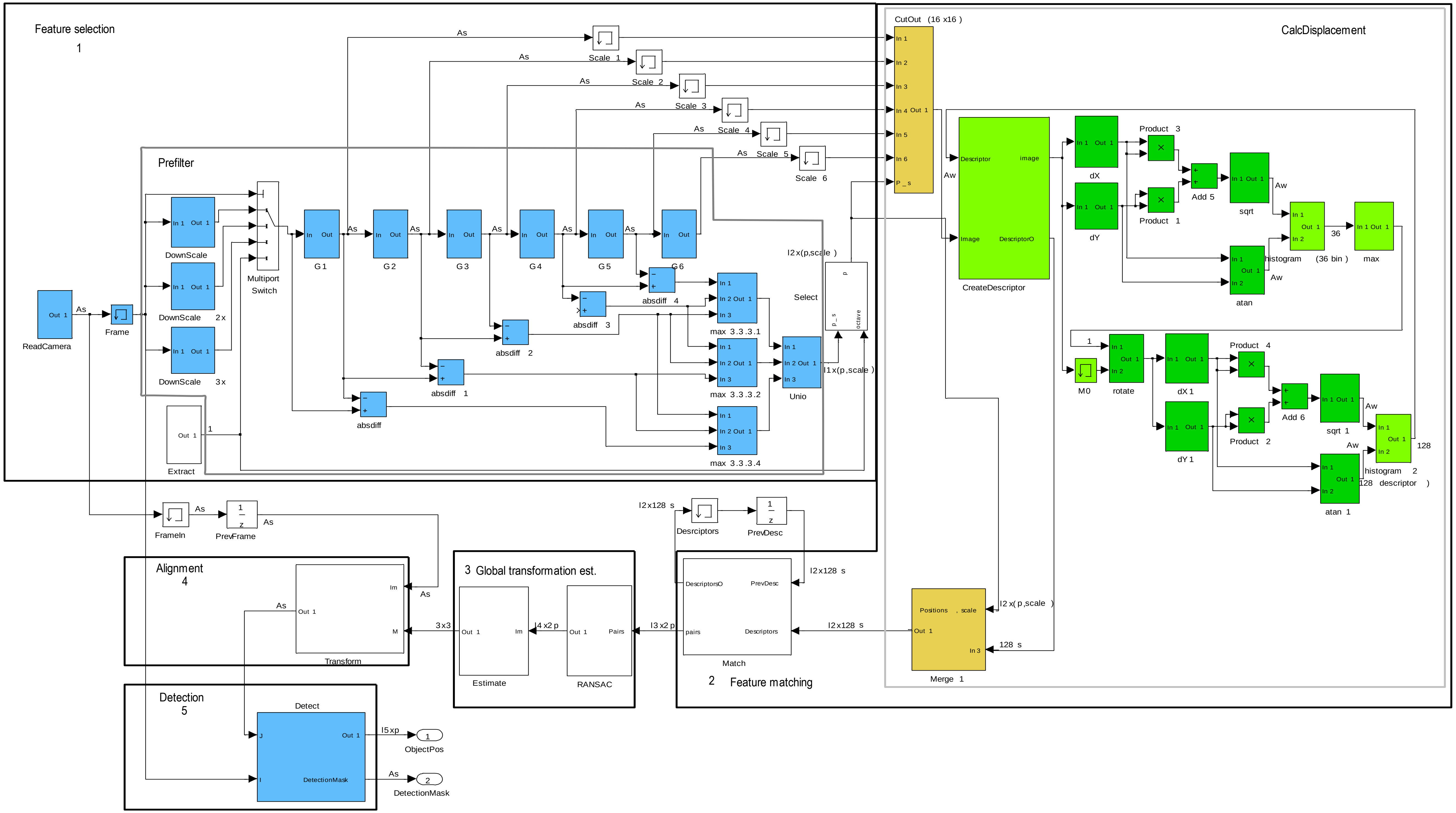
6.5. Block Matching Algorithm



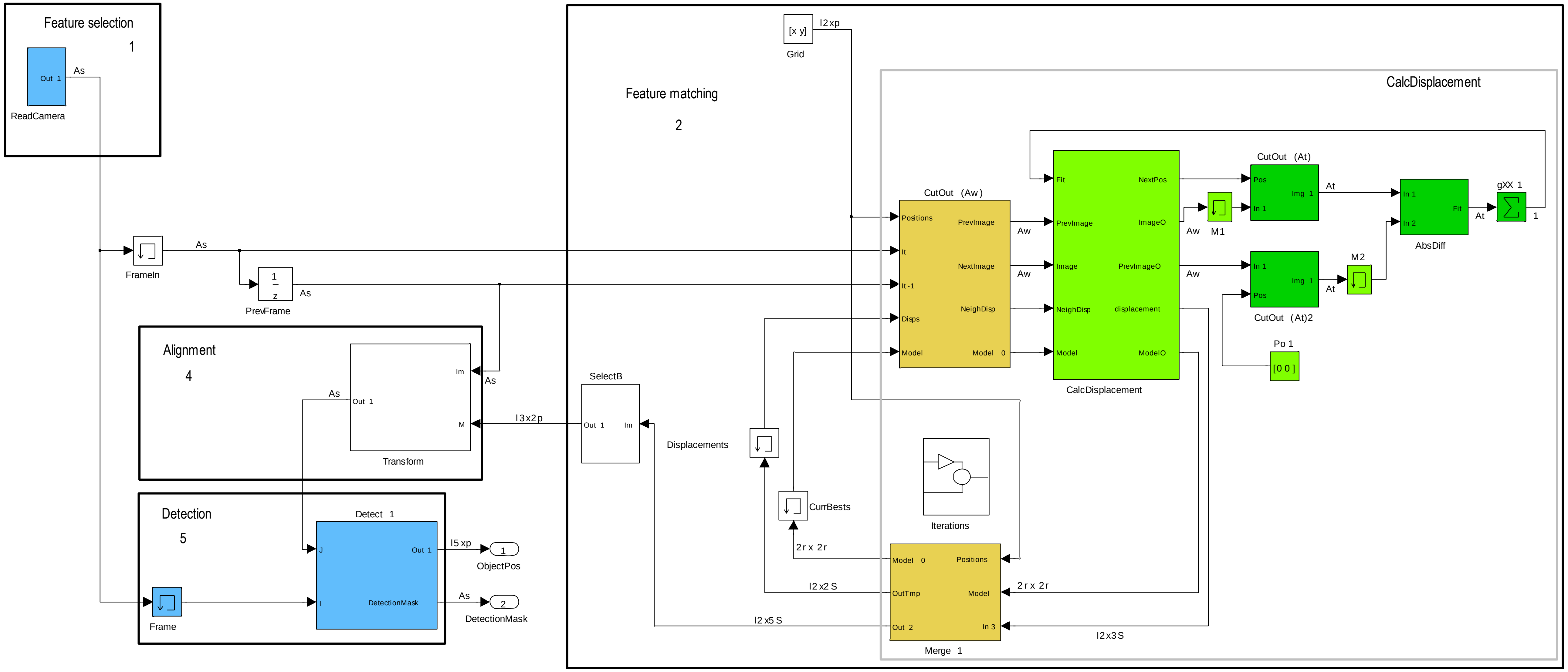
6.6. KLT Algorithm



6.7. SIFT Algorithm



6.8. Elastic Grid Algorithm



LIST OF ABBREVIATIONS

ALU	Arithmetic and Logical Unit
Algo	algorithm
API	Application Programming Interfaces
ASIC	Application Specific Integrated Circuit
BMA	Block Matching Algorithm
BMA-DS	Diamond Search Block Matching Algorithm
BMA-FS	Full Search Block Match Algorithm
BP	Backend Processor
CMVA	Cellular Multi-core Video Analytics
CNN	Cellular Neural/Nonlinear Network
CNN-UM	CNN Universal Machine
CPA	Corner Pairing Algorithm
CPU	Central Processing Unit
CUDA	Compute Unified Device Architecture
DAG	Directed Acyclic Graph
DMA	Direct Memory Access
DoG	Difference of Gaussian
DSP	Digital Signal Processor
e.g.	for example
ELG	Elastic Grid Multi-Fovea Detector or Elastic Grid Algorithm
FPA	Frontend Processor Array
FPGA	Field Programmable Gate Array
FVA	FoVeal Processor Array
GPU	Graphics Processing Unit
GP-GPU	general purpose GPU
i.e.	id est (Latin)
IC	Integrated Circuit
I/O	input/output
KLT	Kanade-Lucas Tracker
LDSP	Large Diamond Search Pattern

LTLM	Long Term Local Memory
MIMD	multiple instruction multiple data
MP	Multiprocessor
MRI	Magnetic Resonance Imaging
PC	Personal Computer
PCB	Printed Circuit Board
PCI local bus	Peripheral Component Interconnect local bus
PCIe	PCI Express
DLT	Direct Linear Transform
PE	Processing Element
PET	Positron Emission Tomography
RANSAC	RANdom SAmple Consensus
SAD	sum of absolute differences
SDSP	Small Diamond Search Pattern
SIFT	Scale Invariant Feature Transform
SIMD	single instruction multiple data
STLM	Short Term Local Memory
SVD	Singular Value Decomposition
UAV	Unmanned Air Vehicle
UMF	Universal Machine on Flows
VLSI	Very Large Scale Integration

Könyvjelző