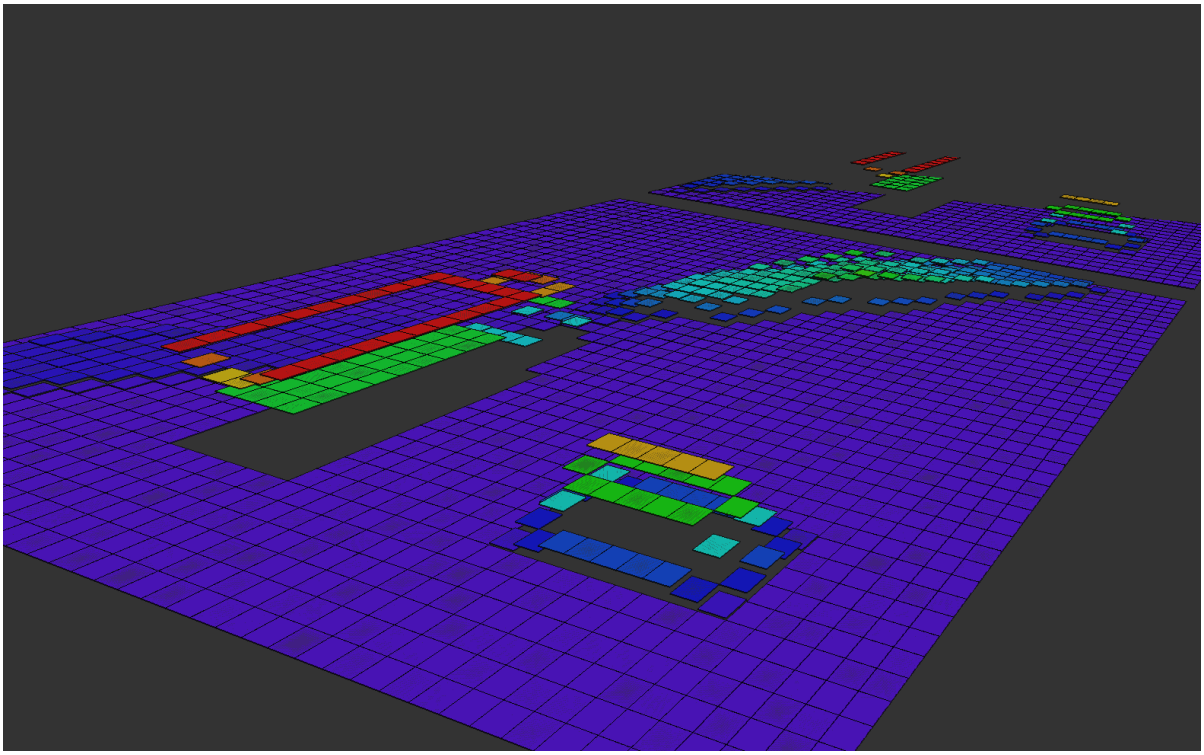
Robotics Research Lab
Department of Computer Science
University of Kaiserslautern

Bachelor's Thesis



A Versatile Spatiotemporal Working Memory for Autonomous Robots

Felix Laufer

Contents

1 Introduction and Motivation **1**
- 1.1 THOR Project . . . 2
- 1.2 Goal . . . 3

2 Computational Intelligence and Memory **5**
- 2.1 Two Competing but Complementary Paradigms . . . 6
  - 2.1.1 Symbolic Approach . . . 6
  - 2.1.2 Connectionist Approach . . . 8
  - 2.1.3 Overview and Comparison . . . 11
- 2.2 Memories and their Purposes . . . 12
- 2.3 Cognition from the Perspective of Computational Vision . . . 13

3 Related Work **15**
- 3.1 Convolutional Neural Networks . . . 15
  - 3.1.1 Origins and Motives . . . 15
  - 3.1.2 Architecture and Functional Principles . . . 16
  - 3.1.3 Applications and Implementation . . . 18
- 3.2 Cognitive Maps Architecture . . . 18
  - 3.2.1 Basic Approach and Building Blocks . . . 19
  - 3.2.2 Aspect Maps . . . 19
  - 3.2.3 Processing . . . 21
  - 3.2.4 Data Flow and Data Hierarchy . . . 24
  - 3.2.5 Example Network . . . 26
- 3.3 Summary and Evaluation . . . 26

4 Concept **29**
- 4.1 Discretization of Information and Grid Aspect Definition . . . 30
- 4.2 Structure of an Aspect Working Memory . . . 31
- 4.3 Forgetting . . . 33
  - 4.3.1 Temporal Filtering . . . 33
  - 4.3.2 Temporal Subsampling . . . 34
  - 4.3.3 Spatial Filtering . . . 35
  - 4.3.4 Spatial Subsampling . . . 36
  - 4.3.5 Summary — The Entire Forgetting Process . . . 37
- 4.4 Remembering . . . 38
  - 4.4.1 A Naive Approach: Brute Force Template Matching . . . 39
  - 4.4.2 An Efficient Method: Phase Correlation . . . 39
  - 4.4.3 Rotation-Invariant Phase Correlation . . . 41
  - 4.4.4 Summary — The Entire Remembering Process . . . 42

5 Implementation **45**
5.1 Preliminary Remarks . . . 45
5.1.1 Finroc Framework . . . 45
5.1.2 Parallel Processing . . . 46
5.2 Grid Aspect Implementation . . . 48
5.2.1 Grid Aspect Maps . . . 48
5.2.2 Grid Aspect Processing . . . 49
5.3 Architecture of the Aspect Memory . . . 52
5.4 Forgetting . . . 53
5.4.1 Separable Convolution Filters . . . 54
5.4.2 Fast Fourier Convolution Filters . . . 54
5.4.3 Practical Considerations: Hybrid Approach . . . 55
5.5 Remembering . . . 55
5.5.1 Batched Separable Fast Fourier Transforms on the GPU . . . 56
5.5.2 Parallel Reduction Operations on the GPU . . . 57
5.5.3 Rotations on the GPU . . . 58

6 Experiments and Results **59**
6.1 Functional Tests . . . 59
6.1.1 Forgetting . . . 59
6.1.2 Remembering . . . 63
6.2 Runtime Tests . . . 64
6.2.1 Forgetting . . . 64
6.2.2 Remembering . . . 68
6.3 Aspect Memory Integration Test . . . 69
6.4 Advanced Application: Motion Detection and Optical Flow . . . 71

7 Conclusion **73**
7.1 Evaluation . . . 73
7.2 Outlook . . . 74

Bibliography **77**

A Appendix **83**
A.1 Related Work . . . 83
A.1.1 Relations between Cognitive Maps Architecture, ANNs and CNNs . 83
A.1.2 Hybrid and Unaligned Aspect Map Combinations . . . 84
A.2 Implementation . . . 85
A.2.1 Separable Convolution Filters . . . 85
A.2.2 Separability of the Fourier Transform . . . 85
A.2.3 Bilinear Interpolation . . . 85
A.3 Experiments . . . 86
A.3.1 Accumulation using Weighted Additions . . . 86
A.3.2 Noise Compensation using Spatial Smoothing . . . 86
A.3.3 Runtime Results . . . 87

1. Introduction and Motivation

Over recent decades, remarkable successes have been achieved in the continually growing field of autonomous robotics. One of the most conspicuous examples of technological evolution in this domain might be the driverless car. Nevertheless, there are numerous fields of applications for autonomous robot systems executing their tasks in markedly less structured environments, such as agricultural areas, construction sites or danger zones. A major vision for current and future research is to increase the robots' degree of autonomy when interacting with each other and their complex surroundings, and likewise to reduce the need of human guidance. Evidently, even fundamentally intelligent behavior such as pathfinding, safe navigation and collision avoidance in unstructured and changing environments require extensive perception abilities. Managing more abstract or high-level constructive tasks, accordingly, involves acquiring detailed insight about the surrounding, reacting to external events and evaluating the system's own behavior with due regard to the desired goals.

Preferable capabilities associated with true intelligence such as judging, reasoning, reflection, directed attention, forgetting and remembering require the ability to retain and retrieve information. As the latter are inherent features of memories, it is not surprising that cognitive science agrees on a strong linkage between intelligence and memory (see [Colom 08]). Autonomous robots must in any case keep an appropriate internal representation of their operational space which is traditionally implemented using different types of maps. In that context, one might be inclined to read the term "map" as a synonym of a robot's working memory, emphasizing the fact that its contents are continually updated depending on the surroundings being central for the robot's perception.

However, one could argue that truly intelligent behavior arises from observations of the environment over time, which implies the need for a memory storing not only recent knowledge but also previously perceived data. In a first approximation, an extended working memory's structure can be characterized by a temporal axis, adding an entirely new dimension to the basically spatially conditioned maps—whether concerning metrical, topological or hybrid maps. This distinct dimension is fundamental to temporal retrieval queries for filtering, classification, recognition or learning tasks and helps to overcome the limitations of purely reactive behavior.

Whereas the aforementioned motivation has been focused on functional aspects regarding intelligent behavior, some considerable non-functional benefits of implementing a memory can be identified additionally. As nowadays a variety of highly integrated but yet reasonably priced sensors offer the opportunity to collect vast amounts of required environmental information, a major challenge is to process, store and maintain all this data. From a software engineering point of view, a memory architecture can be seen as a framework providing access to that data in an efficient, structured and centralized manner. Distributed robot systems composed of several subsystems that process the same data or exchange intermediate results can profit by such a shared data source.

Beyond these two classes of requirements, there is finally the crucial challenge for intelligent autonomous systems to deduce higher-order knowledge from various raw sensory data; namely the problem of perception, knowledge extraction and reasoning itself. This might be technically and architecturally supported by an above-mentioned memory framework but is thereby essentially not yet solved. Concepts in the area of artificial intelligence such as symbolic reasoning, neural or behavior based networks, on the other hand, provide appropriate solutions for this purpose. Thus, it may be advisable to investigate approaches which integrate or at least guide the implementation of those requirements aiming for an overall framework or so called cognitive architecture.

Regardless of any attempt to emulate the relation between biologically motivated intelligence and memories, a "memory" is also simply a fundamental technical concept to store and maintain data and include past information in current computations. In order not to get confused by the different meanings, the term *working memory*[1] will be used in this thesis, indicating the intended application as a robot memory architecture which provides more than basic data storage and manipulation capabilities. Emphasis should be consequently laid on changing sensory and operational data as well as on more"intelligent" (to be determined) processing features.

1.1 THOR Project

`THOR`—short for Terraforming Heavy Outdoor Robot—is a cooperative long-term research project of the Robotics Research Lab at the University of Kaiserslautern and Volvo CE, Konz. The overall objective is the development of an fully autonomous bucket excavator for typical landscaping tasks based on the mobile excavator `Volvo EW-180B`. To this end, the machine is equipped with sensors, particularly two planar laser scanners, actuated with electro-hydraulic valves and controlled by a software system. Figure 1.1 shows `THOR` in action, performing a usual loading task autonomously.

Previous work addressed the interconnection between the control architecture and the basic mechatronic machine model [Schmidt 13], the bucket and arm control [Pluzhnikov 12] and the development of a simulation and environment model [Schmidt 10] having regard to soil particles [Schmidt 09] allowing for testing and verification without the use of the physical machine. A behavior-based `iB2C` control system has been implemented [Proetzsch 10], which provides `THOR` with a reactive behavior in order to dynamically deal with changes and disturbances within its surroundings.

[1] Although being not consistent with the term "working memory" as the description of a memory model introduced by [Baddeley 74], cognitive science occasionally uses the term in a sense closely related to the reading mentioned before.

Figure 1.1: THOR bucket excavator autonomously loading up a construction site truck.

One focal point of current research besides the biologically motivated reactive approach is the development of advanced perception methods. Inspired by human perception, rather simple and lower level raw data gathered from various sensory sources is to be processed, acquired and combined with the aim of obtaining a sufficient environmental model with higher level information. As part of that, the two laser scanners which were primarily intended for reasons of safety, stopping the machine as soon as humans enter the workspace of the excavator arm, can be utilized to collect point clouds resulting in elevation maps of the currently processed soil surface. These planar scanners are mounted vertically aligned on both sides of the cabin such that a three-dimensional point cloud must be successively aggregated during rotational movements of the upper part of the excavator. [Zolynski 12] describes in this context the gathering of point clouds in dynamic surroundings.

This application is paradigmatic for an autonomous system which highly interacts with its environment and must properly fuse variant sensory data in order to keep track of the scene. Thus, a working memory as mentioned above might provide an architecturally and functional basis for extensive perception approaches as required within this project.

1.2 Goal

This thesis suggests a working memory approach based on a Cognitive Maps Architecture introduced by [Zolynski 15] (dissertation, to be published) which integrates both map-like data storage and processing features in the sense of elementary, biologically inspired perception and deduction. The underlying principal paper emphasizes the successive deduction of higher knowledge by means of a data flow network connecting different data sources and storage with simple computational operations. By contrast, this work will focus on basic approaches to actively store, forget and remember homogeneous but time-variant data from the same sources. Rather technically speaking, an aforementioned time axis will be added to the spatial cognitive maps, allowing for efficient fuzzification or generalization of previously received data and recognition of patterns therein. Later, those space and time dimensional memory units may be reconnected again and support

additional explicit temporal processing. Since such a distributed memory structure should be expected to become complex and computationally expensive, this work will aim for an efficient and, if advisable, a highly parallel GPGPU[2] implementation.

[2]General-purpose computing on graphics processing units: utilization of a GPU for applications traditionally handled by the CPU in order to exploit the parallel hardware structure for parallel (parallelizable) algorithms.

2. Computational Intelligence and Memory

Albeit affiliating with the engineering point of view aiming to contribute to the development of intelligently operating robots, the quite opposing incentive of cognitive science, trying to comprehend, model and emulate biological intelligence, may not be omitted. The following chapter will briefly outline fundamental concepts of *computational intelligence* and point out certain, not always obviously related aspects of temporal knowledge and working memories. This part is not an indispensable presupposition for comprehending the ideas proposed within this thesis. But for the interested reader it might illustrate the path of motivation and development as to the essence of this work. Moreover, an extensive fundamental discussion is helpful in order to assess the strengths and weaknesses of the established approaches.

Numerous concepts and specific implementations of artificially intelligent systems, cognitive frameworks or architectures have been elaborated and investigated in various disciplines and from different perspectives and motivations. Whenever there is talk of "artificial intelligence", one should differentiate between *weak* and *strong artificial intelligence* (see [Russell 03]): whereas the former one focuses on developing systems or machines whose acting and behavior is justified to be called intelligent from the outside (human) observer, the second one pursues the vision to animate them with true, inherent consciousness. While the latter attempt is still poorly charted territory and questionable due to the mere controversial concept of artificial consciousness, weak AI achieved very encouraging successes. The term "intelligence" within the scope of this work addresses some of the weak methods and will employ them.

The Challenge of Intelligence from a Weak AI Viewpoint

The modern self-conception of AI is the challenge of creating intelligent agents where the term "agent" denotes an entity interacting with its environment in such a manner as to perceive information, make rational decisions, execute actions and possibly change the environment (see [Russell 03], chap. 2). This definition is quite compliant with robotics since it immediately suggests a cyber-physical system and allows to replace "agent" by the

term “autonomous robot”. Basically, two fundamental concepts of intelligence arise from that definition: logically correct, target-oriented *decision making* based on the currently available information on the one hand and *cognition* as both the awareness of the own internal state and the process of perceiving environmental information on the other hand.

Decision making is certainly highly dependent on the agent’s superordinated goals or its intended purpose and occurs on a tactical or strategic layer whereas perception is literally an underlying and fundamental task. It is exactly that contribution to intelligence that provides an agent with the required (pre)processed information from its surroundings. Admittedly, it may appear that this is a simple and straightforward process of data preparation but when considering the human visual perception, the necessity of complex mechanisms such as feature extraction, pattern and object recognition becomes apparent. Human perception suggests the assumption that plenty of spadework is done within the perceiving process and also encourages the attempt to implement sophisticated artificial techniques in order to support higher-level reasoning (see [Wilson 01], chap. 2, [Russell 03], chap. 1).

2.1 Two Competing but Complementary Paradigms

The two most prevalent approaches in artificial intelligence are the established *symbolic* paradigm, often referred to as the traditional or classical AI, and the more recently revived but actually prior *connectionist* paradigm. While giving an overview on those seemingly conflicting approaches, not only their relation to respective concepts of memory will become apparent. In fact, it will turn out that both concepts are rather complementary and contribute essential ideas in order to solve general problems in practice.

2.1.1 Symbolic Approach

The classical symbolism originates in the physical symbol system hypothesis formulated by Newell and Simon [Newell 76] which claims that “a physical symbol system has the necessary and sufficient means for general intelligent action”. Herein the physical symbol system is specified as a system mapping real world objects or, more generally, physical entities to so called *symbols* that can again occur as components of *expressions*. Thus, representative discrete symbols as primitives can be combined syntactically to formulate propositions organized in arbitrary complex taxonomies and allow for a modeling of real world aspects, physical rules, factual knowledge, etc. Coherent symbol and expression manipulations within this structure conduce to new conclusions or *inferences* which form the successive steps of an overall problem solving mechanism. Symbolic approaches often represent knowledge in tree-like structures, where symbols correspond to nodes and their relations to interconnecting links.

Reasoning as a Search

Symbolic reasoning corresponds to *deductive inference* meaning that conclusions can be deduced from a set of premises through the application of certain *inference rules*. The figurative thinking process of sequential inferences consequently corresponds to a search within the knowledge encoding structure. Each inferring step across the structure leads to a new *state* of the problem solving process, where inferring actually implies expedient

branching at links of the graph or, in other words, applying a valid *rule* or *operator*. Those steps must be repeatedly executed until either a goal state, likewise marking the solution, or a state without any applicable operators, indicating the unsolvability of the problem, is reached (see [Sun 99]). Hence, it seems obvious that incipient approaches used well known graph search methods such as depth-first search and breadth-first search in order to inspect the entire state space, hopefully resulting in a path to the solution. However, not only are those techniques computationally expensive for major search spaces, not uncommonly they are even simply lost in the combinatorial explosion of problem structures that are far too complex to be exhaustively explored within decent time. Instead of a brute force approach, *heuristics* proved beneficial to control and abbreviate the search by choosing promising shortcuts on the path through the search space. One prominent example is the *means-end analysis* which aims to recursively reduce the difference between the current state and the goal state or, to put it another way, tries to reach subgoals at first by removing just those obstacles in the way between the actual and the target state (see [Newell 59]).

Knowledge Representation and Limitations

Basically, symbolic systems initially relied on logic or, more specifically, first-order-logic, which is quite expressional but grows cumbersome as knowledge increases or gets indefinite. Where appropriate, the all-purpose principle of reasoning is abandoned in favor of simpler description languages right down to knowledge management systems such as smart databases (see [Wilson 01]). Although various *representations* have been developed, almost all symbolic reasoning models share the substantial requirement of strictly formalizable knowledge and goals. What seems to be possible in a straightforward manner for inherently formal problems such as mathematical proofs, chess or other well comprehensible decision problems, turns out to be exceedingly difficult for most real world problems that feature virtually unrestricted spaces of actions and environment.

Another characteristic of actual, naturalistic problems (in robotics) is the necessity to deal with incomplete and noisy information, uncertainties and probabilities instead of only discrete entities and rigid rules. Newell and Simon experienced those limitations when developing their software called *general problem solver* and have been criticized for their technique operating less generally than its name suggested (see [McDermott 76]). Nevertheless, there are well known approaches aiming to slightly "de-discretize" the internal representation of knowledge and reasoning in order to incorporate the fuzzy nature of the real world and human reasoning therein. To these belong the *fuzzy logic* (see [Zadeh 94]) and *probabilistic approaches* (see [Jain 09]). The former admits vagueness to conceptual symbols such as "hot" and "cold" assigning a rate of membership to a specific thermal measurement and hence allowing for a continual logic whereas the latter adds probabilities to certain beliefs resulting in states that are more attractive for inference than others. For instance, diagnostic systems typically require additional probabilistic information in order to come to useful results: a sensitive smoke detector in a public building might likely give an alarm signal because of cigarette smoke, yet the case of emergency must be excluded.

Temporal Knowledge

The preliminary findings have not considered any temporal nature of knowledge, although in practice information is usually tied to points in time or is even only valid for a certain

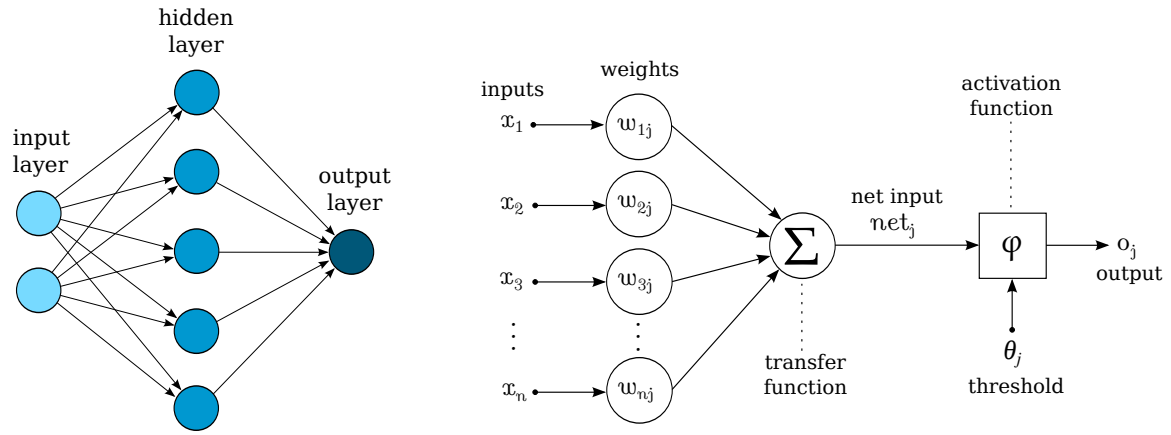
period of time. Two basic cases of application are conceivable: knowledge from various points in time must be maintained, taking account of temporal meta information such as timestamps that identify moments of adding, editing, removing, etc., which is usually the case for databases. Secondly, knowledge can be temporally constrained from its own accord as is the case with the proposition "the roadway will be wet with rain *until* the sun shines *for at least half an hour*". In such cases it would be desirable to have the possibility of logical inferring with regard to those temporal modifiers. Arthur Prior [Prior 03] established the basis of *temporal logic* and proposed his concept named *tense logic* as a variant of modal logic. Alternatively, temporal logic could be emulated using classical logic in addition of temporal meta information associated with symbols and inferring rules (see [Borges 12]). Anyway, maintaining temporal knowledge in the scope of symbolic reasoning suggests a possibly large time frame of relevant data and thus advises to make use of memories.

2.1.2 Connectionist Approach

The roots of connectionism reach back to the early 1940s when McCulloch and Pitts [McCulloch 88] adapted the nature of the highly interconnected human brain to an abstract and simplistic model. The basic modules—*neurons*—of such an *artificial neural network* consist of units with several incoming binary signals and one output signal. The weighted sum of all input signals is compared to a certain threshold. Another famous and simple concept of neural networks is the so called *perceptron* proposed by Rosenblatt [Rosenblatt 58] which is likewise built of weighted inputs that are aggregated and offset. A network composed of those units is capable of simulating any logical function or, equivalently, building a complete basis of boolean algebra.

Although this approach is reminiscent of the symbolic knowledge encoding structure, it must be mentioned that the *data-flow-oriented* model composed of a vast number of simple units differs essentially from the symbolic paradigm. This criterion is also immanent in models aside from various types of pure artificial neural networks and emphasizes the superordinated term "connectionism". Due to the *massively parallel* structure, reasoning in the connectionist sense is not as sequential as with symbolic inferring but rather simultaneous. Instead of taking local steps from one to another adjacent state, connectionist problem solving means striding through various concurrent paths of the entire network. Therefore the term reasoning should be better replaced by *processing*. Not only the activity of a particular problem solving process is distributed over many parts of the network but also the encoded knowledge is, which makes it difficult to reconstruct the reasons for a final result subsequently (see [Haykin 98]).

Connectionism experienced a renaissance in the 1980s driven by the book "Parallel Distributed Processing" by McClelland and Rumelhart [McClelland 87] and yielded a range of models and variants since then. They mostly share the neural principle but differ for example in accepting continuous instead of discrete signals, in the type of activation functions controlling the output signals or in the structure of the overall network (e.g. multi-layered, feedback). A few years back, the work of Minsky and Papert [Minsky 87] had shown that the basic perceptron was not capable of simulating the logical XOR function—and actually only linearly separable functions in general. Initially, this difficulty led to a great discouragement but has finally been coped with by introducing an additional,



(a) Schematic structure of a multilayer neural network with an input layer, a hidden layer and an output layer.

(b) Internal model of an artificial neuron: $o_j = \varphi(\sum_{i=1}^{n} x_i w_{ij} - \theta_j)$

Figure 2.1: Generic model of an artificial neural network

so called *hidden layer* to the network. Most models of artificial neurons can be expressed by the scheme shown in figure 2.1, where 2.1a shows a generic structure of neurons arranged in three layers: the input layer containing all neurons directly linked with input signals, the output layer respectively linked with outgoing signals and an inner hidden layer with externally inaccessible neurons. Figure 2.1b shows the inner formula of a single neuron as the sum of all weighted inputs, offset with a constant bias or threshold and finally fed into an activation function resulting in the output.

Learning and Knowledge Representation

In contrast to the symbolic approach, concepts of connectionism have always been intertwined with the issue of machine learning or autonomous, empirical knowledge acquisition. The idea of connectionist learning is the inversion of the deductive principle—thus called induction—such that general rules are derived from exemplary observations. Since learning is to some extent also the inverse process of reasoning, it operates non-locally and massively parallel as well, which implies that an explicit modeling of knowledge is hardly possible. It should be noted that learning in some areas of application might be in fact the only way to establish an intelligent system whether because of the complexity of knowledge itself or its encoding structure. Technically, it is implemented based on the *Hebbian theory* and emulates changes to the synaptic signal transmission between neurons by adapting the edge weights of the corresponding connected units (see [Hebb 02]). To put it simply, learning changes the distributed parameters of a network in such a manner that its computed resulting outputs successively satisfy certain given outputs in the hope that the true, underlying cause-effect relationships can be inferred and later on successfully applied to unknown data. In the scope of connectionism, learning takes the place of the deliberate knowledge modeling known from symbolism and is inextricably linked with representation—and for this very reason this paragraph has got its distinct superscription (see [Sun 99, Wilson 01]). For the sake of completeness, it should be mentioned that there

exist a few approaches that allow a local knowledge representation resulting in network units which process rather complete data structures instead of single, unrelated values (see [Lange 89, Sun 99]).

The majority of learning algorithms correspond to network-wide optimization problems and are concerned about minimizing the errors between actual and desired results by updating the parameters (weights) of units involved in computation. The best known example is the *back-propagation algorithm* which controls the weight tuning process through multiple network layers. An optimization step itself is mostly based on the *gradient descent* method trying to shrink the remaining error in terms of a particular desired outcome—a subset of the encoded total knowledge. The complete procedure called *training* consists of a number of the former optimization steps (see [Haykin 98]).

A distinction is made between three categories of learning algorithms in respect of the used type of *feedback* (see [Russell 03, Sun 99, Wilson 01]): *supervised*, *unsupervised* and *reinforcement* learning. Presenting a definite favored outcome for a fixed input as mentioned above belongs to the class of supervised learning and requires always tuples of defined inputs and target outputs. A simple example might be a logic AND gate trained with the following (complete) set of *training data*: $(\{0,0\},0),(\{0,1\},0),(\{1,0\},0),(\{1,1\},1)$. However, a typical real world application is optical character recognition where a presented image is to be classified as an alphabetic character. Unsupervised algorithms, by contrast, give no feedback about the computation results at all but aim to identify structures or features within data without giving any guarantees of whether those findings are useful or not. Metrics of features being watched for, as for instance the Euclidean distance of data points, are explicitly described in such algorithms. Clustering is thus a classical representative of this class. Reinforcement learning, finally, does not employ any target result but rates the actual output according to certain criteria, thus giving back a grade but no details on how to improve the processing. A function that maps the output to a scalar-valued rating is sometimes called fitness function. This approach is favored in the field of evolutionary algorithms. Genetic algorithms, for instance, modify (parts of) the solution by repeated selection, mutation and recombination and receive their feedback only after a while.

Especially the class of supervised algorithms demonstrates that comprehensive knowledge can be learned from only some chosen examples—in many cases it would even not be possible to provide a complete training set. In combination with the high degree of parallelism of the underlying network, this allows for the strong generalization and abstraction capabilities and noise robustness which are characteristic of connectionist models. Thus, a primary feature of neural networks is the inherent ability to manage fuzzy, incomplete and noisy data in marked contrast to the symbolic models where respective mechanisms must be intentionally integrated. However, it must be clearly stated that these features highly depend on the quality, selection and amount of training data. A too extensive training procedure with a fixed data set soon results in undesirable effects such as curve fitting where the network perfectly reproduces the stated target outputs but looses any generalization abilities regarding new, unknown data (see [Tetko 95, Haykin 98]).

Temporal Knowledge

Temporal knowledge, in the sense of temporally annotated information stored in data base systems, is less of an issue in connectionism due to the complex distributed in-

ternal structure. Instead, temporal data processing is all the more worth considering. In order to include previous information, output signals can be back-coupled to input signals of preceding units, forming cyclic structures also referred to as *recurrent networks* (see [Haykin 98, Borges 12]). For deliberately modeling such networks, apart from those recurrent links, so called *delay units* are introduced. Holding back their incoming data for a certain number of time units, they implement single memory cells or, in their entirety, a kind of short-term memory. Similar concepts can be found in all fields of signal processing as for example with infinite impulse response filters. More vividly, the output value of a neuron is fed back to the input of a neuron of the same or a previous layer and affects in this way the computations of all layers starting from the injected layer. Thus, the computational behavior of the network is not only depended on current but also on previous input values. By this means, temporal data dependencies are implicitly included and allow for universal computing in contrast to classical *feed-forward* networks (see [Siegelmann 91]).

2.1.3 Overview and Comparison

Apparently, models work just as well as they match their scope of application: therefore, higher level, closely comprehensible, strategic problems including for instance planning tasks, which concern complete, intact and definite data, are suitable applications for symbolic approaches. Their typical area of responsibility is thus the decision making part of intelligence. In contrast, connectionist principles naturally show their strengths in lower level (pre)processing tasks of noisy and incomplete data, where knowledge is learned from certain observations. So, they can be assigned to the subdomain of cognition and perception. The two opposing principles of inference, deduction and induction, correspond moreover to different architectural conceptions: the deliberate, top-down processing and the reactive, bottom-up processing, respectively. These are also closely related to the particular methodologies of knowledge acquisition, namely reasoning as an act of deriving concrete conclusions on the one hand and learning as a process of generalization and abstraction on the other hand. Those characteristics and others mentioned above are once again concisely compared in table 2.1.

Points of Intersection between Symbolism and Connectionism

However contrary the paradigms at a first glance may appear, indeed, they feature some interesting points of intersection. Within the scope of both approaches, techniques have been developed to overcome the own weaknesses by adapting advantages of the respective other. Examples include fuzzy logic and probabilistic approaches on the part of symbolism as well as attempts to apply primal connectionist learning algorithms to symbolic knowledge structures. Connectionism provides exactly those preferable properties, symbolic methods struggle with. Conversely, symbolism requires in the first place some preprocessing in order to transform any analog real world data to definite, discrete symbols (see [Wilson 01]). It seems quite idealistic to expect that raw data is at any time as noiseless and comprehensive as to be useful within the strict, logic sphere of symbolism—especially in the field of robotics. Instead, it might be appropriate to combine both principles resulting in a connectionist subsystem which processes sensory data as far as possible and passes hereby obtained higher level knowledge through to another symbolic reasoning subsystem.

Strong capabilities of knowledge acquisition and robustness enable connectionist models to undertake the cognitive (perceptive) part of artificial intelligence. Referring to the lower

Characterization	Symbolism	Connectionism
concept of intelligence:	decision making	cognition, perception
type of inference:	deductive	inductive
strengths of thinking process:	deriving conclusions	generalization, abstraction
direction of operation:	top-down	bottom-up
principle of cognition:	deliberate	reactive
driving flow of information:	control flow	data flow
knowledge acquisition:	reasoning	learning
knowledge structure:	tree-like	distributed units
knowledge cohesion:	local	non-local / global
knowledge deployment:	centralized	massively parallel
process operation:	sequential	concurrent
technical implementation:	symbolic manipulations	arithmetical processing
quality of suitable data:	definite, plain, complete	fuzzy, noisy, incomplete
type of associated memory:	rather long-term	(very) short-term

Table 2.1: Characteristics of symbolic and connectionist approaches in comparison.

level of abstraction and a more granular representation of data, connectionism is also known as *subsymbolic* approach. At the interface between both paradigms, two classes of several complementary conceptions are brought into contact as can be seen from table 2.1. Therefore, the boundary between subsymbolism and symbolism can also be considered as a representative distinction between the two branches of weak artificial intelligence and their associated types of memory, requiring a distinct dimensioning of the temporal axis. These types are the short-term memory storing contemporary information for the purposes of perceptive processing and a rather long-term memory required to maintain more persistent symbolic knowledge. The next section of this chapter explains related issues of memory in detail.

2.2 Memories and their Purposes

In psychology and cognitive science memory models often distinguish roughly between *short-term* and *long-term* memory depending on the period of time information can be stored. An early cognitive model of memory proposed by Atkinson and Shiffrin [Atkinson 68] adds another type of very limited storing duration and capacity called *sensory memory* which is beyond conscious control and allows to memorize currently perceived information. From a robotics point of view, this term suggests a kind of memory supporting typical (pre)processing tasks on incoming sensory data such as filtering or accumulation. Actually, data processing up to a derived elementary behavior relates to a rather short-term memory whereas strategic reasoning and decision making requires much longer durations of storing and data maintenance. So, issues such as early perception and thus forgetting and fuzzification are associated with the former type and higher level reasoning depends on more abstract, already learned fundamental knowledge retained in the long-term memory.

Connectionism and symbolism and their respective relation to memories coincides with the above mentioned findings. However, both paradigms address memories first and foremost

at best as a necessity of secondary importance. This may originate from the fact that some typical applications have only implicit temporal attributes but involve no *time-invariant* data. For instance, a trained feed-forward network for pattern recognition is steady and stateless and changes the output whenever a new input is available regardless of previous data. It is therefore in the end just a (potentially complex) mapping of input signals to output signals. Of course, also a pure feed-forward network can incorporate temporal input data to some extent by adding another delayed input to a respective undelayed signal. But this method only increases the network's structural complexity without a semantic, temporal differentiation of the inputs. This method might be called temporal signal feeding but is far from modeling temporally dependent intermediate states inside the network that would allow for true temporal processing.

On the contrary, other applications admittedly require back-coupling computationally but need not to keep their computations available for a long period of time. Likewise, typical logical reasoning involves universal knowledge without any temporal dependencies. As opposed to this a cognitive architecture implementing those approaches must very well maintain knowledge for a certain time depending on its purpose. With this in mind, the structure of an appropriate memory should be brought into focus whereat two dimensions can be identified: *time and content* (see [Wienke 10]). An explicit representation and graduation of a *temporal axis* is frequently neglected in models grounded on subsymbolic or symbolic methods. Introducing such a distinct dimension and expanding these models along the temporal axis, places special emphasis on the representation and processing of *time-variant knowledge*.

The basic idea of this work is to assign a central role to these memory related architectural issues. Ideally, actual temporal data processing can be distinguished from passive data storage, related, granular contents are kept local and encapsulated and the depth of the memory can be adapted for the needs without yet forfeiting favorable subsymbolic conceptions in particular. The working memory targeted in this thesis can be considered as a very short- to medium-term memory originated in the idea of a sensory memory but temporally and functionally enhanced, trying to evolve its favorable features towards computational cognition as far as possible and reasonable. Eventually, it aims to overcome the gaps between very basic data processing, more advanced perception and a higher level symbolic representation.

2.3 Cognition from the Perspective of Computational Vision

Human visual cognition is highly developed and often quite ahead of the remaining sensory perception. Actually, even non-visual problems, relations and facts are deliberately visualized in order to obtain a clearer understanding. Certainly obvious are the influences on inherently visual problems such as spatial perception and orientation which represent, taken by themselves, considerable parts of human cognition. *Cognitive maps* are an example of visualized knowledge which is reduced to the essential and devoid of details and strict scales (see [Kitchin 94]). For instance, people have cognitive maps of their hometowns in mind, which are sufficient to find their way trough the streets by foot or by car, but differ fundamentally from metrical city maps. Instead of exact distances

between city blocks or angles between crossing streets, only the existence of these turn-offs and their rough direction is essential. Likewise, subjectively relevant areas such as the residential neighborhood might contain more detailed knowledge than rarely visited parts of the city. This example covers manifold aspects of visual perception, representation and data retention: precise information must be visually perceived, processed on demand and stored in a visually retrievable representation where knowledge can be partial, incomplete and fuzzy. This raises the question of whether these markedly human abilities of visual thinking and mind can be transferred to engineering techniques.

Technically, approaches of computational vision require a mapping—or rather a projection—of real visual scenes onto a 2-dimensional image (see [Wilson 01]) where visual characteristics can be analyzed in terms of methods such as lower level filtering operations or more complex clustering, segmentation, feature extraction or object recognition algorithms. These allow for an interpretation of the actual scene by taking account of certain aspects and are thus another kind of information extraction. When considering subsymbolic unsupervised learning, an immediate overlapping with those typical issues of computer vision becomes apparent. Also, from this computer vision point of view, there are analogies between basic image processing and subsymbolism on one hand and higher level recognition algorithms and symbolic representations on the other hand. Even so, it is worth to mention that in this context there are some techniques providing translations from raw information such as pixel data into abstract entities such as clusters or objects which can also be understood as symbols.

Image processing and computer vision in general are not subject to restrictions regarding the projection plane since the respective pixels, a real visual scene is mapped onto, can easily be converted into another (world) coordinate system if necessary. When trying to take advantage of suchlike methods, especially within the scope of robotics, it might yet be favorable to align the image with the horizontal plane of the robot's world model. In this way, an image can be interpreted as a spatial map (and vice versa), representing the most significant sphere of action of most autonomous robots from an aerial perspective. Interestingly, this assumption is quite consistent with the structure of biological cognitive maps encoding particular contents within a 2-dimensional plane rather than the experiencable 3-dimensional space (see [Ulanovsky 11, Hayman 11]). The third dimension's coordinates can be additionally stored with the actual information if necessary or, in case of altitude data, it even represents the actual payload itself. Approved methods of image processing and related fields can therefore be applied to the projected spatial map and, in contrast to pure subsymbolic approaches, their effects are immediately visible or at least easily visualizable. As indicated above, pattern or object recognition provides in theory a solution to transform parts of a subsymbolic representation to abstract symbols. The solution explained within the next chapters addresses this idea and combines it with a working memory as elaborated in the section before.

3. Related Work

3.1 Convolutional Neural Networks

An emerging special type of feed-forward networks, addressing the visual principles mentioned in section 2.3 on visual cognition, is the *Convolutional Neural Network*. This approach will show essential similarities to the concept of a Cognitive Maps Architecture introduced later but contributes a substantially different strategy to learn the network's processing structure from the presented data in a hybrid unsupervised and supervised manner.

3.1.1 Origins and Motives

Convolutional Neural Networks were originally introduced by Fukushima [Fukushima 80, Fukushima 88] in 1980 and considerably advanced by LeCun et al. [LeCun 98]. This brief introduction will mainly focus on the latter source although the approach has been generalized further in recent years [Simard 03] and found entrance into speech recognition and big data challenges establishing the state-of-the-art domain of *deep learning* [Bengio 09]. Developed for the purpose of optical recognition tasks such as character recognition, the *CNN* is intuitively inspired by biological visual perception—or, more precisely, principles of the animal visual cortex [Hubel 68].

The challenging handprint character recognition task entails the problem of classifying presented images into characters of a given alphabet. A possible approach might consist of a manually designed preprocessing, extracting relevant subareas such as edges or line end points—also called *features*—while other image information is ignored. The set of identified features—the *feature vector*—can then be fed into a common trainable neural network mapping specific feature vectors to final character categories. However, a designated feature extraction chain is inflexible on the one hand and not necessarily easy to implement on the other hand since relevant features might be only deficiently known. Both arguments give point to learn the feature extraction process directly from presented data which could be also realized by use of a neural network. While this obvious approach might be quite successful, several methodical drawbacks must be accepted:

Fully-connected networks cannot take into account the spatial locality of the input image since every input pixel is handled independently as a separate variable. In fact, neighboring image pixels are certainly correlated, forming visual features of a higher semantic level, which is why feature extraction is considerably helpful in the first place [Trier 96]. Neglecting this property is moreover aggravating as connections of distant (and thus rather uncorrelated) pixels increase the network's complexity tremendously in terms of free parameters without substantially contributing to classification results. But yet, all useless parameters or weights must be adequately trained and require an accordingly extensive training set. For that reason, it would be wiser to prohibit non-local relations in advance, likewise reducing the network's parameter capacity (see [Rumelhart 86] chap. 8 and [LeCun 98] chap. 2). CNNs implement the very same idea by definition and map the problem's topological texture onto a proper, locally restricted network.

Real world data is not only noisy but also highly variable and, referring to the motivating example, handwriting may vary from character to character with the result that identical characters appear with different orientations and locations. Since input data must be adapted to the size of an input layer, more or less identical but shifted character images cause conceivable difficulties. Neural networks, indeed, tolerate minor variations but when shifts become large enough, translated images present the system with a seemingly new input. This problem requires structural redundancy within the classical network approach: several neurons, responsible for the detection of the same features located at different spatial positions, are necessary, thus, increasing the network's complexity and training effort (see [Fukushima 80, LeCun 98] chap. 2). CNN techniques such as successive convolution and subsampling cover this issue.

3.1.2 Architecture and Functional Principles

Convolutional Neural Networks employ first and foremost the architectural idea and learning principle of the multilayer perceptron. In distinction from common neural networks, CNNs assert a structural locality by arranging the input signals in a 2-dimensional plane referring to the visual field of animals or humans. This idea is perpetuated through all network layers such that units receive inputs from neighboring units of the previous layer. A neighborhood of spatially correlated inputs is called *receptive field* and allows to detect local features. Since each unit of a certain layer is connected to contiguous regions of the previous layer, units contain redundant information caused by overlappings of the receptive fields of the precedent layer. These overlapping areas ensure that the network's results are shift-invariant to a certain extent which is a highly desirable property within this scope of application.

Convolution Layer: Feature Extraction

Considering a specific feature of interest, several adjacent neural units with weights trained to detect this feature are required to cover the entire input plane. These units share the same weights, are just replicated next to each other and organized in a plane. The set of the plane's outputs—each unit has one output—is the *feature map* in terms of one specific feature and the entire precedent input plane. Due to their identical weights, units on the same plane process identical calculations on different parts of the input and their outputs are fed into a common feature map. The term Convolutional Neural

Networks is derived from this uniform processing method which turns out to correspond to a convolution operation: for calculating each output of a feature map, the corresponding unit applies a kernel composed of the unit's weights to the respective receptive field [1]. As a convolution allows to (figuratively) spread the processing of subareas to the entire image, the structural redundancy of cloned units can be replaced by a single appropriate convolution implementation. Extracting several different features requires multiple planes resulting in the same number of related feature maps. Such a layer, capable of extracting different features, is called *convolutional layer* [Fukushima 80].

Subsampling Layer: Resolution Reduction

Up to this point, each particular feature can be detected at any input position, which is quite similar to the idea of a conventional feature extraction process. In order to find representative compound features distributed over the image plane, additional network layers must combine those interim results. However, the pattern recognition task firstly suggests a preceding subsampling step which reduces the spatial resolution of a feature map. This is based on the assumption that the precise positions of features are less important than their positioning relative to each other. A so called *subsampling layer* implementing the subsampling operation in form of an averaging filter[2] contributes to the system's robustness against shifts and distortions. Since subsampling reduces the resolution of input data, it might be looked upon as a process that counteracts the spatial inflation of feature maps resulting from overlappings within a precedent convolutional layer. Commonly, convolutional and subsampling layers are connected in an alternating sequence. Therefore, the number of feature maps increases with each layer as their resolution decreases. Both effects in combination facilitate the network's tolerance of spatial variations [Fukushima 88, LeCun 98].

Classification and Learning

A *fully-connected layer* with trainable connection weights finally combines all feature maps and performs the actual classification task. It should be pointed out that the weights of all previous locally connected layers are also trainable and, thus, represent an adaptive feature extractor. Back-propagation can be used as a comprehensive training algorithm for the entire network allowing to learn both characteristic features and their relations. Figure 3.1 shows an exemplary architecture with an input image and layers, where each plane depicts a feature map resulting from convolution or subsampling operations. The first convolutional layer extracts 6 distinct features resulting in 6 maps followed by a subsampling layer reducing the maps' resolutions twofold. Again, a pair of both layers increases the number of feature descriptors while reducing the spatial accuracy. All connections between those layers—or their units respectively—are locally restricted. In contrast, the subsequent layer is connected to all outputs of the precedent units and represents the network's classification part together with potential further fully-connected layers [LeCun 98].

[1] To be exact, a feature map's output is the result of the subsequent application of the unit's kernel to the corresponding receptive field, an addition of a bias—the neuron's threshold—and a so called squashing function—similar to a neuron's activation function.

[2] Here, too, the filtering operation is actually followed by a multiplication with a trainable weight, added with a bias and smoothed with an activation function in the fashion of a neural unit.

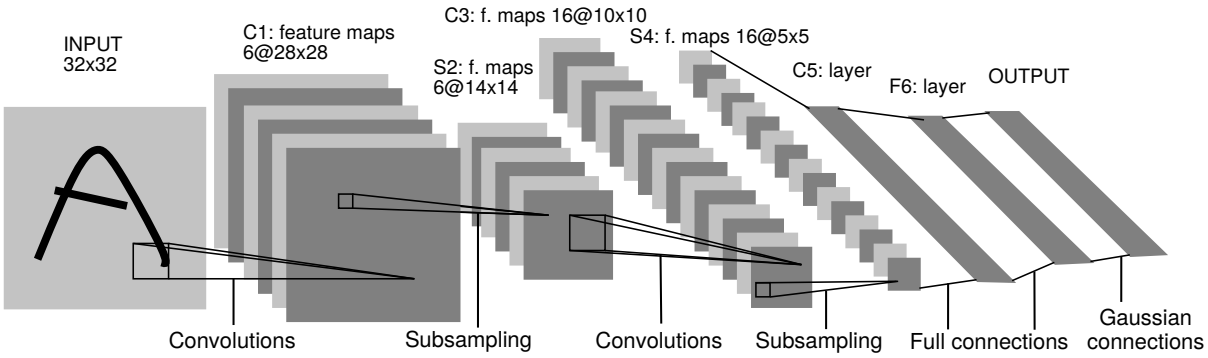


Figure 3.1: Convolutional Neural Network according to the `LeNet-5` architecture. Planes correspond to feature maps, each resulting from a set of units with fixed, identical weights for the purpose of a global detection of a specific feature.[LeCun 98]

3.1.3 Applications and Implementation

Convolutional Neural Networks have been successfully used for various optical recognition tasks such as character, pattern, face or general image recognition but are not restricted to inherently visual applications. The approach is convenient for any problem suitable for subsymbolic methods, which can be moreover mapped onto a 2-dimensional plane. This includes, for instance, phoneme word and speech recognition when transforming raw data into the frequency domain. But also Time-Delay Neural Networks processing 1-dimensional time series or even networks for higher dimensional input data, as in the case with video analysis, can be built based on the principles of CNNs (see [Ji 13, LeCun 98] chap. 2).

Different implementations of CNNs are available such as the `LeNet-5` described above and shown in figure 3.1 developed by LeCun [LeCun 98]. One of the most popular solutions might be the `Caffe` framework [Jia 14] which is basically a `C++` library for building general deep learning architectures and provides `Python` and `MATLAB` interfaces. It also supports `GPGPU` computation in order to take advantage of the parallel nature of neural processing. Due to their spatial, local restrictions, the grid-like arrangement of neural units and the properties of convolution operations, CNNs are eminently suited for a computational deployment on graphics processing units. Another implementation completely based on parallel computing with `C++/CUDA` is `cuda-convnet`[3].

3.2 Cognitive Maps Architecture

This work makes essential use of a Cognitive Maps Architecture proposed by [Zolynski 15] within the scope of the `THOR` project introduced in section 1.1. It will provide both a paradigmatic and a technical basis for a working memory implementation according to most of the favorable characteristics stated in the previous chapters. For this reason, the Cognitive Maps Architecture, as a main inspirational source for this thesis, is paraphrased from the mentioned thesis and discussed in detail at this point.

[3] `http://code.google.com/p/cuda-convnet/`

3.2.1 Basic Approach and Building Blocks

The presented approach is, in short, a data-flow driven, hierarchical network of interconnected components composed of *data storage* and *processing units*. A key feature is the data representation in the form of spatial, cognitive maps retaining a set of spatially and temporally related data which has in its entirety ideally only one semantic meaning or interpretation. For instance, altitude data of an areal consists of many data points but represents the surface of this particular region at a specific time. This data entity, retaining possibly many pieces of elementary information, is called an *Aspect*, indicating the fact that it encapsulates data with a common superordinated purpose. It is hence literally just one of potentially several extractable Aspects in so far as underlying raw data can often be manifoldly interpreted: for example, a gradient field arising from the same data source as the surface data results in another Aspect of steepness. [Zolynski 15] describes an Aspect as a "feature, fact, or facet of the perceived world or of the conclusions about the same". In accordance to cognitive maps, the spatial representation is also based on a projection of real scenes onto a 2-dimensional plane that is easily inspectable and visualizable and also computationally advantageous. Deliberately connected modules for *unary operations* on Aspects and *combinations* of several of them allow for a *top-down design* of desired perception processes incorporating established task-oriented methods and algorithms. But yet, the architecture maintains a natural *bottom-up data flow* starting from Aspects of the lowest data hierarchy level supplied by sensory sources up to cognitive conclusions. The entire approach might be seen as an architectural middle ground between object maps and the parallel processing of neural networks and consequently between the connectionist and symbolic paradigms as well but explicitly leaves out the issue of object recognition and classification—in other words a transformation of representations from the subsymbolic to the symbolic universe. Conceptually, the idea arises clearly from the subsymbolic approach.

3.2.2 Aspect Maps

In order to process Aspect data of various meanings, abstraction levels and data types, a plain but versatile storage format is chosen. To that end, an idea of [Barkowsky 97] and [Berendt 98], to represent individual, semantic Aspects of particular interest contained in cartographic data by the use of spatial *Aspect Maps*, has been revisited and modified: Formally, an Aspect Map A is a 2-dimensional plane that is congruent with the horizontal plane of the robot's world coordinate system and consists of one or more elements, which assign a piece of information (value) to a spatial location or area [Zolynski 15]:

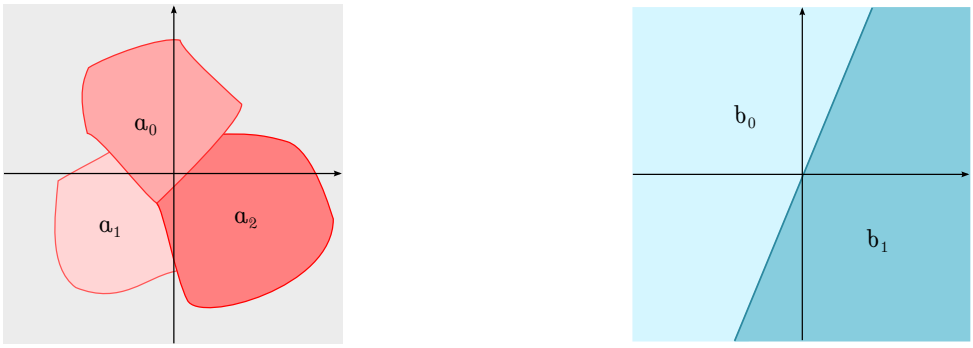
$$A := \bigcup a_i. \tag{3.1}$$

Each element a_i is a tuple comprised of an area Ar_i and a value v_i:

$$a_i := (Ar_i, v_i). \tag{3.2}$$

The area component Ar_i describes both a spatial location and an limited or unlimited extent and enclose regions which have the same encoded information value v_i. Since each area is a subset of the 2-dimensional Euclidean plane $Ar_i \subseteq \mathbb{R}^2$, or $Ar_i \in \mathrm{Pot}(\mathbb{R}^2)$, the power set of $\mathbb{R}^2$. The areas of the elements a_i are pairwise disjoint:

$$\forall\ (Ar_i, v_i), (Ar_j, v_j) \in A, \quad i \neq j : \quad Ar_i \cap Ar_j = \emptyset. \tag{3.3}$$



(a) Aspect A **(b)** Aspect B

Figure 3.2: Graphical representations of two Aspects A, B: the three elements a_i enclose limited areas whereas the two elements b_i divide the Aspect plane into two halve planes [Zolynski 15].

The elements' information value or magnitude v_i could be in principle of any arbitrary data type. However, in no case actual object-like values or symbolic representatives should be stored since this would subvert the central idea of any subsymbolic perception process operating with rather environmental features or evidences than definite symbols. Therefore, it is assumed that every information can be represented with a finite amount n of real-valued numbers, such that $v_i \in \mathbb{R}^n$. Values encoded in different Aspects certainly may have different dimensions. Thus, the complete definition of an element a_i with a value-dimensionality of n is:

$$a_i := (Ar_i, v_i) \in \mathrm{Pot}(\mathbb{R}^2) \times \mathbb{R}^n. \tag{3.4}$$

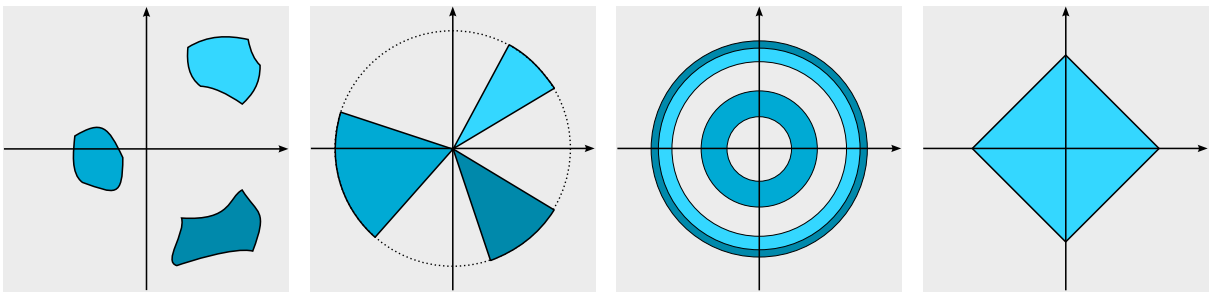
Figure 3.2 shows two example Aspect Maps with elements of limited and unlimited area components. The point of origin should typically coincide with the origin of the robot-centered coordinate system. This definitional alignment will come in useful when combining Aspect Maps of different dimensions and different types as introduced subsequently.

Spatial Structures of Aspect Maps

When considering spatial maps in robotic systems apart from rigid grid maps, directed data without exact distance information often requires directional maps, storing only the angle relative to the system. Acoustic data is an example of this class, requiring storage with an angular mapping. Furthermore, air temperature is hardly spatially dependent at all but represents one single scalar value. In order to operate consistently with different classes of data within an architecture based on maps, four types of full and partial spatiality—*Aspect types* where some spatial information is not available—can be identified [Zolynski 15]:

Spatial Aspect: The *Spatial Aspect* provides fully spatial information and uses Cartesian addressing. Maps using this mode can be both sparse or dense, depending on the application. Each region in the Aspect encodes one piece of information and can have a unique shape and extent. See figure 3.3a.

Directional Aspect: *Directional Aspects* provide a solution for storing data without a distance information as mentioned above and allow for a directional addressing. Each



(a) Spatial Aspect **(b)** Directional Aspect **(c)** Distance Aspect **(d)** Uniform Aspect

Figure 3.3: Graphical representation of Aspect types with different spatial structures [Zolynski 15].

element has an angular width but no distance information. As for the Directional Aspects, the spatial representation is comprised of circle segments (with undefined, possibly infinite radius). See figure 3.3b.

Distance Aspect: The *Distance Aspect* is basically the same as the Directional Aspect despite of the reading (interpretation) in terms of the contained information: complementing the previous case, no directional information but only distance data is available, which might be the case for data received from ultrasonic sensors. The spatial representation of a Distance Aspect's regions can be thought of as concentric rings where only the radial distance is definite but the associated data is invariant of any directional angle. See figure 3.3c.

Uniform Aspect: *Uniform Aspects* finally represent the simplest case of data without any spatial component such as temperatures, time stamps or any network parameters. Storing just one entity of locally invariant information can also be seen as retaining the same value for any spatial position. See figure 3.3d.

3.2.3 Processing

A fundamental assumption, motivating the cognitive maps architecture, is that every perceived information or any derived conclusion might be a useful data source of knowledge, which is in the end worth knowing for the system. Deliberate data processing in the sense of successive extraction, abstraction and (re)combination of particular data yielding more complex, superior information is, therefore, a core requirement of this approach. The first step of this undertaking has to be a translation from raw sensory data to entities that can be processed by the architecture. As mentioned earlier, raw data commonly entails various facets of information content, requiring an initial *extraction step*, which appropriately encodes data of importance into an Aspect. Obtained Aspects may then be processed further in a unary manner as for example in the case of *filtering* or combined with other Aspects by use of *operations*. These different types of *processing units* and their functional principles are introduced below.

Extractions

Extraction units can apply any encoding method or complex algorithm in order to transform raw data into one or multiple different Aspect Maps. For example, a point cloud gathered

from a laser scanner might be transformed into an Aspect encoding the maximum height per horizontal unit area and into a second one representing the minimum height respectively. Also a simple one-to-one mapping of sensory data or even a complex object detection algorithm is conceivable. Generally, an extraction node is the first element in the processing chain and has exactly one input for incoming data and possibly many outputs which deliver Aspects of *primary data*, each encoding ideally only one facet of the underlying data. A second type of extraction node can be located at any arbitrary processing step, takes one input Aspect and yields multiple Aspects containing extracted information as for instance the steepness derived from a maximum height Aspect as mentioned before. If incoming Aspects are unchangedly passed to multiple outputs, an extractor actually degenerates to a simple distribution node. Every data arising from any further extraction or processing of primary Aspects is called *secondary data*. An extraction node is shown in figure 3.4a.

Operations

Up to this point, several distinct Aspects providing different features of the environment can be extracted from primary data sources. The combinations of those Aspects, ideally generating entirely new information, is a central part of this architecture. A *combination node* takes two or more input Aspects and performs a *operation Op* using the information encoded within the Aspects as parameters. Any combination of N Aspects can be broken up into a recursive combination of a single Aspect with the combination of the remaining $N-1$ ones:

$$Op\ (A_0, A_1, \ldots, A_{n-1}) \equiv Op_0\ (A_0, Op_1\ (A_1, \ldots, Op_{n-2}\ (A_{n-2}, A_{n-1}))). \tag{3.5}$$

Thus, the subsequent formal definition of a combination of exactly two Aspects is sufficient. An *Aspect Operation Op* takes two input Aspect Maps A_0, A_1 and combines them applying a *binary operator op* on each two elements' information values $v_{0,i}, v1, j$ whose regions $Ar_{0,i}, Ar_{1,j}$ intersect. Formally, an operation Op can be written as:

$$A_2 = Op\ (A_0, A_1)\ , \tag{3.6}$$

or, concerning the element level:

$$\begin{aligned} \forall\ a_{0,i} = (Ar_{0,i}, v_{0,i}) \in A_0 :\ \forall\ a_{1,i} = (Ar_{1,j}, v_{1,j}) \in A_1,\ Ar_{0,i} \cap Ar_{1,j} \neq \emptyset : \\ a_{2,k} = (Ar_{2,k}, v_{2,k}) := (Ar_{0,i} \cap Ar_{1,j},\ op\ (v_{0,i}, v_{1,j}))\ , \end{aligned} \tag{3.7}$$

where *op* denotes the actual binary operator and i, j are corresponding element indices of A_0, A_1 and k is the resulting element's index of A_2. The indices i, j are generally distinct in the case of combining different Aspect storage types and / or different Aspect dimensions and resolutions. At this point, it is only important to note that they refer to matching regions which are combined by means of any binary operator *op*. For example, a weighted addition and thresholding operator can be defined as follows:

$$op_{\text{wadd}} :\ v_{2,i} = \alpha v_{0,i} + (1-\alpha) v_{1,j},\ \alpha \in [0,1]\ , \tag{3.8}$$

$$op_{\text{threshold}} :\ v_{2,i} = \begin{cases} 1 & \text{if} \quad v_{0,i} > v_{1,j} \\ 0 & \text{else} \end{cases}\ . \tag{3.9}$$

While combining two Aspects types using the same storage mode is easy, the combination of different Aspects, however, requires some additional thought. Furthermore, a combination of any Aspect types with unaligned origin points and / or different scales requires an appropriate matching of corresponding elements. Combinations of Aspect types with different storages modes are called *hybrid combinations* whereas combinations of Aspects with displaced origin points or different scales are referred to as *unaligned combinations*. These special cases are elaborated in detail in appendix A.1.2.

Filters

While operations combine two distinct Aspects subset by subset, some standard applications especially known from image processing imply only one Aspect Map but modify each element as a function of some of its neighboring elements. These *filtering* tasks such as smoothing, edge detection or normalization require access to spatially surrounding elements for computing the resulting element's value. The extrapolation of dense data from sparse data or a convolution (or similarly a correlation) are also common filtering operations. *Aspect Filters* are hence *unary processing* units, working in the spatial domain of only one Aspect Map at a time as shown in figure 5.4a. Formally, an Aspect Filter applies a filter kernel $K(r)$ with a certain radius r subject to an appropriate norm to an Aspect Map A_0:

$$A_1 = Filt_{K(r)}\ (A_0)\ . \tag{3.10}$$

The kernel can be any spatially dependent, discrete function which has also access to the Aspect's elements. For each Aspect element $a_{0,i}$ the kernel's origin point (central element) is aligned with the element's position and all kernel elements $k_j \in K$ are element-wise combined with all elements within a radial neighborhood r of $Ar_{0,i}$:

$$\begin{aligned} \forall\ a_{0,i} &= (Ar_{0,i}, v_{0,i}) : \\ a_{1,i} &= (Ar_{1,i}, v_{1,i}) := (Ar_{0,i},\ filt_{K(r)}\ (Ar_{0,i}))\ . \end{aligned} \tag{3.11}$$

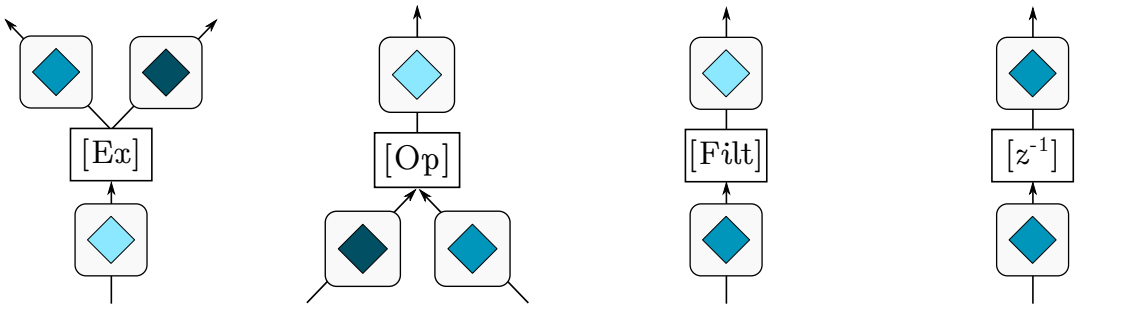
For example, a simple maximum filter and a mean smoothing can be defined as follows:

$$filt_{K(r),\ \max} :\ v_{1,i} = \max_{k_j \in K(r)} (Ar_{0,i})\ , \tag{3.12}$$

$$filt_{K(r),\ \text{mean}} :\ v_{1,i} = \frac{1}{|K(r)|} \sum_{k_j \in K(r)} v_{0,j}\ . \tag{3.13}$$

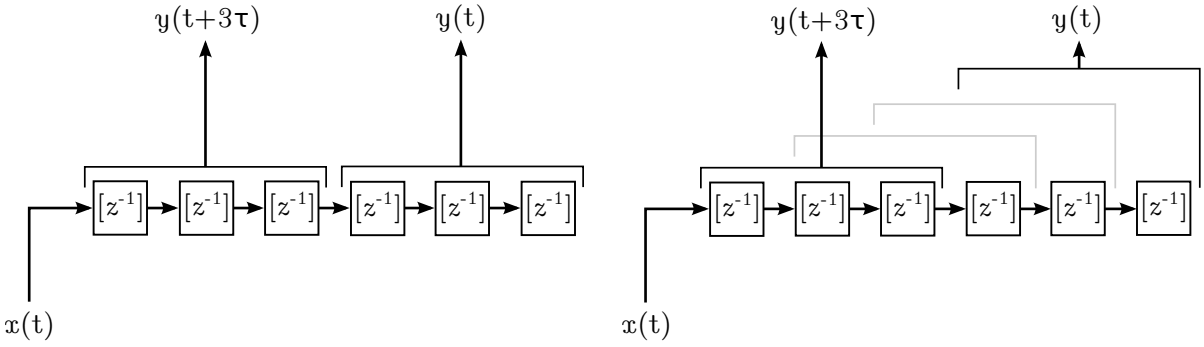
Temporal Processing

The Cognitive Maps Architecture provides another unary processing unit, which is in contrast to the spatial filtering unit responsible for the temporal domain of data. Any temporal processing requires storage or delay elements storing data for a specific period of time and thus introducing a kind of memory. A *delay node* temporary retains an input Aspect Map as a whole and releases the data after the expiration of a certain time span. Technically, a basic delay node holds its information for the system's minimum unit of time while longer storage times can be realized by a concatenation of multiple basic delay nodes. However, this might be implemented quite differently. Referring to signal processing conventions, a delay node is denoted by z^{-1} as depicted in figure 3.4d. Within the scope of



(a) Extraction node **(b)** Operation node **(c)** Filter node **(d)** Delay node

Figure 3.4: Basic binary and unary processing nodes of the architecture. Aspects are illustrated as diamond shapes on the nodes' edges.



(a) Bucket fill accumulation **(b)** Rolling accumulation

Figure 3.5: Bucket fill and rolling accumulation based on concatenated delay nodes [Zolynski 15].

this approach, temporary processing involves always entire Aspects and no singled subset data in order to maintaining the spatial connectedness of Aspect Maps at all times.

Combining delay units and operation units as building blocks, *temporal filtering* in a signal processing sense can be implemented: *Accumulation* reduces several successively incoming Aspects to a new aggregate Aspect causing a temporal averaging (in the broadest sense) of information. Buffering several Aspects and combining them only after a certain amount has arrived, is called *bucket fill accumulation* and depicted in figure 3.5a. A *rolling accumulation*, in contrary, combines a specific number of recently retained Aspects and releases a resulting aggregate Aspect as soon as a new input Aspect arrives and is shown in figure 3.5b. The latter type of accumulation corresponds therefore to an online (filtering) algorithm while the bucket fill method must wait for complete data sets and works offline, reducing the frequency of data propagation on the entire processing path of all successive nodes. The combination operation itself might be any Aspect Operation but is usually a weighted addition.

3.2.4 Data Flow and Data Hierarchy

Conceptionally, the Aspect Maps Architecture is based on a data flow paradigm and models a network of interconnected processing nodes with Aspects flowing along the

edges. Process chains—or paths of processing units—can be reused for different purposes, easily modified or inspected by tapping network links at any process step and visualizing arriving Aspects. A processing path must always start from a (raw) data source followed by an extractor transforming the data's facets of interest into Aspects, which are then spatially or temporally processed (filtered) and combined among each other. Finally, the last processing node on a path leads to a data sink providing clients with desired higher level information.

Following a processing path from the source node upwards, each processing unit constitutes also a higher abstraction level of data. Edges of nodes on lower abstraction levels carry almost unaffected sensory data and completely processed information arriving at data sinks may be abstract binary values, representing, for instance, a computed solution of a decision problem. For that reason, Aspects should not only be distinguished in terms of their spatial structure but also having regard to the abstraction level of the encoded information. Three hierarchical types are hence proposed by [Zolynski 15]:

Simple Aspects: *Simple Aspects* correspond to not or only marginally preprocessed raw data after being extracted and encoded into an Aspect Map. Containing primary data, simple Aspects represent the lowest hierarchy level and are—despite their seemingly misleading name—allowed to store expressive original data with potentially complex data types of unrestricted value ranges. So, simple Aspects might be complex as to their information content but plain in view of the acquisition process.

Complex Aspects: *Complex Aspects* store secondary data obtained from processing within the network and is derived from an arbitrary combination of filtering steps and combinations with Aspects of any other hierarchical level. Encoded information is more simple, low-dimensional and often normalized in order to have a pertinent representation for further modular processing. Complex Aspects embody specific interpretations of the underlying early primary data and are therefore not only informatively enriched but also potentially distorted. They are the major part of the network's hierarchy types and can be found on all abstraction levels between data sources and sinks.

Cognitive Aspects: *Cognitive Aspects* are finally the results of a network path at the highest level: desired Aspects encoding outcomes of the entire processing which can be used by other clients at the data sinks' interfaces. These Aspects contain the most abstracted information in a very simple and restricted data format. Typically values are binary corresponding to solutions of decision problems or normalized to the range $[0, 1]$ encoding for example probabilities.

The different types of nodes have also different impact on the local network topology: extractor nodes fan out incoming edges, while operation nodes combine incoming edges into one outgoing edge, and filtering or delay nodes leave unary edges structurally unchanged. Arranging an arbitrary network graph such that data sources are located on the bottom and data sinks on the top, illustrates two different tree-like structures within the graph.

From a bottom-up perspective, source data edges fed into extractor nodes branch through the network and, from a top-down perspective, combination nodes form also trees joining those branches again. While the first viewpoint highlights the structural influence of a typical subsymbolic bottom-up architecture, the combination trees might be seen as a

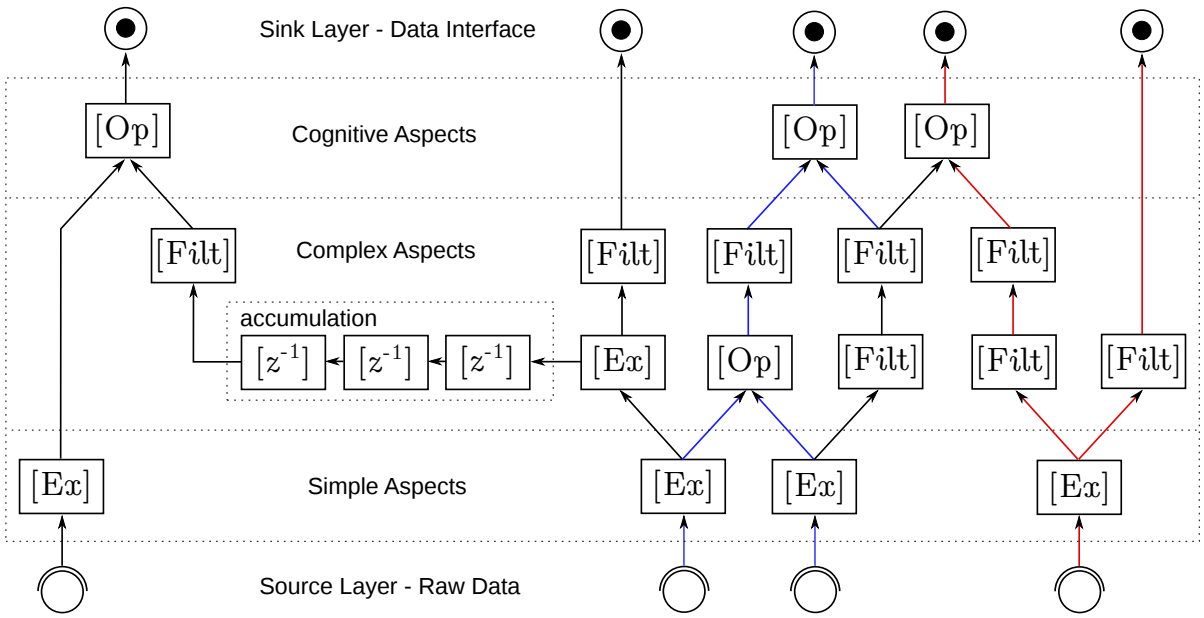


Figure 3.6: Hierarchical data flow network of an exemplary cognitive maps network with unary and binary processing.

manifestation of deliberate top-down design decisions modeling the network's externally incorporated process knowledge.

An exemplary data flow network with hierarchical Aspect layers is shown in figure 3.6. A selected *extraction tree* from the bottom-up perspective of extraction is represented by red edges whereas another *combinational tree* from the top-down perspective of combination is shown in blue.

3.2.5 Example Network

With the aid of the introduced processing nodes, arbitrary complex cognitive networks can be set up. After the general network topology has been shown previously, a specific exemplary network, deriving new, distinct information from rather simple basic data sources, is presented in figure 3.7. The network's objective is to extract a spatially dependent slipping probability for the `THOR` excavator (see section 1.1) when traversing a construction site during excavation tasks. As the slipping probability includes the soil conditions, the networks aims to acquire this information from a combination of local soil properties such as the slide of soil as well as the soil moisture. Both, for their part, are Spatial Aspects that can be derived from the soil granularity. Additionally, the slide of soil depends on the steepness of the ground, extracted from a height map of the construction site, and the moisture depends on the volume of rainfall per area, extracted from recent weather data. At the bottom level, the respective raw data sources are shown.

3.3 Summary and Evaluation

In this chapter two subsymbolic approaches have been introduced, both relying on a visual view of the problem domain. Within this work, the scope of application is a

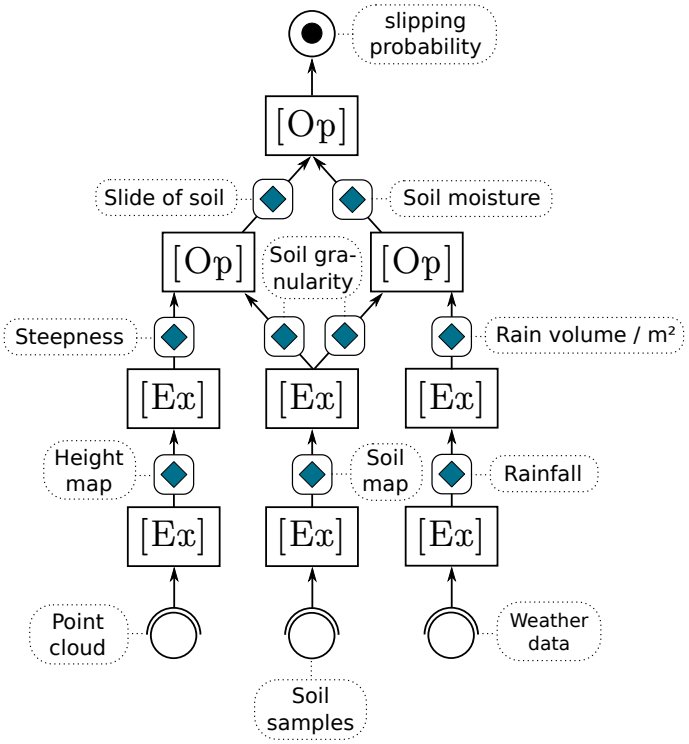


Figure 3.7: Exemplary network deriving THOR's slipping probability from a point cloud, soil information and weather data.

cognitive computing approach by use of (time-variant) sensory data perceived from a robot's surroundings. For this purpose, spatial information must be in either case mapped onto a horizontally aligned, 2-dimensional plane corresponding to an aerial perspective of the world. Both approaches rely on the spatial structure of those input planes and provide convenient, locally restricted processing methods. Aspect Maps in one case and feature maps in the other are the basic entities of spatially dependent but semantically homogeneous information and represent a particular facet of environmental knowledge. Although they might seem to be similar regarding their storage format, there are some disparities: firstly, Aspect Maps provide different addressing modes as with Directional and Distance Aspects. But Aspects are also much more expressive than feature maps, which are only outcomes of a specific extraction process encoding the location of a specific feature. Beyond any question, the Cognitive Maps Architecture is in several respects the markedly more general and versatile approach. As the name suggests, it is an entire architecture composed of many processing paths and various data sinks, each supplying a distinct Aspect of the robot's world. The Aspect network as a whole is aimed at extensively modeling the robot's overall cognitive processing. In contrast to that, a Convolutional Neural Network is typically comprised of only one path with one data sink. A CNN is not designed for diverse processing and rather geared to the needs of a single subdomain of cognition such as pattern recognition. Cognitive Maps Networks, on the other hand, are capable of combining two heterogeneous Aspects of distinct types, dimensions, resolutions and semantic meanings and turn those combinations to advantage in order to generate higher level knowledge. It is exactly this versatility and abstraction, which makes the Cognitive Maps Architecture as powerful and applicable for virtually any processing task.

Desired behavior can and must be deliberately implemented in a top-down manner and it is up to the user to design the processing blocks and their interconnections. The approach intends not to provide any techniques in order to generate an entire cognitive network autonomously, which would anyway seem quite dubious in consideration of the complexity of such a demand. CNNs, albeit considerably restricting the operations of units and the network's rough structure, are able to find good configurations with respect to certain desired outputs. While such a training procedure is indeed conceivable for particular processing paths taken by themselves, a complete autonomously adaptive Aspect network seems visionary; not least because training would likely be lost in the combinational complexity of major networks and even assorting a sufficient training set might involve more effort than directly developing a suitable top-down implementation.

However, the advantages that can be benefited from the Cognitive Maps Architecture cannot be denied: The approach guides a clean, consistent and comprehensive implementation of a subsymbolic, data-flow oriented cognitive framework. Established algorithms can be integrated while redundant processing and intermediate data are avoided unlike in separate, closed subsystems. Instead, Aspect edges can be branched off anywhere and fed into another processing path. Aspect networks are understandable and inspectable since their structure as well as all Aspects passing through the edges can be visualized if necessary, thus, supporting the design and debugging process. Zolynski describes the architecture aptly as a "glue between [...] components" [Zolynski 15] and furthermore as an architectural "middle ground between object maps and the parallel processing of neural networks". Consequently, it is an interface between the connectionist and symbolic paradigms and explicitly omits the presupposition of object recognition and classification as prestage of any cognitive processing. Instead, an Aspect Map network might be seen as a form of feature, pattern or even object recognition per se since subsymbolic raw data on the level of Simple Aspects is transformed into Cognitive Aspects that indicate the existence of patterns or symbolic objects either binary or probabilistically. The very quintessence of this approach might be the insight that an appropriate combination of information from diverse sensory sources can yield rewarding higher-level knowledge. CNNs, on the other hand, excel at extracting features scattered over the input plane and teach the awareness of problems such as shifts and distortions of data. Furthermore, they show the possibility of self-configured networks by means of machine learning, which might be hardly applicable for entire cognitive networks but maybe for sub-networks or selected processing paths. Finally, CNNs encourage a highly parallel implementation in consequence of their underlying data and processing structure.

The Cognitive Maps Architecture with its operations and filters is expressive enough to implement an artificial neural network and, as a CNN is just a special case of a restricted neural network, also CNNs could be emulated (see A.1.1). The concept's general applicability, useful mentioned properties and restrictions such as the semantic and spatial cohesion of information stored in Aspect Maps let the Cognitive Maps Architecture appear to be an appropriate basic architecture for a working memory.

4. Concept

Since a brief introduction and general remarks about computational intelligence, subsymbolic, cognitive and visual computing and the implications of temporal processing and memory have been given in chapter 2 and two inspiring, specific concepts have been presented in chapter 3, the foundations for a working memory are laid. This chapter is aimed at explaining the essence of this thesis—a conception of a working memory based on the Cognitive Maps Architecture. To this end, the 2-dimensional Aspect Map is expanded by a temporal dimension forming thus a 3-dimensional cuboid or a *spatiotemporal memory*. This entity composed of temporally related Aspects can now reflect both the spatial and temporal cohesion of environmental information. Within this structure, continuously arriving Aspect Maps of identical informational semantics are retained and kept available for other processing units.

As storage is generally limited, a memory's depth must also be restricted. In the simplest case, consistently deleting the oldest preserved data would serve this purpose. But biologically motivated models of memory show a gradual process of *forgetting* which is more than a technical necessity. Instead, forgetting opens a chance to memorize important features, abstract from negligible details and can also be seen as a form of productive temporal processing. Moreover, associating currently perceived stimuli with partially forgotten, "faded" information might be virtually seen as the inverse process of *remembering*.

As it is beyond debate that forgetting and remembering, especially from a perspective of cognitive science, cannot be addressed exhaustively within this work, the presented approach has to be a narrow and technical interpretation of those issues. Whereas the subsequent section is meant to introduce the structure of the basic memory, the last two sections of this chapter will investigate forgetting and remembering from the perspective of subsymbolic processing and propose specific, selected concepts.

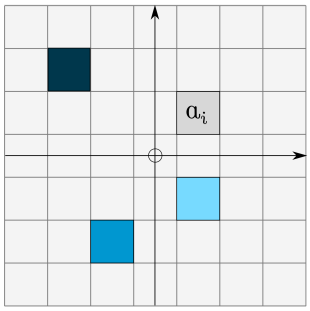


Figure 4.1: Graphical representation of a Grid Aspect composed of cells as a discretized special case of the Spatial Aspect.

4.1 Discretization of Information and Grid Aspect Definition

The implementation of an Aspect working memory has been limited to a special case of grid-like Spatial Aspects, henceforth referred to as *Grid Aspect*, without limiting the generality of the concept presented in this chapter. A Grid Aspect can be seen as a discretized version of a Spatial Aspect composed of elements that enclose limited square regions—or *cells*—of equal sizes. The terms *element* and *cell* may be used synonymously. While technically similar to a grid map, a Grid Aspect implies the semantic and functional consequences of an Aspect in the sense of the Cognitive Maps Architecture. Figure 4.1 shows a Grid Aspect with its cells arranged in a Cartesian grid structure.

The Grid Aspect type has been chosen for the reference implementation of this thesis for several reasons: Unrestricted Spatial Aspects keep their information enclosed in arbitrary regions contained in the 2-dimensional plane and can thus be thought of as continuous 2-dimensional signals. While conceptually simple, a proper representation of such data would be technically quite challenging. In contrary, a discrete grid-like addressing mode can be easily implemented. Furthermore, a grid structure is very preferable from a computational point of view as will be discussed later in section 5.1.2. Grid Aspects can be interpreted as images where each cell corresponds to a pixel. Thus, concepts of computational vision and image processing motivated in section 2.3 are inherently suitable. Stored Grid Aspects—and thus the state of an entire working memory—can be easily visualized and inspected by the user. Last but not least, a working memory composed of Grid Aspects perceived at several consecutive points in time is a realistic model for retaining real sensor data since most (more complex) sensors provide inherently discrete data at a certain update frequency. Hence, both the spatial and temporal dimension are usually implicitly discretized and, as any computation involves a discretization of continuous data in practice sooner or later, a sampling by means of equidistant intervals along each dimension—a 3-dimension grid structure—seems natural and is certainly sufficient for most applications in robotics. The subsequent fundamental concepts are hence introduced in the discrete domain and utilize Grid Aspects which may also forge a bridge to the actual implementation chapter and help to illustrate the presented ideas graphically.

When translating the remaining Aspect types introduced in section 3.2 into the discrete domain as well, the Directional, Distance and Uniform Aspect types can be simply defined

as 1-dimensional or respectively 0-dimensional special cases of Grid Aspects which is explained in detail in section 5.2.1 of the implementation chapter. Therefore, within the scope of this work, a Grid Aspect is the most general implemented, discretized Aspect type and provides fully spatial information restricted to a grid-like addressing mode. Henceforth, whenever the term *Aspect* is used within the following chapters, it shall implicitly mean *discrete Aspect* from a practical, implementational perspective.

However, all mathematical concepts proposed in this chapter are basically derived from signal processing and are also well-defined in the continuous domain[1] and in principle suitable for a working memory containing unrestricted Spatial Aspects. A proper implementation—if reasonable—would be less a conceptual but rather a (albeit not obvious) technical issue.

4.2 Structure of an Aspect Working Memory

Temporal processing of the Cognitive Maps Architecture makes use of delay units introduced in section 3.2.3 of the previous chapter. In the simplest case a single delay unit forms an *Aspect Memory* of depth one which allows to memorize an Aspect for one unit of time. Multiple concatenated delay nodes can store many Aspects from consecutive points in time and constitute a temporal processing path that can also be interpreted as a discrete timeline with an equidistant scaling at an interval of the network's cycle time. In this way, buffers or memory blocks can be implemented. An Aspect Memory is composed of several Aspect Maps of the same semantic Aspect and encapsulates spatiotemporally correlated information within a distinct, decentralized unit in the network. Each different Aspect might require its own Aspect Memory if time-variant instances of this Aspect are needed for processing. The system's entire memory state depends therefore on the particular Aspect Memories. This design decision seems appropriate in view of the network approach of the underlying architecture and provides some benefits: the memory depths of individual units can be different and adapted for demands and each memory's responsibilities are restricted to a certain Aspect. Whereas a buffer of scalar-valued Uniform Aspects—as for example temperature values—corresponds to a time series, a series of delayed 2-dimensional Aspects forms a spatiotemporal cuboid as shown in figure 4.2. The temporal axis starts in the present time and points towards past along the path of delay nodes. In general, an Aspect Memory is spanned by three types of dimensions: time, space and information content, where space and content are already encapsulated within the Aspect Map entity. An Aspect's space can be of any dimensionality between 0 and 2, where 0 corresponds to Uniform Aspects, 1 to Directional or Distance Aspects and 2 finally to Spatial Aspects.

Aspect Memories afford the opportunity to access previously perceived Aspects and, thus, to look into the past of the robot's environment. Technically, this is an requirement to apply temporal filtering and to take temporal changes of data into consideration. Possible applications are highly diverse: a typical task is low-pass filtering for the purpose of noise reduction, another one might be a comparison of two successive Aspect Maps in order to calculate a rate of change or to detect changing subareas in the case of Spatial Aspects. Also a basic segmentation that classifies Spatial Aspects into static and dynamic segments

[1] A continuous representation can be achieved by replacing regions associated with a certain areal extent (cells or pixels) with an infinite number of infinitesimal area elements such that all discrete variables translate to continuous variables and all discrete sums translate to integrals.

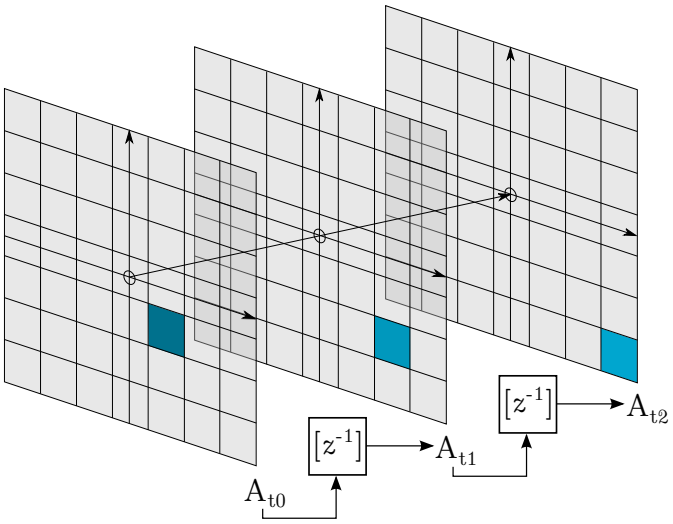


Figure 4.2: Basic spatiotemporal Aspect Map Memory containing three temporal instances of a Grid Aspect and the corresponding processing path composed of delay units. A memory's Aspects may be of any (but identical) storage mode in general.

can be implemented by simply observing a heightmap over time. Figure 4.2 shows a set of Grid Aspects where each temporal instance contains cells of the same magnitude at different locations. A possible interpretation might be a moving object where the direction of movement between two successive frames can be computed. Knowing the cycle time of the network, also a velocity can be estimated. In this way, the potential object can be tracked through the spacetime by means of the trace of its translation vectors. Furthermore, an estimation of the future movement can be derived etc.

Including a history of Aspects introduces various possibilities for cognitive processing but also allows for storing consequential results or cognitive inferences. Hence, the system has a complex state consisting of all sub-states induced by memories distributed over the network. This might be seen as a prerequisite for any fundamental contemplative or cognitive processing since without states or memories the network would be restricted to a straight, time-invariant mapping between inputs and outputs. Beyond that, the ability to detect changes over time by comparing successive Aspect Maps enables the network likewise to recognize time-invariant periods where no events occur within the environmental subdomain the Aspect is dedicated to. Vice versa, any significant change can be seen as an event that is worthwhile to observe and process whereas no events require no attention in most cases. Hence, with the help of memories, a basis for *attention-directed processing* is established. Events can trigger or attract particular processing for example by dynamically connecting temporally active network nodes via edges to certain processing paths. This helps to use computational resources more purposeful as well. Although this subject is not part of this work, it shows a very different, simple and interesting application of distributed Aspect Memories introduced here.

4.3 Forgetting

The Aspect Memory's timeline is discretely divided into slots of temporal Aspect instances and has an individual but constant depth. Thus far, each time a new Aspect is available and added to the front of the memory, the entire queue moves by one slot and the last Aspect is expelled. One of the earliest and most discussed theories in the context of biological forgetting is the *decay theory* which claims that contents of memories fade over time such that information belonging to the distant past is less retained than recently stored data. Frequently, a so called *forgetting-curve* which describes the relation as a function of availability—the probability that a certain information is available—over time or memory depth is assumed to be logarithmic [Henderson 99]. This basic idea is taken up for the Aspect Memory concept but slightly modified. In order to achieve a gradual forgetting process, the decision has been made to fade the Aspect's contentual dimension by successively coarsening the spatial and temporal resolution over time. Technically, both dimensions will be treated separately as a temporal and spatial filtering with a respective subsequent temporal and spatial subsampling. The particular concepts are discussed hereinafter in detail.

4.3.1 Temporal Filtering

In section 3.2.3 temporal accumulation has been introduced as a from of temporal filtering that reduces a batch of Aspects to one resulting Aspect. Using a bucket fill accumulation, the reduction cannot be performed until a certain amount n of Aspects have arrived, thus reducing the subsequent frequency f of outgoing Aspects on that path to $\frac{f}{n}$. A rolling accumulation, instead, combines the last n Aspects each time a new Aspect arrives and generates successively combinational Aspects without changing the frequency. In either case, a set of Aspects perceived at different points in time are combined into a single representative. When considering the temporal axis, an accumulation can thus be thought of assigning several time slices to one single time block that spans all slices. The resulting Aspect can no longer be associated with a precise point in time but only with a longer period. Aspect accumulations correspond therefore to a temporal filtering where the filter characteristics are dependent on the chosen accumulation operation. Using an averaging or a weighted addition implies a temporal smoothing and in the latter case the influence of recent information on the accumulation result can be controlled by a respective weight factor:

$$A_{\text{acc}} = Op\ (A_t, A_{t-1})\ , \tag{4.1}$$

or on element level:

$$\begin{aligned} \forall\ i:\ v_{\text{acc},i} &= op_{\text{wadd}}\ (v_{t,i}, v_{t-1,j})\ , \\ op_{\text{wadd}}:\ v_{\text{acc},i} &= \alpha\ v_{t,i} + (1-\alpha) v_{t-1,j},\ \alpha \in [0,1]\ . \end{aligned} \tag{4.2}$$

As an accumulation is technically a set of Aspect Operations, contentual modifications are performed element-wise such that each Aspect element is treated independently and filtered along the temporal axis only. This method is not only motivated by the temporal forgetting curve but also based on the idea that most noise or incorrect information is a temporal phenomenon: that is, temporally corrupted data caused by sensor faults, inconvenient

light conditions or sensor perspectives might be recovered using accumulations where a single faulty piece of information can be compensated by its temporal predecessors and successors.

Note that a continuous rolling accumulation based on a recursively applied weighted addition corresponds to a exponential smoothing. A bucket fill accumulation, in contrary, is similar to a finite exponential smoothing of depth n. This conception will focus on the latter one since the bucket fill mechanism provides the favorable property of frequency reduction and thus a reduction of required storage on the one hand and partitions the temporal axis of the memory in a plain and comprehensible manner on the other hand. This temporal fragmentation—or subsampling—is elaborated subsequently.

4.3.2 Temporal Subsampling

Delay nodes of an Aspect Memory can now be grouped into chunks of a certain *bucket size* n and each one is equipped with a reduction operation such that *accumulation stages* are formed along the temporal axis. Since each accumulation reduces the processing path's data frequency and likewise fuses data assigned to different points in time, the temporal resolution decreases with each stage or, in other words, the time blocks of reduced Aspects and their temporal uncertainty increase.

Assumed a series of m delay nodes is grouped into $\left\lfloor \frac{m}{n} \right\rfloor$ buckets or accumulation stages of size n, the frequency after delay unit m is then:

$$f(m) = \frac{f_0}{n^{\left\lfloor \frac{m}{n} \right\rfloor}} \ , \tag{4.3}$$

where f_0 is the initial frequency at the beginning of the path. Respectively the uncertainty time period associated with each accumulation stage is:

$$T\left(\left\lfloor \frac{m}{n} \right\rfloor\right) = \frac{1}{f\left(\left\lfloor \frac{m}{n} \right\rfloor\right)} \ . \tag{4.4}$$

A fixed bucket size causes therefore an exponential increase of time interval sizes. This implies that, with the passing of time, stored information is dated more and more imprecisely.

Note that choosing $n = 1$ leads to a structure without any reductive accumulation where $f(m) = f_0$. In order to keep the implementation as flexible as possible, the accumulation operation for temporal filtering is optional. If no operation is defined, the recently arrived Aspect of a bucket is passed while the others are ignored. In the case of $n = 1$ without any operation, the entire memory structure degenerates to a simple delay path or buffer storing the last m unchanged Aspects.[2]

[2] In contrary, allowing a weighted addition after each delay node that combines the arriving Aspect with the retained one, interestingly corresponds again to a true (infinite) exponential smoothing where the last m smoothed Aspects are stored in the particular nodes.

4.3.3 Spatial Filtering

Since Aspects consist of spatially dependent subsets, their expressiveness can also be changed by modifying locally correlated information. Aspect Filters described in section 3.2.3 are exactly aimed at affecting elements subject to their local neighborhood. Choosing a smoothing filter such as a mean filter for example, simply replaces each value with the average value of its neighboring elements and corresponding to a low-pass filter. As low-pass filters limit the upper frequency band, the spatial details or the granularity of Aspect information is likewise reduced in the spatial domain. This induces vagueness and thus robustness against minor displacements and is—applied over time—another contribution to the forgetting process. Original Cognitive Maps introduced in section 2.3 show the feature of metrical inaccurateness and are sometimes even markedly distorted in favor of their inherent, subjective expressive intention. For this reason, it seems worthwhile to deliberately modify Aspect Memories within their spatial dimension using filters. Considering Convolutional Neural Networks, feature extraction is an advantageous cognitive process based on filtering operations. Consequently, not only smoothing filters might be of interest but depending on the memory's stored data and its purpose also any extracting filters such as edge detectors can help to abstract form spatial details and bring essential features into focus. An appropriate filter function is certainly highly dependent on the type and quality of data and its application. Whereas roughly estimated positions of obstacles at a distance disturbed by bad light conditions are likely good candidates for a moderate smoothing, a high-resolution height map might be high-pass filtered in order to emphasize contrasts and edges for object recognition tasks.

Convolution Filter

Up to this point, a predefined filter kernel is bound to a particular filter type but in order to evaluate the effects of different filters, it would be favorable to pass any Aspect as a kernel. Therefore, a more versatile filtering implementation than the presented Aspect Filter is preferable. A special *convolution filter* is introduced which takes two Aspect Maps and convolves them using the second one as kernel:

$$\begin{aligned} A_2 &= Conv\ (A_0, A_1) = A_0 * A_1 \\ &= Filt_{K(r)}\ (A_0), K(r) := A_1\ , \end{aligned} \tag{4.5}$$

or element-wise:

$$\begin{aligned} &\forall\ i =: (x, y),\ j =: (u, v)\ : \\ &v_{2,(x,y)} = \sum_{u=-\infty}^{\infty} \sum_{v=-\infty}^{\infty} v_{0,(x-u,y-v)} v_{1,(u,v)}\ , \end{aligned} \tag{4.6}$$

where (x, y) and (u, v) are spatial coordinates of any addressing mode which can be mapped uniquely to an index i or j respectively. Note that this definition is valid for any discrete Aspect type including 0-dimensional Uniform Aspects.

In this way, the original unary filtering is amended by a binary convolution operation. Typical smoothing filters such as a mean or Gaussian filter can now be defined by the second

kernel Aspect. Since discrete kernels correspond to matrices, they can be represented as follows:

$$K_{mean,3} := \frac{1}{9}\begin{bmatrix}1 & 1 & 1\\ 1 & 1 & 1\\ 1 & 1 & 1\end{bmatrix} ,$$

$$K_{gaussian,3} := \frac{1}{16}\begin{bmatrix}1 & 2 & 1\\ 2 & 4 & 2\\ 1 & 2 & 1\end{bmatrix} .$$

But also a high-pass edge detector as for instance the *Sobel operator*, known from image processing, can be written in matrix form corresponding to a Grid Aspect:

$$K_{sobel,x} = {K_{sobel,y}}^T := \begin{bmatrix}1 & 0 & -1\\ 2 & 0 & -2\\ 1 & 0 & -1\end{bmatrix} .$$

4.3.4 Spatial Subsampling

The final component of the forgetting process is a subsampling step motivated by the CNN approach. After both temporal and spatial filtering have been applied, the spatial resolution can be reduced. In this way, the information stored in Aspects becomes more and more vague in terms of their addressable local positions. A subsampling step is thus a contribution towards the idea of cognitive maps: in case of a precedent spatial low-pass filtering or smoothing, an appropriate subsampling ensures that the resulting Aspect contains only desired, blurred main components of lower frequencies. On the other hand, in case of a high-pass filtering such as an edge detection, one might expect that the final, coarser Aspect Map contains especially interesting environmental features. In either case, subsampling reduces the amount of elements of an Aspect and helps to save limited storage.

Subsampling can technically be reduced to an Aspect Operation already introduced in section 3.2.3, where the operation is a copy operation and the first Aspect is an empty Aspect Map with a reduced target resolution:

$$A_{\text{subsampled}} = Op\ (A_{\text{target\_res}}, A_{\text{initial}}) , \tag{4.7}$$

or on element level:

$$\begin{aligned} \forall\ i:\ v_{\text{subsampled},i} &= op_{\text{copy}}\ (v_{\text{target\_res},i}, v_{\text{initial},j}) ,\\ op_{\text{copy}}:\ v_{\text{subsampled},i} &= v_{\text{initial},j} . \end{aligned} \tag{4.8}$$

The degree of spatial subsampling or coarsement is called *coarsement factor* and measures the ratio of the initial and resulting resolution in a spatial dimension before and after subsampling. In this process, the implementation of Aspect combinations implicitly ensures that element's regions with matching world coordinates are properly subsampled. Since generally multiple elements of the initial Aspect are associated with one target region of the coarser, subsampled Aspect, the mapping is ambiguous. It is the responsibility of the previous spatial filter to resolve this ambiguity for the purpose of the particular intended forgetting process.

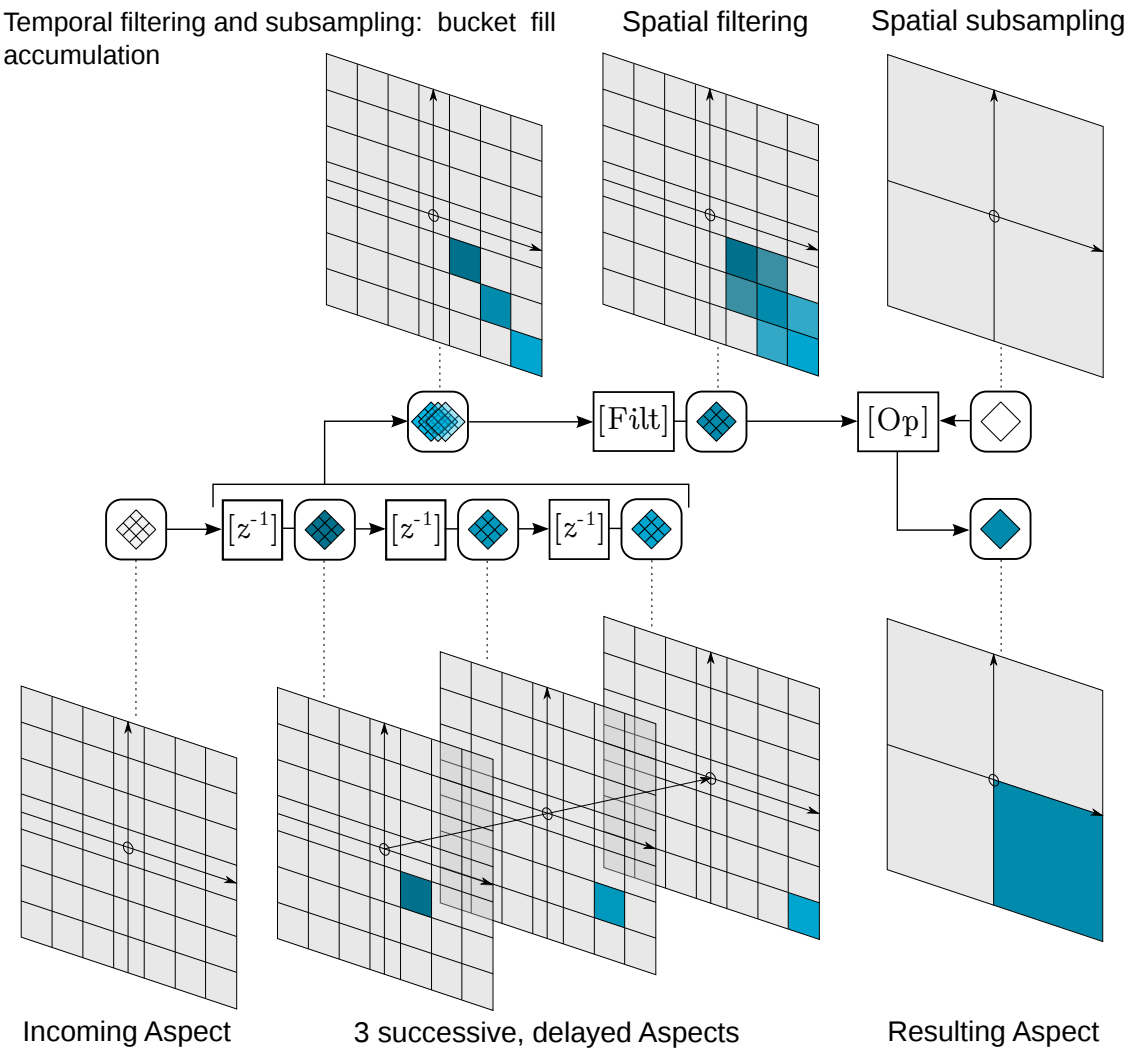


Figure 4.3: Illustration of the forgetting process within a single memory unit: delayed Grid Aspects are filtered along the temporal axis and subsampled by means of a bucket fill accumulation and finally spatially filtered and subsampled.

4.3.5 Summary — The Entire Forgetting Process

The temporal fragmentation into accumulation stages suggest also an appropriate implementation: a spatiotemporal subspace of an Aspect Memory is therefore encapsulated in a *memory unit*. By definition, the *depth* of an Aspect Memory is then the number of memory units, not to be confused with the number of basic, underlying delay nodes. All accumulation operations of a certain Aspect Memory as well as all Aspect Filters are constrained to be identical. Each memory unit's bucket size is a function of the entire memory's depth and describes the depth of a particular accumulation stage and thus the degree of temporal subsampling. Similarly, the coarsement factor describing the degree of spatial subsampling is a parameter of a particular memory unit and might also be a function dependent on the memory depth.

Note the similarity of temporal and spatial subsampling as a resolution reduction of the respective dimension and a technical manifestation of the vagueness introduced by the

respective previous filtering. Both bucket size and coarsement factor are parameters to control the extent of coarsening from one memory unit to the next and its contribution to the entire forgetting process. Subsampling also reduces in either case the required memory storage: each memory unit accumulates a set of Aspects along the temporal axis and finally subsamples the result to an Aspect with fewer elements. Therefore, a memory unit maps several Aspects of a higher spatial resolution into one reduced Aspect with a lower spatial resolution. This might be thought of as spatiotemporal subsampling or a lossy data compression. By design, only those compressed Aspects of each memory unit are externally accessible for other components of the processing network. The forgetting process within a single memory unit is shown in figure 4.3.

4.4 Remembering

Now that a forgetting process has been introduced, the question arises of how to utilize and evaluate gradually forgotten Aspects stored in an Aspect Memory. One might be simply interested in the contents of an Aspect perceived some time ago such that a simple query to the responsible Aspect Memory including a timestamp of interest delivers the desired information. However, a more interesting issue emerges if an Aspect Memory is searched for a certain Aspect and inquired if it is stored and, if so, how often and at which points in time it can be found. As soon as "similar" Aspects are of interest as well such that small deviations of Aspect subsets are acceptable, this problem cannot be addressed with a simple search in terms of equality testing anymore. The question of similarity is deeply interesting in the scope of a robot working memory: as the forgetting process already introduces vagueness that helps to abstract from noise and pointless details it would be favorable to also have a fuzzy search technique in order to examine those compressed, memorized Aspects in a nonrigid, flexible manner.

In this way, information similar to already known or expected Aspects could be identified, which corresponds to a pattern matching. In case of height information, 2-dimensional environmental settings could be recognized. Even more challenging but also more interesting is the extension of the fuzzy Aspect search to a fuzzy partial, sub-Aspect search such that smaller Aspects can be found within larger ones. This is similar to the problem of *template matching* known from image processing where a potentially small pattern has to be found within a larger search image (see [Brunelli 09], chap. 1). By this means, patterns encoded in Aspect Maps could be located within more comprehensive Aspect Maps and, if matching, their relative displacement could be determined. Furthermore, applying a template matching for each memory unit, patterns could be found and registered within both spatial and temporal dimensions resulting in a *spatiotemporal matching* approach. For instance, a height map representing a certain object such as a construction vehicle can now be presented to an Aspect Memory storing environmental height data. Similar occurrences within the spatiotemporal cuboid of memorized Aspects are then valid matchings—given that past, aged Aspects are not overly forgotten. Consequently, the vehicle can be tracked through the spacetime. Considering that Aspect Maps are very general and flexible entities, not only for encoding information associated with objects but rather subsymbolic, abstract and more subtle patterns a spatiotemporal template matching is a powerful approach in order to extract, recognize or, in other words, *remember* informational patterns.

This section will firstly propose the template matching approach as the 2-dimensional problem of registering a pattern within a larger image and later on describe how to extend

this method to match templates within several search images corresponding to a set of memorized Aspects. Here, the term pattern is associated with an Aspect Map, thus, the presented approach is intended to match spatial features—environmental snapshots—within a spatiotemporal memory. Instead, one might even want to allow spatiotemporal patterns, i.e. a set of temporally succeeding Aspects—a subspace of an Aspect Memory—which can be interpreted as an environmental situation. In the latter case it would be possible to recognize complex events. This is not part of this work, albeit it can be implemented using the very same fundamental principle of template registration introduced below.

4.4.1 A Naive Approach: Brute Force Template Matching

The aim of 2-dimensional template matching is to locate or register a smaller template image within a larger search image such that the error between the search image and its best match is minimized. The error metric or distance function is commonly the sum of the element-wise absolute or squared distances.

Assuming the elements of the search image $s(x, y)$ of size $n \times n$ and a template image $t(x, y)$ of size $m \times m$ are addressable by the coordinates (x, y), the origin of the template image can be moved over the search image and an overall error is evaluated by the sum of all element-wise errors subject to the chosen distance function. The error for a certain displacement (x, y) is given by:

$$E(x, y) = \sum_{u=0}^{m-1} \sum_{v=0}^{m-1} dist(x + u, y + v, u, v) \ , \tag{4.9}$$

where $dist(x_s, y_s, x_t, y_t)$ is a distance function such as $\|S(x_s, y_s) - T(x_t, y_t)\|$.

The displacement (x', y') yielding the smallest error determines the best match:

$$(x', y') = \underset{(x,y)\in\mathbb{N}^{n\times n}}{\arg\min} \ E(x, y) \ . \tag{4.10}$$

The optimal match can now be accepted or rejected depending on the magnitude of the associated remaining error. A high error indicates that the best match is quite different from the template pattern and might even be not contained in the search image.

This naive approach is computationally quite expensive since for each n^2 displacement candidates the sums of all m^2 distances between elements of template and search image must be evaluated resulting in a complexity of $O(n^2m^2)$ or $O(n^4)$ in the worst case if $n = m$ [Brunelli 09].

4.4.2 An Efficient Method: Phase Correlation

The Fourier domain, which is related with the time or spatial domain via the Fourier Transform, provides some useful properties for image registration: translation, rotation and scaling are transfered into their equivalents in frequency domain and can be determined more conveniently [Zitova 03]. Furthermore, a method in the Fourier domain is robust against frequency dependent noise in both the template and search image and occlusions [Sarvaiya 12]. A technique which exploits the translation property of the shift theorem [De Castro 87] in order to match two images translated relative to each other is the *phase correlation* method [Reddy 96],[Sarvaiya 09],[Sarvaiya 12].

Let $t(x, y)$ and $s(x, y)$ be two images both of size $n \times n$ where $t(x, y)$ is a circular shift of $s(x, y)$ with displacement $(\Delta x, \Delta y)$, i.e.:

$$t(x, y) := s((x - \Delta x) \bmod n, (y - \Delta y) \bmod n) \ . \tag{4.11}$$

Their *Discrete Fourier Transforms*[3] $T(u, v) := \mathcal{F}\{t(x, y)\}$ and $S(u, v) := \mathcal{F}\{s(x, y)\}$ are according to the shift theorem related by the corresponding phase shift

$$T(u, v) = S(u, v)e^{-2\pi i(\frac{u+\Delta x}{n} + \frac{v+\Delta y}{n})} \ . \tag{4.12}$$

By contrast, the Fourier magnitudes of both transformed images are identical.

It turns out that the phase of the *normalized cross power spectrum* $R(u, v)$ which is the Fourier transformed *normalized cross correlation* is equivalent to the phase shift caused by the shift in time domain:

$$\begin{aligned} R(u, v) &= \frac{S(u, v)T^*(u, v)}{|S(u, v)T^*(u, v)|} \\ &= \frac{S(u, v)S^*(u, v)e^{-2\pi i(\frac{u+\Delta x}{n} + \frac{v+\Delta y}{n})}}{|S(u, v)S^*(u, v)e^{-2\pi i(\frac{u+\Delta x}{n} + \frac{v+\Delta y}{n})}|} \\ &= \frac{S(u, v)S^*(u, v)e^{-2\pi i(\frac{u+\Delta x}{n} + \frac{v+\Delta y}{n})}}{|S(u, v)S^*(u, v)|} \\ &= e^{-2\pi i(\frac{u+\Delta x}{n} + \frac{v+\Delta y}{n})} \ , \end{aligned} \tag{4.13}$$

since the magnitude of any e^{ix} is one and the phase of the product of the complex conjugate pair $S(u, v)S^*(u, v)$ is zero.

Thus, the inverse Fourier Transform of $R(u, v)$ which is a delta function[4] located at the displacement $(\Delta x, \Delta y)$ indicates the initial translation. Therefore, the best match according to the phase correlation method can be generally found at the coordinates (x', y') associated with the peak value of $\mathcal{F}^{-1}\{R(u, v)\}$:

$$(x', y') = \underset{(x,y)\in\mathbb{N}^{n\times n}}{\arg\max}\ \mathcal{F}^{-1}\{R(u, v)\} = \underset{(x,y)\in\mathbb{N}^{n\times n}}{\arg\max}\ r(x, y) \ . \tag{4.14}$$

Note that the indirect way using the Fourier Transform is in principle not necessary since $r(x, y)$ could have been directly obtained in time domain. But calculating the normalized cross correlation involves a 2-dimensional convolution operation with computational complexity $O(n^4)$ similar to the naive approach (for images of the same size). Instead, calculating $R(u, v)$ in the frequency domain as stated in equation 4.13, requires only $O(n^2)$ operations. A 2-dimensional transformation from time domain to Fourier domain can be realized by means of the *Fast Fourier Transform* with $O(n^2 \log(n))$ effort (see appendix A.2.2). Hence, the entire method requires only $O(n^2 \log(n) + n^2) \subseteq O(n^2 \log(n)$ operations, where the second addend of the first complexity class arises from the search for the maximum peak value and the element-wise multiplications.

[3] Discrete Fourier Transform of a 2-dimensional image $f(x, y)$ of size $n \times m$:
$\mathcal{F}\{f(x, y)\} = F(u, v) = \frac{1}{nm} \sum_{x=0}^{n-1} \sum_{y=0}^{m-1} f(x, y)e^{-2\pi i(\frac{ux}{n} + \frac{vx}{m})}$.

[4] Kronecker delta: $\delta_{ij} := \{if\ i = j : 1, \quad \text{else} : 0$

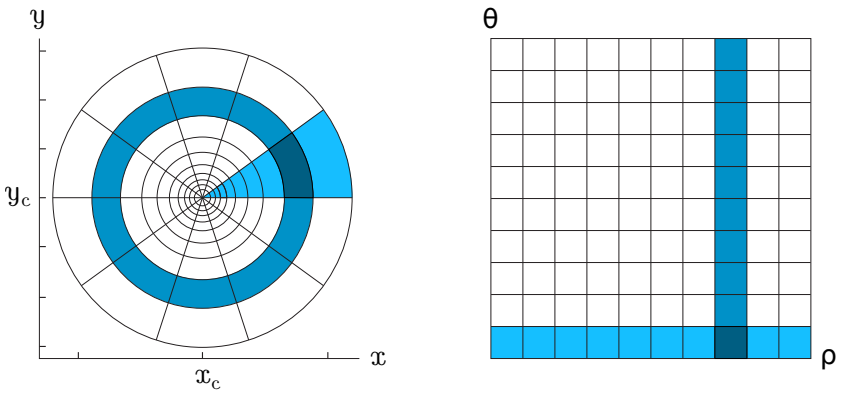


Figure 4.4: Approximate, discretized mapping from Cartesian space to polar space. The lighter-colored horizontal area in the (ρ, θ)-space indicates equiangular cells in the Cartesian space whereas the vertical area along the θ-axis corresponds to cells of the same radial distance [Sarvaiya 12].

The proposed Fourier Transform approach actually assumes a periodic 2-dimensional function instead of a truncated image. Likewise, translations are in fact expected to be linear shifts rather than circular shifts so that peaks in $r(x, y)$ will be distorted. This can be compensated by applying a *window function* and / or *zero-padding* to the images in order to reduce those edge effects.

4.4.3 Rotation-Invariant Phase Correlation

The presented phase correlation method provides an efficient solution for image registration where both images are translated relative to each other. However, finding or remembering more complex patterns of dynamic objects such as height map information of vehicles is likely to involve also a rotational component. Small rotations are certainly compensated by the memory's subsampling steps but one might also wish to detect images that are considerably more rotated and not to treat them as two separate patterns.

The *polar transform* is a mapping from the Cartesian domain to the polar domain providing the property of rotation invariance where Cartesian coordinates (x, y) are conformally mapped to polar coordinates (ρ, θ). Any rotation within the former image space corresponds to a translation in the polar domain [Sarvaiya 09],[Sarvaiya 12],[Wolberg 00].

Let ρ denote the radial distance from a center (x_c, y_c) of an image and θ the polar angle. Any Cartesian point (x, y) can then be expressed in a polar coordinate system by

$$\begin{pmatrix} \rho \\ \theta \end{pmatrix} = \begin{pmatrix} \sqrt{x - x_c)^2 + (y - y_c)^2} \\ \tan^{-1} \frac{(y-y_c)}{(x-x_c)} \end{pmatrix} . \tag{4.15}$$

Radial lines—or circles—in the Cartesian space map to vertical lines in the polar space as shown in figure 4.4. Therefore, the presented phase correlation method applied to images transformed to the polar domain allows for a rotation-invariant template matching where a rotation angle can be identified by a translational displacement along the θ-dimension within the (ρ, θ)-space.

Assuming two images are translated versions of each other, where $(\Delta x, \Delta y)$ is the respective translational component, (ρ, θ) are the polar coordinates of the first, original image and

(ρ', θ') those of the displaced image. Obviously, the corresponding polar coordinates of the second image are affected by the Cartesian translation relative to the first image:

$$\begin{pmatrix} \rho' \\ \theta' \end{pmatrix} = \begin{pmatrix} \sqrt{(\rho\cos\theta - \Delta x)^2 + (\rho\sin\theta - \Delta y)^2} \\ \tan^{-1}\frac{(\rho\sin\theta - \Delta y)}{(\rho\cos\theta - \Delta x)} \end{pmatrix} . \tag{4.16}$$

Hence, if the second image is additionally rotated against the first one, both translational and rotational components will not be distinguishable from each other in polar space. For this reason, the polar domain is not suitable for the translation extraction. In order to obtain both parameters properly, the entire approach has to be two-tiered: for the purpose of rotation extraction, the polar transform is only applied to the magnitude spectra of the Fourier transformed images. As mentioned previously, the magnitude spectra of purely translated images are identical. Rotational varieties, by contrast, appear as the same in this domain as well and clearly emerge as translations in polar space where a phase correlation gives the respective rotation angle. Subsequently, the identified relative rotation is corrected by back-rotating the second image and the remaining translational displacement can be finally found by means of an ordinary phase correlation.

4.4.4 Summary — The Entire Remembering Process

An efficient, rotation-invariant method for template registration has been introduced which provides the basis for the working memory's remembering algorithm. The complete template matching process is briefly summarized below and depicted in figure 4.5:

Rotation-Invariant Template Matching

1. Both search and template Aspects $s(x, y),\ t(x, y)$ are loaded and adjusted in size by means of zero-padding if necessary.

2. A 2-dimensional Fast Fourier Transform is applied on both images yielding the Fourier space counterparts $S(u, v),\ T(u, v)$.

3. Optionally, both FFT images can be high pass filtered in order to emphasize edges and contours in support of the subsequent rotation detection.

4. The magnitude spectra $|S(u, v)|,\ |T(u, v)|$ of both (filtered) images are calculated in order to omit translational components.

5. $|S(u, v)|,\ |T(u, v)|$ are transformed to polar space.

   (a) Define an appropriate (ρ, θ)-space depending on the image size and the desired or expected angular resolution. Compensate the loss of precision along the radial axis in polar space by choosing a sufficient resolution.

   (b) Find for all defined polar coordinates (ρ, θ) the corresponding Cartesian coordinates (u, v) of the magnitude images according to equation 4.15.

   (c) Interpolate both images accordingly since in general no perfect Cartesian matches are to be expected.

6. The phase correlation method is applied on both magnitude spectra $|S_p(u_p, v_p)|,\ |T_p(u_p, v_p)|$ in polar space.
   (a) A 2-dimensional Fast Fourier Transform is applied on both images yielding $S'(u', v'),\ T'(u', v')$.
   (b) The normalized cross power spectrum $R(u', v')$ of $S'(u', v'),\ T'(u', v')$ is computed according to equation 4.13.
   (c) The Inverse Fast Fourier Transform is applied on $R(u', v')$ yielding the normalized cross correlation $r(u_p, v_p)$.
   (d) Search for the peak value within $r(u_p, v_p)$ in order to find the corresponding coordinates indicating the best match.
7. Calculate the estimated rotation from the θ-coordinate of the peak value.
8. Correct the rotation by rotating back the template image using an appropriate interpolation method.
9. Perform a second phase correlation on the initial search image and the corrected template image in order to detect translational components.
   (a) Similar to steps 6 (a) – (d).
10. Calculate the estimated translation from the (x, y)-coordinates of the peak value.
11. Correct the translation by shifting back the template image.
12. Calculate the remaining error between both registered images.

The template image is finally registered at the best match and the remaining error can be used to decide if the registration is valid or must be rejected. Furthermore, evaluating the normalized cross correlation map directly at all significant peak values, allows for finding multiple occurrences of the same template image within the search image.

Combining several (successive or parallel) matching operations for each memory unit or stage, allows to search for arbitrarily displaced and rotated patterns within an entire Aspect Memory in a fuzzy manner. The next chapter will address the issue of an efficient, parallel implementation including the remembering process in particular.

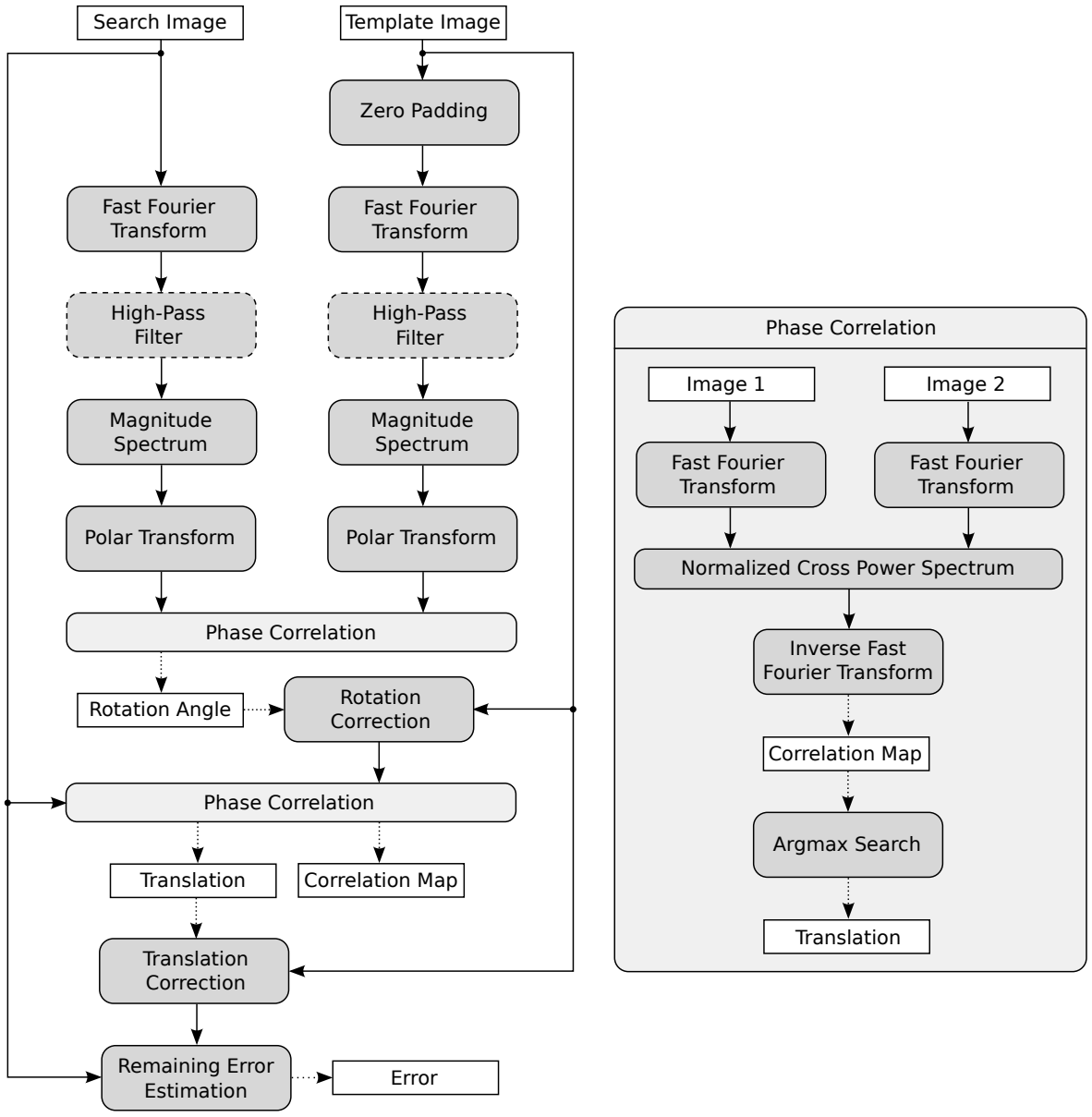


Figure 4.5: Diagram of the entire rotation-invariant phase correlation process. Rounded boxes depict functional blocks, cornered boxes are data sources or results and dashed blocks are optional. The phase correlation method is shown in detail on the right. Cf. [Sarvaiya 12].

5. Implementation

This chapter proposes an efficient and versatile implementation of the presented concepts. To this end, the content structure of the previous chapter is taken up again but considered from a practical, implementational viewpoint. Initially, the implemented solution will be introduced on a comprehensive architectural level whereas the remarks on both forgetting and remembering processes will elaborate some selected algorithmic and technical optimizations. As part of that, a highly parallel `GPGPU` implementation tailored to the problem structure is presented.

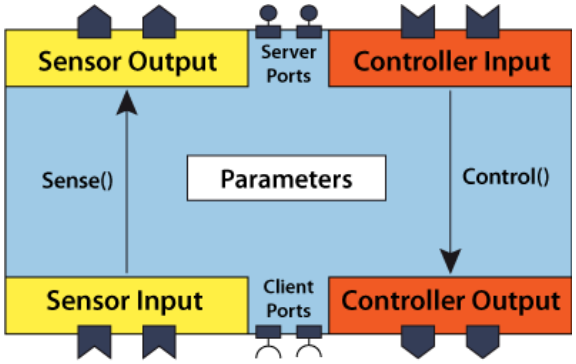
5.1 Preliminary Remarks

5.1.1 Finroc Framework

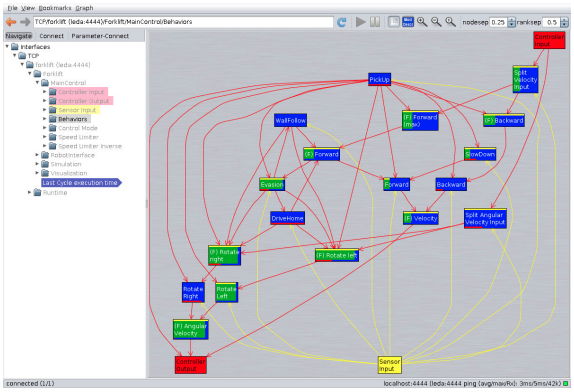
All software components of the Aspect Memory have been implemented with `Finroc`[1] (Framework for Intelligent Robot Control), a modular, distributed real-time framework developed by the Robotics Research Lab at the University of Kaiserslautern as a replacement for the `MCA2`[2] framework. `Finroc` is also used for all simulation and control tasks within the `THOR` project introduced in section 1.1.

The `Finroc` Framework is available in native `C++` and `Java` and guides a component-based implementation with *modules* as basic parts which can be organized in *groups*. Modules communicate via data ports connected with edges that can be managed and inspected at runtime. Additionally, simple and complex data types such as parameters, maps or point clouds can be visualized in graphical tools. A particular control architecture is composed of hierarchically organized modules where two crucial types of opposed data flows can be identified: sensor data flowing upwards along the hierarchies and top-down control data generated in modules and passed to lower level components or actuators. The basic structure of a `Finroc` module is shown in figure 5.1a and a visualized structure of several modules connected with data edges is shown in figure 5.1b. A more extensive introduction to `Finroc` is given by [Reichardt 12] and the benefits of real-time data visualization are stated by [Reichardt 14].

[1] `http://www.finroc.org`
[2] `http://www.mca2.org`



(a) `Finroc` module with parameters, ports and methods to process sensor and control data flows.



(b) Finstruct, a graphical `Finroc` tool for component visualization and data flow management.

Figure 5.1: Basic module and component structure of the `Finroc` framework.

5.1.2 Parallel Processing

Both the Aspect filtering an combination operations of the Cognitive Maps Architecture are element-wise operations which apply an operator or respectively a kernel on each element of the initiating Aspect. Since the computations on each element's value are independent from each other, a parallel implementation is feasible and perfectly possible as well. In the following, two different parallelization techniques will be introduced: initially an interface for parallelization on conventional multi-core processors and finally a highly parallel computation technique using graphics processing units.

Open Multiprocessing

Open Multiprocessing (`OpenMP`)[3] is a convenient application programmable interface for shared memory multi-processing in `C`, `C++` and `Fortran` supported on many platforms. Among other multi-threading methods, `OpenMP` supports the preprocessor directive `#pragma omp parallel` to fork several threads in order to distribute the workload of an annotated parallel region. Loop constructs such as `omp do` and `omp for` can be used to parallelize so called *embarrassingly parallel* problems which can be easily decomposed into independent subtasks [Samadi 14]. As mentioned above, the element-wise filtering and combination operations obviously belong to this category. Therefore, the respective iteration loops can be simply annotated with the `#pragma omp parallel for` directive. This enables speedups of the order of the number of available threads depending on both external workloads and the allocatable hardware resources.

GPGPU Processing

General-purpose computing on graphics processing units (`GPGPU`) stands for the GPU computation of problems commonly handled by CPUs. The highly parallel nature of the GPU's hardware, traditionally used for typical computer graphics tasks involving vast amounts of pixel manipulations, also allows for substantial speedups of suitable parallelizable problems of other scopes.

[3] `http://www.openmp.org`

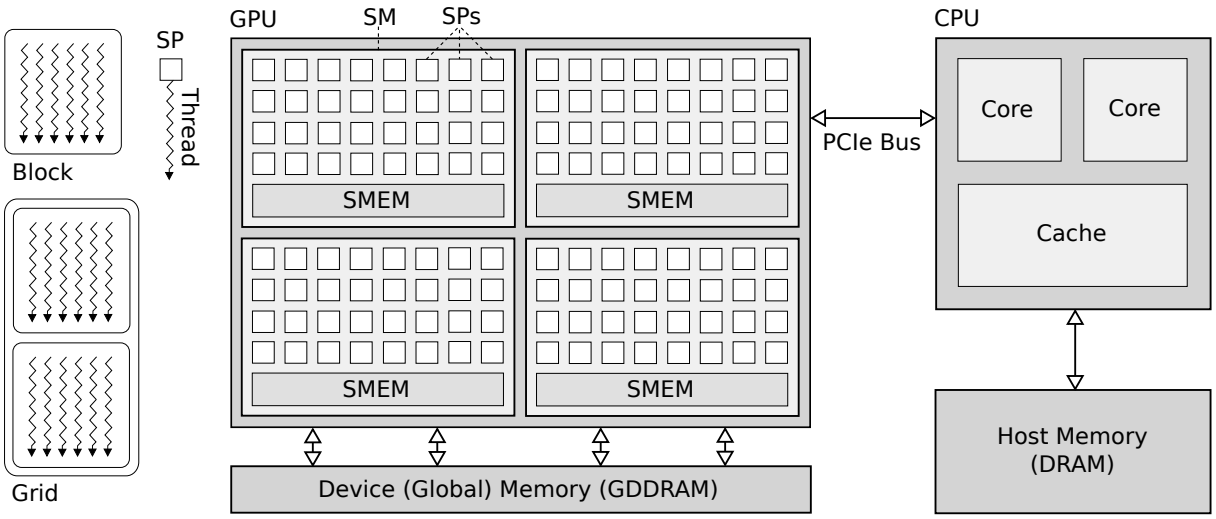


Figure 5.2: Abstract model of CPU and GPU: several stream processors (SPs) form a stream multiprocessor (SM) as part of the entire GPU device, which is internally equipped with a global memory and communicates externally with the host side and its memory via the `PCIe` bus. Cf. [Nvidia 13].

An abstract GPU model is shown in figure 5.2 where *stream processors* (SPs)—the GPU's elementary processing units—form groups of *stream multiprocessors* (SMs) each assigned to a *shared memory* (SMEM) that can be accessed by all its stream processors. Each GPU *device* contains multiple stream multiprocessors and can access a global memory which is significantly slower than the separate shared memory. Lightweight *threads* associated with single stream processors execute arbitrary operations—ideally floating point operations and can store local variables in *registers*. Thread groups are called *blocks* and assigned to stream multiprocessors. Several blocks form *grids* which are executed on single devices. Many threads (thousands) organized in hierarchical levels of blocks and grids can be efficiently launched by a so called *kernel*[4] which is a function whose code is executed by all its threads in parallel on possibly different paths (see [Nvidia 13, Sanders 10], chap. 1).

Basically, a problem appropriate for `GPGPU` should be composed of many independent, rather simple operations which can be easily deployed on the GPU's hardware structure. Any (element-wise) vector or matrix operations are therefore eminently suitable as long as a useful and computational efficient mapping to the hierarchy of threads, blocks, grids and their associated memory types can be found. However, attention should be paid to the memory transfer process between GPU devices and the CPU host: during the overall computation, the CPU is the controlling initiator and data has to be transfered from host memory to device memory and vice versa each time intermediate or final results are required on the host side. External memory transfer is rather slow compared to internal transactions and can indeed cancel out the performance benefits of parallelization for smaller problem sizes (see [Schüle 11] chap. 8).

Several application programmable interfaces for GPUs are available such as `OpenGL`[5],

[4]Not to be confused with filter kernels although they might be implemented using `GPGPU` kernels.
[5]`http://www.opengl.org`

OpenCL[6], OpenACC[7] and CUDA[8]. The latter one has been chosen for implementations within this work. The CUDA toolkit provides also several additional GPU accelerated libraries such as cuFFT[9] which is an interface for parallelized Fast Fourier Transforms used by the remembering process.

5.2 Grid Aspect Implementation

In section 4.1, the Grid Aspect as a discretized special case of the Spatial Aspect has been introduced. Within this section, implementation details considering the addressing mode as well as the particular filtering and combination algorithms will be added.

5.2.1 Grid Aspect Maps

A Grid Aspect Map A divides the 2-dimensional spatial plane into limited, equally sized regions or cells arranged in a Cartesian grid structure and stores information in elements a_i that are associated with those regions. The cell Ar_i of each element $a_i = (Ar_i, v_i)$ can be identified by a Cartesian coordinate (x, y) where $\|x\|, \|y\| \leq r$ and r is the radius of the Grid Aspect Map in L_∞ norm. The cell size is defined to be metrical and determines—together with the radius—the spatial resolution of a Grid Aspect Map. In order to ensure the existence of a center cell $(0, 0)$ and thus a proper combinational alignment for processing, the dimension of a Grid Aspect is enforced to be $(2r + 1) \times (2r + 1)$. The cells of a Grid Aspect Map can be accessed by means of their internal coordinates (x, y) using the method `GetCell()` or by means of the respective coordinates (x', y') in the robot's world coordinate system using the method `GetCellByWorldCoords()`. The function `CoordsToWorldCoords()` is responsible for an appropriate mapping.

Directional, Distance and Uniform Aspect as Technical Special Cases

Note that with the definition above also the discretized versions of the Directional, Distance and Uniform Aspect types introduced in section 3.2 can be implemented as special cases of 1-dimensional or respectively 0-dimensional Grid Aspects:

Directional Aspect: *Directional Aspects* provide a solution for storing data without a distance information allow for a directional addressing. Considering polar coordinates $(r \cos \varphi, r \sin \varphi) = (x, y)$, only the directional sector zone $\Delta\varphi$ is of importance while no radial distance r is known. The cells can be thought of as sectors of a circle with a uniform angular width resulting in an angular resolution $\Delta\varphi^{-1} = \frac{n}{2\pi}$ where n is the total number of cells. The Directional Aspect is conveniently implemented as a $n \times 1$ grid map or array and is hence internally just a special case of a non-square Grid Aspect—yet being a semantically different data representation.

Distance Aspect: The *Distance Aspect* is basically the same as the Directional Aspect despite of the reading (interpretation) in terms of the contained information: in contrast to the former type, no directional information but only distance data is

[6] http://www.khronos.org/opencl/
[7] http://www.openacc-standard.org
[8] http://developer.nvidia.com/cuda/
[9] http://developer.nvidia.com/cufft/

available, which might be the case for data received from ultrasonic sensors. While technically equivalent to the Directional Aspect, the cells of Distance Aspects can be thought of as concentric rings where only the radial distance is definite but the associated data is invariant of any directional angle.

Uniform Aspect: *Uniform Aspects* represent the simplest case of data without any spatial component such as temperatures, time stamps or any parameters. Storing just one entity of locally invariant information can also be seen as retaining the same value for any spatial position, thus, $\forall i : a_i = A$. However, a Uniform Aspect might be simply implemented as virtually 0-dimensional Aspect by means of a 1×1 Grid Aspect.

Therefore, any discretized Aspect type can be technically derived from the Grid Aspect implementation.

5.2.2 Grid Aspect Processing

Now that the Grid Aspect implementation has been defined, the appropriate implementation of the combination and filtering operations, generically introduced in section 3.2.3, can be discussed. Up to now, some technical details such as the proper combination of unaligned Grid Aspects or the filter behavior in edge regions, where no overlapping elements can be matched, are still unconsidered and will be clarified subsequently.

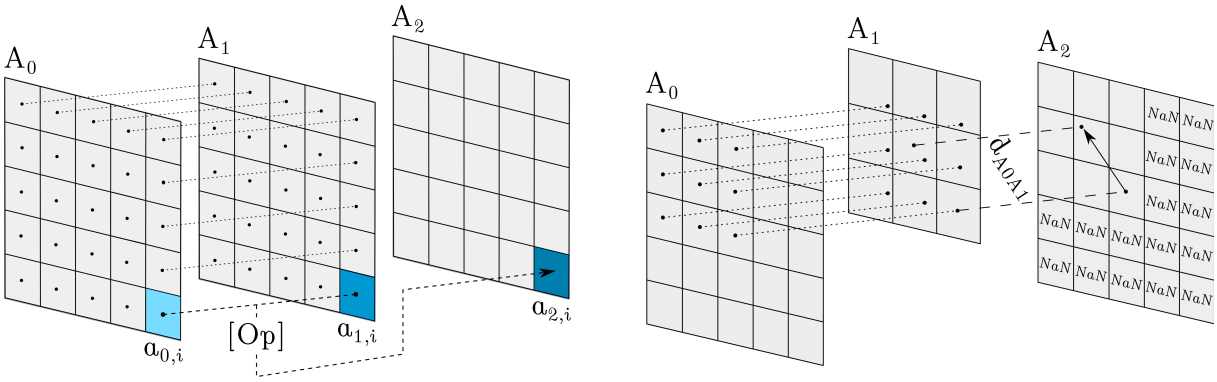
Grid Aspect Operations

By definition, the first Grid Aspect of a *Grid Aspect Operation* will be overwritten by the resulting Aspect such that the first Aspect implicitly defines the result's dimension and resolution. To this end, the element-wise value operator *op* are applied beginning with each element of the first Grid Aspect while trying to find an appropriate match within the second Grid Aspect[10].

Two Grid Aspects with different resolutions are combined using a sampling mechanism where the first one determines the resolution of the outcome. For each element $a_{0,i} := (Ar_{0,i}, v_{0,i})$ of Grid Aspect A_0 the world coordinates related to the region or cell $Ar_{0,i}$ are computed and a respective candidate region $Ar_{1,j}$ at the same spatial position within Grid Aspect A_1 is looked up. If successful, the values $v_{0,i}$ and $v_{1,j}$ can be combined using the value operator *op*, otherwise, the resulting value $v_{2,i}$ is set to NaN. Algorithm 5.1 describes the procedure in detail and figure 5.3a illustrates the combination process graphically.

As mentioned earlier, despite the case of *aligned combinations* it is also possible to combine Grid Aspects where their origin points are not aligned, i.e. there is a displacement vector $\vec{d}_{A_0A_1} = \vec{o}_{A1} - \vec{o}_{A0}$ between the two Grid Aspect's origin points. Thus, both Aspects are offset against each other in respect of the world coordinate system. *Unaligned combinations* must take the displacement into consideration when calculating matching elements but behave apart from that basically exactly like aligned combinations. An unaligned combination can also be explicitly enforced by passing an additional displacement vector to the operation: $A_2 = Op\ (A_0, A_1, \vec{d}_{A_0A_1})$. This might be useful when constructing a larger Aspect Map from several existing sub-areas. Figure 5.3b shows an unaligned combination of Grid Aspects with different resolutions.

[10]For this reason, the implemented version of an operation *Op* is indeed not commutative even if its operator *op* is.



(a) Grid Aspect Operation $A_2 = Op\ (A_0, A_1)$.

(b) Grid Aspect Operation with distinct cell resolutions and displacement $\vec{d}_{A_0 A_1}$.

Figure 5.3: Graphical representation of aligned and unaligned Grid Aspect Operations.

Note that due to the definition of the Grid Aspect (see paragraph 5.2.1) all explanations above are also valid for the Directional, Distance and Uniform Aspect.

Algorithm 5.1: Grid Aspect Operation

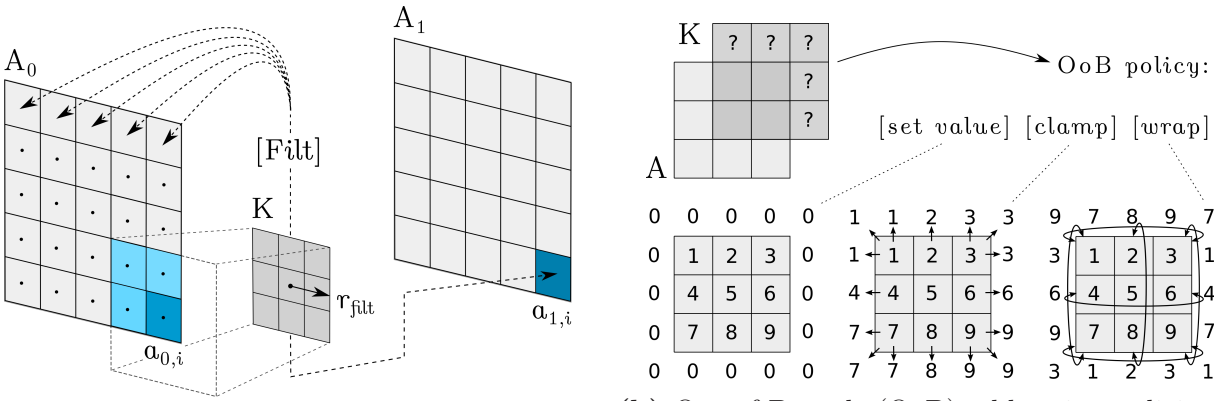
Data : Grid Aspects A_0, A_1; Combination operator $op\ (v_{0,i}, v_{1,j}$
Result: Grid Aspect A_2

for $x \leftarrow -A_0.radius$ **to** $A_0.radius$ **do**
  for $y \leftarrow -A_0.radius$ **to** $A_0.radius$ **do**
    $(x', y') \leftarrow A_0$.`CoordsToWorldCoords`$((x, y))$;
    $a_{0,i} = (Ar_{0,i}, v_{0,i}) \leftarrow A_0$.`GetCell`$((x, y))$;
    $a_{1,j} = (Ar_{1,j}, v_{1,j}) \leftarrow A_1$.`GetCellByWorldCoords`$((x', y'))$;
    $Ar_{2,i} := Ar_{0,i}$;
    $v_{2,i} := NaN$;
    if $Ar_{0,i} \cap Ar_{1,j} \neq \emptyset$ **then**
      $v_{2,i} := op\ (v_{0,i}, v_{1,j})$;
      $a_{2,i} \leftarrow (Ar_{2,i}, v_{2,j})$;
    end
    A_2.`SetCell`$((x, y),\ a_{2,i})$;
  end
end
return A_2;

Grid Aspect Filters

A *Grid Aspect Filter* applies a grid-like filter kernel $K(r)$ with a certain radius r in L_∞ norm to an Grid Aspect Map A_0. For each element $a_{0,i}$ the kernel's central cell must be aligned with the element's cell $Ar_{0,i}$ and all kernel values $k_j \in K$ are element-wise combined with all elements' values $v_{0,i}$ within a radial neighborhood r of $Ar_{0,i}$.

Technically, as with combinatorial operations, the data encoded in Grid Aspect A_0 will be overwritten after a successful application of the filter kernel. Grid Aspect filtering relies



(a) Grid Aspect Filter $A_1 = Filt_{K(r)}\ (A_0, OoB)$.

(b) Out-of-Bounds (OoB) addressing policies: set value, clamp or wrap around.

Figure 5.4: Graphical representation of the filtering process applied to Grid Aspects.

Algorithm 5.2: Grid Aspect Filter

Data : Grid Aspect A_0; Filter $filt_{K(r)}\ (Ar_{0,i}, OoB)$ with kernel $K(r)$ and OoB policy
Result: Grid Aspect A_1

```
for x ← −A0.radius to A0.radius do
    for y ← −A0.radius to A0.radius do
        a0,i = (Ar0,i, v0,i) ← A0.GetCell((x, y));
        Ar1,i := Ar0,i;
        v1,i := NaN;
        v1,i := filtK(r) (Ar0,i, OoB);
        a1,i ← (Ar1,i, v1,i);
        A1.SetCell((x, y), a1,i);
    end
end
return A1;
```

on the same cell matching procedure as discussed above. Figure 5.4a shows the filtering process for a Grid Aspect.

However, edge regions of Grid Aspect Maps require a closer consideration when applying filter kernels: some kernel elements (at least one) are covered by Aspect elements but others might find no matches. In this case, the overall cell value for the resulting Aspect's element would be apriori undefined. To overcome the indefiniteness, an *Out-of-Bounds* (OoB) addressing policy is consulted for edge regions. Typically, three established techniques are appropriable: missing subset matches beyond the Aspect's borders can be either set to a specific value (e.g. $0, 1$) clamped from the Aspect's edges or obtained by wrapping around the Aspect area—artificially closing the Aspect's boundaries. Algorithm 5.2 describes the procedure in detail and the figures 5.4a and 5.4b illustrate the filtering process and the edge policies.

All explanations above are again valid for the Directional, Distance and Uniform Aspect.

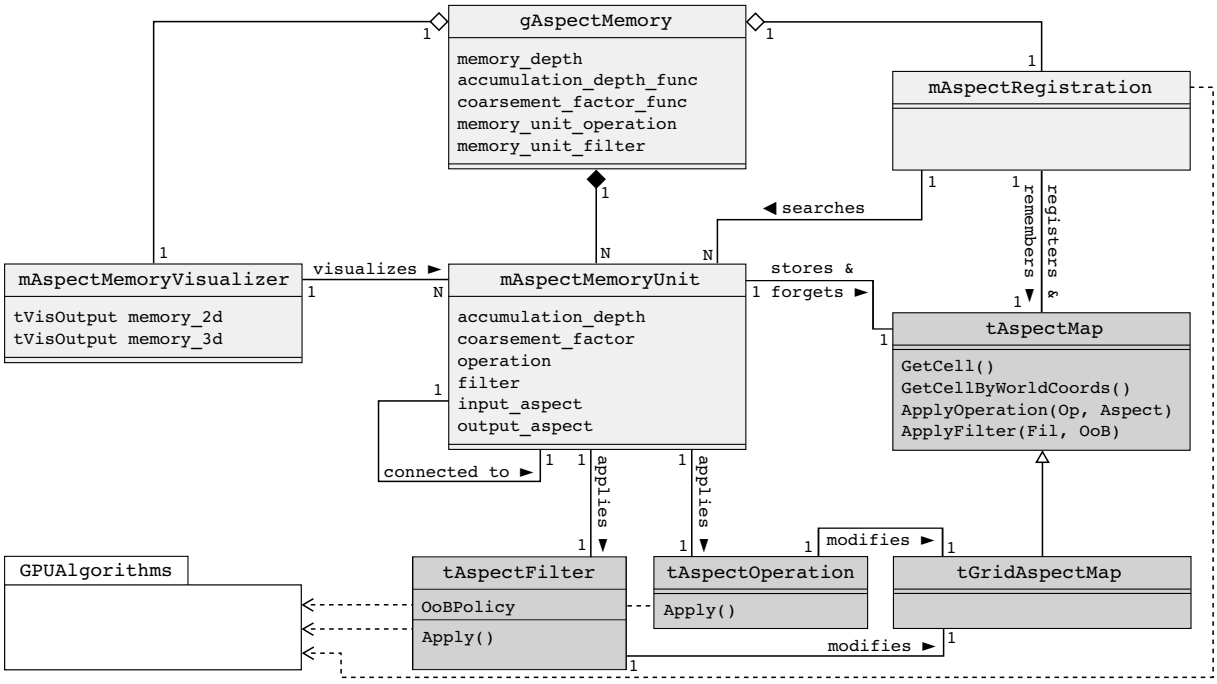


Figure 5.5: UML diagram of the Aspect Memory Architecture: `Finroc` modules implementing the Aspect Memory core are depicted in lighter gray and classes of the Cognitive Maps Architecture in darker gray.

5.3 Architecture of the Aspect Memory

The structural core of the Aspect Memory is implemented with `Finroc` modules and groups interconnected with data ports whereas basic functionalities such as Aspect Operations, Aspect Filters and GPU accelerated algorithms are encapsulated in common `C++` classes. Figure 5.5 shows the architecture with its components detailedly described below:

Aspect Memory: The `gAspectMemory` component groups can be associated with an entire particular Aspect Memory composed of memory units containing Aspects of identical types and semantic meanings but perceived at different, consecutive points in time. Those Aspects may be partially forgotten over time, i.e. their temporal and spatial resolutions may coarsen as they are passed through the memory units. While the actual processing is done within the memory units, the Aspect Memory group can be seen as the managing container object. A memory depth N has to be defined which determines the number of concatenated memory units. Two lambda functions[11] `accumulation_depth_func` and `coarsement_factor_func` dependent on $n \in [0, N-1]$ control the individual memory unit's temporal and spatial subsampling factors. Finally, the type of Aspect Operations and Aspect Filters used for temporal and spatial forgetting are also defined on this group level.

Aspect Memory Unit: `mAspectMemoryUnit` modules are components of the `gAspectMemory` group as mentioned above and store the individual temporal Aspect instances. Concatenated memory units, each containing exactly one delayed, (partially) forgotten Aspect, form the actual memory. They can thus be seen as complex delay

[11] `http://en.cppreference.com/w/cpp/language/lambda/`

nodes implementing their specific part of the entire forgetting process at a certain temporal stage. Memory units hold instances of Aspect Operations and Aspect Filters used for the forgetting process.

Aspect Memory Visualizer: The `mAspectMemoryVisualizer` has access to all memory units of the memory group and visualizes the entire memory and its stored temporal Aspect instances providing a 2- or 3-dimensional image.

Aspect Memory Registration: The `mAspectMemoryRegistration` module implements the remembering process, i.e. a fuzzy template matching method, which searches all memory units for given Aspect patterns.

Aspect Map: `tAspectMap` is an abstract `C++` class, part of the Cognitive Maps Architecture, which implements the generic Aspect Map including common storage and addressing functionalities independent from any specific Aspect type.

Grid Aspect Map: `tGridAspectMap` is a specific grid implementation of the abstract `tAspectMap` class according to the definition of the Grid Aspect type described in section 5.2.1.

Aspect Operation: `tAspectOperation` is an abstract class defining the generic Aspect Operation Op of the Cognitive Maps Architecture and can be specialized to specific operator implementations op such as weighted additions. The implementation for Grid Aspects is similar to the algorithm described in section 5.2.2.

Aspect Filter: `tAspectFilter` is an abstract class defining the generic Aspect Filter $Filt$ of the Cognitive Maps Architecture and can be specialized to specific filter implementations $filt$ such as Gaussian smoothing. The implementation for Grid Aspects is again similar to the algorithm described in section 5.2.2.

GPU Algorithms: The `GPUAlgorithms` package is essentially used by remembering and template matching methods of the `mAspectMemoryRegistration` module but is also intended to accelerate implementations of the basic `tAspectOperation` and `tAspectFilter` modules involved in the forgetting process. It provides optimized `GPGPU` routines implemented in `C++/CUDA` for efficient filtering tasks and a parallelized version of the entire remembering process proposed in section 4.4.4.

5.4 Forgetting

Forgetting in the Aspect Memory sense is according to section 4.3.5 a combination of subsequent Aspect Operations and Aspect Filters each followed by a subsampling step. Large Aspect Maps and / or those with a high resolution such as environmental high maps are likely to contain thousands of elements per instance which implies several thousands of elementary computations each time a new Aspect instance is perceived and stored in its respective memory. Since forgetting is a continuous process performed at each point in time for each Aspect Memory as part of the robot system's entire memory, the question arises whether those basic element-wise operations can be implemented more efficiently.

As suggested above in section 5.1.2, both processing operations can be easily parallelized in order to distribute the overall workload to several threads either executed by the CPU or the GPU in the `GPGPU` sense. While the Aspect Operation requiring $O(n)$ computations is

an asymptotically optimal[12] algorithm, the Aspect Filter incorporates for each element all elements within a radial neighborhood or those of the respective kernel. Assuming a typical smoothing filter used for forgetting with k kernel elements, a filtering task involves $O(nk)$ basic operations. Subsequently, algorithmic optimizations for specific but common filter types will be discussed which can speed up filtering tasks and thus the overall forgetting process substantially.

5.4.1 Separable Convolution Filters

The kernel matrices of some convolution filters typically used for spatial filtering presented in section 4.3.3 share the interesting property of *separability*[13], i.e. they can be decomposed into two consecutively applicable 1-dimensional kernels yielding the same result with less computational effort [Podlozhnyuk 07b]. The resulting partial kernels are column and row vectors whose outer product is equal to the original kernel matrix as shown in the following example of the Gaussian kernel:

$$\frac{1}{4}\begin{bmatrix}1\\2\\1\end{bmatrix} \otimes \frac{1}{4}\begin{bmatrix}1 & 2 & 1\end{bmatrix} = \frac{1}{16}\begin{bmatrix}1 & 2 & 1\\2 & 4 & 2\\1 & 2 & 1\end{bmatrix} \tag{5.1}$$

It turns out that the outer product of a column vector $\vec{v}$ and a row vector $\vec{h}$ is equivalent to their convolution[14] (see appendix A.2.1):

$$\vec{v} \otimes \vec{h} = \vec{v} * \vec{h} \tag{5.2}$$

Thus, a separable filter can be applied by first convolving a 2-dimensional Aspect with the 1-dimensional vertical kernel and finally convolving the result with the horizontal kernel or vice versa (since convolution is associative).

Assuming a $n \times n$ Grid Aspect matrix and a $k \times k$ kernel matrix, an ordinary filtering requires n^2k^2 multiplications and additions according to equation 4.6. Instead, a separable filter requires only $2n^2k \in O(n^2k)$ operations, which implies a substantial theoretical speedup of $\frac{k}{2} \in O(k)$.

5.4.2 Fast Fourier Convolution Filters

Convolution filters introduced in section 4.3.3 compute 2-dimensional convolutions of Aspects and kernel matrices in time domain. In section 4.4.2, where a fast template matching procedure for the purpose of remembering has been discussed, the Fourier Transform has already been used for a efficient correlation operation. Since correlation and convolution are mathematically similar[15], both operations can be significantly faster applied in the Fourier domain. A convolution of two 2-dimensional images $f(x, y), g(x, y)$

[12] $O(n)$ is obviously a lower bound for computing n element-wise operations.

[13] In order to be separable, a matrix K must have $rank(K) = 1$.

[14] `http://blogs.mathworks.com/steve/2006/10/04/separable-convolution/`

[15] Correlation can be expressed as convolution and vice versa by flipping one of both involved series or images along each dimension.

in time domain maps to a simple element-wise multiplication of their corresponding spectra $F(u,v), G(u,v)$ in the Fourier domain:

$$f(x,y) * g(x,y) = \mathcal{F}^{-1}\{\mathcal{F}\{F(u,v)G(u,v)\}\} \ . \tag{5.3}$$

Thus, a 2-dimensional convolution of two images of size $n \times n$ can also be computed in time $O(n^2 \log(n) + n^2) \subseteq O(n^2 \log(n))$ using Fast Fourier Transforms (see appendix A.2.2), which is asymptotically optimal. Note that for this method both images—or the Grid Aspect and kernel matrices respectively—must be equally sized. If not, n is chosen to be the maximum of both sizes and the smaller matrix is zero-padded properly. This transformation provides not only a theoretical, algorithmic optimization but is also practically favorable since the quite complex convolution operation involving several basic operations for each resulting element is mapped to a pure element-wise operation which can be easily parallelized for `GPGPU` processing. Of course, also the two 2-dimensional Fourier Transformations and the inverse back transformation have to be taken into account, which can be efficiently implemented using `CUDA`'s parallel Fast Fourier library `cuFFT` as described in detail in [Podlozhnyuk 07a].

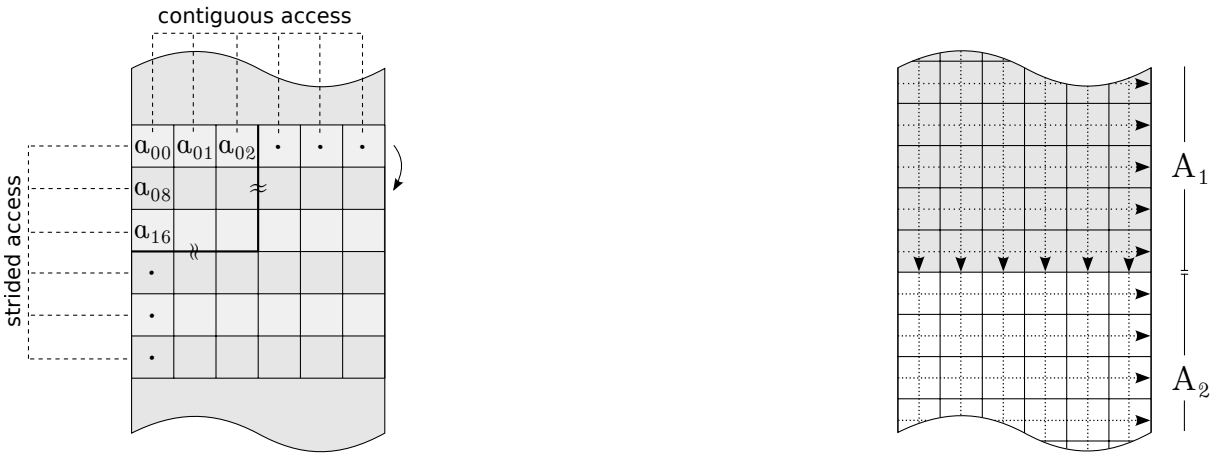
5.4.3 Practical Considerations: Hybrid Approach

Two different approaches allowing for efficient implementations of convolution filters have been presented whereas the first solution using separability in the time domain seems to be inferior to the second one in terms of computational efficiency for large values of the kernel size k. But the O-notation omits any potentially major factors: the second method requires actually three additional Fourier Transforms. Hence, it is to be expected that the second version is practically only preferable for larger Aspect sizes.

For this reason, it has been decided to compute convolutions of smaller Aspect and kernel sizes directly in time domain by means of the separable convolution and to parallelize iteration loops with the aid of `OpenMP` on the CPU. Instead, (very) large Aspects can be transfered to the GPU and convolved in the Fourier domain according to the second method using `cuFFT` for Fourier Transforms and specific `GPGPU` kernels for the remaining parts of the procedure. As mentioned earlier in section 5.1.2, GPU processing is only worthwhile for larger problem sizes as a matter of principle due to expensive memory transfers. Aspect Operations are treated similarly and can be transfered to the GPU as well dependent on their number of elements. Consequently, incorporating this case differentiation, a convenient and efficient complete solution has been made available.

5.5 Remembering

In contrary to forgetting, remembering is a more complex process as stated in section 4.4.4 and composed of several phase correlations which should be computed in the Fourier domain not only for reasons of efficiency but also in order to reduce noise and allow for rotation invariance. Since remembering is basically a fuzzy search within an Aspect Memory or technically a set of template matchings for each memory unit, several iterations of the algorithm shown in figure 4.5 are necessary. Furthermore, albeit computationally more expensive, it can be expected to be a less frequently requested process compared to forgetting. Thus, it is appropriate to transfer the entire remembering process to the GPU



(a) Elements of horizontal rows can be contiguously accessed whereas column elements require strided accesses.

(b) Multiple Grid Aspects stored consecutively in memory and their 1-dimensional FFT decompositions.

Figure 5.6: Grid Aspect in memory: elements are sequentially stored such that horizontally neighbored elements are still neighbors in the memory but vertically neighbored elements are distant.

with the advantage of workload reduction on the CPU. Efficient `GPGPU` implementations of more extensive procedures, which are not obviously parallelizable, involve also different algorithmic approaches. For example, a common sequential maximum search is not suited for GPU computing. Those and other selected issues of the parallelized remembering approach will be discussed in this section.

5.5.1 Batched Separable Fast Fourier Transforms on the GPU

The phase correlation method necessarily requires 2-dimensional Fast Fourier Transforms which are decomposed (separated) into batches of horizontal and vertical 1-dimensional Fast Fourier Transforms similarly to the fast convolution filtering discussed above in section 5.4.2. This provides not only an algorithmically optimal time complexity as indicated in appendix A.2.2 but also helps to better use the GPU to capacity since those smaller 1-dimensional sequences can be independently transformed. [Al Umairy 12] explains the benefits of smaller Fourier Transforms on GPUs in detail.

The `cuFFT` library provides a so called *batched mode* which can efficiently process several Fast Fourier Transforms of the same sizes in parallel. After transforming all horizontal image rows, the second batch containing all vertical columns must be processed. Commonly, this requires an image or matrix transposition since some libraries expect the elements to be contiguously stored in memory. However, this would imply additional computational effort. In contrast, `cuFFT` allows custom memory layouts where the memory distance (offset) between consecutive elements of a single Fast Fourier Transform can be specified which allows to access the initial matrix in a *strided* manner and transform the columns directly as shown in figure 5.6a. It should be noted that strided memory accesses are significantly slower than contiguous accesses[16] but in this particular case still more favorable than an

[16]Recent GPU architectures alleviate those problems by use of sophisticated caching techniques.

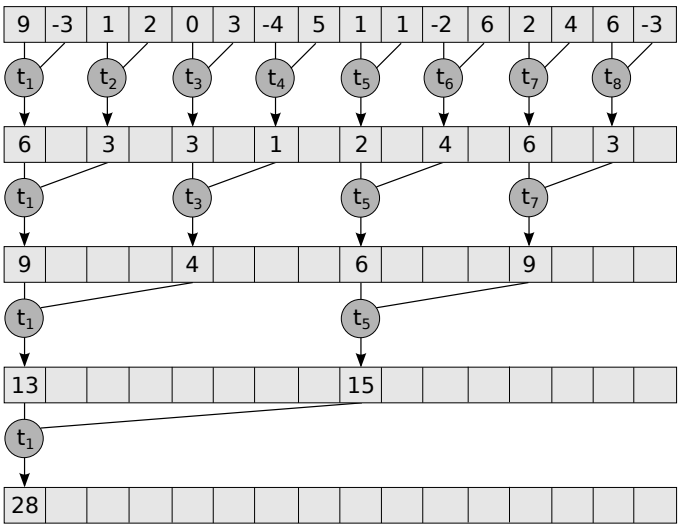


Figure 5.7: Reduction tree for a sum operation: each operator node reduces two elements of the preceding level to a new interim result such that the problem size is divided by half at each level.

additional transposition. After the entire process the transformed matrix (or sequence in memory) needs twice as much memory as before since real values are transformed to complex values comprised of an real and an imaginary part.

The approach can be even further optimized for the entire remembering process where several template matchings are performed and thus all Aspects stored within an Aspect Memory are required in Fourier domain. Before transferring all single Aspects from to the GPU device, they can be copied in succession into a contiguous host memory block which helps to exploit the available memory bandwidth and compensates the latencies of several smaller memory transfers. Furthermore, contiguously stored Aspects (of the same sizes) can be easily Fourier transformed in batched mode across their boundaries as illustrated in figure 5.6b. In this way, the number of independent 1-dimensional Fourier Transforms that can be processed in parallel is again increased.

5.5.2 Parallel Reduction Operations on the GPU

As mentioned above, the phase correlation requires a search of (the argument of) the maximum contained in the correlation map, which is a linear algorithm that can be easily implemented by iterating over all elements and memorizing the interim maximum. A similar operation is the summation of all elements used for computing the total remaining error after the best match has been found. These so called *reduction operations* reduce a sequence of elements to a single output value in respect of a specific operator. It might be admittedly possible to adapt these methods by utilizing one single `GPGPU` thread just as with a sequential CPU version but this would not nearly use the GPU to its capacity. As a consequence, appropriate parallel algorithms must be considered.

The basic idea of parallel reductions is to firstly reduce subsets of the entire sequence before reducing the partial results of those subsets again. Typical reduction operators are binary operators such as an addition or a maximum function taking two arguments.

Thus, reductions trees are mostly binary trees dividing the remaining problem size in half at each level. Figure 5.7 illustrates the procedure for the case of a sum reduction. The number of nodes on each level of the tree indicates likewise the degree of parallelization at the related reduction step: initially, a sequence of n elements is processed by $\left\lceil \frac{n}{2} \right\rceil$ threads reducing the sequence to the respective number of interim results and so forth. Thus, the reduction tree has a height of $\lceil \log_2(n) \rceil$. Early steps of the algorithm benefit from the great number of available `GPGPU` threads, when many independent subsequences of length two can be reduced, whereas the last steps are hardly parallel anymore. However, most of the work is done in the upper part of the tree and, due to its logarithmic structure, only a few steps are required even for large input sequences.

While conceptually simple, efficient implementations of parallel reductions are technically quite challenging. A reduction tree shown in figure 5.7 features equidistant gaps in memory which grow with the number of levels and involve slow strided accesses. Instead, partial reduction results should be ideally stored as neighboring elements and located within the same shared memory of the same thread block such that related threads can access them efficiently. Furthermore, most of the initially launched threads become idle at further reduction steps which is—as indicated—wasteful on the one hand and causes the unfavorable effect of parallel but divergent execution paths on the other hand. For the latter reason, threads must be synchronized at each reduction level which is again critical in view of performance. Those issues and further optimizations elaborated in [Harris 07] have been considered within the actual implementation. For GPUs equipped with the recent `Kepler`[17] architecture, a more efficient solution using so called *shuffle operations*[18] has been tested. These are very efficient, atomic memory operations enabling threads to directly read registers from other thread in the same *warp*—a group of 32 technically related threads—which obviates the need for extensive synchronization and shared memory usage.

5.5.3 Rotations on the GPU

The rotation-invariant template matching algorithm requires image or matrix rotations in order to correct the template image according to the estimated rotation angle. An initial vector (x, y) or likewise a respective pixel can easily be rotated by an angle α as follows:

$$\begin{pmatrix} x_\alpha \\ y_\alpha \end{pmatrix} = \begin{pmatrix} \cos\alpha & -\sin\alpha \\ \sin\alpha & \cos\alpha \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} \tag{5.4}$$

Thus, for each rotated target coordinate the value at the corresponding source pixel is copied from the initial image. Since image pixels or Aspect cells are discretely arranged in a grid structure, values that do not (exactly) map onto a data point must be interpolated. A simple *bilinear interpolation* has been implemented for this purpose (see appendix A.2.3). The procedure can be computed independently for each target pixel and is hence an ideal candidate for a `GPGPU` kernel.

However, a sophisticated approach for low-loss rotations in the Fourier domain using a decomposition of rotations into shears and Fourier properties is discussed in [Larkin 97].

[17] `http://www.nvidia.com/object/nvidia-kepler.html`
[18] `http://devblogs.nvidia.com/parallelforall/faster-parallel-reductions-kepler/`

6. Experiments and Results

Several experiments and tests have been done in order to evaluate the Aspect Memory and its closely related forgetting and remembering processes. The general approach proposed in this thesis is basically capable of storing any data that can be encoded using the Cognitive Maps Architecture. It is hence impossible to extensively test the Aspect Memory and all its potential applications. Test data of the experiments presented within this chapter has been limited to height data of objects from construction site scenes of the THOR simulation. This might be a reasonable choice since height data makes fully use of the spatial dimensions and is both vividly representable and easy to visualize. For this purpose, a 3-dimensional visualization for Grid Aspects— similar to surface plots—has been implemented.

At first, the basic core components and algorithms involved in forgetting and remembering have been tested in regard to their functionality in section 6.1. Subsequently, results of runtime tests, which focus on the mentioned algorithmic and technical optimizations, are discussed in section 6.2. The functional interaction of all components has been evaluated within a integration test of the entire Aspect Memory in section 6.3. Finally, an advanced scenario that demonstrates further possible applications of the Aspect Memory and its remembering functionality is shown in section 6.4.

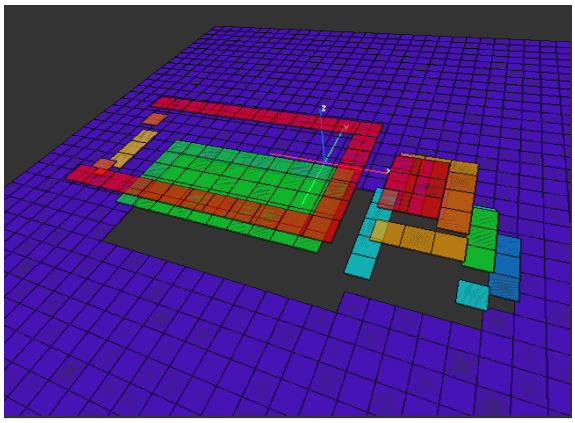
6.1 Functional Tests

This section presents functional tests that have been executed in order to ensure the correctness of the basic units and to evaluate the effects of different kernels and parameters in case of the forgetting process. For this purpose, a Grid Aspect is used as test data which stores height data of a truck model derived from a construction truck of the THOR simulation. Figure 6.1 shows a simulated construction site scene with a construction truck and the respective Grid Aspect model.

6.1.1 Forgetting

As the forgetting process is composed of two major parts—temporal and spatial forgetting, the corresponding technical components have been tested independently in detail as discussed in the following.

(a) Construction site scene in the THOR simulation.



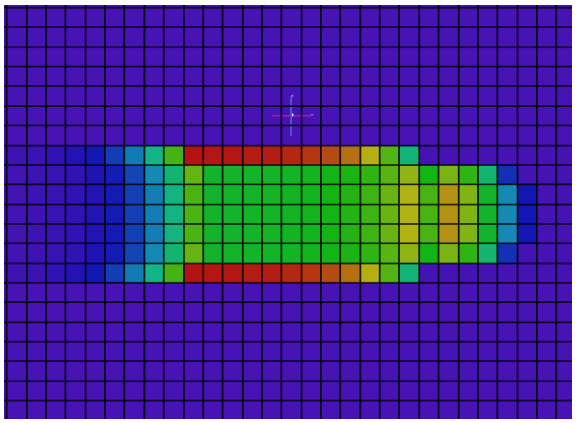
(b) Simplified Grid Aspect model of a construction truck.

Figure 6.1: Model of a construction truck in a THOR simulation scene (a) and a Grid Aspect model derived from height data (b).

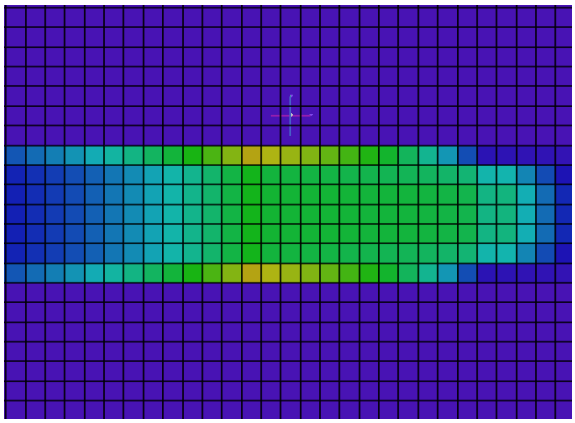
Temporal Filtering and Subsampling

Temporal filtering and subsampling constitute the temporal forgetting process, where the former component is technically an accumulation operation and the latter one simply a reduction of the frequency. The temporal subsampling has been successfully tested but is hardly visualizable with static images; hence, only the temporal subsampling is mentioned.

Typically, a weighted addition is used for the accumulation operation (see section 4.3.1) where the parameter α can be interpreted as the weight factor for already stored data and $1 - \alpha$ as the weight factor for new data respectively. As part of a test scenario, a moving truck model has been observed over time and accumulated using weighted additions with different values of α.

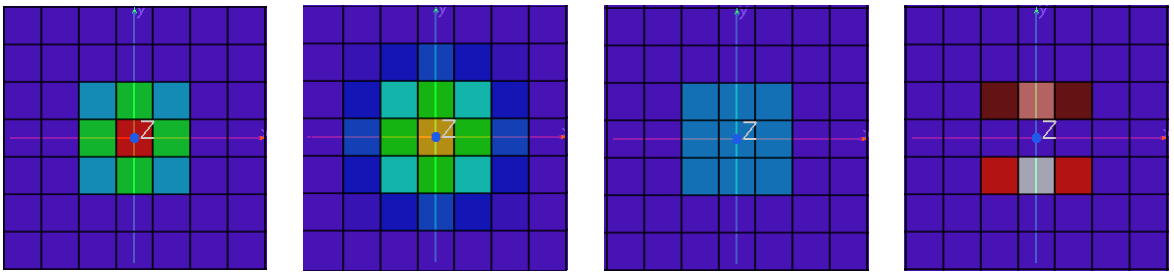


(a) Weighted addition with $\alpha = 0.60$ yielding a slight temporal blurring.



(b) Weighted addition with $\alpha = 0.90$ yielding a marked temporal blurring.

Figure 6.2: Temporal filtering applied to a moving truck model (left to right) using an accumulation based on a weighted addition.



(a) Gaussian kernel **(b)** Gaussian kernel **(c)** Mean kernel **(d)** Sobel kernel

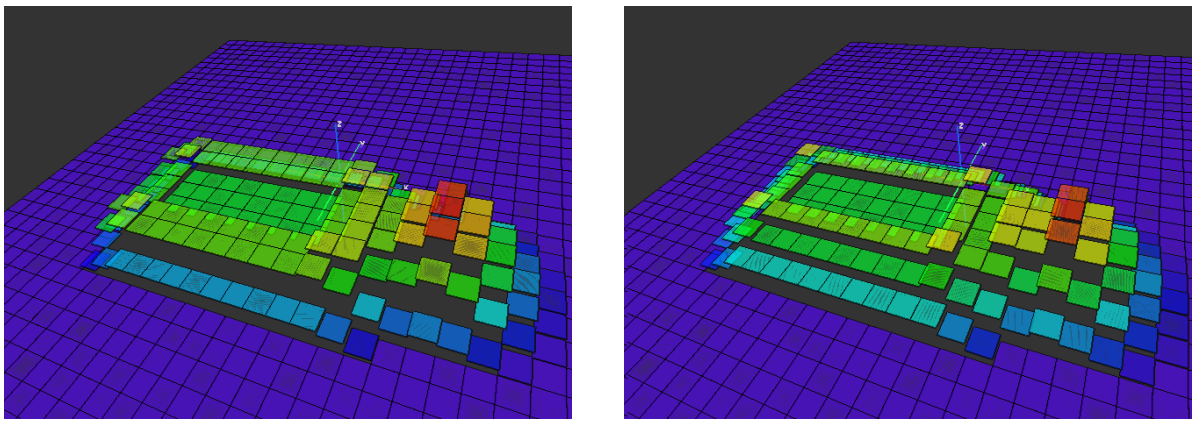
Figure 6.3: Tested spatial filter kernels: Gaussian and mean smoothing and Sobel edge detection (x-direction).

Figure 6.2 depicts two accumulation results which feature different temporal blurring or uncertainties for a model moving from left to right with the same velocity in either case for $\alpha = 0.60$ and $\alpha = 0.90$ (see appendix A.3.1 for additional results).

Spatial Filtering

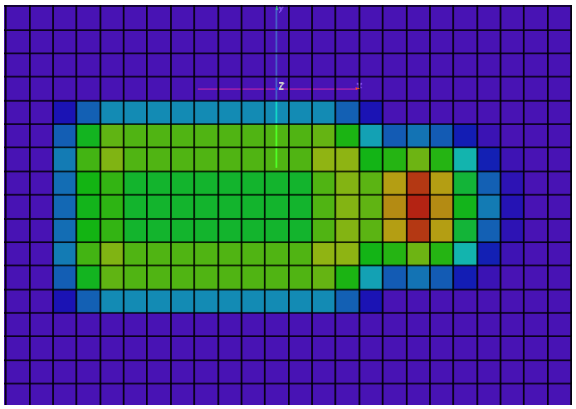
Spatial filtering as part of the spatial forgetting requires convolution operations as proposed in section 4.3.3. Depending on the underlying data and the desired behavior of the forgetting process, any filter kernel may be chosen. Several kernels have been tested in order to validate the correctness of the implementation on one hand and to investigate the filter properties and effects on the other hand. Figure 6.3 shows a selection of Gaussian and mean smoothing kernels as well as a Sobel edge detection kernel. Additionally, also a minimum, maximum and median filter are available but not depicted.

The results of Gaussian and mean smoothing are illustrated in figure 6.4. Obviously, the truck model's edges are blurred which might be unfavorable in this specific case for recognition tasks but might prove useful in case of noisy data (see appendix A.3.2 for noise compensation) or when processing information that is associated with probabilities rather than with objects.

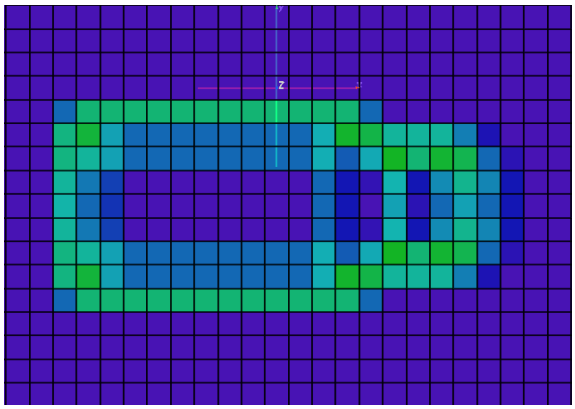


(a) Gaussian kernel applied to the truck model. **(b)** Mean kernel applied to the truck model.

Figure 6.4: Truck model convolved with two types of smoothing filters.



(a) Gaussian kernel applied to the truck model.



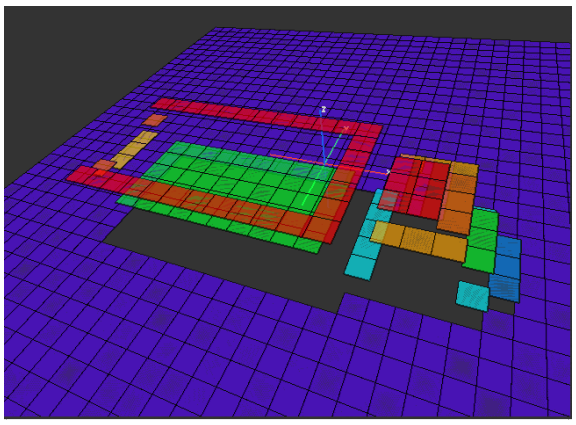
(b) Sobel kernels applied to the truck model.

Figure 6.5: Areal view of the truck model convolved again with a Gaussian kernel (a) and two Sobel kernels for both directions (b).

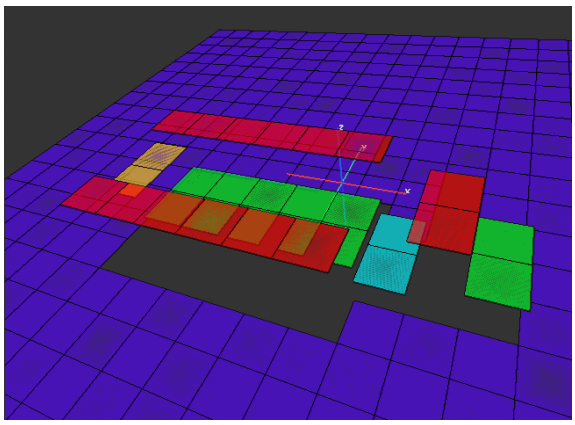
Figure 6.5 compares the results of a Gaussian filtering and a Sobel edge detection in both directions. As an instance of a high pass filter, and thus an example of a feature extractor, the Sobel kernel emphasizes the model's contours and suppresses areas of steady values which is favorable for the purpose of recognition.

Spatial Subsampling

Within the spatial forgetting process, spatial subsampling is typically applied after spatial filtering but both steps are in principle independent form each other. The spatial subsampling has hence been tested separately. Figure 6.6 shows a subsampled version of the original truck model where the resolution has been reduced by a factor of 2.

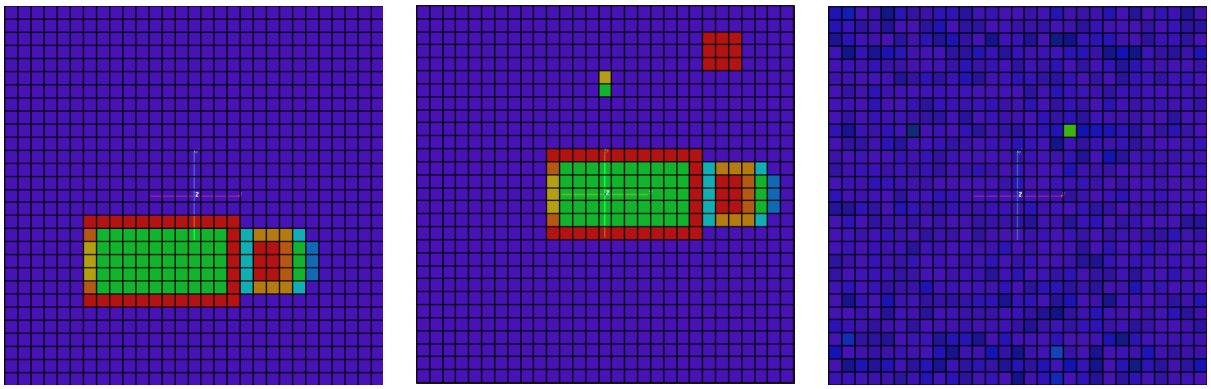


(a) Truck model with original resolution.



(b) Spatially subsampled truck model.

Figure 6.6: Spatial subsampling of a truck model with higher resolution (a) to a version with lower resolution (b).



(a) Template image: initial truck model.

(b) Search image: displaced truck model and other objects.

(c) Correlation map with a peak value at cell $(4, 5)$.

Figure 6.7: Basic phase correlation detecting a translational displacement between two Aspects.

6.1.2 Remembering

Template matching is the core functionality of the remembering process as discussed in section 4.4 and hence, two versions have been tested: a basic phase correlation capable of detecting translations and a rotation-invariant phase correlation which can also identify and compensate rotational components. The test implementation expects two Aspects of any sizes which can be translated and / or rotated against each other. A correlation map is computed where the coordinates of the peak value correspond to the most accurate match subject to a minimum sum of squared errors. Translational and rotational offsets can be estimated, the registered template and the search image are then aligned accordingly and a remaining error is computed.

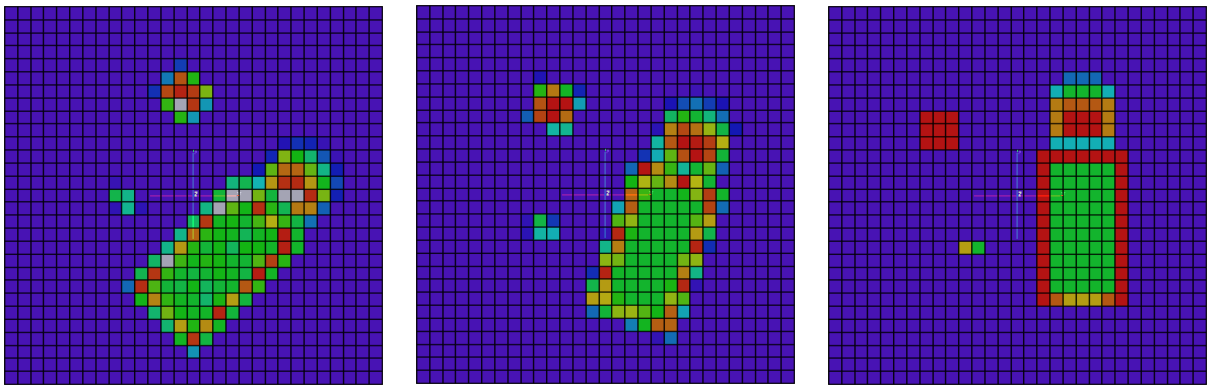
Basic Phase Correlation Template Matching

The basic phase correlation can only identify translational displacements as shown in figure 6.7. Both Aspects are offset against each other with the displacement vector $(4, 5)$. Figure 6.7a illustrates the template image and figure 6.7b the search image. A maximum peak value within the correlation map is found at the corresponding coordinates as shown in figure 6.7c.

The basic version can register template images with any displacement properly, as long as the search image contains no confusing patterns similar to the template image. In the latter case, the correlation map shows multiple peaks at coordinates that correspond to the respective candidates. However, allowing several peaks and changing the argument maximum search to a search for k largest elements, enables the algorithm to cope with this cases as well. Noisy search images with values that deviate randomly within a range of up to 30 % from their reference value have also been successfully registered, which illustrates the benefits of this fuzzy approach.

Rotation-Invariant Template Matching

The rotation-invariant phase correlation allows for registering translated and rotated template images. In contrary to the basic version, two phase correlations are applied



(a) Search image: truck model rotated by 45 °.

(b) Search image: truck model rotated by 70 °.

(c) Search image: truck model rotated by 90 °.

Figure 6.8: Rotation-invariant template matching: rotated Aspects.

resulting in two correlation maps: the first one for the rotational component in the polar domain and another one for the final translational offset. Figure 6.8 shows a scene containing rotated versions of the truck model and some clutter objects. Note that a bilinear interpolation is used in order to map the rotated data to the Aspect cells. Hence, even a perfect match yields a small remaining error except for rotations aligned with the grid structure (90 °, 180 °, 270 °).

Rotated images can be successfully registered for any rotational angle, as long as the Aspect's Cartesian resolution allows an accurate angular resolution. For the Aspects depicted in figure 6.8 with a radius of 14, the magnitude of the largest deviation between any given and detected angle was smaller than 4 °. Again, noisy rotated images have been tested but can only be successfully registered if the values deviate from their reference value within a maximum range of about 15 %, which is due to the additional distortion caused by the interpolation.

6.2 Runtime Tests

Runtime measurements have been done for all (computationally expensive) core concepts of the Aspect Memory. All tests have been executed on an `Intel i7-3930K` 6-core CPU clocked at 4.0 GHz with 32 GB of RAM available and a `NVIDIA 780 Ti` GPU with 2880 `CUDA` cores and 3 GB of dedicated memory. In order to obtain significant results, all runtime measurements have been averaged over at least 10 iterations for expensive test sets and otherwise over 1000 iterations.

6.2.1 Forgetting

Again, temporal and spatial forgetting will be discussed separately due to their substantially different implementations and computational costs.

Temporal Filtering and Subsampling

Temporal forgetting is based on accumulations as proposed in sections 4.3.1 and 4.3.2 which use internally several Aspect Operations (typically weighted additions) in order

to combine all delayed Aspects. The efficiency depends thus on the runtime of a single Aspect Operation which is linear in the number of elements. Since Aspect elements are stored contiguously in memory and can also be accessed sequentially and combined interdependently from each other, this is operation is suitable for both sequential CPU and parallel GPU processing. However, the costs for memory transfers might negate the benefits of the latter method. Three implementations have been tested: the obvious, sequential CPU version, the same version parallelized by means of `OpenMP` and finally the GPU version using a `GPGPU` kernel. Table 6.1 shows some runtime results for an accumulation operations applied on Aspects of different sizes (see appendix A.3.3 for all runtime results).

Size (Radius) $n \times n$ (r)	**Naive ST**	**Naive MT** `OpenMP`	**GPU Kernel** `CUDA`
11×11 (5)	[$\ll 0.01$ ms]	[$\ll 0.01$ ms]	0.26
101×101 (50)	0.01	[$\ll 0.01$ ms]	0.42
201×201 (100)	0.04	0.01	0.56
601×601 (300)	0.37	0.05	3.61
1001×1001 (500)	1.03	0.19	6.61

Table 6.1: Runtime in milliseconds for accumulation operations using different acceleration approaches (ST: single-threaded, MT: multi-threaded).

Apparently, the multi-threaded CPU version which requires less than 0.2 ms for more than 1000×1000 elements is preferable. The GPU implementation, instead, must be discarded due to serious memory transfer overheads, albeit the kernel execution time for computing the actual operation is markedly below 0.01 ms even for more than 1000×1000 elements.

Spatial Filtering and Subsampling

As mentioned in the implementation chapter, spatial forgetting requires computationally expensive convolutions for the purpose of filtering. The additional spatial subsampling is technically only a copy operation and can be neglected from a performance perspective (compared to the convolution operation). Two optimized approaches have been introduced in section 5.4: the Separable Convolution Filter which can be accelerated using `OpenMP` and the Fast Fourier Convolution Filter suited for GPU computation. Both require significantly less operations than the naive implementation as shown in table 6.2. The former approach is expected to be faster for small Aspect and kernel sizes since on the one hand no memory transfers to the GPU are necessary and on the other hand kernels need not to be zero-padded as is the case with the latter method. However, the GPU implementation might scale markedly better on larger problem sizes.

	Naive	**Separable**	**FFT**
Complexity	$O(n^2k^2)$	$O(2n^2k)$	$O(n^2 \log(n))$

Table 6.2: Algorithmic complexity of the naive, separable and Fast Fourier convolution for Aspects of size $n \times n$ and kernels of size $k \times k$.

Both methods have been tested and compared against the naive implementation, while differentiating two scenarios: the typical spatial filtering as part of the forgetting process

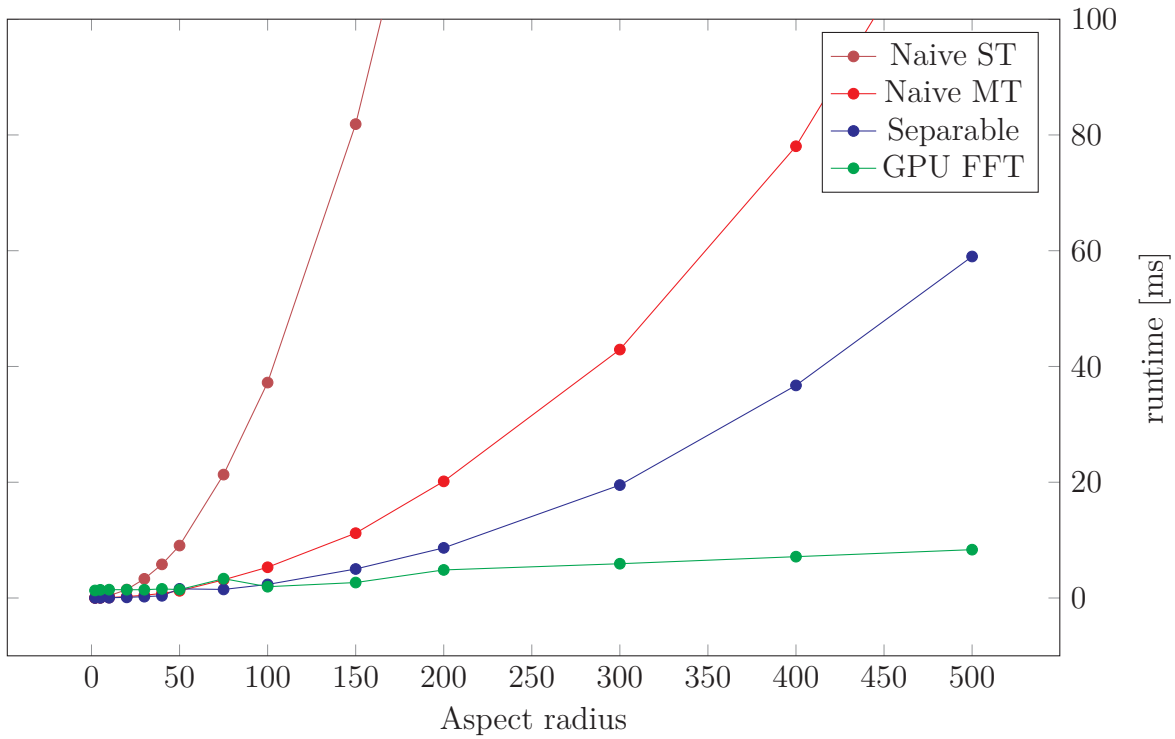


Figure 6.9: Runtime plot of the different convolution approaches for small kernels with $k = 5$ (ST: single-threaded, MT: multi-threaded).

requires rather small kernels such as a Gaussian or Sobel kernel where $k \ll n$. But yet, sometimes larger kernels with $k \approx n$ are useful when considering correlation operations, global minimum / maximum filters, etc. Besides that, the second scenario provides also an estimate of the minimum runtime required for the remembering process since the phase correlation as a basic component is technically similar to a large convolution where $k = n$. Hence, the runtime test can give a hint whether a CPU implementation for remembering comes into question in general.

Size (Radius) $n \times n$ (r)	**Naive ST**	**Naive MT** `OpenMP`	**Separable** `OpenMP`	**GPU FFT** `CUDA, cuFFT`
11×11 (5)	0.12	0.03	0.02	1.41
101×101 (50)	9.06	1.24	0.58	1.45
201×201 (100)	37.22	5.31	2.34	1.95
601×601 (300)	322.46	42.92	19.51	5.92
1001×1001 (500)	893.71	128.02	59.00	8.35

Table 6.3: Runtime in milliseconds for small kernels: different convolution approaches for a typical small kernel size $k = 5$ (ST: single-threaded, MT: multi-threaded).

Table 6.3 shows some measurement results (see appendix A.3.3 for all results) for the first scenario where a mean smoothing filter with a fixed kernel size $k = 5$ has been applied to Aspects of different sizes using different convolution approaches. The first column refers to results of the naive implementation executed on a single thread; the second column

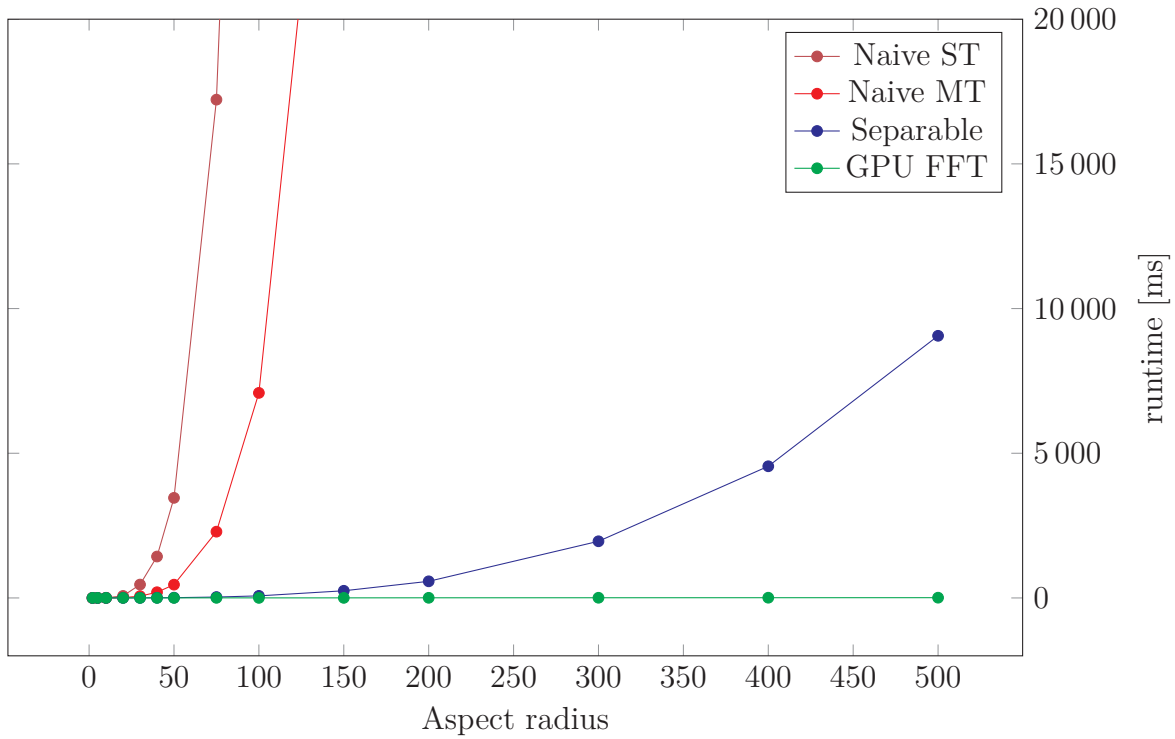


Figure 6.10: Runtime plot of different convolution approaches for large kernels with kernel sizes $k = n$ identical to Aspect sizes (ST: single-threaded, MT: multi-threaded).

contains results for the same algorithm but parallelized with `OpenMP`. The next columns appertain to the optimized approaches where the Separable Convolution Filter has also been accelerated in terms of multi-threading and the Fast Fourier Convolution Filter utilizes the `cuFFT` library. Figure 6.9 shows the runtime results plotted against the Aspect radius.

Size (Radius) $n \times n$ (r)	**Naive ST**	**Naive MT** `OpenMP`	**Separable** `OpenMP`	**GPU FFT** `CUDA, cuFFT`
5×5 (2)	0.03	0.02	0.01	1.28
101×101 (50)	3458.96	459.06	9.71	1.46
201×201 (100)	54164.15	7085.87	71.66	2.13
601×601 (300)	[$\gg$ 1 min]	[$\gg$ 1 min]	1958.23	6.60
1001×1001 (500)	[$\gg$ 1 min]	[$\gg$ 1 min]	9058.90	8.93

Table 6.4: Runtime in milliseconds for large kernels: different convolution approaches for identical Aspect and kernel sizes $k = n$ (ST: single-threaded, MT: multi-threaded).

Table 6.4 shows some measurement results (see appendix A.3.3 for all results) for the second scenario where Aspects of different sizes are convolved with a mean filter of the same size $k = n$ using again the same four approaches mentioned above. Figure 6.10 shows the runtime results plotted against the Aspect and kernel radius.

In both scenarios the optimized implementations are substantially faster whereas the separable convolution is superior for smaller problem sizes as expected. In case of a

small, constant kernel size the separable convolution's speedup compared to the naive version is also constant. Instead, for kernel sizes similar to the Aspect sizes the speedup is proportional to the size and confirms the theoretical speedup factor of $\frac{k}{2}$. However, while acceptable for single filtering operations, the runtime of the separable convolution for large Aspect sizes might be too high in case of several Aspect Memories processed simultaneously. For this reason, the hybrid approach envisaged in section 5.4.3 proves advantageous in practice. The runtime of the second scenario executed on the CPU is not acceptable in any case for (very) large Aspect and kernel sizes. In contrary, the runtime of the Fast Fourier approach computed on the GPU grows only linearly[1] at least up to a radius of 500 which means that the GPU is not yet utilized to capacity. The decision to compute similar remembering operations completely on the GPU is hence justified.

6.2.2 Remembering

Remembering is implemented as a set of template matching procedures applied on each memory unit. The efficiency of the remembering process depends thus on the runtime of the basic template matching. A rotation-invariant phase correlation algorithm as introduced in section 4.4.4 and a basic phase correlation neglecting rotational components—both implemented as `GPGPU` versions using `cuFFT`—have been tested. Table 6.5 shows some runtime measurements (see appendix A.3.3 for all results) for both versions applied on search Aspects of different sizes including the times for all memory transfers. Note that template Aspects which are smaller than search Aspects need to be zero-padded, which is the reason why the runtime is solely determined by the size of the search Aspect.

Size (Radius) $n \times n$ (r)	**GPU PC** `CUDA, cuFFT`	**GPU RIPC** `CUDA, cuFFT`
11×11 (5)	1.41	3.98
101×101 (50)	1.96	6.63
201×201 (100)	3.75	7.30
601×601 (300)	6.32	17.36
1001×1001 (500)	10.96	21.57

Table 6.5: Runtime in milliseconds for the basic phase correlation (PC) and the rotation-invariant phase correlation (RIPC) for different Aspect sizes.

The runtime results of the first version are only marginally greater than those of the GPU convolution filter for large kernels (see table 6.4) since a basic phase correlation method is composed of a similar correlation operation followed by a maximum search. The rotation-invariant approach requires two phase correlations, a polar transform and some matrix rotations and translations and is thus approximately twice as expensive as the former one. It can be assumed that a substantial amount of time is consumed for memory transfers. However, when applying multiple template matchings for an entire Aspect Memory as is the case with the remembering process, memory transfers can be combined and the actual matching procedures can be executed in parallel.

[1] Due to the increasing effort for memory transactions.

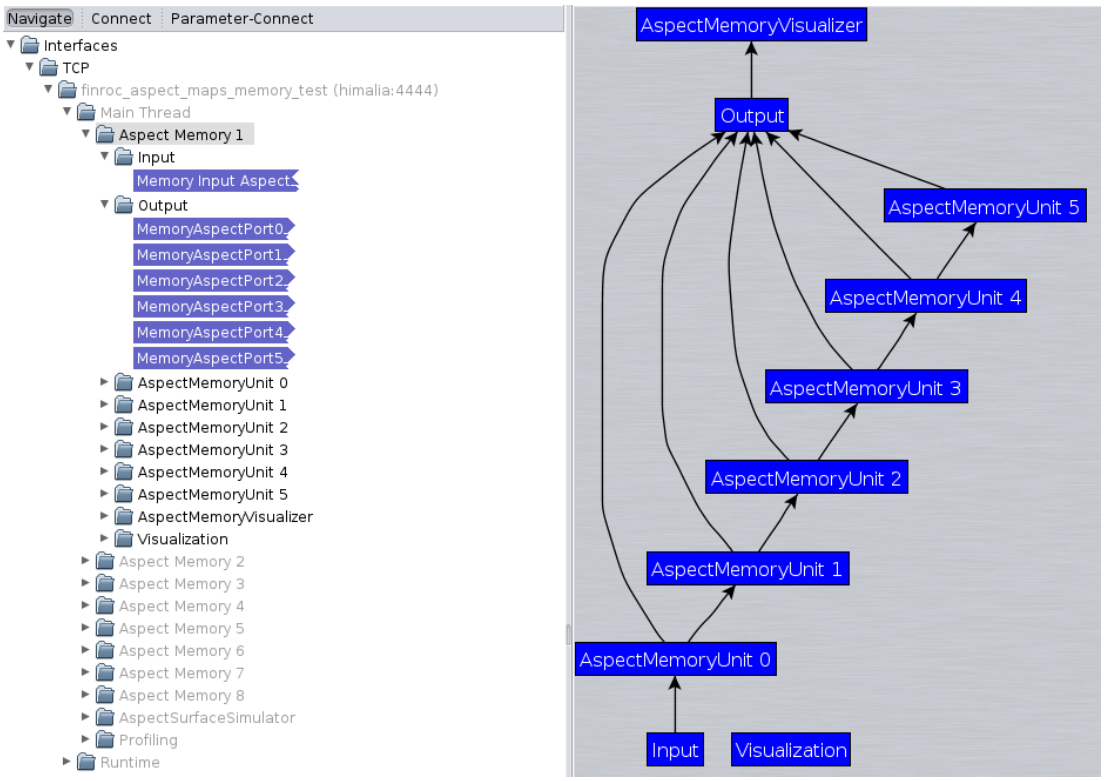
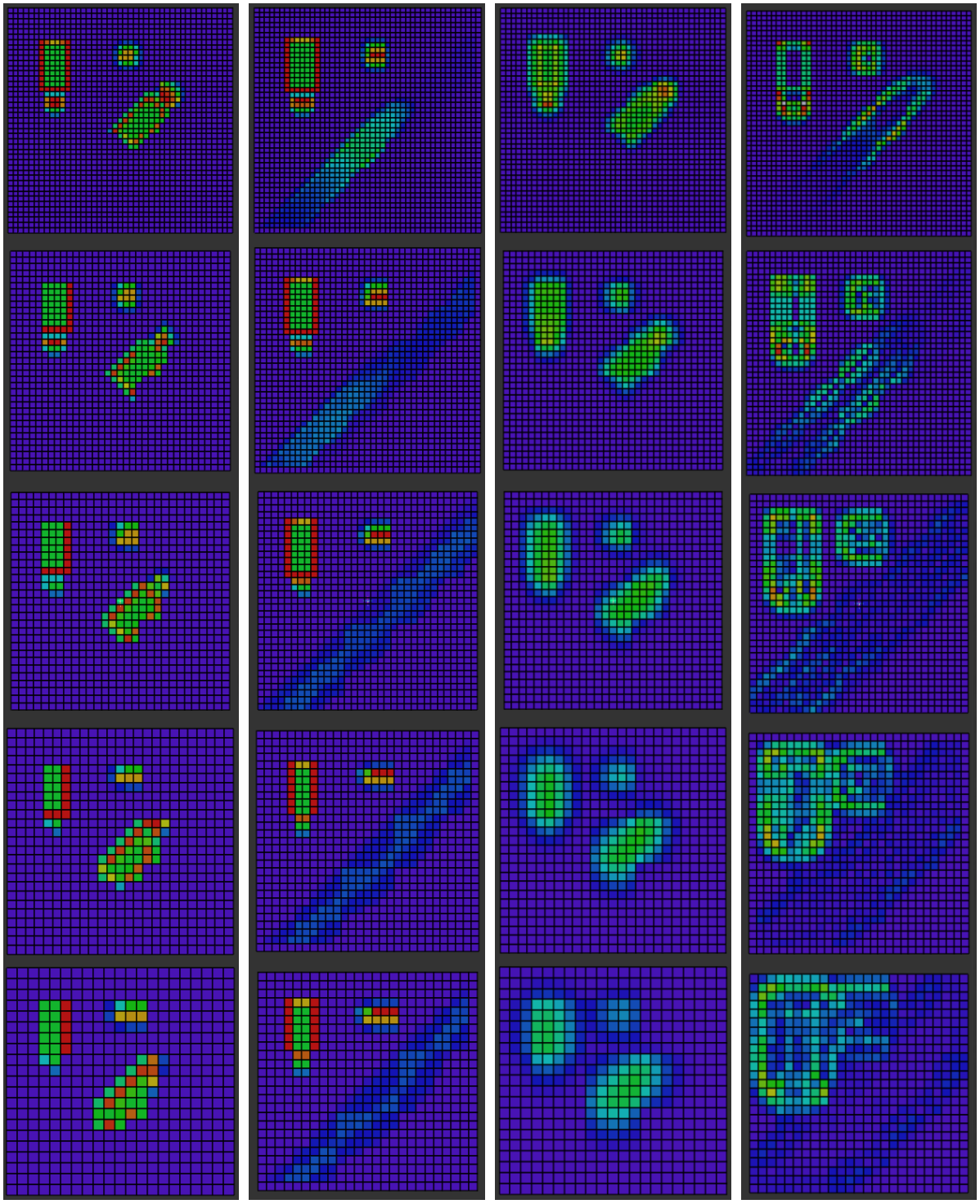


Figure 6.11: An Aspect Memory of depth 6 and its interconnected memory units shown in `Finstruct`.

6.3 Aspect Memory Integration Test

An Aspect Memory is composed of several concatenated memory units as described in section 4.2 where each unit implements the temporal and spatial filtering and subsampling operations. The particular functional components involved in the forgetting process must be integrated within the memory unit and the Aspect Memory group must finally be able to initialize and interconnect all its units properly. Figure 6.11 depicts the structure of an Aspect Memory group visualized with the component visualization tool `Finstruct`.

Several settings have been tested in dynamic scenarios with static and moving objects. Figure 6.12 shows a selection of Aspect Memory instances, each of depth 5 but with different filter kernels, accumulation weights and coarsement factors, where the temporally recent Aspect is depicted at the top. The same scenario with a dynamic object moving from the lower left to the upper right has been presented to all memory instances. Figure 6.12a illustrates a setting without any filtering but with spatial subsampling controlled by a coarsement factor of 1.15 such that the presented scene is only delayed and coarsened. Temporal filtering with a weight factor of 0.70 and spatial subsampling with a coarsement factor of 1.20 is shown in figure 6.12a. A blurred trace indicates the path of the moving object at older memory stages. Figure 6.12c, instead, depicts only spatial filtering and subsampling using a Gaussian smoothing kernel of size 3×3 and again a coarsement factor of 1.20. Finally, figure 6.12d illustrates the case of both temporal and spatial filtering using a weight factor of 0.70, a Sobel filter and a coarsement factor of 1.15, which yields a contoured trail and sharpened edges of the static objects.



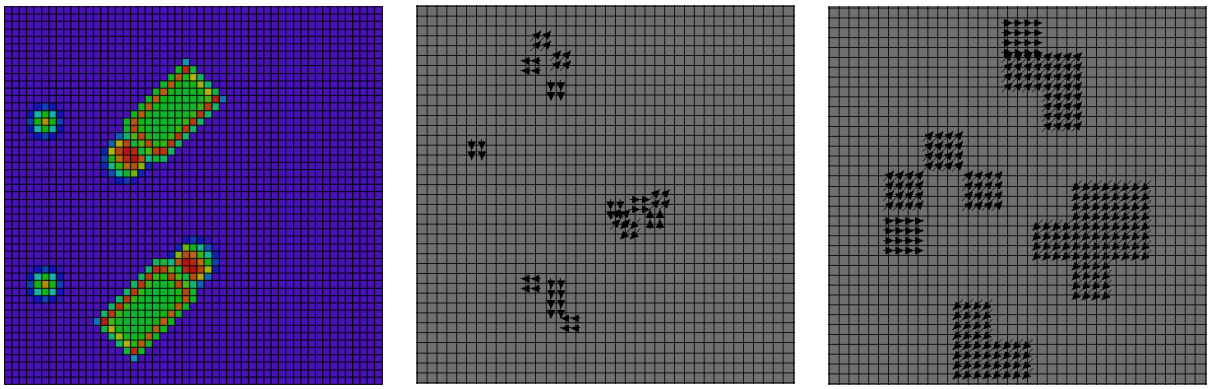
(a) Spatial subsampling only

(b) Temporal filtering, spatial subsampling

(c) Spatial filtering, spatial subsampling

(d) Complete filtering and subsampling

Figure 6.12: Aspect Memories of depth 5 with different configurations visualized using the `mAspectMemoryVisualizer` module and the `FinGUI` tool.



(a) Scene with two moving trucks and static objects.

(b) Motion vector field with a block size of 2.

(c) Motion vector field with a block size of 4.

Figure 6.13: Motion detection of two moving trucks using a slightly modified remembering algorithm.

6.4 Advanced Application: Motion Detection and Optical Flow

The entire remembering process is implemented by means of several template matching procedures for each memory unit or each temporal Aspect instance respectively. In order to exploit the GPU's memory bandwidth, all Aspects are contiguously concatenated in memory as discussed in section 5.5.1 before applying multiple phase correlations in parallel for each matrix. Interestingly, this procedure (with some technical adaptations) can also be used to compute another problem known as *optical flow* where relative motions between visual scenes projected onto an image plane are to be detected. The optical flow of two image frames is a vector field of the relative velocity vectors between all pairs of points which can be identified in both images and associated with each other. A more formal definition and common algorithms can be found in [Beauchemin 95].

Wide fields of image processing and robotics make use of a variant called *block motion estimation* where corresponding blocks of pixels (or sub-images) are searched for and provided with an appropriate motion vector[2]. In order to find those vectors—or translational displacements—between two blocks, a phase correlation can be applied. Since an entire image is divided into several blocks which can be independently processed, the remembering algorithm is perfectly suited: two initial Aspects from different points in time are decomposed into several partial Aspects and each of those from the first temporal instance is registered within its (expanded) matching partial Aspect from the second temporal instance. This corresponds exactly to the remembering algorithm with the difference that template and search images are associated with pairwise related, partial image blocks. The result of this procedure is an Aspect composed of 2-dimensional vectors.

However, some technical details must be considered: assuming a template block width of n, search blocks must be obviously expanded to the size $m > n$ such that potential movements are covered. Furthermore, edge effects are of importance, especially for small

[2]This technique is also used for motion compensation or video compression applications.

block sizes, and must be compensated using zero-padding and / or window functions. An appropriate algorithm motivated by the work of [Liang 00] has been implemented an successfully tested. Figure 6.13 shows a dynamic scene of two truck models, one moving from the lower left to the upper right and the second one in the opposite direction, and two motion vector fields with different template block sizes. For instance, an Aspect containing a motion vector field might again be stored in an Aspect Memory such that the optical flow could be observed over time, objects tracked or future movements predicted from recent velocity vectors.

7. Conclusion

Within the scope of this work, a robot working memory motivated by a cognitive network approach, utilizing practices of both connectionist and symbolic computing and thus bottom-up and top-down processing, has been designed and implemented in an efficient manner. Characteristics and approaches of both worlds have been discussed at first and then, based on an evaluation, the Cognitive Maps Architecture has been chosen as a promising foundation of a memory conception. As part of the question of how to support "intelligent" processing features and how a working memory might distinguish itself from basic temporal networks, the concepts of forgetting and remembering have been identified. A specific, technical interpretation of those terms has been chosen and implemented using a generic combination of temporal and spatial filtering as for the first case and a template matching approach as for the second case. Due to the computational complexity of the respective algorithms and the expected data volume of an entire system composed of multiple memories, algorithmic and technical optimizations have been aimed. Finally, the functionality of the forgetting and remembering features as well as the working memory's basic applicability as a spatiotemporal data structure have been tested.

Certainly, similar to the underlying cognitive approach, the Aspect Memory is a versatile concept or a generic tool which can and must be deliberately configured and applied for the data to be stored and processed and the respective fields of application. A more detailed discussion in the context of the current status of the working memory's conception and implementation is presented in the subsequent section whereas technical and conceptual improvements and further ideas beyond that are addressed in the last section.

7.1 Evaluation

The implementation of the forgetting process as a combined filtering along each dimension of the Memory might be seen as a kind of video filtering where both spatial and temporal filter components are optional and can be customized. Additionally, a frequency reduction and a spatial reduction—temporal and spatial subsampling—can be defined. Initially, the forgetting process was mostly motivated by the decay theory claiming that a memory's contents (or its information details) fade over time which is why a combination of spatial

Gaussian smoothing, temporal accumulation and spatiotemporal subsampling seems natural. However, there are cases where smoothing filters and / or strong subsampling are unfavorable: data which encodes objects per point or pixel, as for instance height data, may forfeit important features such as edges. In this case, high-pass filters such as the Sobel operator can be employed, implying a spatial forgetting where features are emphasized and steady information is lost over time. An accurate motion detection and the computation of optical flows are further examples where a high degree of any filtering or subsampling is not advisable. As opposed to this, especially a strong temporal filtering excels at the segmentation of dynamic and static objects: moving objects leave a blurry trail, vaguely indicating both the path and velocity of movement, or even fade out depending on the accumulation settings whereas static objects remain over time.

Remembering, on the other hand, as a process which is to some extent opposed, relies on rotation-invariant template matchings or, more precisely, on phase correlations. It is hence a kind of spatial pattern matching assessing the similarity of two (partial) Aspects by means of a cross correlation and allows to register one Aspect within the other. In this way, known spatial patterns can be recognized and located within the memory. Interestingly, the remembering procedure might also be seen as a potential technique bridging the gap between subsymbolic data (Simple Aspects) to symbolic data (Cognitive Aspects): detecting patterns within information distributed along the spatial dimensions corresponds to an object recognition and maps rather detailed and low-level data to a single abstract symbol or a scalar probability of existence. Furthermore, the remembering implementation could be utilized for computing a problem that seemed rather different at a first glance: block motion estimation and optical flows within temporal Aspect instances.

The algorithms of both processes have been successfully optimized in respect to their efficiency: in terms of spatial filtering, a hybrid approach, applying either a CPU version, which exploits the separability of common filter kernels, or a GPU processing depending on the problem size, has been chosen. Due to the higher effort of (rotation-invariant) template matchings, remembering procedures are entirely computed on the GPU.

7.2 Outlook

Based on the current status, certainly, various further improvements are conceivable. Firstly, as for forgetting, a fully unrestricted definition of filtering operations and kernels within the same Aspect Memory may be desirable—currently, the same temporal and spatial filter types are applied on each memory stage—allowing an even more generic spatiotemporal filtering referring to the Convolution Neural Networks. This depends, however, highly on the respective case of application.

Regarding the remembering process, an easy but effective improvement is suggested: changing the polar transform used for rotation invariance into a *log polar transform*, where both axis of the target polar image plane are also logarithmized, additional scale invariance can be achieved. This is useful when registering template patterns within scaled or subsampled versions of search Aspects. An appropriate implementation is proposed in [Sarvaiya 09, Sarvaiya 12].

Additionally, the bilinear interpolation method used for rotation correction within the rotation-invariant template matching can be replaced with a bicubic version which probably

adds less distortion to rotated Aspect and thus to the remaining error. Alternatively, rotations could be implemented in the Fourier domain by means of three shears, which corresponds technically to three multiplications of twiddle factors and is hence both quite efficient and distorts data only marginally according to [Larkin 97].

As indicated in the implementation chapter, a substantial part of the runtime for GPU computations originates from the memory transfers between CPU host and GPU device. Therefore, it would be advisable to keep Aspect data permanently on the GPU memory as long as possible; for example, for the time of multiple forgetting computations at several memory units. In this case, also a GPU computation of Aspect Operations would be favorable. However, the Aspect Memory's state cannot be inspected and visualized for this amount of time and, anyway, computation results are of course required on the host side sooner or later. Thus, a caching method can provide a remedy: each Aspect Map might be equipped with a flag that indicates if the Aspect's storage content has changed on the host memory. If not, computations can be done efficiently on the GPU using a cached version; otherwise, the CPU processes the operations and transfers the changed data to the device cache. Whenever Aspects are actually needed on the host side, they are requested from the GPU cache. `CUDA` provides extensive features for such an asynchronous hybrid processing[1].

Currently, the remembering process is implemented as a 2-dimensional pattern matching, capable of registering blocks of the spatial plane within other spatial search blocks. The fundamental concept is, however, not limited to the 2-dimensional space. Interpreting a block of multiple (successive) temporal Aspect instances as a *dynamic situation* within the robot's environment, arises the question of whether an entire spatiotemporal pattern can be detected within an Aspect Memory. Conceptually, only an extension of the proposed procedures to an additional dimension is required to this end. Since such an approach would allow for a direct processing of temporal sequences of spatial patterns or, in other words, registering partial Aspect Memories within larger Aspect Memories, spatiotemporal situations could be recognized. This might be advantageous whenever characteristic dynamic processes or events are of special interest for the robot's behavior. Technically, this requires, in addition to a certainly sophisticated memory layout, an efficient 3-dimensional Fast Fourier Transform as discussed in [Nvidia 12].

Finally, the coincidence of several patterns detected at the same (period of) time within different Aspect Memories could be again encoded within a Cognitive Aspect pattern and recognized as a scattered, complex pattern in this way. An example is a scenario where local patterns within different, independent data sources indicate in their entirety a security-critical situation as is the case with, for instance, an ongoing excavation task while a construction operative runs up to the construction site and an alarm signal sounds.

[1] See `http://devblogs.nvidia.com/parallelforall/how-overlap-data-transfers-cuda-cc/`.

Bibliography

[Al Umairy 12] S. A. Al Umairy, A. S. Van Amesfoort, I. D. Setija, M. C. Van Beurden, H. J. Sips, "On the Use of Small 2d Convolutions on GPUs", in *Computer Architecture*, Springer. 2012, pp. 52–64.

[Atkinson 68] R. C. Atkinson, R. M. Shiffrin, "Human memory: A proposed system and its control processes.", *In K. W. Spence and J. T. Spence (Eds.), The Psychology of learning and motivation: Advances in research and theory (vol. 2).*, pp. 89 – 105, 1968.

[Baddeley 74] A. D. Baddeley, G. Hitch, "Working memory", *Psychology of learning and motivation*, vol. 8, pp. 47–89, 1974.

[Barkowsky 97] T. Barkowsky, C. Freksa, B. Berendt, S. Kelter, "Aspektkarten—Integriert räumlich-symbolische Repräsentationsstrukturen", in *Perspektive in Sprache und Raum*, Springer, 1997, pp. 147–168.

[Beauchemin 95] S. S. Beauchemin, J. L. Barron, "The Computation of Optical Flow", *ACM Comput. Surv.*, vol. 27, no. 3, pp. 433–466, sept. 1995.

[Bengio 09] Y. Bengio, "Learning Deep Architectures for AI", *Found. Trends Mach. Learn.*, vol. 2, no. 1, pp. 1–127, jan. 2009.

[Berendt 98] B. Berendt, T. Barkowsky, C. Freksa, S. Kelter, "Spatial representation with aspect maps", in *Spatial Cognition*, Springer. 1998, pp. 313–336.

[Borges 12] R. Borges, "A neural-symbolic system for temporal reasoning with application to model verification and learning.", Dissertation, City University London, January 2012.

[Brunelli 09] R. Brunelli, *Template Matching Techniques in Computer Vision: Theory and Practice*, Wiley Publishing, 2009.

[Colom 08] R. Colom, F. J. Abad, M. Quiroga, P. C. Shih, C. Flores-Mendoza, "Working memory and intelligence are highly related constructs, but why?", *Intelligence*, vol. 36, no. 6, pp. 584–606, 2008.

[Cormen 01] T. H. Cormen, C. Stein, R. L. Rivest, C. E. Leiserson, *Introduction to Algorithms*, 2nd ed., McGraw-Hill Higher Education, 2001.

[De Castro 87] E. De Castro, C. Morandi, "Registration of translated and rotated images using finite fourier transforms", *IEEE transactions on pattern analysis and machine intelligence*, vol. 9, no. 5, p. 700—703, May 1987.

[Fukushima 80] K. Fukushima, "Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position", *Biological cybernetics*, vol. 36, no. 4, pp. 193–202, 1980.

[Fukushima 88] K. Fukushima, "Neocognitron: A hierarchical neural network capable of visual pattern recognition", *Neural networks*, vol. 1, no. 2, pp. 119–130, 1988.

[Harris 07] M. Harris, "Optimizing Parallel Reduction in CUDA", `http://developer.download.nvidia.com/compute/cuda/1.1-Beta/x86_website/projects/reduction/doc/reduction.pdf`, 2007. [Online; accessed 25-April-2015].

[Haykin 98] S. Haykin, *Neural Networks: A Comprehensive Foundation*, 2nd ed., Upper Saddle River, NJ, USA: Prentice Hall PTR, 1998.

[Hayman 11] R. Hayman, M. A. Verriotis, A. Jovalekic, A. A. Fenton, K. J. Jeffery, "Anisotropic encoding of three-dimensional space by place cells and grid cells", *Nature neuroscience*, vol. 14, no. 9, pp. 1182–1188, 2011.

[Hebb 02] D. O. Hebb, *The organization of behavior: A neuropsychological theory*, Psychology Press, 2002.

[Henderson 99] J. Henderson, *Memory and Forgetting*, ser. Cognitive psychology, Routledge, 1999.

[Hubel 68] D. H. Hubel, T. N. Wiesel, "Receptive fields and functional architecture of monkey striate cortex", *The Journal of physiology*, vol. 195, no. 1, pp. 215–243, 1968.

[Jain 09] D. Jain, S. Waldherr, M. Beetz, "Bayesian Logic Networks", IAS Group, Fakultät für Informatik, Technische Universität München, Tech. Rep., 2009.

[Ji 13] S. Ji, W. Xu, M. Yang, K. Yu, "3D convolutional neural networks for human action recognition", *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 35, no. 1, pp. 221–231, 2013.

[Jia 14] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, T. Darrell, "Caffe: Convolutional architecture for fast feature embedding", in *Proceedings of the ACM International Conference on Multimedia*, ACM. 2014, pp. 675–678.

[Kitchin 94] R. M. Kitchin, "Cognitive maps: What are they and why study them?", *Journal of environmental psychology*, vol. 14, no. 1, pp. 1–19, 1994.

[Lange 89] T. E. Lange, M. G. Dyer, "Dynamic, Non-Local Role Bindings and Inferencing in a Localist Network for Natural Language Understanding", in *Advances in Neural Information Processing Systems 1*, D. S. Touretzky, Ed., Morgan-Kaufmann, 1989, pp. 545–552.

[Larkin 97] K. G. Larkin, M. A. Oldfield, H. Klemm, "Fast Fourier method for the accurate rotation of sampled images", *Optics Communications*, vol. 139, no. 1, pp. 99–106, 1997.

[LeCun 98] Y. LeCun, L. Bottou, Y. Bengio, P. Haffner, "Gradient-based learning applied to document recognition", *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[Liang 00] Y. Liang, "Phase-correlation motion estimation", *EE392J Project report*, 2000.

[McClelland 87] J. L. McClelland, D. E. Rumelhart, G. E. Hinton, "The Appeal of Parallel Distributed Processing", in *Parallel Distributed Processing: Volume 1: Foundations*, D. E. Rumelhart, J. L. McClelland et al, Eds., Cambridge: MIT Press, 1987, pp. 3–44.

[McCulloch 88] W. S. McCulloch, W. Pitts, "Neurocomputing: Foundations of Research", J. A. Anderson, E. Rosenfeld, Eds., Cambridge, MA, USA: MIT Press, 1988, ch. A Logical Calculus of the Ideas Immanent in Nervous Activity, pp. 15–27.

[McDermott 76] D. McDermott, "Artificial Intelligence Meets Natural Stupidity", *SIGART Bull.*, no. 57, pp. 4–9, april 1976.

[Minsky 87] M. L. Minsky, S. A. Papert, *Perceptrons - Expanded Edition: An Introduction to Computational Geometry*, MIT press Boston, MA, 1987.

[Newell 59] A. Newell, H. A. Simon, *The simulation of human thought*, Rand Corporation, 1959.

[Newell 76] A. Newell, H. A. Simon, "Computer Science As Empirical Inquiry: Symbols and Search", *Commun. ACM*, vol. 19, no. 3, pp. 113–126, march 1976.

[Nvidia 12] Nvidia, "Aatish, Kumar and Zhang, Boyan", `http://cseweb.ucsd.edu/~baden/classes/Exemplars/cse260_fa12/3DFFT.pdf`, 2012. [Online; accessed 20-May-2015].

[Nvidia 13] Nvidia, "Standard Introduction to CUDA C Programming", `http://rt.uits.iu.edu/ci/training/files/cuda-talk.pdf`, 2013. [Online; accessed 25-April-2015].

[Pluzhnikov 12] S. Pluzhnikov, D. Schmidt, J. Hirth, K. Berns, "Behavior-based Arm Control for an Autonomous Bucket Excavator", K. Berns, C. Schindler, K. Dreßler, B. Jörg, R. Kalmar, G. Zolynski, Eds. Kaiserslautern, Germany: Shaker Verlag, March 13–15 2012, pp. 251–261.

[Podlozhnyuk 07a] V. Podlozhnyuk, "FFT Based 2D Convolution", `http://developer.download.nvidia.com/compute/cuda/2_2/sdk/website/projects/convolutionFFT2D/doc/convolutionFFT2D.pdf`, 2007. [Online; accessed 25-April-2015].

[Podlozhnyuk 07b] V. Podlozhnyuk, "Image Convolution with CUDA", `http://developer.download.nvidia.com/assets/cuda/files/convolutionSeparable.pdf`, 2007. [Online; accessed 25-April-2015].

[Prior 03] A. N. Prior, "Papers on time and tense", 2003.

[Proetzsch 10] M. Proetzsch, T. Luksch, K. Berns, "Development of Complex Robotic Systems Using the Behavior-Based Control Architecture iB2C", *Robotics and Autonomous Systems*, vol. 58, no. 1, pp. 46–67, January 2010. doi:10.1016/j.robot.2009.07.027.

[Reddy 96] B. S. Reddy, B. N. Chatterji, "An FFT-based technique for translation, rotation, and scale-invariant image registration", *IEEE transactions on image processing*, vol. 5, no. 8, pp. 1266–1271, 1996.

[Reichardt 12] M. Reichardt, T. Föhst, K. Berns, "Introducing FINROC: A Convenient Real-time Framework for Robotics based on a Systematic Design Approach", Robotics Research Lab, Department of Computer Science, University of Kaiserslautern, Kaiserslautern, Germany, Technical report, July 2012.

[Reichardt 14] M. Reichardt, G. Zolynski, M. Arndt, K. Berns, "On the Benefits of Component-Defined Real-Time Visualization of Robotics Software", ser. Lecture Notes in Artificial Intelligence (LNAI), D. Brugali, J. F. Broenink, T. Kroeger, B. A. MacDonald, Eds., vol. 8810. Bergamo, Italy: Springer, October 20–23 2014, pp. 376–387.

[Rosenblatt 58] F. Rosenblatt, "The perceptron: a probabilistic model for information storage and organization in the brain.", *Psychological review*, vol. 65, no. 6, p. 386, 1958.

[Rumelhart 86] D. E. Rumelhart, G. E. Hinton, R. J. Williams, "Learning Internal Representations by Error Propagation, Parallel Distributed Processing, Explorations in the Microstructure of Cognition, ed. DE Rumelhart and J. McClelland. Vol. 1. 1986", 1986.

[Russell 03] S. J. Russell, P. Norvig, *Artificial Intelligence: A Modern Approach*, 2 ed., Pearson Education, 2003.

[Samadi 14] M. Samadi, D. A. Jamshidi, J. Lee, S. Mahlke, "Paraprox: Pattern-based Approximation for Data Parallel Applications", *SIGPLAN Not.*, vol. 49, no. 4, pp. 35–50, feb. 2014.

[Sanders 10] J. Sanders, E. Kandrot, *CUDA by Example: An Introduction to General-Purpose GPU Programming*, 1st ed., Addison-Wesley Professional, 2010.

[Sarvaiya 09] J. N. Sarvaiya, S. Patnaik, S. Bombaywala, "Image registration using Log polar transform and phase correlation", in *TENCON 2009 - 2009 IEEE Region 10 Conference*, IEEE. 2009, pp. 1–5.

[Sarvaiya 12] J. N. Sarvaiya, S. Patnaik, K. Kothari, "Image Registration Using Log Polar Transform and Phase Correlation to Recover Higher Scale", *Journal of Pattern Recognition Research*, vol. 7, no. 1, pp. 90–105, 2012.

[Schmidt 09] D. Schmidt, "Soil Simulation and Behaviour-Based Control for Landscaping Tasks of an Autonomous Bucket Excavator", Diploma thesis, Robotics Research Lab, Department of Computer Sciences, University of Kaiserslautern, April 24 2009. [Unpublished].

[Schmidt 10] D. Schmidt, M. Proetzsch, K. Berns, "Simulation and Control of an Autonomous Bucket Excavator for Landscaping Tasks". Anchorage, Alaska, USA, May 3–8 2010, pp. 5108–5113.

[Schmidt 13] D. Schmidt, T. Juhasz, K. Berns, U. Schmucker, "Interconnection of the behavior-based control architecture and a detailed mechatronic machine model for realistic behavior verification of the Thor project", ser. Lecture Notes in Informatics (LNI). Koblenz, Germany: Springer, September 16–20 2013.

[Schüle 11] J. Schüle, *Paralleles Rechnen: Performancebetrachtungen zu Gleichungslösern*, Oldenbourg Verlag, 2011.

[Siegelmann 91] H. T. Siegelmann, E. D. Sontag, "Turing Computability With Neural Nets", *Applied Mathematics Letters*, vol. 4, pp. 77–80, 1991.

[Simard 03] P. Y. Simard, D. Steinkraus, J. C. Platt, "Best practices for convolutional neural networks applied to visual document analysis", in *2013 12th International Conference on Document Analysis and Recognition*, vol. 2, IEEE Computer Society. 2003, pp. 958–958.

[Sun 99] R. Sun, "Artificial intelligence: Connectionist and symbolic approaches", 1999.

[Tetko 95] I. V. Tetko, D. J. Livingstone, A. I. Luik, "Neural network studies, 1. Comparison of overfitting and overtraining", *Journal of Chemical Information and Computer Sciences*, vol. 35, no. 5, pp. 826–833, 1995.

[Trier 96] Ø. D. Trier, A. K. Jain, T. Taxt, "Feature extraction methods for character recognition-a survey", *Pattern recognition*, vol. 29, no. 4, pp. 641–662, 1996.

[Ulanovsky 11] N. Ulanovsky, "Neuroscience: How Is Three-Dimensional Space Encoded in the Brain?", *Current Biology*, vol. 21, no. 21, pp. R886 – R888, 2011.

[Wienke 10] J. Wienke, "A Spatiotemporal Working Memory for Humanoid Robots", Master's thesis, Bielefeld University, 2010.

[Wilson 01] R. A. Wilson, F. C. Keil, *The MIT Encyclopedia of the Cognitive Sciences*, ser. A Bradford book, MIT Press, 2001.

[Wolberg 00] G. Wolberg, S. Zokai, "Robust image registration using log-polar transform", in *Image Processing, 2000. Proceedings. 2000 International Conference on*, vol. 1, IEEE. 2000, pp. 493–496.

[Zadeh 94] L. A. Zadeh, "Soft Computing and Fuzzy Logic", *IEEE Softw.*, vol. 11, no. 6, pp. 48–56, nov. 1994.

[Zitova 03] B. Zitova, J. Flusser, "Image registration methods: a survey", *Image and vision computing*, vol. 21, no. 11, pp. 977–1000, 2003.

[Zolynski 12] G. Zolynski, C. Schank, K. Berns, "Point Cloud Gathering for an Autonomous Bucket Excavator in Dynamic Surroundings", K. Berns, C. Schindler, K. Dreßler, B. Jörg, R. Kalmar, G. Zolynski, Eds. Kaiserslautern, Germany: Shaker Verlag, March 13–15 2012, pp. 262–271.

[Zolynski 15] G. Zolynski. "Cognitive Maps For Autonomous Robots in Weakly-structured Dynamic Environments". [Unpublished, Effective May 2015], 2015.

A. Appendix

A.1 Related Work

A.1.1 Relations between Cognitive Maps Architecture, ANNs and CNNs

Ensuing from the presented generic artificial neural network model $o_j = \varphi(\sum_{i=1}^{n} x_i w_{ij} - \theta_j)$, the Cognitive Maps Architecture can emulate a neural unit in the following way:

1. Define a Spatial Aspect where each element $a_{0,i} := (Ar_{0,i}, v_{0,i})$ is associated with the inputs of a neuron such that $v_{0,i} = x_i$. A second Spatial Aspect of the same size with elements $a_{1,i}$ contains the respective weight factors w_{ij} and is aligned with the input Aspect such that $\forall\ i:\ Ar_{0,i} = Ar_{1,i}$.

2. Combine input and weight Aspects:

   (a) Combine both Aspects using a multiplication operation $op_{\text{mult}}:\ v_{2,i} = v_{0,i} v_{1,i}$ and process the resulting Aspect using a global summation filter $filt_{K(r),\ \text{sum}}:\ v_{3,i} = \sum_{k_i \in K(r)} v_{2,i}$ where r is identical to the Aspects' radii.

   (b) **Or**, equivalently, convolve both input and weight Aspects using a convolution filter (see paragraph 4.3.3).

3. The central element $v_{3,(0,0)}$ of the resulting Spatial Aspect can now be combined with a Uniform Aspect containing the threshold θ_j using a subtraction operation.

4. Finally, the activation function φ can be mapped to an appropriate filter kernel definition which is applied to the foregoing Uniform Aspect yielding another Uniform Aspect whose value is identical to the neuron's output o_j.

An entire network of several neurons is constructed by simply interconnecting multiple processing paths.

Since Convolutional Neural Networks are just a special case of neural networks, where inter-neural connections are kept locally, the Cognitive Maps Architecture can also emulate them similarly. The essential convolution operation can either be implemented by replicating emulated feature maps and combining their outputs properly (similar to step 2a above) or using a convolution filter mentioned above.

A.1.2 Hybrid and Unaligned Aspect Map Combinations

Generally speaking, the combinations of apriory incompatible Aspects relies on matching related elements with the aid of their world coordinates and always requires a proper definition of origin points. In the Cognitive Maps Architecture, the alignment of Aspects is handled by defining a scale and an origin point as the geometric center for Spatial, Directional, and Distance Aspects. For Uniform Aspects there is no need for a scale or an origin point at all. These so called *hybrid combinations* of Aspects with different storage modes are listed below:

Spatial Aspect with any other type: As Spatial Aspects contain the most detailed spatial information, their hybrid combinations should also yield Spatial Aspects. However, this requires the Spatial Aspect to be the first Aspect, otherwise the other type is determining.

Uniform Aspect with any other type: Storing only one information entity, a Uniform Aspect is combined with all elements of the other Aspect if the other one is passed as first argument. Otherwise, the resulting Aspect will also be a Uniform Aspect containing only the combinatorial result of the original piece of information and one other matching element.

Directional Aspect with Distance Aspect: The spatial addressing of Directional and Distance Aspects is to some extent orthogonal: Directional Aspects provide an angular resolution and Distance Aspects contain radial distance information which yields altogether a fully spatial position in the sense of polar coordinates. Therefore, a combination of these two specific types allows for restoring an exact Cartesian coordinate from two partial, complementary types and results in a Spatial Aspect.

Despite the case of *aligned combinations* considered up to this point, it is also possible to combine *unaligned* Aspects; i.e. there is a displacement vector $\vec{d}_{A_0A_1} = \vec{o}_{A1} - \vec{o}_{A0}$ between the two Aspect's origin points such that both are offset against each other subject to the world coordinate system. Apart from that, they are similar to aligned combinations, albeit there are two different special cases:

Unaligned Directional Aspects: Two directional angles of respective Aspect elements and a displacement vector between both Aspect's origin points are sufficient in order to restore a fully spatial location. This represents in fact the principle of a triangulation and as a consequence the resulting Aspect is a Spatial Aspect.

Unaligned Distance Aspects: The combination of two distance Aspects yields a Spatial Aspect but, in contrast to the above case, two radial distances and a displacement distance are not enough to ambiguously determine a spatial position. The condition of two displaced Distance Aspects corresponds to intersecting two circles for each matching element pair where possible outcomes are two intersection points, one boundary point or no overlapping at all. Thus, the resulting Spatial Aspect data contains candidates that are spatially indefinite.

A.2 Implementation

A.2.1 Separable Convolution Filters

The outer product of a column vector $\vec{v}$ and a row vector $\vec{h}$ is equivalent to their convolution:

$$\vec{v} \otimes \vec{h} = \begin{bmatrix} v_{(0,0)} \\ v_{(1,0)} \\ \vdots \end{bmatrix} \otimes \begin{bmatrix} h_{(0,0)} & h_{(0,1)} & \dots \end{bmatrix} = \begin{bmatrix} c_{(0,0)} & c_{(0,1)} & \cdots \\ c_{(1,0)} & c_{(1,1)} & \\ \vdots & & \ddots \end{bmatrix} , \tag{A.1}$$

where

$$c_{(x,y)} = v_{(x,0)} h_{(0,y)} = \sum_{i=0}^{0} v_{(x,i)} h_{(i,y)} = \sum_{u=0}^{0} \sum_{v=y}^{y} v_{(x-u,y-v)} h_{(u,v)} \ . \tag{A.2}$$

A.2.2 Separability of the Fourier Transform

A 1-dimensional Fourier Transform can be computed with $O(n \log(n))$ effort using the Fast Fourier Transform algorithm (see [Cormen 01], chap. 30). The 2-dimensional transformation of an image of size $n \times m$ can be decomposed into two subsequent batches of m 1-dimensional Fourier Transforms of the image's rows followed by n 1-dimensional Fourier Transforms of its columns or vice versa due to the separability of the Fourier Transform:

$$\begin{aligned} \mathcal{F}\{f(x,y)\} &= \frac{1}{nm} \sum_{x=0}^{n-1} \sum_{y=0}^{m-1} f(x,y) e^{-2\pi i (\frac{ux}{n} + \frac{vx}{m})} \\ &= \frac{1}{nm} \sum_{x=0}^{n-1} \sum_{y=0}^{m-1} f(x,y) e^{-2\pi i \frac{ux}{n}} e^{-2\pi i \frac{vx}{m}} \\ &= \frac{1}{nm} \sum_{x=0}^{n-1} \left[\sum_{y=0}^{m-1} f(x,y) e^{-2\pi i \frac{ux}{n}} \right] e^{-2\pi i \frac{vx}{m}} \ . \end{aligned} \tag{A.3}$$

Thus, the 2-dimensional version requires $O(m(n \log(n)) + n(m \log(m)))$ operations or, in the more common case of square images where $n = m$, $O(2n(n \log(n))) \subseteq O(n^2 \log(n))$ operations.

A.2.3 Bilinear Interpolation

Suppose that the value $f(x,y)$ of a unknown continous function f is to be interpolated from values of the four data points $v_{x_0,y_0}, v_{x_0,y_1}, v_{x_1,y_0}, v_{x_1,y_1}$ on a discrete grid next to (x,y). First, two linear interpolations in x-direction can be done resulting in two interpolated values $f(x,y_0), f(x,y_1)$:

$$\begin{aligned} f(x,y_0) &= \frac{x_1 - x}{x_1 - x_0} v_{x_0,y_0} + \frac{x - x_0}{x_1 - x_0} v_{x_1,y_0} \ , \\ f(x,y_1) &= \frac{x_1 - x}{x_1 - x_0} v_{x_0,y_1} + \frac{x - x_0}{x_1 - x_0} v_{x_1,y_1} \ . \end{aligned} \tag{A.4}$$

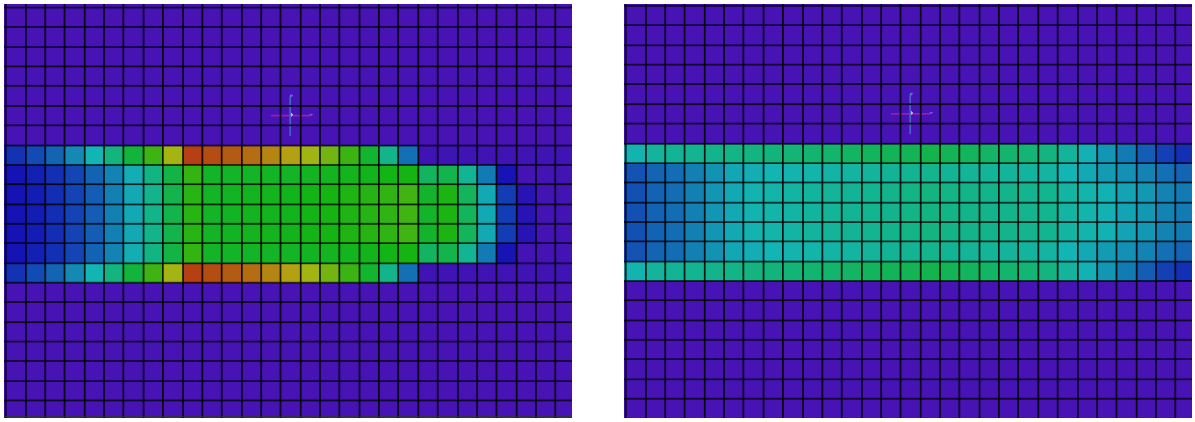
A final linear interpolation in y-direction using those support values yields the bilinear interpolation value $f(x,y)$:

$$f(x,y) = \frac{y_1 - y}{y_1 - y_0} f(x,y_0) + \frac{y - y_0}{y_1 - y_0} f(x,y_1) \ . \tag{A.5}$$

A.3 Experiments

A.3.1 Accumulation using Weighted Additions

Figure A.1 depicts two additional accumulation results for a model moving from left to right with the same velocity in either case which feature different temporal blurring or uncertainties for $\alpha = 0.80$ and $\alpha = 0.99$.



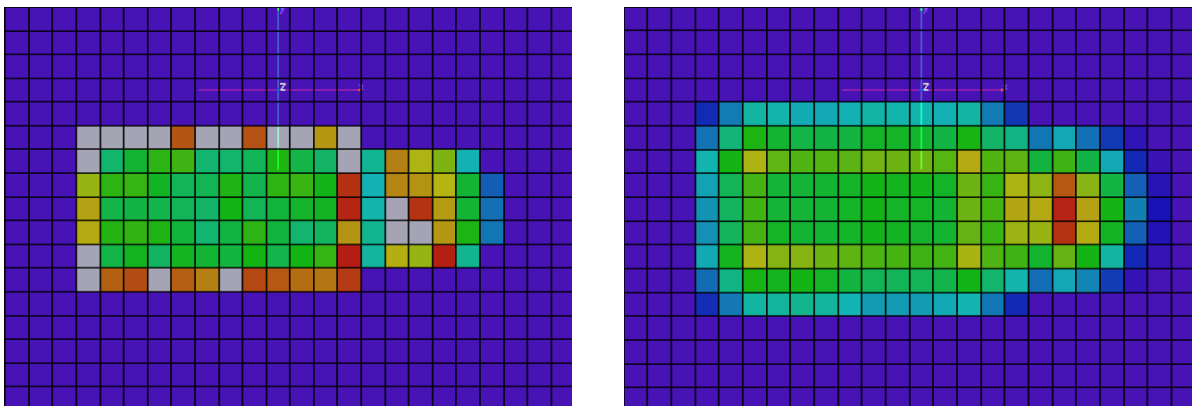
(a) Weighted addition with $\alpha = 0.80$ yielding a moderate temporal blurring.

(b) Weighted addition with $\alpha = 0.99$ yielding a strong temporal blurring.

Figure A.1: Temporal filtering applied to a moving truck model (left to right) using an accumulation based on a weighted addition.

A.3.2 Noise Compensation using Spatial Smoothing

Figure A.2 shows a noisy version of the truck model where values are allowed to deviate randomly within a range of 20 % from the model's reference value. A mean filter is able to compensate those disturbances quite well.



(a) Noisy version of the truck model.

(b) Mean filtered noisy truck model.

Figure A.2: Areal view of the noisy truck model (a) and a smoothed version after mean filtering (b).

A.3.3 Runtime Results

Runtime Results for Accumulation Operations

Table A.1 shows all runtime results in milliseconds for an accumulation operations applied on Aspects of different sizes.

Size (Radius) $n \times n$ (r)	Naive ST	Naive MT `OpenMP`	GPU Kernel `CUDA`
11×11 (5)	[$\ll 0.01$ ms]	[$\ll 0.01$ ms]	0.26
21×21 (10)	[$\ll 0.01$ ms]	[$\ll 0.01$ ms]	0.27
101×101 (50)	0.01	[$\ll 0.01$ ms]	0.42
101×101 (75)	0.02	[$\ll 0.01$ ms]	0.47
201×201 (100)	0.04	0.01	0.56
401×401 (200)	0.16	0.02	1.09
601×601 (300)	0.37	0.05	3.61
801×801 (400)	0.65	0.14	4.74
1001×1001 (500)	1.03	0.19	6.61

Table A.1: Runtime for accumulation operations using different acceleration approaches (ST: single-threaded, MT: multi-threaded).

Runtime Results for Small Convolution Kernels

Table A.2 shows all measurement results in milliseconds for the first scenario where a mean smoothing filter with a fixed kernel size $k = 5$ has been applied to Aspects of different sizes using different convolution approaches.

Size (Radius) $n \times n$ (r)	Naive ST	Naive MT `OpenMP`	Separable `OpenMP`	GPU FFT `CUDA`, `cuFFT`
11×11 (5)	0.12	0.03	0.02	1.41
21×21 (10)	0.39	0.08	0.04	1.44
41×41 (20)	1.49	0.25	0.11	1.43
61×61 (30)	3.31	0.47	0.23	1.43
81×81 (40)	5.81	0.79	0.38	1.57
101×101 (50)	9.06	1.24	0.58	1.45
151×151 (75)	21.32	3.14	1.48	3.34
201×201 (100)	37.22	5.31	2.34	1.95
301×301 (150)	81.87	11.20	5.01	2.67
401×401 (200)	144.14	20.18	8.66	4.85
601×601 (300)	322.46	42.92	19.51	5.92
801×801 (400)	573.41	78.05	36.73	7.14
1001×1001 (500)	893.71	128.02	59.00	8.35

Table A.2: Runtime for small kernels: different convolution approaches for a typical small kernel size $k = 5$ (ST: single-threaded, MT: multi-threaded).

Runtime Results for Large Convolution Kernels

Table A.3 shows all measurement results in milliseconds for the second scenario where Aspects of different sizes have been convolved with a mean filter of the same size $k = n$ using different convolution approaches.

Size (Radius) $n \times n$ (r)	**Naive ST**	**Naive MT** OpenMP	**Separable** OpenMP	**GPU FFT** CUDA, cuFFT
11×11 (5)	0.59	0.28	0.02	1.43
21×21 (10)	6.78	1.30	0.21	1.45
41×41 (20)	96.74	13.10	1.14	1.44
61×61 (30)	462.22	61.01	2.50	1.42
81×81 (40)	1431.30	197.39	6.48	1.53
101×101 (50)	3458.96	459.06	9.71	1.46
151×151 (75)	17219.40	2290.25	31.29	3.42
201×201 (100)	54164.15	7085.87	71.66	2.13
301×301 (150)	[$\gg$ 1 min]	35138.26	247.58	2.73
401×401 (200)	[$\gg$ 1 min]	[$\gg$ 1 min]	574.59	5.12
601×601 (300)	[$\gg$ 1 min]	[$\gg$ 1 min]	1958.23	6.60
801×801 (400)	[$\gg$ 1 min]	[$\gg$ 1 min]	4550.27	7.98
1001×1001 (500)	[$\gg$ 1 min]	[$\gg$ 1 min]	9058.90	8.93

Table A.3: Runtime for large kernels: different convolution approaches for identical Aspect and kernel sizes $k = n$ (ST: single-threaded, MT: multi-threaded).

Runtime Results for Template Matching

Table A.4 shows all runtime measurements in milliseconds for both implemented template matching versions applied on search Aspects of different sizes.

Size (Radius) $n \times n$ (r)	**GPU PC** CUDA, cuFFT	**GPU RIPC** CUDA, cuFFT
11×11 (5)	1.41	3.98
21×21 (10)	1.55	4.21
41×41 (20)	1.67	4.32
61×61 (30)	1.72	5.14
81×81 (40)	1.83	5.72
101×101 (50)	1.96	6.63
151×151 (75)	2.89	7.06
201×201 (100)	3.75	7.30
301×301 (150)	4.51	11.29
401×401 (200)	5.10	15.51
601×601 (300)	6.32	17.36
801×801 (400)	8.56	19.13
1001×1001 (500)	10.96	21.57

Table A.4: Runtime for the basic (PC) and the rotation-invariant phase correlation (RIPC).