A COMPUTATIONAL MODEL OF SPATIAL ENCODING IN THE HIPPOCAMPUS

by

Fabian Schönfeld

A thesis submitted in partial fulfillment of the requirements for the degree of

Philosophiae Doctoris (PhD) in Neuroscience

from the International Graduate School of Neuroscience

Ruhr University Bochum

September 5, 2016

This research was conducted at the Institute for Neural Computation at the Ruhr University under the supervision of Prof. Dr. Laurenz Wiskott

Printed with the permission of the International Graduate School of Neuroscience, Ruhr University Bochum

Statement

I certify herewith that the dissertation included here was completed and written independently by me and without outside assistance. References to the work and theories of others have been cited and acknowledged completely and correctly. The Guidelines for Good Scientific Practice according to 9, Sec. 3 of the PhD regulations of the International Graduate School of Neuroscience were adhered to. This work has never been submitted in this, or a similar form, at this or any other domestic or foreign institution of higher learning as a dissertation.

The abovementioned statement was made as a solemn declaration. I conscientiously believe and state it to be true and declare that it is of the same legal significance and value as if it were made under oath.

Name / Signature

Bochum, September 5, 2016

PhD Commission

Chair: Prof. Dr. Irmgard Dietzel-Meyer

1st Internal Examiner: Prof. Dr. Laurenz Wiskott

2nd Internal Examiner: Prof. Dr. Denise Manahan-Vaughan

External Examiner: Associate Professor Dr. Kechen Zhang

Non-Specialist: Prof Dr. Robert Kumsta

Date of Final Examination: 21st June 2016

PhD Grade Assigned: Magna cum laude

In memory of

Siegfried Schönfeld

(1949 - 2015)

Contents

List of Figures

Abbreviations

CPU: *Central processing unit*

CUBLAS: *CUDA basic linear algebra subprograms*

CUDA: *Compute unified device architecture*

EEG: *Electroencephalography*

FLOPS: *Floating point operations per second*

fMRI: *Functional magnetic resonance imaging*

GPU: *Graphics processing unit*

ICA: *Independent component analysis*

LTD: *Long-term depression*

LTP: *Long-term potentiation*

MDP: *Modular Toolkit for Data Processing*

NAN: *Not a number*

SFA: *Slow feature analysis*

SIMD: *Single instruction multiple data*

SLAM: *Simultaneous locating and mapping*

Abstract

What are the computational principles behind hippocampal activity? In this thesis the slowness principle is suggested as a fundamental process behind hippocampal spatial activity. Six studies from the experimental literature (originally performed with real animals) are presented and replicated in computer simulations. The chosen studies investigate different aspects of the spatial encoding performed in the hippocampus: adaptation to cue relocation and removal; directional dependent firing in the linear track and open field; and morphing and scaling the environment itself. Simulations are based on the hierarchical application of the slowness principle using the slow feature analysis (SFA) algorithm and a custom software package developed for this thesis. The slowness principle is shown to account for the central findings reported in the selected experimental studies. For future developments, hierarchical SFA is embedded within a new general architecture, designed to address the issues of association and integration of several input modalities during runtime. In addition, the architecture aims to bridge the gap between the hippocampal research in humans – targeting episodic memory – and rodents – targeting spatial encoding. As a proof of concept, the extended model is shown to perform spatial mapping as well as a basic memory task.

1 Introduction

This work examines the spatial encoding found in the rodent hippocampus and how computational modeling may further our understanding of both spatial encoding in particular and hippocampal processing in general. In order to provide the necessary context for this work two questions need to be answered:

- *What is the hippocampus and why is it studied?*
- *What makes hippocampal spatial encoding especially interesting?*

The following provides this context, offers an overview of previous work in the field, and states the research questions of this work. Chapter 2 introduces both the principal computational method and the software package used throughout this work. Afterwards the performed computer simulations and their results are presented in chapter 3, which also sees the original model embedded within a more general architecture (chapter 3.3). Finally, all results as well as future perspectives are discussed in chapter 4 and the manuscript ends with the final conclusions of this work.

Note: *The term "model" is used differently throughout various disciplines in neuroscience (e.g., referring to the species used for an experiment as the "animal model"). For theoretical work such as presented here, "model" refers to a mathematical or computational model, which commonly takes the form of a set of (usually differential) equations, or an algorithm and its implementation.*

1.1 The hippocampus

The hippocampus is an essential structure of the mammalian brain. It is located at the center of the limbic system which is part of the medial temporal lobe and plays a crucial role in the processing of emotions, formation of memories, and other high level functions (Bear et al., 2007, p.744ff). The hippocampus receives highly processed sensory information and performs its computations within a multitude of different neural loops – most notably within the so called trisynaptic circuit (fig. 1): neural signals from the entorhinal cortex enter a section of the hippocampus known as the dentate gyrus. This area exhibits only sparse activity but its neural projections along the mossy fiber pathway form strong connections with hippocampal area CA3. The pyramidal cells in area CA3 form recurrent connections among

themselves via associational fibers and project to the neighboring neurons in area CA1 via the Schaffer collaterals. From here the information is transmitted through the subiculum and back to the entorhinal cortex (Andersen et al., 2006, p.42). The entorhinal cortex thus serves as the main (but not exclusive) input/output structure of the hippocampus. It receives information "from the association areas of the cerebral cortex, containing highly processed information from all sensory modalities" (Bear et al., 2007, p.741). It also allows the hippocampus to directly interact with the prefrontal cortex via connections through the parahippocampal cortex (Preston and Eichenbaum, 2013). Furthermore, not only do mammals share the same hippocampal structures (West, 1990), but the functionality of the structure has also been preserved over the deep time of evolution (Rodriguez et al., 2002). These anatomical observations and evolutionary findings of the hippocampal formation support the following:

- The hippocampus as a structure is virtually identical across mammals.
- Its function has remained largely unchanged during the evolutionary process.
- It takes highly processed sensory information as its input.
- It is directly involved in the highest levels of cognitive function.
- Information is processed in a loop, i.e., being iterated on instead of simply propagating through a feed forward network with a distinct beginning and end.

These general findings allow for the following reasoning:

- The hippocampus did not develop to suit the needs of one particular species.
- The computation it performs can thus be assumed to be general purpose and flexible enough to accommodate the sensory input from a variety of different species such as bats, seals, or primates – each of which experience their surroundings in a fundamentally different manner.

While the actual function of the hippocampus and its neural implementation have not been fully clarified as of yet, we do know of at least two essential roles it plays in humans and rodents respectively: In rodents, the hippocampus is involved in spatial encoding; in humans it is critical to episodic memory. The latter became

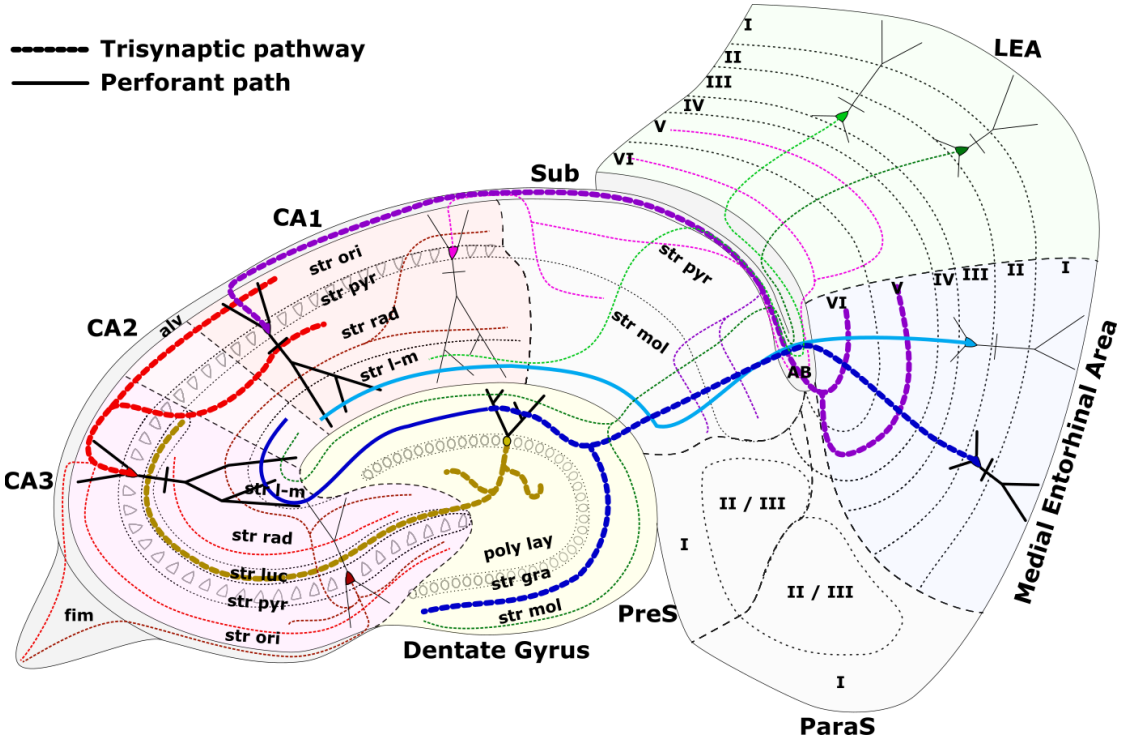

Figure 1: Anatomy of the hippocampus. The hippocampus is a complicated structure that receives input from the entorhinal cortex and consists of the dentate gyrus, areas CA3, CA2, CA1, and the subiculum. Shown in thick lines are the trisynaptic pathway and the perforant bath. The former projects input from the medial entorhinal area layer II to both the dentate gyrus and area CA3; the latter projects input from entorhinal layer III to area CA1. [This image is a slightly altered version of the original created by Stefanie Bothe which is freely available in vector format at https://github.com/MartinPyka/NeuroSVG (accessed 17th of January, 2016).]

apparent in 1953, when the American epilepsy patient Henry Gustave Molaison (1926-2008) underwent a surgical procedure that resulted in the complete removal of the hippocampal formation, as well as large parts of the entorhinal cortex and amygdaloid complex. Even though the operation was successful in the sense that his epileptic seizures were reduced, it became quickly apparent that it also resulted in severe anterograde amnesia in "patient HM." Henry Molaison became unable to form new episodic memories and access memories less than about two years old at the point of his surgery. He was still able to access long term memories, however, and his ability to form new procedural memories was seemingly left intact as well – as demonstrated by him learning to play the piano after his operation while being

unable to recall ever having taken any piano lessons.[1] Henry Molaison worked with scientists in countless studies and in doing so made an invaluable contribution to our understanding of human memory. For the purposes of this introduction, however, it suffices to note the crucial role the hippocampus plays for both the formation of new episodic memories and the access of already established ones.

While the case of Henry Molaison helped our understanding of the human hippocampus, it is problematic to apply this knowledge to non-verbal animals. Because of the obvious difficulties of cross-species communication it is challenging to confirm human-like episodic memory in other mammals – and consequently just as difficult to confirm that the mammalian hippocampus fulfills the same role in animal brains as it does in the human brain. Note, however, that the "true" episodic memory label hinges primarily on a single component that technically cannot be proven in humans either. In (Tulving, 1984), Endel Tulving coined the term "episodic memory" and presented his criteria for a proper memory episode. According to him, a key element of episodic memory is "autonoetic consciousness", the ability of the human consciousness to place itself inside a recalled memory episode. Because of this definition the most scientists can hope for is to show "episodic-like" memory in animals, i.e., to show that animals can recall the different aspects (*what, where, when*) of an episodic memory episode, without claiming the animals consciously relive a previously experienced episode (Hampton and Schwartz, 2004; Dere et al., 2006; Fouquet et al., 2010; Pause et al., 2013; Templer and Hampton, 2013).

Despite this limited capability to examine episodic memory in animals – and thus the main role the hippocampus seems to play in human behavior – an enormous body of research has gone into the study of the animal hippocampus as well. The majority of this research does not target episodic memory, however, but focuses on the issue of navigation and spatial encoding.

1.2 Spatial encoding in the rodent hippocampus

Eighteen years after the fateful surgery of Henry Molaison, Professor John O'Keefe and his group published their initial report of hippocampal place cells in rodents

[1] After working with Henry Molaison for almost 50 years – and introducing herself to him every day anew – Dr. Suzanne Corkin recounts the experience in her book "*Permanent Present Tense*" (2014).

(O'Keefe and Dostrovsky, 1971) – arguably one of the next milestones in hippocampal research[2] and a discovery that resulted in his receipt of the 2014 Nobel Prize in Physiology and Medicine. Place cells are pyramidal neurons found in the hippocampal CA3 and CA1 areas (O'Keefe and Dostrovsky, 1971); as the name suggests they are primarily active when an animal is located within a specific region of a given environment – their place field. When placed in an unknown setting, it usually takes rodents about 8-12 minutes of exploration to map an open area of 1-2 square meters (Yan et al., 2003; Wilson and McNaughton, 1993), though rats have also been observed to form stable fields as quickly as within four minutes or less (Frank et al., 2004; Andersen et al., 2006). Once such an internal map is established, place fields remain stable and place cells fire reliably if the animal finds itself within their respective boundaries. When placed in a different setting, place fields remap seemingly at random, and cells active in one environment may or may not be reused when the animal maps a different environment (Leutgeb et al., 2005b). Different mappings can be remembered for weeks at a time (Ziv et al., 2013) which links place cell research directly back to the memory research done in the human hippocampus. Since place cell activity is by definition recorded from the cells themselves, place field data in humans is much more difficult to come by. Such studies are usually performed with epilepsy patients who agree to invasive monitoring and confirm place field-like spatial activity in the human hippocampus (Ekstrom et al., 2003).

The hippocampal place code does not solely consist of place cell activity though. In 1990 Dr. Jeffrey Taube published the findings of his group about head direction cells found in the postsubiculum (Taube et al., 1990b,a). These cells are sensitive to the direction the animal is facing at any given time, irrespective of the actual location the animal finds itself in. The head direction signal itself is not computed within the hippocampus, however, but is assumed to originate in the dorsal tegmental nucleus (Sharp et al., 2001) with input from the vestibular system being a critical component (Brown et al., 2002). Accordingly, the subiculum is not the only area where head direction cells can be found; since the original discovery in 1990 cells

[2]Just shortly before the discovery of place cells in 1971, Timothy Bliss and Terje Lomo learned about long term potentiation (LTP) in 1969 (Bliss and Lomo, 1973). LTP refers to long a lasting change in synaptic efficacy observed after repeated stimulation and represents another essential result in the history of memory research (Lomo, 2003).

sensitive to head direction have also been recorded in parts of the thalamus (Taube, 1995), the mammillary bodies (Stackman and Taube, 1998), and the striatum (Wiener, 1993), among others.

Next to the ability to encode specific locations via place cells and maintaining a sense of direction via head direction cells, the third component of the hippocampal place code was discovered in 2005 – and its discovery shared the 2014 Nobel Prize in Physiology and Medicine with place cells: In (Hafting et al., 2005) the group around professors May-Britt and Edvard Moser reported the finding of grid cells. These neurons are found in the entorhinal cortex, i.e., upstream of the place cells in the hippocampus proper, and display multiple firing fields aligned with the vertexes of a regular hexagonal grid spanning the environment. The different grids expressed by individual grid cells vary in scale along the the entorhinal cortex with grid spacing and firing fields being larger at the ventral end of the structure and smaller towards the dorsal border (Brun et al., 2008). The offset of the grid varies from cell to cell and while firing fields naturally overlap in parts, the overall grids do not. Grid orientation, however, is locked throughout the population. In case the spatial representation of an animal remaps – such as when an unknown environment is explored for the first time – grid cells may change both their offset and orientation synchronously (Fyhn et al., 2007). One particularly interesting facet of grid cell activity is the non-continuous nature of their firing: When entering distinct sections of an environment, and without any physical obstacle obstructing the path, the grid population remaps seemingly in reaction to the animal entering a different part of the surroundings (Derdikman et al., 2009). Note that this might also apply to place field encoding, but due to the already non-continuous nature of place cells this would be impossible to observe. In contrast to place cells, grid cells and head direction cells do not require a period of time in order to stabilize themselves when entering a new environment (Hafting et al., 2005). As with place cells, recent studies were also able to isolate grid cell-like activity in the human hippocampus thanks to epilepsy patients who agreed to take part in the experiments (Jacobs et al., 2013).

In contrast to most humans, rats are nocturnal animals and the hippocampal place code has been shown to maintain its function even in darkness. Head direction cells maintain their heading in the dark and, due to the unavailability of visual cues,

become more dependent on vestibular cues based on the self-motion information of the animal (Blair and Sharp, 1996). Place cells are similarly affected but also show additional behavior: while some place fields remain intact – albeit usually becoming more diffuse over time – others can be seen to remap, or turn on only when the rat is visiting the area in question multiple times (Quirk et al., 1990; Markus et al., 1994). Remapping behavior further depends on whether animals are located within an apparatus when the lights are switched off, or are placed there in darkness to begin with. In case of the latter, close to 50% of the place cells remap to develop new place fields, while in case of the former, most place cells do not alter their firing behavior (Quirk et al., 1990). In either case place fields remain stable during periods of darkness and allow animals to navigate without the help of visual cues. Grid cell firing also persists during darkness, with grids retaining both their spacing and average firing rate and no remapping taking place (Hafting et al., 2005).

Rats have evolved to navigate under low-light conditions and controlling for their sensory capabilities is a difficult task in itself. It is usually assumed – including in this work – that navigation in darkness largely relies on the ability of an animal to integrate self-motion information, a process known as path integration or dead-reckoning (Etienne and Jeffery, 2004; McNaughton et al., 2006). As a consequence of this the internal spatial representation is subject to a certain drift over time as the inevitable error of unreliable sensory information accumulates (McNaughton et al., 1996; Müller and Wehner, 1988). In order to maintain accurate functionality during normal circumstances, the system is thus assumed to use visual cues to anchor and recalibrate this internal representation (McNaughton et al., 1996). For path integration to be examined, experiments taking place in darkness have to ensure that animals truly lose their access to any salient external cues, including cues outside of the human sensory range. As an example, rats commonly leave small urine marks throughout the environment to mark territory and aid in navigation. While the smell of such droplets might dissipate quickly, urine is visible under ultraviolet light and rat vision[3] is known to extend into the ultraviolet end of the spectrum, possibly turning urine markers into visible navigation aids (Desjardins et al., 1973). While common experimental practices should easily prevent this, laboratories often include a variety of electronic equipment which emit

[3]For an excellent overview of the sensory capabilities of rats visit www.ratbehavior.org [accessed January 2016]

electromagnetic fields that rodents have been shown to be sensitive to (Mather and Baker, 1981). In addition, the hearing capabilities of rats also include ultrasound (Kelly and Masterton, 1977) which is emitted from speakers that, ironically, are commonly used to emit white noise in order to prevent animals to make use of auditory cues. This can be problematic since more than one speaker is required to mask the direction of the audio source and different speakers emit different levels of ultrasound, offering the animals a directional cue during navigation in darkness (Burn, 2008). However, this is not to discredit any experimental findings – experimental protocols are usually well aware of these issues – but rather to demonstrate **(a)** how hippocampal spatial encoding is stable under different sensory conditions, and **(b)** that high level features such as an encoded orientation, awareness of place, and grid cell functionality do not rely on visual cues alone but rather seem to derive from a whole hierarchy of different sensory modalities.

To conclude this introduction to spatial encoding in the hippocampus, there is a collection of additional cell types that have been recorded since the discovery of grid cells in 2005. These are listed in the following for the sake of completeness, but are not part of the modeling efforts and the discussion that follows.

- **Boundary vector cells (2010):** After predicting boundary vector cells with the help of computer simulations in (Barry et al., 2006), neurons fulfilling those properties were recorded one year later in the subiculum (Lever et al., 2009), i.e., downstream of the place cells in the CA1 and CA2 areas. Boundary vector cells are active whenever an animal is close to a specific border of the environment, with activity decreasing as the animal increases its distance to it. They function in darkness and their firing activity is independent from the direction the animal is facing at the time. Interestingly, they also remain stable in the face of environmental changes that lead to remapping in the place cell population (Lever et al., 2009).
- **Time cells (2013):** In (MacDonald et al., 2011) cell recordings were reported that suggest the encoding of (delay) times during experimental procedures. This temporal activity cannot be linked to external parts of an experimental protocol and is claimed to "represent the flow of time in specific memories" (Eichenbaum, 2014). As such neurons have been recorded during spatial

trials in rodents this discovery could go a long way in linking hippocampal spatial encoding with the memory research done in humans. However, more research will be required to substantiate this claim and examine how it may be connected to episodic-like memory in rodents.

- **Speed cells (2015):** In order for grid cells to perform their function they need to know about the direction the animal is facing and how fast it is going. While the directional component is encoded within the different populations of head direction cells found in the brain, the speed signal has remained more elusive. In (Kropff et al., 2015), however, it is reported "that running speed is represented in the firing rate of a ubiquitous but functionally dedicated population of entorhinal [cortex] neurons."

Finally, it is important to note that place fields are not the exclusive mechanism to encode space in the mammalian brain. Theta rhythm activity can be measured to travel as a wave through the hippocampus along the septotemporal axis (Lubenov and Siapas, 2009) and has been intrinsically linked to place field activity (Buzsaki, 2005; Mizuseki et al., 2012). In addition, long-term potentiation (Bliss and Lomo, 1973; Bliss and Collingridge, 1993) and long-term depression (Lynch et al., 1977; Dayan and Willshaw, 1991) are fundamental mechanisms to not only maintaining spatial representations, but hippocampal learning in general (Andersen et al., 2006, p.343-420). Both theta (and other) oscillations and synaptic plasticity are important fields of study by themselves and have seen a tremendous amount of resources dedicated to investigating them. Since both of these mechanisms operate on the neuronal level, however, they are not a part of the modeling efforts presented in this work.

What makes hippocampal spatial encoding especially interesting? Going back to the initial question posed in this introduction, what makes hippocampal spatial coding uniquely fascinating is not the fact that it is happening in the first place, but rather the fact that we, as scientists, can actually *observe* the hippocampus performing spatial encoding. When compared to the raw input from nerve cells in, for example, the vestibular system, or the receptors forming the first stage of the visual system in the retina, elements like "*place*", "*orientation*", or "*intersection that requires a decision*" are highly abstract concepts. Even if fundamental and

obvious to our understanding of the world, computing these concepts from nothing but the original input of noisy neural spikes is a tremendous achievement by the mammalian nervous system. Thus, being able to directly observe the result of this computation is very different than recording the preference of a visual receptor for a distinct input pattern. While the latter furthers our understanding of our visual senses, the former helps our understanding of the mental processes whose results we are actually consciously aware of. The spatial elements being encoded and processed in the hippocampus are immediate in our very experience of the outside world. Place cells in particular allow for an insight into this process. It is, again, not about the fact that place cells encode distinct locations, but about us being able to directly observe this fact. In other words, there exists a direct relation between the input, in the form of of the outside environment, and the output, in terms of place cell activity – activity that we can not only observe but also *interpret*. Apart from the cells involved in the hippocampal place code, there are no other individual neurons in the mammalian brain that we know of that directly encode such high level concepts.[4]

What is the hippocampus and why is it studied? To address this second question, it is necessary to take one step back from spatial encoding and consider this ability in the rodent hippocampus in conjunction with what has been learned from the human hippocampus, namely its involvement in episodic memory. Recall that the anatomical makeup of the hippocampus is virtually identical in mammals and has been this way over the deep time of evolution. And yet this one structure serves as the central component in at least two different functional areas of two different species that vary widely in elemental properties such as physical shape, cognitive ability, and natural habitat. To illustrate this point consider bats – nocturnal animals a fraction of the size of a human, that navigate truly three dimensional space via echolocation. Not only do experiments show grid cells in the entorhinal cortex of bats (Yartsev et al., 2011), but there are even reports of three dimensional, volumetric "place fields" (Hayman et al., 2011; Yartsev and Ulanovsky, 2013). No studies should be required to argue that the sensory input across the different modalities in animals such as rats, bats, and humans can be

[4]There are the so called grandmother cells (Gross, 2002), or Jennifer Aniston neurons of course (Quiroga et al., 2005), but these do not dynamically adapt to input manipulations.

assumed to be vastly different. And yet the same prevailing structure operates at the center of the brain of each of these species, processing this information and providing crucial functionality. This allows for one general conclusion and one central assumption:

- **Conclusion:** During mammalian evolution the hippocampus did not adapt to individual species to a degree that would leave its structure drastically changed. The architecture of the hippocampus was preserved even while the phenotype of individual species changed dramatically over time. Consequently, the computation performed by the hippocampal formation – even if adapted to the individual input statistics of a particular species – should be expected to be similar, if not identical across species.
- **Assumption:** In order to be as universally useful as the computation of hippocampal data loop appears to be, it has to be assumed to be a high level, general purpose process. "High level" hereby refers to the hippocampus primarily dealing with the more abstract elements that are communicated by the preceding brain structures, e.g., "face" instead of the underlying "roundish shape of various colors." The second term, "general purpose", refers to the hippocampal computational loop being flexible enough to be essential for at least two high level processes that we know of – navigation in rodents and episodic memory in humans – as well as its to accept a wide range of different input compositions.

Bridging the gap between episodic memory in humans and spatial encoding observed in rats is one of the major goals in contemporary neuroscience and especially in hippocampal research. Though these are two largely separate research fields, the underlying assumption of this work is that scientists are observing two aspects of the same computation as it is being expressed by the behavior of the respective host species, rats/mice and humans. Furthermore, as argued above, it may be postulated that this core computation, in order to be generally applicable, is not likely to be complicated. This is not to be confused with its implementation, which should be expected to be a sophisticated process. But a description of the overall functionality of the system might be surprisingly simple – and thus be expressed by an equally simple, if highly abstract, model.

While this is pure speculation, computational neuroscience is well equipped to confirm or disprove such assumptions. Theoretical models cannot hope to capture the intricate observations collected over decades of experimentation on a meaningful scale. They are limited by available processing power and memory resources, and have to adhere to a computational architecture that operates very differently from the mammalian brain. As such, computational models, by necessity, abstract heavily from biological observations and attempt to replicate them using systems that are significantly less complex. This should not be seen as a disadvantage, however, but rather be understood as an approach that is looking to answer a different set of questions. In the case of this work, an extensive body of hippocampal research was taken into account to find support for the assumption outlined above: Can a case be made for computational principles, that are general purpose, to explain (parts of) hippocampal function?

1.3 Previous work on modeling the hippocampus

Note: *From here on "experiment" or "experimental study" refers to real life experiments done with actual animals, while "simulation" refers to experiments done in virtual environments and other computer simulations.*

Previous work in modeling the hippocampus has focused on the same two main experimental areas: its role in episodic memory in humans, on the one hand, and spatial encoding in rats, on the other. In both experimental and theoretical studies, comparatively little work has been published in bridging the gap between the two. When comparing and discussing computational models they can be distinguished by a number of criteria: Is the model accepting realistic input (e.g., plain images) or working with abstract data carrying additional information (e.g., distance information to surrounding obstacles); is it the intent of the model to explain biological data or employ biologically inspired processes to solve an engineering problem (the latter will not be discussed here); what is the targeted system and the concomitant level of abstraction (single cell, cell population, overall system); is the output of the model based on the current input alone (feedforward) or also takes into account previous output (recurrent); does the model aim to provide new predictions for future experiments or validate a hypothesis by replicating known experimental results?

The modeling efforts presented in the following are almost exclusively based on artificial neurons, commonly called "units." While this approach keeps a computational model close to the biological system it attempts to simulate, it also restricts modeling efforts to subsystem scale at the most. To provide some context for this approach to modeling, briefly consider large scale simulations and their trade-offs: In 2005 Eugene Izhikevich took 50 days to simulate one second of activity of a hundred billion neurons, the estimated neuron count of the human brain. While technically simulating individual neurons, the model "only" simulated neural dynamics and no neural learning took place – in fact units where immediately discarded after use during runtime in order to allow the model to fit into the available memory at all (Izhikevich, 2005).[5] A large scale network actually capable of learning was published in (Le et al., 2012). Ten million images were used in an unsupervised training phase to allow a hierarchical network to learn $22,000$ object categories. Note, however, that in contrast to the simulation done by Izhikevich, no actual time frame was being simulated. The model passed a training stage and was then frozen and tested to evaluate its object recognition capabilities. To perform this large scale simulation, $16,000$ processing cores were used in a parallel cluster running for three days (Le et al., 2012). As can already be seen by looking at these examples, different models at different levels of abstraction have to consider different tradeoffs that dictate what questions can be asked at which level of modeling.

1.3.1 Models of spatial encoding

Computational models aimed at simulating spatial representations can be very simple. The model presented in (Sharp, 2013) receives both distance and orientation information regarding a number of set up distal cues. This information is used to compute the activity of individual units and results in place fields that display the irregular shape often observed in real neurons, remain stable even after the removal of some of the cues after learning, and stretch appropriately with the environment. The model is also able to account for directional unit activity depending on the movement statistics during the initial learning phase (Sharp, 2013). While this

[5] Unfortunately this can only be sourced by Eugene Izhikevich's personal website, a local copy of which can be found on the enclosed disc [http://www.izhikevich.org; mirrored 17th of January 2016].

model operates on input that already contains high level information compared to raw visual image data, it nevertheless shows how stable place cell-like activity may be produced by a feedforward algorithm and the principle of competitive learning.

In (Fuhs and Touretzky, 2000) different learning rules are examined to determine their success in modeling the observed remapping properties in real neurons. Their model consists of $22,000$ units split into different populations to represent the entorhinal cortex, dentate gyrus, and hippocampal area CA3. The authors report that both traditional Hebbian learning and Hebbian covariance learning proved to be unsuccessful in modeling the desired outcome. However, when using a third learning rule – originally presented in (Bienenstock et al., 1982) – the model is able to produce the remapping behavior recorded in actual cells (Fuhs and Touretzky, 2000). Their work demonstrates how computational studies are able to efficiently compare and evaluate different approaches and determine which hypotheses should be considered more closely.

Excursion: ***The cost of computation, a matter of scale.*** If the model presented in (Samsonovich and McNaughton, 1997) was to be extended to the actual size of the rat hippocampus, it would have to include about $n := 4.26$ million artificial neurons (Andersen et al., 2006, p.100). According to equations (1) and (2) in (Samsonovich and McNaughton, 1997) the computation of one time step takes, at the very least, $j := (5+2n)n + (2n+n) = 2n^2 + 8n$ floating point operations per individual time step, defined as 6 milliseconds. Simulating a ten minute time interval – the time required for rats to acquire a stable spatial representation – would thus require $100,000j$ operations, amounting to about 3.6×10^{18} operations. At the time of publication, the cost of one Gigaflop (10^9 FLOPS) amounted to about $\$50,000$ (Warren et al., 1997) and a contemporary Gigaflop-capable machine would take 3.6×10^9 seconds, or 114 years, to compute the simulation. Technology moves quickly, however, and in 2015 a contemporary Intel Celeron G1830 (valued at $60 as of January 2016) may reach up to 11.5 teraFLOPS – thus theoretically able to compute the same calculation in $31,000$ seconds, or about 8.5 hours. While this seems reasonable, modeling techniques have progressed as well and require much more operations per second than the model in (Samsonovich and McNaughton, 1997) – thus pushing large scale simulations at neuronal resolution yet again out of reach.

The discovery of recurrent connections in hippocampal area CA3 serves as the basis for recurrent neural networks, i.e., networks that feed the output of each time step back into themselves as an addition to the external input of the next time step.

Such models do not compute a result in one go and stop, but instead iterate their computation until they converge to a stable state. Their original use in modeling is based on the fact that such neural networks can be trained to store distinct patterns which can be restored with only partially complete versions of the original – hence their designation of autoassociative memory. This property is of obvious use when it comes to model memory, but has also been used to model spatial encoding. In (Skaggs et al., 1995) a linear attractor model based on recurrent connections is used to model the firing patterns of head direction cells. To prove the validity of the core principle, they aimed "to develop the simplest possible architecture consistent with the available data" (Skaggs et al., 1995).[6] Often a clean and elegant model will be preferred over one that, using additional mechanisms, is able to match experimental data more closely. As noted above, this is to demonstrate the validity of a computational building block which, once established, may then be used and built upon in future work. The one dimensional line attractor approach used in (Skaggs et al., 1995), for example, was later extended to be used in two dimensional continuous attractor models (Zhang, 1996). The idea is elaborated on in (Samsonovich and McNaughton, 1997) where place cells are modeled using a recurrent network linked with output from an artificial path integration system. As in (Skaggs et al., 1995), the authors emphasize a lean design philosophy and introduce their work as "A minimal synaptic architecture" (Samsonovich and McNaughton, 1997). Nevertheless their model maps to a number of anatomical structures and is designed to capture the general behavior of hippocampal spatial encoding instead of focusing on replicating a specific experimental observation – thus taking one important step towards modeling the hippocampus as a whole.

One specific question computational models have been focusing on is the remapping behavior of the hippocampal space code in the face of geometric changes. This is because modeling such a task does not require the simulation of any high level processes such as goal oriented behavior or navigation. In both the real life experiment and the virtual simulation, the agent in question only needs to look around and allow its internal spatial representation to adapt. In (Kli and Dayan, 2000) a novelty signal is used to trigger place cell remapping in an unknown environment

[6] Though not always available, this "let's understand the simple version first"-approach is also used by experimentalists with great effect – as exemplified by Eric Kandel and his well-known experiments with the nervous system of the sea slug *Aplysia californica* to understand learning processes (Carew et al., 1971).

after the model has established a spatial representation in a previous environment. The novelty signal allows the model to display both global and rate remapping in different environments of varying similarity (Kli and Dayan, 2000). Note that abstract signals such as this one, however, would serve as powerful cues to any model and thus have to be motivated carefully if added to any simulation.

1.3.2 Models of episodic memory

Computational models of hippocampal episodic memory initially focused on how an artificial neural network might be able to store any memory at all. The common abstraction is to define a distinct pattern of activity within a neural network as a single memory. A model is said to have learned this "memory" if its pattern constitutes an attractor state of the network, i.e., when, provided with an incomplete version of the pattern, the network converges to the original version and then remain stable. In David Marr's seminal work (Marr, 1971) – described as "an epochal contribution to theoretical neuroscience" (Willshaw et al., 2015) – he has outlined the basic architecture of such a system based on Hebbian learning (*"what fires together wires together"*).[7] He has also introduces a theory of how memory enters long term storage and proposes the hippocampus to only store a reference ("simple representation") to a full memory residing in the neocortex. This has lead to the later suggestion of the dentate gyrus working as a pattern separator (McNaughton and Morris, 1987). More precisely, the dentate gyrus is proposed to compute a compressed hash key,[8] or reference, to be stored in the hippocampus and used to recall an actual full scale memory from the cortex. Anatomical findings seem to agree with this idea: the larger number of neurons in the dentate gyrus compared to the upstream/downstream areas; the sparse firing patterns exhibited by the dentate gyrus; and the so called "detonator" synapses of dentate gyrus neurons projecting to CA3 (Blackstad and Kjaerheim, 1961; Jonas et al., 1993). The latter owe their name to the unusual size of the mossy fiber presynaptic boutons, which are assumed to form connections strong enough to make CA3 pyramidal

[7] This well-known phrase was not actually introduced by Donald Hebb himself, but only coined in 1992 by Siegrid Löwel (Löwel and Singer, 1992).

[8] In computer science, a hash function takes a package of data of arbitrary size and returns a much smaller hash value (or key) to be used to address the original data. It allows for rapid data lookup by searching not for the original data but the expected hash value instead. The hash value then, in return, points to the actual data.

cells fire with the input of one detonator synapse alone (McNaughton and Morris, 1987). Taken together, the dentate gyrus exhibits the ability of both computing a sparse representation, and "imprinting" it on the downstream CA3 area. If this overall architecture is presumed to be true, however, the full scale memory would not actually enter the hippocampus proper through the dentate gyrus – as this structure would only relay the hash key – and thus would have to be otherwise transported to the cortex (e.g., the perforant path as seen in fig. 1, p. 3). It also implies that a damaged hippocampus would not, in fact, lead to memory loss, but rather to losing the ability to index and access otherwise intact memory structures.

Modeling work after Marr largely kept the fundamental paradigms and architecture he proposed intact and focused on either **(a)** adding mechanisms and modifications to allow the model to account for additional properties of memory encoding or **(b)** examine individual aspects of the model to suggest future improvements. This includes matching the overall architecture closely to anatomical data (Rolls, 1996); considering the role of context compared to content (Raaijmakers and Shiffrin, 1981); the properties of memory access by reference, or "header"[9] (Morton et al., 1985); the necessity for discrete system states for encoding and retrieval (McNaughton and Morris, 1987); and the role of the hippocampus in (cortical) long term storage (McClelland et al., 1995), among others.

Even though David Marr's original ideas have been expanded on, his approach is still at the heart of the so called "standard framework" (Nadel and Moscovitch, 1997) of hippocampal memory function (Treves and Rolls, 1992, 1994), widely regarded as the current state of the art.[10] However, it is important to remember that the original work intended to show how an artificial neural network might be able to store anything at all and what the implications of the proposed approach would be. Consequently, most research using the standard model excludes the use of realistic input (though different modalities have been proposed to be weighted differently) and attempt to adhere as closely to the biology as possible – restricting modeling efforts to neural networks and the limited scale they are able to describe (see above).

[9] Introducing yet another term for the same principle: "simple representation" (David Marr), "header" (John Morton), "hash value" & "reference" (computer science). While all terms refer to the same principle each implies a different interpretation of their assumed implementation.

[10] For a critical review and a proposed alternative to the standard model see (Cheng, 2013).

Further, while modeling efforts have cited episodic memory research and explored the role of the hippocampus in context encoding, the primary attribute of episodic memory – storing episodic sequences of events – is not straightforward to reconcile with the "standard framework." Finally, while autoassociative networks and their ability to reproduce learned patterns using partial cues are an attractive model for memory, they also include significant drawbacks: first, the storage capacity of the default Hopfield network using Hebbian learning is limited to 13.8% (Hertz et al., 1991), allowing the human CA3 area (containing 2.83 million neurons (Andersen et al., 2006, p.100)) to store no more than about $400,000$ different patterns during its lifetime; second, reaching the limits of storage does not mean that new patterns simply cannot be stored anymore, but rather that learning them would eradicate all previously learned patterns, a process known as catastrophic interference (Cl. and C., 1989).

1.3.3 Linking spatial encoding and episodic memory

Computational models trying to link together the two functions of spatial encoding and episodic memory are, unfortunately, not numerous. Often it is merely pointed out how a model targeting navigation also includes features that might be interpreted as aspects of episodic memory, or vice versa. In (Redish and Touretztky, 1998), for example, a model focusing on the Morris water maze (D'Hooge and De Deyn, 2001; Vorhees and Williams, 2006) is presented. It is based on the principles of self localization and post-training internal replay. The latter requires the ability to store and retrieve pattern sequences which can be interpreted as the recollection of an episode. Similarly, in (Jensen and Lisman, 2005) a model of different memory mechanisms (short and long-term) sequence encoding based on LTP are presented. To link this approach with navigation, the authors note the possibility of such a sequence to be made up of a series of previously visited locations (Jensen and Lisman, 2005). In (Burgess et al., 2001) an intricate system is presented in an attempt to tie together spatial context and long-term memory: Places are encoded by linking together landmark information such as distances, texture information, and relative headings. A variety of individual cell populations is dedicated to code for these different properties. To represent a specific location the required input data is not gathered by a sensory module of the system, however, but rather cued internally. The model is then designed to use such a set of

local cues to retrieve the associated data of a scene and reconstruct the missing elements. While the initial cues are encoded from an egocentric perspective, the model maintains an allocentric representation in long-term memory and is able to translate from one representation to the other. To link this model back to episodic memory, the allocentric encoding is interpreted as a (spatial) context and "retrieval of the rich context of real-life events is a central characteristic of episodic memory" (Burgess et al., 2001).

1.4 Outline of the modeling approach

The computational modeling work presented in this work is primarily aimed at the spatial encoding aspect of the hippocampus as it is observed in mice and rats. In contrast to the models reviewed above, however, the fundamental element of computation is not an artificial neuron. In common neural networks individual units usually compute the sum over their input and use a nonlinear function, such as a sigmoidal, to determine their output. A trained neural network stores the acquired knowledge not within the units themselves, but within the connections between them. In contrast to such units, the individual *nodes* forming the model used throughout this work perform a much more sophisticated computation and store the knowledge gained during training within themselves. This computation is not to be confused with how artificial neurons function and is not intended to model the activity of single cells.

A hierarchical network architecture is used to apply this computation repeatedly and extract a set of increasingly abstract features from the original input. When trained in a virtual environment, this approach leads to the development of a spatial representation. Since no artificial neurons are being used and the model is designed to simulate the behavior of an overall system rather than any one particular structure, it is said to operate on a *systems level* of abstraction. The model further uses only natural image sequences as its input and does not receive any other information such as directions, boundaries, or distances relating any visual or geometric features. The architecture of the model is also left unchanged in between simulations and no internal parameters are tuned to better fit the network to any one simulation trial.

1.4.1 The slowness principle

The computation performed by the individual nodes of the network is the application of the slowness principle – a general principle aimed at computing invariant features from a stream of data (Wiskott and Sejnowski, 2002). It states that while individual input channels convey data as quickly as they are able to, any meaningful information they communicate changes much slower in time. To illustrate this principle, consider noisy TV reception: The flickering noise over the actual image changes as quickly as the hardware of the TV is able to produce it – which is much faster than any relevant element of the actual scene being shown, and thus may be discarded without the loss of meaningful information.

The slowness principle suggests to ignore the quickly changing information of a given input and to focus instead on the slowly varying components in order to extract the meaningful elements encoded within the data. Note that the nouns used to describe this process are general terms not restricted to any specific context. The slowness principle may be applied to essentially any stream of data – it is a general principle – and has been shown to be applicable to object recognition (Franzius et al., 2011), hand writing recognition (Berkes, 2005), gender and age estimation (Escalante-B and Wiskott, 2010), blind source separation (Blaschke and Wiskott, 2004), and many others (Escalante-B and Wiskott, 2013).

The original idea of focusing on those parts of the data that change slowly over time dates back as far as 1989 when it was termed "smoothness" and proposed to be used in unsupervised learning (Hinton, 1989). Two years later a neural network model of the visual system was presented that makes use of the "running average of the activation of [a] unit," which "embodies the assumption that the desired features are stable in the environment" (Földiak, 1991). In (Wiskott and Sejnowski, 2002) the *slow feature analysis* (SFA) algorithm for unsupervised learning is presented – a closed form implementation of the slowness principle that does not get trapped in local optima and extracts a set of uncorrelated features ordered by their level of slowness. SFA is the central computational element used throughout this work and the individual nodes forming the hierarchical network each run an instance of the SFA algorithm.

The slowness principle has been applied to visual processing and spatial encoding before: in (Berkes and Wiskott, 2005) it is shown how SFA can be used to develop receptive fields from natural image data; (Wyss et al., 2006) describes how the use of "the computational principle of optimal stability of sensory representations" yields a place field-like spatial representation on a robot platform; and in (Franzius et al., 2007) a virtual agent using hierarchical SFA is shown to produce not only place cell-like behavior, but also yield head direction and spatial view cell activity.

1.4.2 Research question

To summarize, the modeling approach in this work may be characterized as follows:

- ◇ ***The model*** *used throughout this work can be classified as a general purpose systems model that applies the abstract computational principle of slowness in hierarchical fashion to realistic visual input.*

Recall that the hippocampus did not evolve to adapt to individual species but rather seems to perform a set of generally useful computations. With this both the primary and secondary research questions of this work may now be stated as follows:

- ◇ ***The question*** *this work set out to answer is whether the hierarchical application of the general slowness principle is able to compute a spatial representation that behaves similarly to what is reported in experimental studies.*
- ◇ ***If so****, can these results be used to bridge the gab between the two most examined hippocampal functions of spatial navigation in rodents and episodic memory in humans?*

In order to answer the first question, a set of software tools was developed to allow for the construction of a variety of different virtual environments and perform a diverse range of spatial simulations. This toolkit was used to reproduce a range of real life experiments and demonstrate how successful the slowness principle is in computing a spatial representation that behaves similarly to the place code recorded in the rodent hippocampus. The software was further designed to be usable by scientists of different backgrounds to allow other researches to replicate

and build upon the results presented in this work. To address the second question, hierarchical SFA was embedded into a larger system in an attempt to present an architecture capable of solving both a navigational and an episodic memory task.

2 Materials and Methods

This sections provides an overview of the principal methods of investigation employed in this work: the slowness principle and its implementation in form of the slow feature analysis algorithm, as well as the software tools developed over the course of this work. While the software package was published by itself (Schönfeld and Wiskott, 2013), it was also used to perform the simulations presented in chapter 3 and is thus presented as a method used throughout this thesis, rather than a result thereof.

2.1 The slow feature analysis algorithm

Slow feature analysis (SFA) is an unsupervised learning algorithm that implements the slowness principle and allows its application to any continuous stream of data (Wiskott and Sejnowski, 2002). More precisely, given a multidimensional input signal $\mathbf{x}(t) = [x_1(t), x_2(t), ..., x_N(t)]^T$, SFA computes a set of real-valued functions $g_1(\mathbf{x}), g_2(\mathbf{x}), ..., g_k(\mathbf{x})$ such that for each output signal $y_i(t) := g_i(\mathbf{x}(t))$

$$\Delta(y_i) := \langle \dot{y}_i^2 \rangle_t \quad \text{is minimal} \tag{1}$$

under the following constraints

$$\text{zero mean:} \quad \langle y_i \rangle_t = 0, \tag{2}$$

$$\text{unit variance:} \quad \langle y_i^2 \rangle_t = 1, \tag{3}$$

$$\text{decorrelation and order:} \quad \forall i < j, \langle y_i y_j \rangle_t = 0. \tag{4}$$

Equation (1) defines the term "slowness" in mathematical terms. The slowness score of a signal is defined as the squared average of its derivative, i.e., change, over time. The overall goal of slow feature analysis is to extract signals such that their slowness score is minimized. Note that by this definition the slowest signal is a constant, i.e., no change at all. To avoid SFA extracting such signals that carry no information the output signals $y_i(t)$ have to adhere to constraints (2) and (3). Constraint (4) states the requirement that each extracted signal $y_j(t)$ is decorrelated with every previously extracted signal $y_i(t)$. In other words, once the slowest signal is found, SFA extracts the next slowest signal that is uncorrelated to the preceding one and thus presumed to encode a different slow feature of the original input.

When the linear case of the optimization problem is examined, it can be shown that the solution may be computed by solving a generalized eigenvalue problem (Berkes, 2005). To make use of this insight while, at the same time, not restricting itself to the linear case, the SFA algorithm first expands the input quadratically.[11] That is, the original set of input signals $x_1(t), x_2(t), ..., x_N(t)$ is expanded to $\mathbf{z} := [x_1(t), .., x_N(t), x_1^2(t), .., x_N^2(t), x_1x_2(t), x_1x_3(t), .., x_Nx_{N-1}(t)]^T$. Quadratic expansion is usually the default expansion when using hierarchical SFA, though other expansions might yield superior results, depending on the problem at hand. (See (Escalante-B and Wiskott, 2011) for a review of nonlinearities for SFA.) Once the expanded input $\mathbf{z}$ is computed from $\mathbf{x}$, SFA solves the generalized eigenvalue problem

$$\mathbf{AW} = \mathbf{BW\Lambda} \tag{5}$$

$$\text{with} \quad \mathbf{A} := \langle \dot{\mathbf{z}}\dot{\mathbf{z}}^T \rangle_t, \tag{6}$$

$$\text{and} \quad \mathbf{B} := \langle \mathbf{z}\mathbf{z}^T \rangle_t. \tag{7}$$

Each of the desired output functions $g_j(\mathbf{x})$ is a linear combination of the expanded signals $\mathbf{z}$, using the coefficients stored in the eigenvectors that make up matrix $\mathbf{W}$:

$$g_j(\mathbf{x}) := \mathbf{w}_j^T \mathbf{z} \tag{8}$$

The slowness score of the output signal computed by function $g_j(\mathbf{x})$ using eigenvector $\mathbf{w}_j$ is the eigenvalue λ_j stored in the diagonal matrix $\mathbf{\Lambda}$.

When slow feature analysis is applied to a spatial navigation task, it receives a series of images depicting the view of a virtual agent while it explores a given environment. The individual pixels of the image through time form the input data $\mathbf{x}(t)$. Within a rectangular environment the easiest way to encode location is to encode the x and y coordinates. Assuming a movement paradigm that emphasizes quick changes of (view) direction over slow changes in position, this is precisely what SFA produces. The two slowest components of such a simulation are two signals that steadily decline when sampled from left to right and top to bottom respectively

[11] Or by any other finite dimensional nonlinear set of functions.

(e.g., fig 7, p.35). In (Franzius et al., 2007) the mathematical background of this result is discussed in detail and it is explained how the eigenfunctions computed by SFA are differently parameterized combinations of sine and cosine functions. This can be confirmed by sampling different SFA output signals along one axis of the environment and fitting a cosine wave to it (fig. 2).

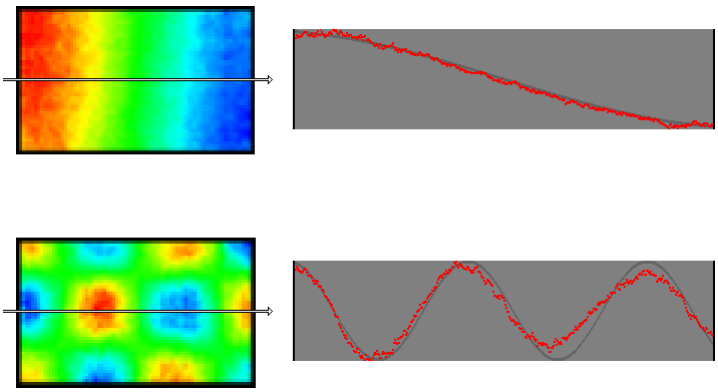

Figure 2: SFA signal cross section. The spatial code produced by SFA in a rectangular environment can be shown to consist of different combinations of cosine waves over the environment. **Left:** Two SFA signals sampled over a rectangular environment. An arrow indicates the line along which the cross section of the signal is plotted. **Right:** Sampling of the same signals along the line indicated by the arrow on the left. The original SFA signal is plotted in red over a fitted cosine signal (dark grey).

Throughout this work, movement is characetrized by rotation and slow translation. Consequently, the agent during exploration tends to stay in roughly the same area for a while before moving on. When this behavior is inverted, i.e., the agent focuses on translation while leaving its direction mostly fixed, view directional changes slower than the position of the agent. In this case SFA output no longer codes for position but for direction instead (e.g., fig. 8, p. 37).

2.2 The ratlab toolkit

In order to perform the necessary simulations to answer the research questions stated in the introduction a number of software tools are required. Note that the capability of hierarchical SFA to produce a spatial representation has already been demonstrated in (Franzius et al., 2007). The shown SFA simulations, however, were restricted to a single rectangular environment, and the developed source code

is not available anymore. It was therefore only partially possible to build on the groundwork laid in earlier publications and software had to be developed anew.

2.2.1 Original software requirements

The initially stated objectives of the software framework – termed "ratlab" during development – were as follows:

1. **Perform virtual experiments:** Create a virtual, three dimensional, environment and populate it with an artificial agent. Both the layout of the environment and the behavior of the agent have to be configurable by the user via a set of optional parameters. While the agent traverses the environment, its visual input, velocity, and position are to be recorded for later use; the simulation also needs to accommodate the unusually wide field of view (320°) of rats (Hughes, 1979).

2. **Train hierarchical SFA:** Create a predefined hierarchical SFA network and train it using the visual data provided by a previously performed simulation. The user may append an additional ICA layer to the network during training or afterwards, and specify which layers of the network are to be trained.

3. **Plot results:** Sample the trained SFA network over the original environment to determine it spatial activity. Sampling results are to be stored as heat map plots on the disk. The user may specify over which directions the SFA signals are plotted (n, ne, e, se, s, sw, w, nw); in addition, the software plots the average activity over all specified directions.

In short, the software framework has to automatically perform all the required steps to allow the user to design an experimental protocol, perform the simulation, and obtain the finished plots in one, unsupervised process. In addition to these functional requirements, there was also a performance requirement: The framework must be able to handle input sizes of up to $100,000$ time steps. Input frames consist of 320 by 40 pixel images, and each pixel is defined by three color components (with a range of 0-255 for each). This results in $100,000 \cdot (320 \cdot 40) \cdot 3$ bytes, or about 3.66 gigabytes of raw data.

Restarting the software development process did offer the chance to include additional design goals that were not part of the previously used software:

- **Ease of use:** From the beginning, the software was intended for release to the public in order to allow anyone to freely experiment with slow feature analysis without the need to develop their own software first. In the multidisciplinary environment of neuroscience, ease of use was not only thought of as a "courtesy" towards new users, but as an actual necessity in order to enable scientists with less programming experience to still be able to perform their own SFA experiments. A thorough documentation (of both the source code and functionality), multiple examples of varying complexity, and a user-friendly graphical user interface (fig. 3) were set as additional goals. While the graphical user interface allows users to set up basic SFA experiments without the need of using the command line, a variety of small examples facilitates the transition to the full tool set, and a comprehensive documentation introduces users to working with the source code itself.
- **Modular architecture:** In contrast to the previously used framework, the new toolkit is designed to enable the implementation of a wide range of experimental protocols. The architecture is a series, or "pipeline," of modules that are executed sequentially. This compartmentalizes the various steps of a given simulation and thus enables different protocols without the need to change the actual source code. For instance, if a user wants to merely change the training procedure, he/she is only required to look up the documentation and parameters of that specific module. Similarly, advanced users who want to modify the source code itself can do so without having to worry about accidentally breaking other parts of the overall software. A modular pipeline design also introduces natural "break points," which allow users to run different steps of the overall process multiple times, again, without having to look into, or change, the source code. This allows for a multitude of different experimental protocols instead of merely variations of the same, monolithic base procedure.
- **Parameterized common use-cases:** While different experimental protocols may differ substantially, there are numerous basic components that should be available right away, as well as readily modifiable. For example, the basic environment within which an experiment takes place is usually a rectangular

box or a circular arena; consequently, such commonly used options are available in a parameterized fashion. This includes on/off switches for frequently used restrictions, configurable templates for routinely used mazes – such as rectangles, T-mazes, n-arm mazes (fig. 4) – and a collection of all the relevant values that define the overall protocol (like the speed and momentum of the rat). While most of these options are available as command line parameters, the latter are stored within a single file (`util/setup.py`) that houses the less commonly used values that usually should not be changed. This includes graphical options, as well as the behavior of the virtual rat. For a complete overview of all available parameters, see appendix A.

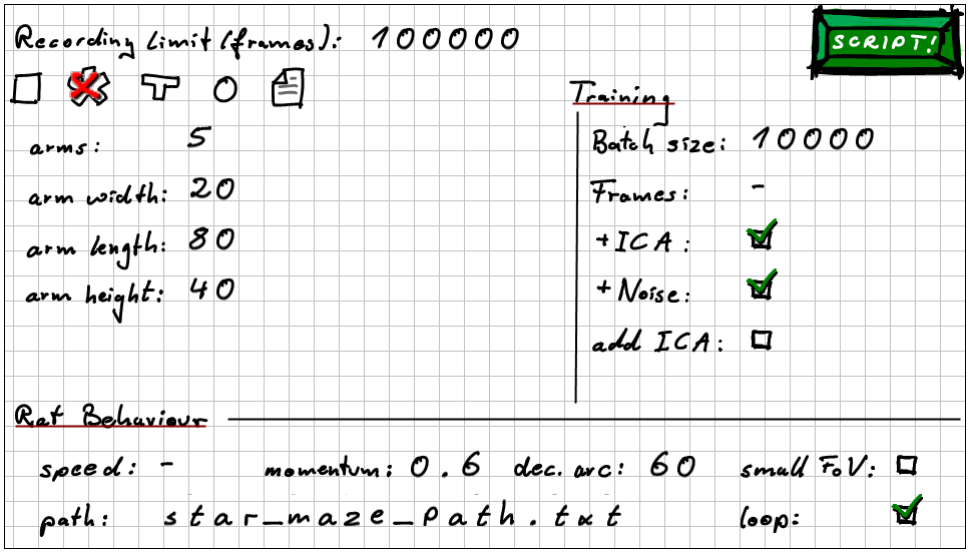

Figure 3: Graphical user interface of the software toolkit. Users can set basic parameters and use the "SCRIPT!" button to let the software generate a (Linux) shell script that runs the corresponding simulation and plots the resulting spatial representation.

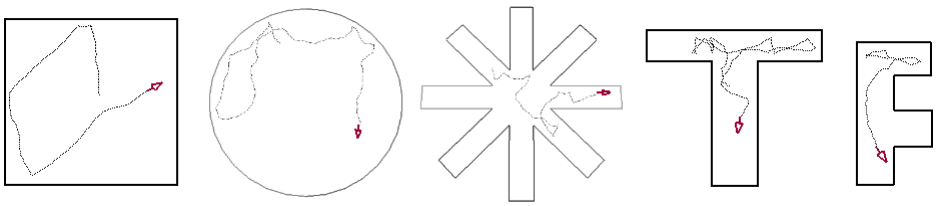

Figure 4: Simulation environments. Users may run experiments within the available and parameterized template environments (rectangular, circular, n-arm maze, and T-maze), or choose to define a free form enclosures (shown on the far right).

2.2.2 Implementation overview

The ratlab toolkit (Schönfeld and Wiskott, 2013) is implemented as a series of four Python (Oliphant, 2007) modules that are called in sequence: `ratlab.py`, `convert.py`, `train.py`, and `sample.py`. Each module executes one central component of the overall simulation: data creation, data accumulation, training the hierarchical SFA network, and sampling & plotting network activity respectively. In addition to these core modules, the toolkit also offers a small number of optional tools that can be used to further process the generated data (see appendix A).

Note: *This chapter includes small excerpts of the ratlab source code. For ease of parsing, such listings are shown as "simplified" versions compared to the actually used code. Simplifications include the exclusion of error checks and debug messages, changed variable names, and more verbose documentation. Third party software used during development is listed as follows:*

- ◇ `Python.....v2.7.11` *Programming language* (Rossum, 1995)
- ◇ `Numpy.....v1.10.1` *Package for scientific computing* (van der Walt, 2011)
- ◇ `MDP.....v3.3` *SFA and network structure implementation* (Zito et al., 2009)
- ◇ `SciPy.....v0.16.0` *Open source scientific tools* (Jones et al., 2001)
- ◇ `PyOpenGL.....v3.0.2` *OpenGL/GLUT graphics bindings for Python* (Fletcher, 2012)
- ◇ `Matplotlib....v1.5.0` *2D plotting library* (Hunter, 2007)
- ◇ `CUDA Toolkit....v7.5` *Graphics card computing API* (Nickolls et al., 2008)
- ◇ `CUBLAS.....v7.5` *Basic linear algebra subroutines for CUDA* (Barrachina et al., 2008)

2.2.3 Generating the data: `ratlab.py` and `convert.py`

The `ratlab.py` module performs the actual simulation and generates the training data for the hierarchical SFA network. It offers the most parameters of all the included software modules to enable users to implement a wide range of simulation protocols. It is responsible for creating the user-defined 3D environment and controlling the virtual rat agent throughout the experiment. The environment itself is rendered using OpenGL (Shreiner and Group, 2009) and the auxiliary OpenGL GLUT (Kilgard, 1996) library. For the maze itself, users may specify any architecture that is based on a 2D blueprint, i.e., does not include more than one level of elevation. Wall segments can be placed arbitrarily to allow users to add cue cards or similar features to the maze; textures may be chosen freely for any wall segment. The available parameters for the `ratlab.py` module control data

gathering, the 3D environment, and behaviour of the virtual agent; they are listed in the following:

- **Data gathering:** The number of time steps (i.e., the length of the training set); color of the image data (RGB color, greyscale, or both for comparison); running without recording (for demoing and testing purposes); running a wall-check (to determine correct alignment of all wall segments prior to performing the actual experiment).
- **Environment:** Use of a template environment (rectangular, circular, N-arm star maze, T-maze); use of a freeform environment (arbitrary wall segments specified in a text file); wall texturing (uniform or mixed); placement of optional obstacles.
- **Virtual rat behaviour:** Speed of the agent; decision arc (turning range limit per time step); momentum (the smoothness of the path); use of waypoints (the agent will run along a pre-determined path instead of random exploration); looping around waypoints (the agent will run back to the first waypoint after reaching the last one); movement bias (force the agent to move faster in a specified direction).

To generate the data, the environment is first set up according to user specifications. The virtual rat is placed at a randomly chosen starting position (if no path was specified) and begins to randomly traverse the environment. Since rats posses a very wide field of view (Hughes, 1979), the normal camera options of OpenGL cannot be used. Instead, at each position the scene is rendered 320 times into one pixel wide frame-slices while the camera scans the 320 degree field of view (lis. 1).

The accumulated slices form one full frame, which is written into an image file and stored to the hard disk. Afterwards the virtual rat takes the next step and generates a random direction that is weighted and added to the current velocity vector of the agent (lis. 2). If the virtual rat is following a predefined set of waypoints instead of randomly exploring the maze, it simply moves one step size towards the next waypoint. In order to avoid the animal following the exact same path on each leg of the specified route, a randomized offset is added during each step. This offset is perpendicular to the original movement direction of the agent, and becomes smaller

as it gets closer to each waypoint to keep the agent from overshooting it.

Listing 1: 320 degree field of view rendering (simplified)

```
x = int( window_width/2 - fov/2 )
for rat_dir in range( int(rat_direction-fov/2), int(rat_direction+fov/2)+1 ):
    # Setup single 1x40 pixel viewport (rendering space) at position (x,80)
    glViewport( x, 80, 1, 40 )
    gluPerspective( 40, 1.0/40, opengl_near_clip, opengl_far_clip )
    # Place camera & focus point and render the scene
    focus = [ rat_pos_x+cos(rat_dir), rat_pos_y+sin(rat_dir), rat_eye_level ]
    gluLookAt( rat_pos_x, rat_pos_y, rat_eye_level, # position of the rat
               focus[0], focus[1], focus[2],        # viewing direction of the rat
               0,0,1 )                              # vector defining "up"
    drawWorld()
    x += 1
```

Listing 2: Virtual rat random walk (simplified)

```
    random_direction = gaussianWhiteNoise2D()
    step  = current_velocity*momentum + random_direction*(1-momentum)
    step /= squareroot(dot(step,step)) # normalizing the step length
    step *= rat_speed_parameter        # setting user-defined speed
```

While `ratlab.py` is running, it displays a sketch of the environment that shows the state of the experiment and the trajectory of the virtual rat so far. This is mainly for the user to confirm that everything is working as intended and can be switched off to speed up the simulation at any time.[12] Once the experiment gets to the second to last time step this display is switched on again and a screenshot of the final state of the experiment is taken (e.g., fig. 5). This can be used to determine the success of the simulation and how evenly the full trajectory of the agent covers the overall environment. After a successful execution of `ratlab.py`, the user is provided with all visual image frames in png format, the full trajectory of the agent in an easy to parse text file, the screenshot of the final state of the simulation, and a setup file that documents all used parameters.

Once all training data in the form of image files was generated, it is accumulated within a single file. Since opening a file requires a system call, opening $100,000$ files will slow down the subsequent processing of the data. Thus all files are being stored within a single file that needs to be accessed only once by the operating

[12] OpenGL, unfortunately, does not allow off-screen rendering and thus the first person view cannot be disabled. Consequently, this part of the simulation cannot run as an invisible background process or be started remotely.

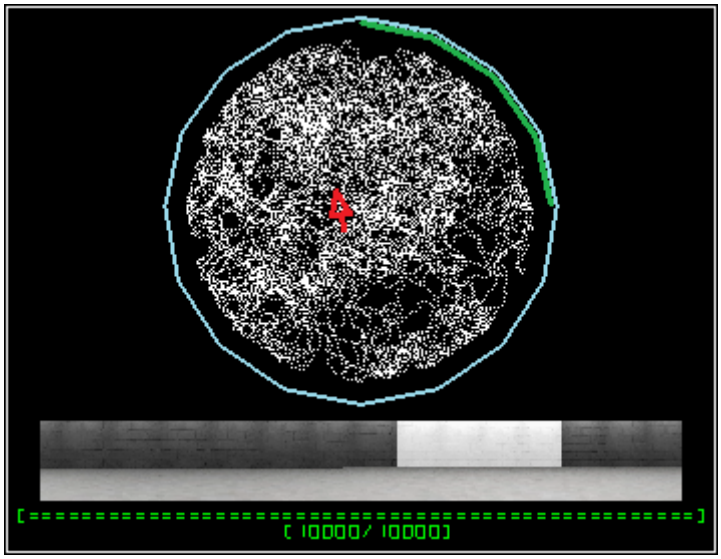

Figure 5: Successful simulation example. Once `ratlab.py` completes data creation a screenshot is made to document the final stage of the simulation. The top shows the borders of the environment (colors are chosen randomly by the software), with the trajectory of the agent depicted in white; a red arrow marks the final position of the agent. Below, the current visual input is seen, in this case a uniform wall with a single white cue card. At the bottom a progress bar shows how many frames have been captured so far, as well as the limit set by the user.

system. Multiple training stages may thus be performed without the need to repeatedly iterate through a large number of small files. This step is performed by the `convert.py` module, which due to the simplicity of its function does not offer any additional parameters.

2.2.4 Training the network: `train.py`

The `train.py` module constructs a hierarchical SFA network and uses the image data provided by the `ratlab.py` and `convert.py` modules as training data. Internally, the network data structure is implemented using the Modular Toolkit for Data Processing (MDP) (Zito et al., 2009), an open source Python library that provides a selection of of machine learning algorithms, including slow feature analysis. MDP also offers the functionality to link algorithm instances together and form hierarchical network structures such as the one employed here.

The network used by the ratlab toolkit is constructed from three SFA layers and one (optional) layer of sparse coding (fig. 6). SFA layers consist of a two dimensional

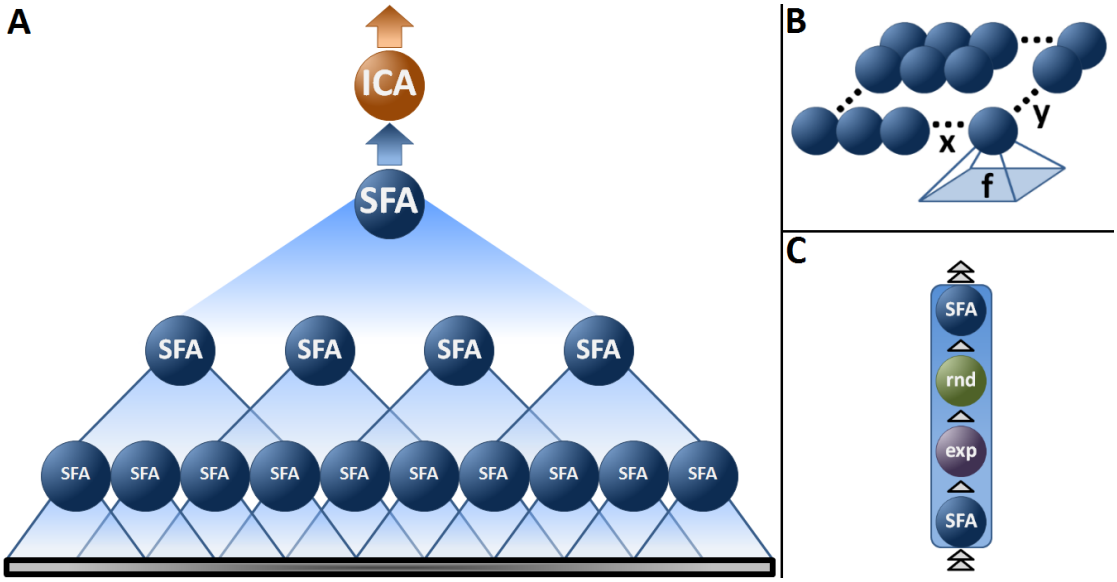

Figure 6: Architecture of the hierarchical SFA network. A: The full network consisting of three SFA layers and an ICA node that performs sparse coding. The grey line below the network represents an input image. **B:** Single SFA layer of width x and depth y. The node in the lower right is depicted with its receptive field f. **C:** Single processing node. All blue nodes in **A** and **B** consist of an MDP *flow* of up to four different sub-nodes, including quadratic expansion (*exp*) and (optional) white noise nodes (*rnd*) to add noise to the input.

array of SFA nodes, each of which receives the output of a set of nodes in the previous layer. Nodes in the first layer receive the color information of a section of the current input image instead. The parameters of the network architecture were chosen such that the input areas of any two adjacent SFA nodes will overlap by half. Note that each of the individual SFA nodes consists of a collection of sub-nodes: (1) an initial node implementing the SFA algorithm to reduce the dimensionality of the raw input, (2) a quadratic expansion node, (3) an optional node to add random noise to the data, and (4) a final SFA node to compute the features that will be relayed to the following layer. This combination was introduced in (Franzius et al., 2007) and will be referred to as a single SFA node throughout this document. The full list of parameters describing the overall network layout is shown in lis. 3.

These parameters specify an input dimensionality of 320 by 40 channels, corresponding to the 320 by 40 pixels of each image frame of the training set (a single channel is set to be either one or three dimensional, depending on whether color or greyscale images are being used). This data is processed by a grid of 63 by

9 SFA nodes, each of which reading a 10 by 8 pixel area of the incoming image. The output computed by this lower SFA layer is read by the upper SFA layer, a grid consisting of 8 by 2 SFA nodes, each of which processing the output of 14 by 6 input channels of the previous layer. The final SFA node integrates the full output of the upper SFA layer and computes 32 output signals – the slowly varying features that form the output of the overall hierarchical network. Omitted from this description are two additional layers of MDP switchboards, which are tasked with sorting the various input channels and implement the "wiring" in between the different processing layers, but otherwise serve no functional purpose within the network.

Listing 3: Parameters determining the structure of the hierarchical SFA network

```
raw_data_dim_x           =320 # frame dimensions and no. of SFA output signals
raw_data_dim_y           = 40
sfa_node_out_dim         = 32
sfa_lower_layer_nodes_x  = 63 # lower SFA node layer
sfa_lower_layer_nodes_y  =  9
sfa_lower_layer_field_x  = 10
sfa_lower_layer_field_y  =  8
sfa_lower_layer_node_out = 32
sfa_upper_layer_nodes_x  =  8 # upper SFA node layer
sfa_upper_layer_nodes_y  =  2
sfa_upper_layer_field_x  = 14
sfa_upper_layer_field_y  =  6
sfa_upper_layer_node_out = 32 # final SFA node
```

As mentioned above, the hierarchical network may also include an additional sparse coding layer. A pure SFA network is capable of computing a spatial representation when trained with the visual stream of a virtual rat. However, the individual slow features of such a representation will consist of slowly varying sine waves spread over the environment (fig. 2, p. 25). This clashes with the sparse representation observed in real life place field measurements, where each place cell is only active within a small area of the overall space. Thus, an additional layer is added to the pure SFA hierarchy in order to transform the SFA representation into a sparse code of the environment using independent component analysis (Hyvarinen, 1999). Note that, after spatial training, the activity of each SFA signal is invariant to the viewing direction in either case, which is an essential property of the spatial representations recorded in the hippocampus (fig. 7).

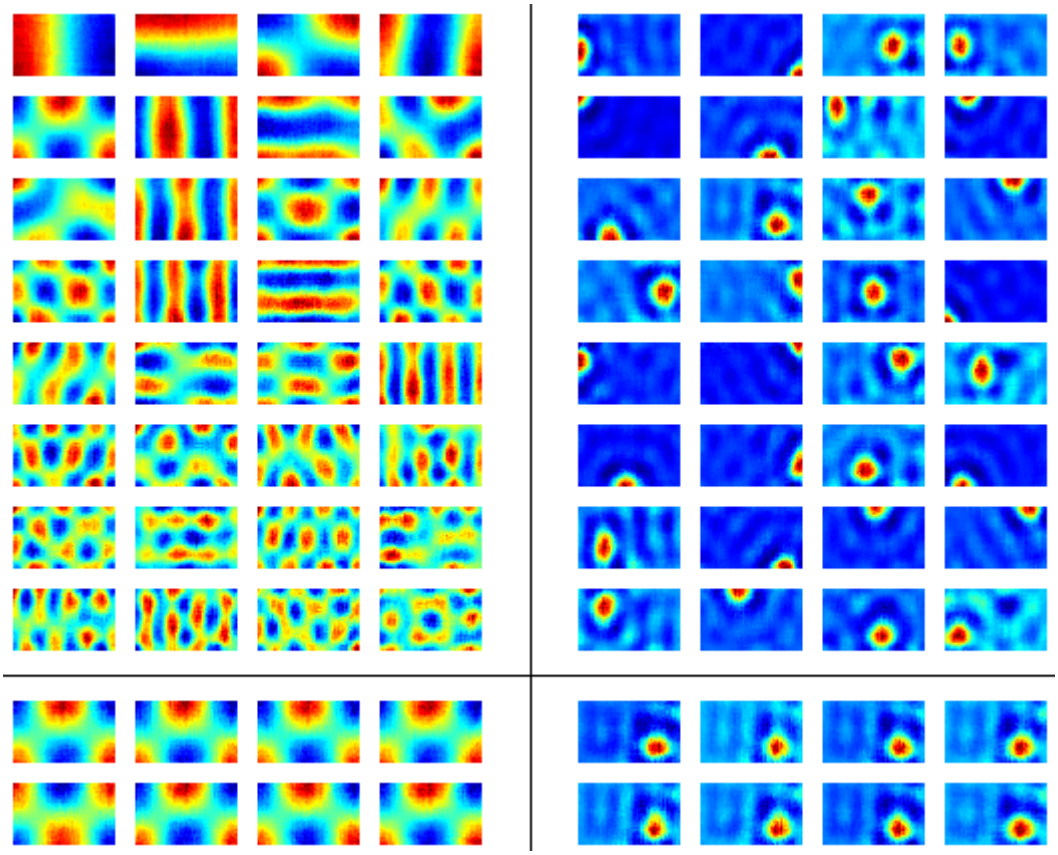

Figure 7: Spatial representation computed by hierarchical SFA. **Top Left:** Activity of 32 SFA output signals, sampled over a rectangular environment. Signals are automatically ordered by slowness (top left to bottom right) and approximate the multiplication of orthogonal sine waves (towards the bottom). **Top Right:** Activity of the 32 signals seen on the left, filtered through an additional layer of sparse coding. **Bottom:** Activity of one randomly chosen (#5 on the left, #10 on the right) signal while the agent faces eight different directions (n, ne, e, se, s, sw, w, nw).

Since the overall training procedure can be time consuming, the `train.py` module offers an alternative way to train a network. By default, a network is trained from the ground up, using the visual input stream from an agent that explores the environment in which the actual experiment takes place. However, this procedure can be broken up into two separate parts, using a *generic training* approach. In this case, only the lower two SFA layers (fig. 6) are trained on a large data set – usually $100,000$ time steps – that consist of images recorded from a multitude of different environments (fig. 10). For any specific experiment such a network can be used to restrict the training phase to the two remaining upper layers (SFA and ICA). This not only dramatically reduces the time to train a network, but

also moves the model one step closer to being biologically plausible: The visual cortex is not reset and re-trained whenever an animal explores a new environment and a model that constructs a new spatial representation on top of, and with the help of, a multi-purpose-trained foundation is much more plausible and intuitive. All the experimental results shown throughout this work are based on networks that resulted from generic training; the used generic network files are part of the distributed software package.

2.2.5 Sampling the network: `sample.py`

The `sample.py` module uses the network trained by the `train.py` module, as well as the world-description provided by the `ratlab.py` module, to reconstruct the environment used during the simulation and sample activity of the network. To do so, the camera is placed at every valid integer position of the environment and snapshots are taken while the camera points in the directions chosen by the user (n, ne, e, se, s, sw, w, and/or nw). These snapshots are fed into the network and the response is recorded: a 32 dimensional vector storing the activity of the 32 slowest signals given a particular visual input, as computed by the trained hierarchical network. This raw data is archived on disk and used to compute the average network activity at each point over all sampled directions. Usually this means all eight available directions, but may also be restricted to a subset – such as northeast, east, and southeast for example – to average over the activity while the rat is facing in a general (or specific) direction. Once all raw values and their averages are computed, the `sample.py` module plots the data (fig. 7) along the commonly used jet color scale (from dark blue for low values over cyan and yellow to deep red for high values). Note that this sampling process is different from the way real cell activity is sampled in actual animals. As mentioned above, a more biologically plausible process can be emulated, but as it merely delivers the same result with a lower resolution, it is not part of the default functionality.

Besides spatial sampling the `samply.py` module also offers directional sampling. This can be selected by adding the `dir` parameter when calling the module, and samples network activity not as a function of space, but as a function of view direction. To do so, the camera is positioned throughout the environment in regular intervals – not every available integer position, but instead every n-th position –

and rotated by 360 degrees, taking one snapshot of the environment per angle. Once this data is collected the activity values of each output signal are averaged over the individual viewing directions. This yields 360 values per signal, which represent the average activity of each signal while the camera is pointed in one specific direction, regardless of the position of the agent in space. Fig. 8 shows several examples of direction sensitive SFA cells sampled from a network trained in a small rectangular environment. Note that not all network signals will code for one distinct direction. In the small number of experiments that have been performed about head direction in this work, about one third of the signals showed a clear preference for a particular direction, while the remaining signals show either inconsistent activity over the sampled space, or react to multiple distinct locations and/or a significantly broader range of directional activity (fig. 8). This does not contradict experimental results, however, as similar firing behavior has been measured in real life head direction cells (Giocomo et al., 2014).

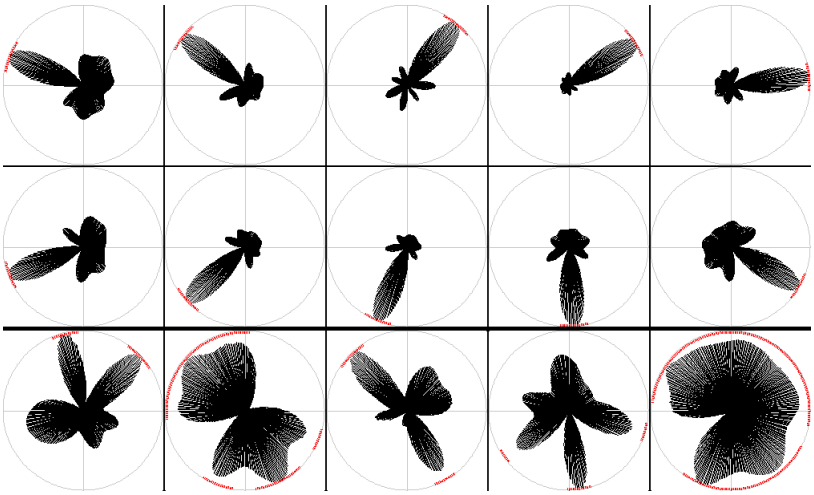

Figure 8: SFA computed head direction signals. When trained with the appropriate input data, SFA signals may code for directionality instead of space. Polar plots indicate the strength of output signals sampled over a 360 degree arc and averaged over the environment. **Top rows:** Directional signals that match head direction cell activity as it is commonly reported in the experimental literature. **Bottom row:** Examples of different classes of directional cells, including more than one preferred direction and broader arcs of preferred activity. Such cells seem to be observed in real life recordings as well but are often excluded in publications (undocumented communications).

2.2.6 Performance and bottlenecks

While putting together a hierarchical SFA network is a straightforward task, several additional steps are necessary to achieve the required performance. Running a simulation should not render the used desktop computer incapable of performing other tasks. In fact, it is desirable to allow for several experiments to run in parallel, and the memory and CPU requirements of the software should not prohibit that. During development three bottlenecks were identified and addressed: The memory requirement of a large number of SFA node data structures; the handling of a large set of visual input data; and the processing cost of training a multi-layered network with a large, high dimensional data set.

SFA node memory consumption. A network such as the one used throughout this work contains a large number of SFA nodes, each of which requires a small, but not insignificant, amount of memory. Due to this, the two lower SFA layers of the network (fig. 6) are not actually built from individual SFA nodes, but from two SFA parent nodes that are cloned across the two layers respectively. These two parent nodes, one per SFA layer, thus pass a number of individual training phases, one for each of the cloned SFA node instances. This drastically reduces memory consumption, as well as the required time to train a full network instance. Cloning parent nodes instead of using individual instances can be done because the statistics of the input can be assumed to be the same across the whole visual field; we do not expect visual features that only ever appear at the same position of the agent's field of view, irrespective of where it looks. (Even though the software used throughout this work was developed from the ground up, this particular optimization was already described and used in other works, such as (Franzius et al., 2007).)

Training data memory consumption. Since training the network requires the repeated processing of a large amount of data, the internally used data structures have to support fast access without blocking all available resources of the machine performing the experiment. Further, even though the SFA algorithm itself poses no limit on the size of the training data, the time it takes to train a network increases rapidly with additional data: the computational complexity of SFA is of order $O(NI^2 + I^3)$, with N denoting the number of samples, and I the input

dimensionality (Escalante-B and Wiskott, 2011). Thus the full training set often is too large to be fed into the network all at once and needs to be split into several batches. These have to support repeated access in order to train the different layers of the network. To address these issues, all training data is first read into a memory map, a Python data structure provided by the commonly used Numpy library (van der Walt, 2011). The purpose of this structure is to store large memory objects partly on the hard disk instead of entirely within the system's RAM. The memory map then masks the larger access delay of reading from the disk by caching the actually used parts of the data within the efficiently accessible RAM modules. This works best if the data can be accessed sequentially and in a uniform pattern, which makes memory maps ideally suited to be used in this case. In order to interface with the network built from MDP components, Python *iterators* need to be used. However, iterator are throwaway data structures that can only be used once, and re-reading the full data set from disc to re-use an iterator is far too time consuming to be an option. Thus a dedicated class is required to encapsulate re-usable iterators. This class further further needs to make use of the Python `yield` statement to force its iterators to return user-defined batches of the data instead of only a single training datum (lis. 4).

Listing 4: Class to iterate training data in batches without requiring it to be re-read from disc

```
class getReusableDataSlicer():
        def __init__( self, data, batch_size ):
                self.data = data
                self.batch_size = batch_size
        def __iter__( self ):
                for i in range( int(self.data.shape[0]/self.batch_size) ):
                        yield self.data[ i*self.batch_size:(i+1)*self.batch_size, ]

#--------------------------------[        use:        ]----------------------------#

training_set = []               # One data slicer per to-be-trained network layer.
for i in range( len(network) ):  # Length of the network == number of layers.
        training_set.append( getReusableDataSlicer(data,batch_size) )
network.train( training_set )
```

Large scale training and CUDA integration. As noted above, training hierarchical SFA networks is a CPU intensive computation that scales with both the dimensionality of the training data as well as the number of time steps contained within it. One of the expensive components at the heart of SFA is the maintenance

and update of two covariance matrices (cf. equations (6) and (7)), which is implemented in the form of matrix multiplications – a notoriously slow operation when performed by a only a single, or small number of CPU threads. At the same time, however, matrix multiplication is an operation referred to as being "embarrassingly parallel:" It naturally scales with the number of processing threads since the individual components of the result can be computed independently from each other. With this operation having a large impact on overall SFA training performance, it is transferred from the central processing unit (CPU) to the graphics processing unit (GPU), i.e., the graphics card of the computer.

Common computer graphics operations are highly parallelizable and, consequently, modern graphics cards are developed as co-processors that perform such computations that benefit greatly from parallelization. Their onboard GPUs feature an increasingly high number of small processing units which have lately been made accessible to developers. High performance graphics cards can thus be utilized as co-processors and used to handle computations that are suited to their hardware specifications. The ratlab software includes a modification for the MDP library that makes it transfer the training data to the graphics card where the aforementioned covariance update will be computed. The desktop computer used to run the simulations presented throughout this work housed a Nvidia Geforce GTX 580 graphics card, which features 512 so-called "streaming processors" – small processing units optimized for use in SIMD parellelization (singe instruction multiple data). The software required to access these processors is known as CUDA (Compute Unified Device Architecture) (Nickolls et al., 2008). The different parts of the implemented CUDA integration are presented in fig. 9, and operate as follows:

- `ratlab/train.py`: To the ratlab toolkit the usage of the GPU instead of the CPU is completely transparent, and simulations can be performed even without a CUDA enabled graphics card. The software merely generates the data and hands it over to the MDP `network.train()` function.
- `mdp/utils/covariance.py`: This file contains the source code of the covariance matrix data structure and is already part of MDP. To modify the matrix operation of its `update()` function, the Python standard `ctypes` library is used to load a shared C library that functions as an intermediate between the

Python code running on the CPU and the C code running on the GPU.

- `libmodule.so`: This file is a shared library compiled from C code. It sets up the access to the graphics card and, upon receiving a pointer reference to the data, facilitates the actual data transfer to and from the graphics hardware. Once the data is transferred, the CUDA library CUBLAS (Barrachina et al., 2008) is used to perform the actual matrix operation.

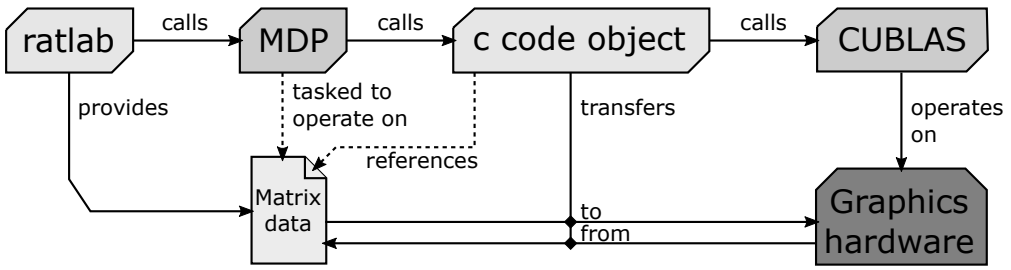

Figure 9: CUDA integration flowchart. The CUDA-enabled graphics card provides a co-processor that is used to outsource the covariance matrix update that is part of the slow feature analysis algorithm.

No thorough benchmarking was performed to accurately determine the precise speedup of this optimization. However, to provide a sense of scale, matrix operations that would take around 12 minutes on an Intel Core i7-3770 Quadcore CPU would take less than a second when performed on the GPU (this includes the time to transfer the data). A complete experiment using around 8000 to 10000 time steps can be run within 60 to 90 minutes, from the initial generating of the data to the finished plots of the results. (Before CUDA integration, such simulations took several hours and were set to run overnight.)

2.2.7 Replicating real life experiments

To replicate real life experiments, the layout and details of their respective apparatuses was copied as closely as the software would allow it. This includes the geometric design, the colors and textures of the surrounding walls, the use of cue cards, and whether or not curtains were used to prevent animals from orienting themselves via the visual cues of the surroundings environment. The individual networks trained during the various simulations presented in chapter 3 all share a common core; their initial two SFA layers were subjected to the generic training

procedure introduced above. The generic training set was created from the visual data resulting from exploring five different environments (fig. 10) and used to train the base layers of a hierarchical network while the final SFA and ICA were left untrained. The upper layers were trained only with the data taken from the different simulations presented throughout chapter 3.

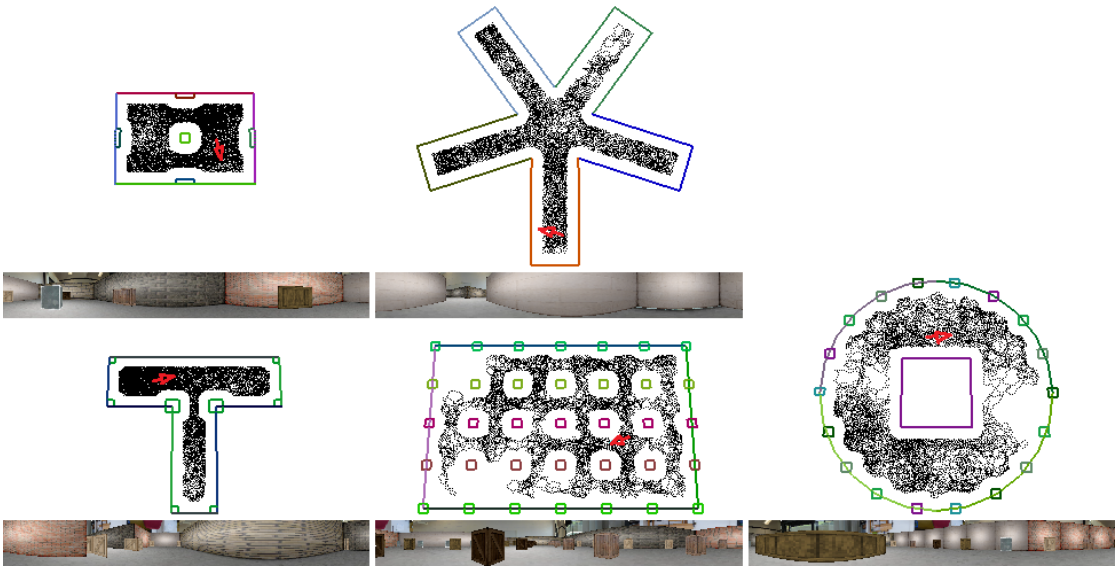

Figure 10: Generic training data. Five different environments were explored, and the recorded image data was used to train the lower two levels of a hierarchical SFA network. Depicted are the final exploration states with the trajectory of the agent, its final position (red arrows), and final frame of the training data for each of the five segments. The full training set consists of $100,000$ image frames, $20,000$ per setting; the network trained in this manner was used in each of the simulations presented below.

All results presented here were taken from a single run per simulation, of which each yielded 32 output signals. From these, the results most similar to the reported experimental results were selected for a direct comparison. Where necessary, additional results are presented to show qualitative differences between SFA activity and real life measurements, including unusual or "misbehaving" signals. All results are plotted on the dark blue to dark red jet scale, which is commonly used to depict place cell activities throughout the experimental literature. Where experimental results are presented in a different format, SFA results are shown in both jet scale format for coherency, as well as the individual representation chosen by the authors of the respective work. During the simulations the agent explored each environment

at a constant speed of 20 cm per second. This corresponds to the average speed observed in experimental trials (Sijie Zhang; personal communication, November 11th, 2013) but should be regarded as an approximation since foraging speed is known to vary between individual animals and, while rarely below 15 cm per second, may reach up to 40 cm per second or more. To compare the covered space of real and simulated rats, fig. 11 shows a side by side comparison of an example trajectory for each.

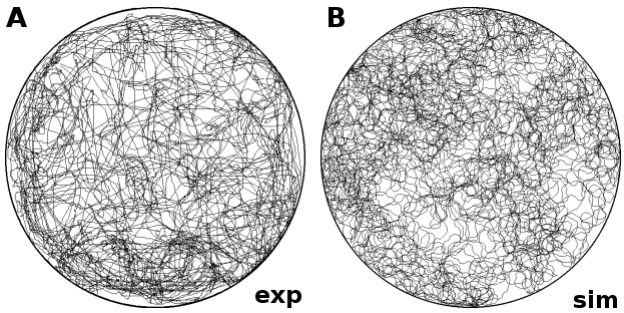

Figure 11: Experimental and simulation trajectories. A: Trajectory of a simulated rat exploring a circular environment (radius 80 cm) for 10 min of simulated time. The speed of the rat is maintained at a constant 20 cm per second. **B:** Trajectory of a real rat traversing a circular environment of the same size for 10 min of real time. The speed of the animal averages to about 20 cm per second.

To quantify the results produced by the network, both *directional consistency* and *spatial consistency* were analyzed. Directional consistency was computed by sampling the full environment with a fixed head direction (n, ne, e, se, s, sw, w, nw) and computing the average correlation of those fixed-direction plots with the average activity over all views. The data is presented in histograms that show how many signals were firing in a consistent manner independent of the viewing direction. Spatial consistency was computed by counting the number of distinct firing areas for each signal over all trials. The number of active fields was determined by segmenting firing activity into connected areas of activity above 50% of the signal's maximum activity. Activity fields of 25 pixels or less - equivalent of an area of 5 by 5 cm - were discarded. The data is presented in histograms that show how many cells featured a specific number of firing fields (including zero).

Note: *For a full list of the available parameters of the individual software modules, as well as detailed examples, and an overview of additional tools to process computed results, please refer to appendix A.*

3 Results

The following chapter will provide an overview the spatial representations that are developed by hierarchical SFA and briefly compare them with the recordings reported in the experimental literature (section 3.1). Afterwards, the reaction of SFA-based spatial representations to a range of environmental manipulations is compared with that of the spatial encoding found in real animals. Modifications include the position and presence of visual cues, the systematic restriction of movement, and changes in the layout of the environment itself (section 3.2). The chapter concludes with presenting a new architecture that embeds hierarchical SFA as one of many possible modules within a functional loop. This architecture is shown to be capable of handling the information from different input modalities and perform both a basic episodic memory task and form a spatial representation (section 3.3).

3.1 Development of SFA-based spatial representations

To show how the spatial encoding of hierarchical SFA develops over time a virtual rat was set to explore a 60×60 cm box for eight minutes of simulation time. Three of the walls were colored in a uniform dark grey, while the fourth wall was covered with a white cue card; walls were high enough to prevent the agent from peeking over them. The activity of the individual SFA signals was sampled over the complete environment after 30 sec and 1, 2, 4, 8 min of exploration time respectively. Since SFA cannot be paused, the simulation went on for 8 min simulation time and five copies of the generically trained base network were set to finish their training using increasingly larger batches of the full training set.

The results of this process are shown in fig. 12; individual place fields can be seen to form as early as two minutes into the simulation. Note that even though the output signals of hierarchical SFA are ordered by slowness, this order is lost once the signals are filtered through the topmost ICA layer. As such, the subplots in fig. 12 are matched by hand in order to provide the depicted progression of the spatial representation.

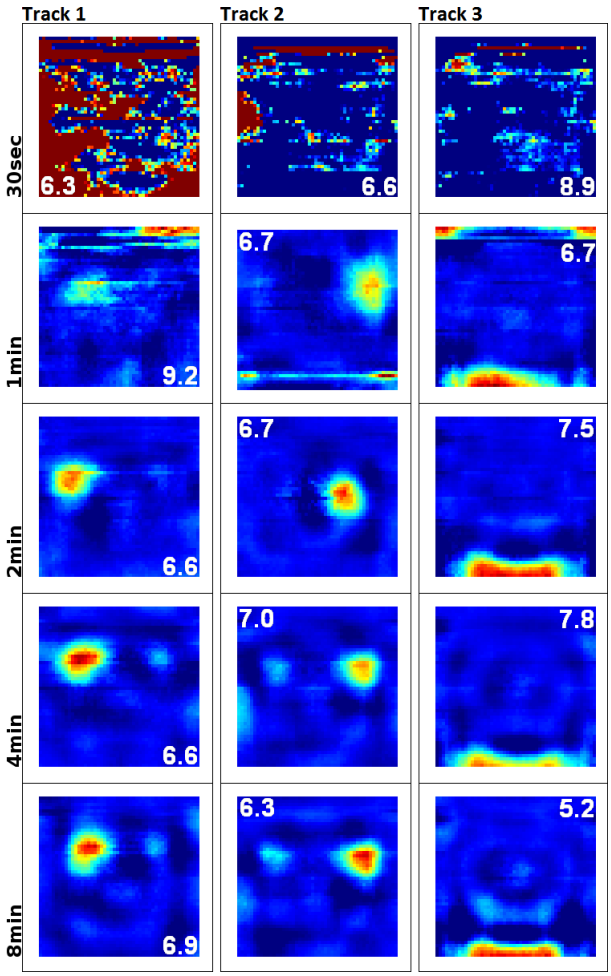

Figure 12: Development of an SFA-based spatial representation. Five SFA networks were trained with differently sized batches of the same overall training set. Each row depicts the activity of three signals for the different networks, ordered by the size of the training data (columns). Peak activity of the individual signals is given in the corner of each plot.

In a second simulation, the virtual agent explores a circular (radius 40 cm) environment, again for 8 min of simulation time. As before, walls are colored in a uniform grey and are high enough to prevent the agent from looking over them. The northern quadrant of the wall was covered with a wide cue card to provide a sense of direction to the agent. This time, spatial activity is sampled only *after* the network is trained with the full training sequence and a spatial representation is fully established. The network is also not sampled over the full environment, but instead only along the original trajectory of the first 30 sec and 1,2,4,8 min of exploration. In addition, the sampling data is processed similarly to how recordings

from real animals are handled: The 32 output values of the model at any given time are treated as the probability for the "cell" to fire. The generated "spikes" are collected in the bins of a regular grid covering the environment and smoothed over. Note that the signals did not have to be matched by hand as before, since, in this case, the training of the underlying network was not split.

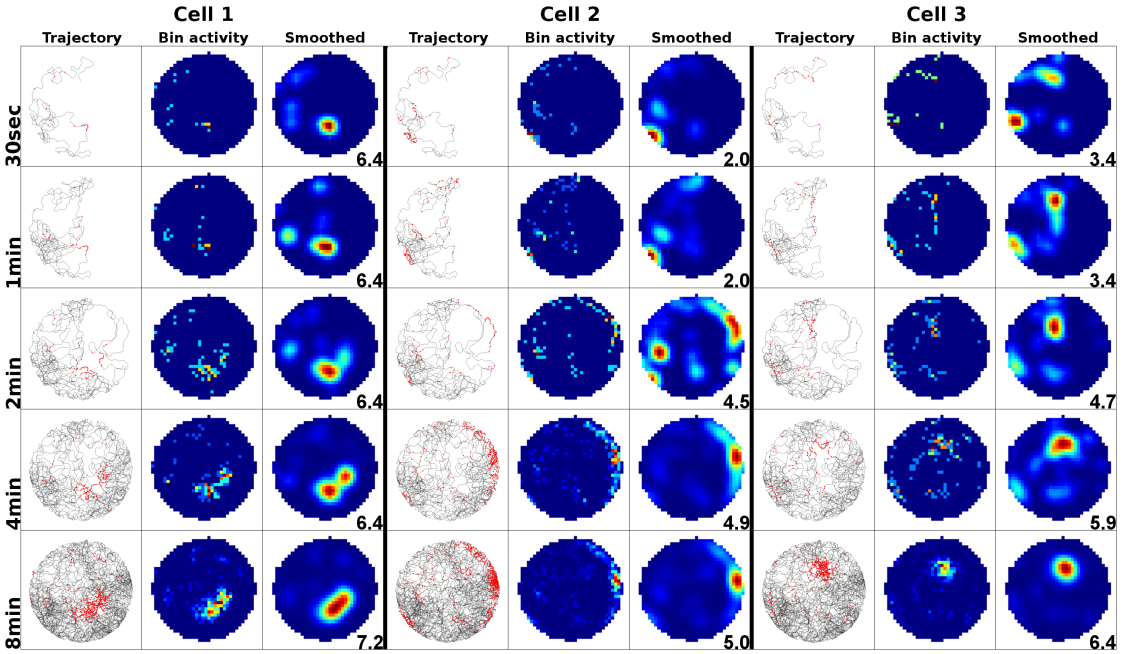

Figure 13: Sampling of SFA-based "spikes" during exploration. Spatial activity of a fully trained network during the first 30 sec and 1,2,4,8 min of exploration. Collections 1, 2, and 3 show the activity over time (rows) of one cell each. For each sampling phase the simulated spikes are shown in the "Trajectory" column; accumulated spikes per spatial bin in the "Bin activity" column, and the smoothed out activity in the "Smoothed" column. Peak firing activity for each cell after the different sampling phases is shown in the lower right hand corner of each row of plots.

As can be seen in fig. 13, this process yields spatial activity that, over time, seems to form moving and merging fields of activity. This behavior is commonly reported throughout the experimental literature. Spatial activity is usually considered stabilized after 8-10 minutes of exploration time (Wilson and McNaughton, 1993). Note that the network itself did not change in between the different sampling runs, and the internal spatial representation is stable from the beginning and remains identical throughout all sampling depicted in fig. 13. The apparent moving and

merging of place fields is purely a result of the area covered by the agent during the time interval considered for sampling.

3.2 Comparison of experimental and simulation results

To study the ability of the slowness principle to capture the properties of the hippocampal space code, six experiments were chosen and replicated using a hierarchical SFA network. Experiments were selected to cover a range of different properties that are commonly observed when recording place cells within the hippocampus. Further, in order to focus on the *spatial* aspect of the hippocampal code, as well as to keep in line with the capabilities of the model, experiments were required to not include any additional task other than the basic exploration and mapping of an unknown space. No sequences, timings, or object interactions were part of the experimental protocols. The list of chosen experiments is as follows:

- **Cue card binding:** A basic experiment that demonstrates how spatial representations bind to salient visual cues in the environment (Knierim et al., 1995). Rats explore a cylindrical, uniformly colored environment that features a white cue card covering a 90° arc of the wall. Rotation of the cue card is shown to rotate the place cell representation of the animals as well.

- **Cue card removal:** To examine the effects of removing salient visual cues from the environment, animals are placed within a closed rectangular box that includes three cue cards of different dimensions mounted on three different walls (Hetherington and Shapiro, 1997). After familiarization single cue cards are removed and place cells are shown to adapt by reducing their activity depending on the spatial relation of their place field to the removed cue.

- **Directional firing in the linear track:** Place fields in the open field usually fire irrespective of the direction animals are facing. This, however, changes when navigating narrow corridors. Directional dependent firing has been demonstrated in (McNaughton et al., 1983) and was also shown to persist when animals navigate a linear track within a virtual environment (Dombeck et al., 2010).

- **Directional firing in the open field:** In (Markus et al., 1995) it is shown that direction dependent firing can also be recorded from animals navigating

an open field. Animals are trained to follow a distinct, rhombus-shaped path within a circular environment and subsequently are shown to develop a spatial representation that would switch if the path was traversed in reverse.

- **Morphing between two familiar environments:** Instead of manipulating elements within a familiar environment, it is also possible to modify said environment itself. In (Wills et al., 2005) animals first explore both a rectangular and a circular environment. Afterwards, they are placed in four new environments, each of which an intermediate configuration between the original rectangular and circular environments. Place cells are measured while animals adapt their spatial representation to these "morphed" stages between the two familiar contexts.

- **Stretching of a familiar environment:** A second experiment to examine the effects of manipulating an established context is reported in (O'Keefe and Burgess, 1996). Animals explore four different versions of a rectangular environment that are stretched versions of each other. Place fields are shown to not react uniformly to the changes, and either stretch along the environment, stay in their relative position within the overall space, or remap in other ways, including becoming silent.

3.2.1 Manipulation and removal of visual cues

The internal representation of space in the hippocampus is linked to salient visual cues found in both the immediate (*local cues*), as well as the distant environment (*global cues*). This is exemplified by the experiment presented in (Knierim et al., 1995), where animals explore a uniformly colored, circular environment without any distinct landmarks except for a large bright cue card covering a 90° arc of the surrounding walls. Once animals establish stable place fields within that environment the cue card is rotated by 90° and recordings show that the animals' internal representation follows that rotation and place fields relocate to keep their relative positions in regards to the visual cue.

To replicate this experiment in a simulation, a circular, uniformly textured arena with a radius of 76 cm was constructed and explored by a virtual agent for the in-simulation equivalent of 8 min. As in the real life experiment, a white cue card

was placed within the arena and a black curtain background was used to prevent the agent to link place fields to cues outside the circular apparatus. A hierarchical SFA network was trained with the data collected during exploration and sampled in both the original setup and the rotated variant.

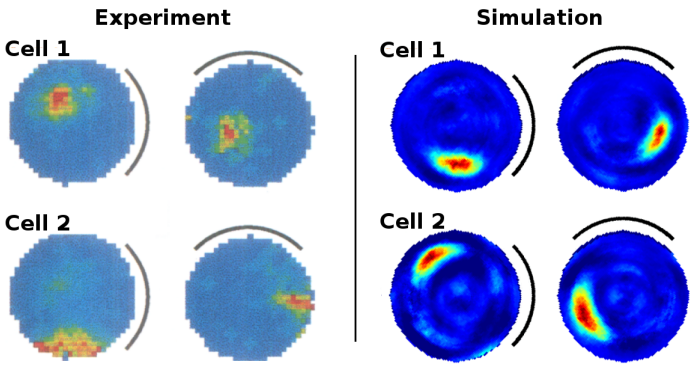

Figure 14: Cue card rotation. Left: Activity of two place cells before and after the rotation of a cue card within an otherwise featureless environment. [Reprinted by permission from the Journal of Neuroscience (Knierim et al., 1995)] **Right:** Activity of two output signals computed by hierarchical SFA and sampled over a circular arena modeled after the real life experiment.

The results of the real life experiment and the virtual simulation are shown in fig. 14. In both cases the place fields rotated to keep their relative position to the cue card. When analyzing the directional consistency of the population, all signals are clustered around 1.0 (fig. 15); spatial consistency is equally high, as all but one signal feature only a single firing field over the course of the simulation.

To further examine the influence of visual cues on animals' spatial representation, their removal after familiarization was investigated. If place fields are closely linked to the position of distinct cues, the question arises how much the internal representation relies on such cues and how it reacts to their removal. In (Hetherington and Shapiro, 1997) animals explored a rectangular space that featured three cue cards of different size mounted on three of the four walls. After the emergence of stable place fields individual cue cards were removed and the reaction of the spatial representation to this introduced change was measured. Recordings show that, while the overall representation appeared to remain stable, cells did react by rate remapping. That is, while their respective fields stayed at the same locations, their firing activity decreased compared to the original environment

featuring the full set of cues. The authors also report that this change in firing rate is proportionate to the distance between place fields and the missing cue card, with fields closer to a removed cue exhibiting a greater reduction in firing rate.

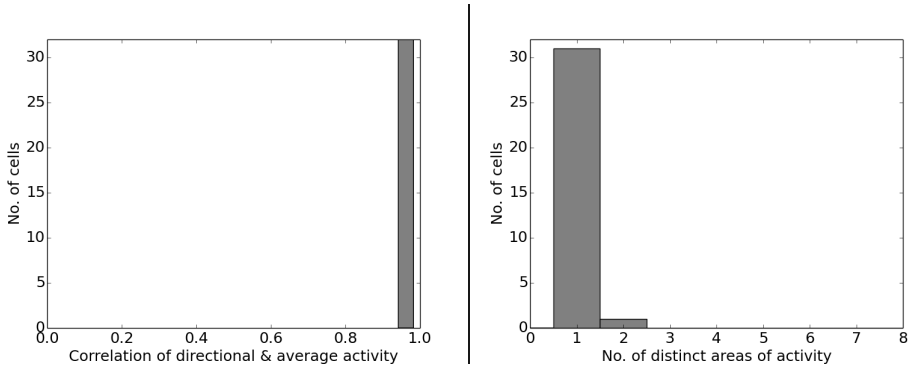

Figure 15: Quantitative analysis for cue card rotation. Left: Directional consistency of the SFA-computed signals after rotating the only available visual cue. **Right:** Spatial consistency of the signal population after rotation.

In simulation, this experiment was replicated by constructing a uniformly dark grey textured square environment with a wall length of 83cm and the addition of three white cue cards of length 12, 40, and 69 cm respectively. The real life apparatus is topped by a wooden ceiling, which was emulated by an increase in wall height that forbids the agent from peeking over the borders of the enclosure. The space was explored for the equivalent of 12 min, after which the arena was sampled several times with different subsets of cue cards removed for each repetition. Note that this includes the removal of more than one cue card at the same time, which, unfortunately, was not part of the original study.

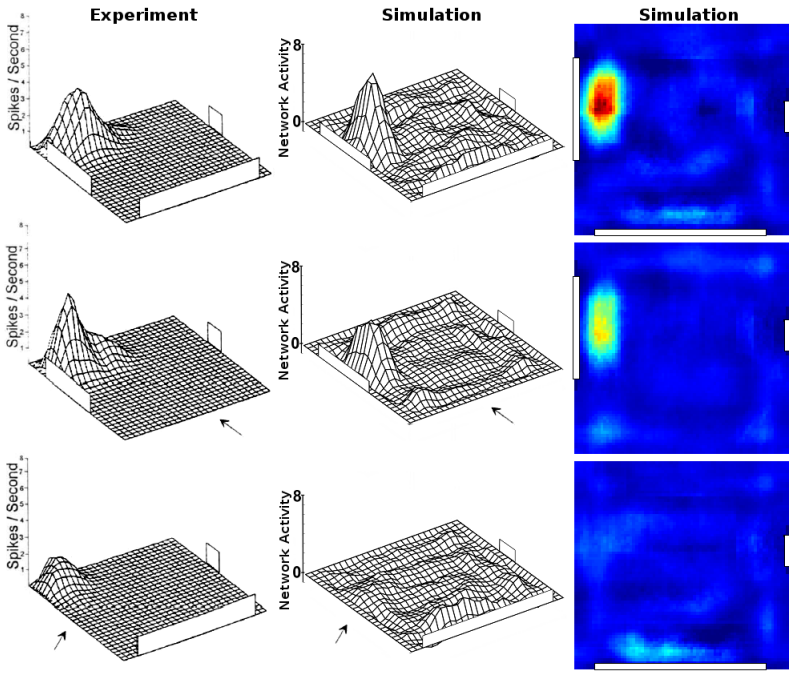

Figure 16: Cue card removal. Left column: Activity of a single place cell within a familiar environment (top); after the removal of a large/distant cue card (center); and after the removal of a smaller/nearby cue card (bottom). [Reprinted by permission from Behavioral Neuroscience (Hetherington and Shapiro, 1997)] **Middle column:** Activity of the closest matching SFA-computed output signal in the simulation of the real life experiment. **Right column:** Activity from the middle column plotted in the more commonly used jet-scale heat map format for coherence.

A comparison of the experimental and simulation results is shown in fig. 16, where a single cell is shown to reduce its firing rate in different amounts depending on whether the removed cue used to be located right next to the firing field or along a more distant wall segment. Fig. 17 shows a selection of eight cells that are representative of the overall population of 32 and depicts their reaction to the removal of one or two cue cards. The SFA-based spatial representation, for the most part, stayed consistent after the removal of a single cue card. As reported in (Hetherington and Shapiro, 1997) different cue cards affected different place fields more than others, depending on their relative position. Firing fields directly next to a missing cue card went almost completely silent, while fields further away were affected to varying degree. Removal of the smallest cue card affected the overall population the least, while removal of the largest cue card – covering almost all of the south wall – significantly affected almost all of the firing fields. Removal of the medium sized cue

card lead to a distinctive loss in firing activity for all nearby cells, while place fields located further away were influenced to a considerably less extent. Removal of two cue cards at the same time turned most place fields silent – but notably did not lead to the representation breaking down, i.e., firing neither degenerated into random activity, nor did the representation lose its smooth transitions between areas of higher and lower activity. Furthermore, place fields next to a single remaining cue card retained their activity. Notably, fields in the center of the environment stayed largely consistent throughout all trials, suggesting that place fields at geometrically distinct locations are less anchored to visual landmarks and more tuned to the overall symmetry of the visual perception at their respective locations. This effect can be seen throughout the other simulations presented in this chapter as well.

When analyzing the overall population, directional consistency behaved as expected: Correlation values were high for the original environment as well as those trials that only saw the removal of one of the smaller cue cards (fig. 18). The more the individual firing activity went down because of cue card removal, the lower the overall directional consistency of the whole population. Spatial consistency values reacted in a similar way: Spatial activity for most signals was confined to a single field of activity, which was lost depending on which and/or how many cues were removed (fig. 19). In some cases the spatial activity split into two areas, which usually meant two symmetrically opposed firing fields.

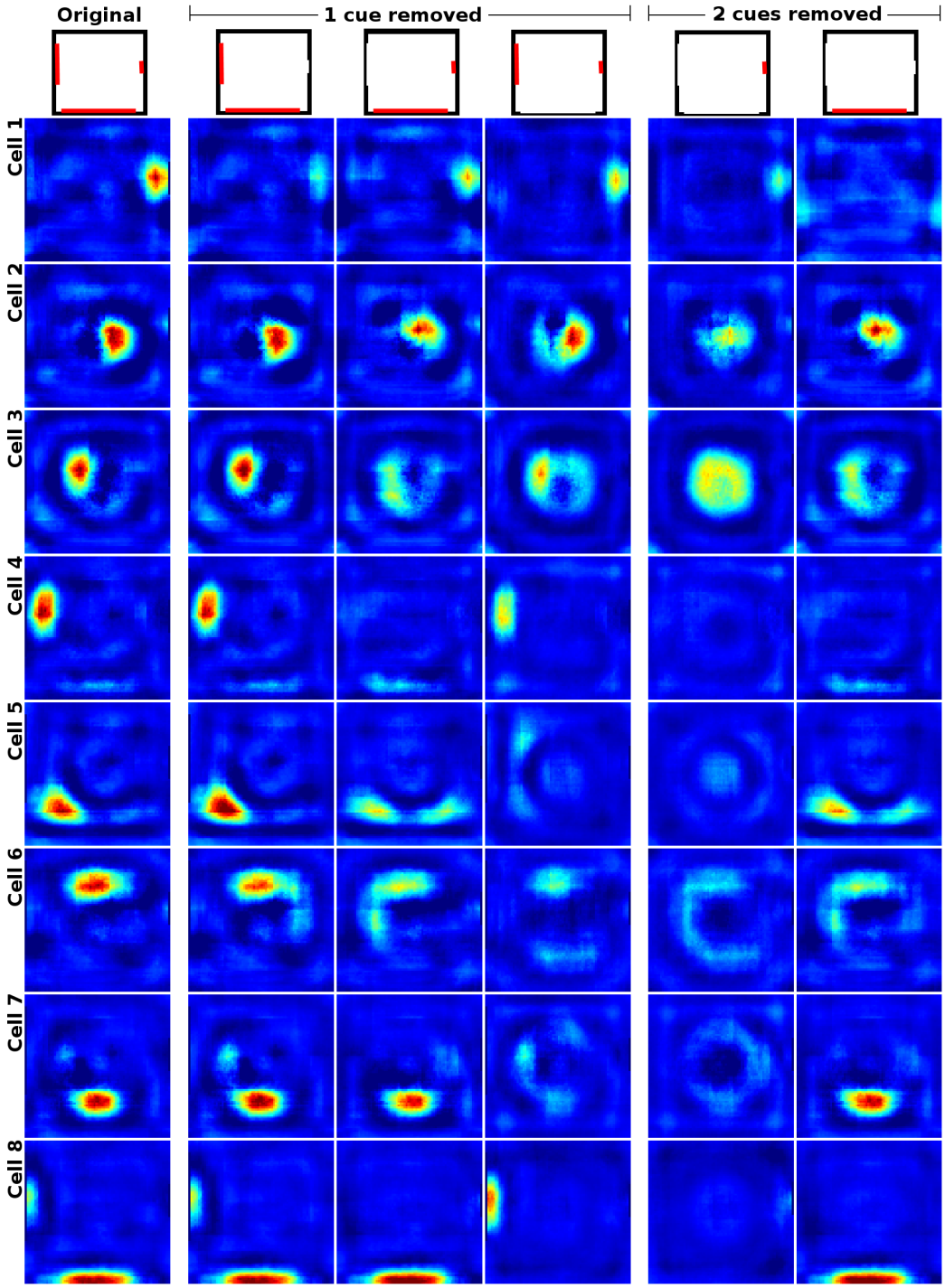

Figure 17: Removal of multiple cue cards. An SFA-computed spatial representation reacts to the removal of multiple cue cards. Rows show the activity of eight output signals sampled over the original environment (leftmost column) and after the removal of either a single (middle columns) or multiple (two rightmost columns) cue cards.

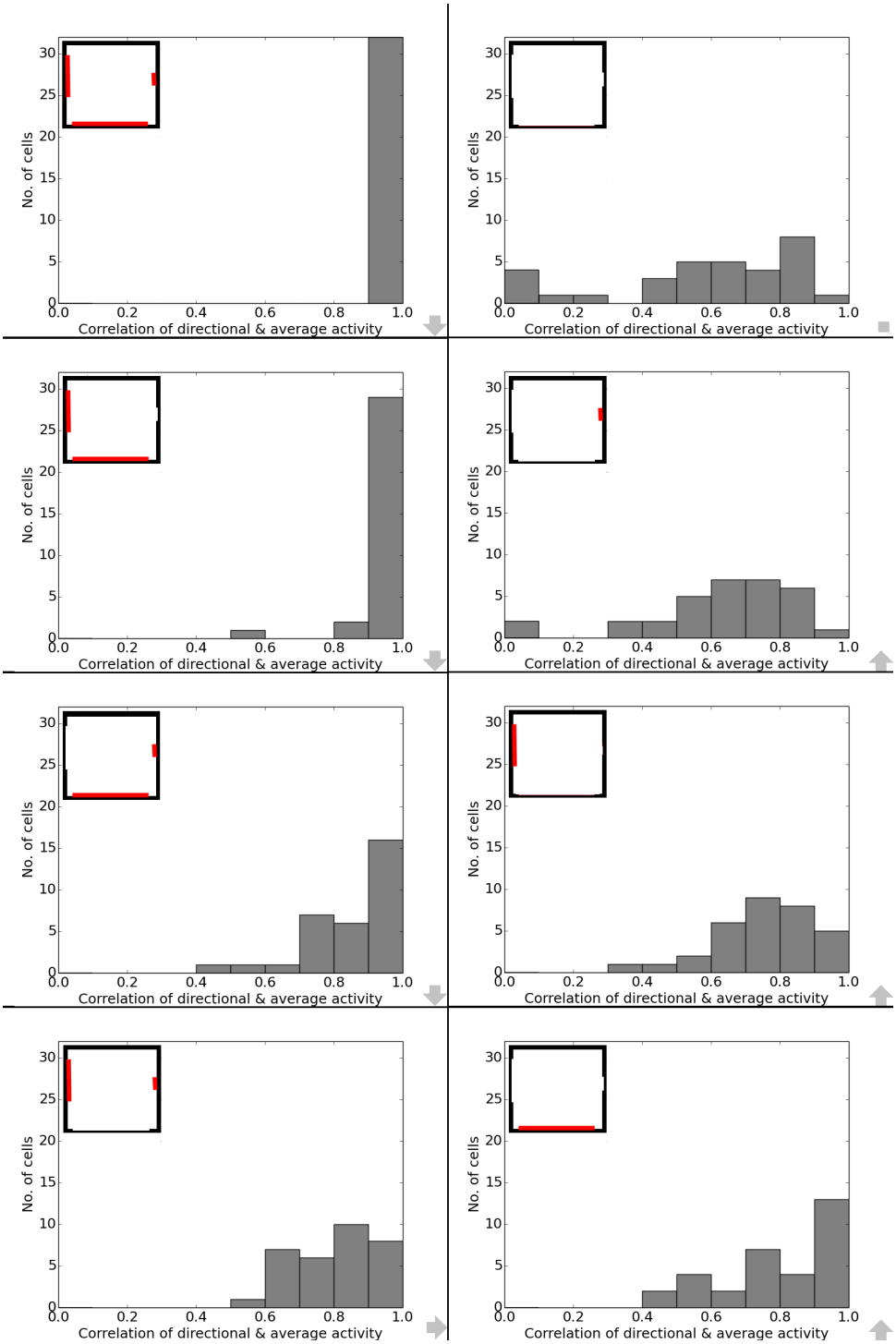

Figure 18: Quantitative analysis for cue card removal, I. Directional consistency of the SFA-computed spatial representation during cue card removal. Plots depict the directional consistency values of the full population of all 32 artificial place cell signals within the original environment and the tested variations with different subsets of the three available cue cards removed. Spatial configuration of each variation is indicated by subplot inserts; grey arrows indicate sequence from smallest to largest change; opposite plots depict inverse configurations.

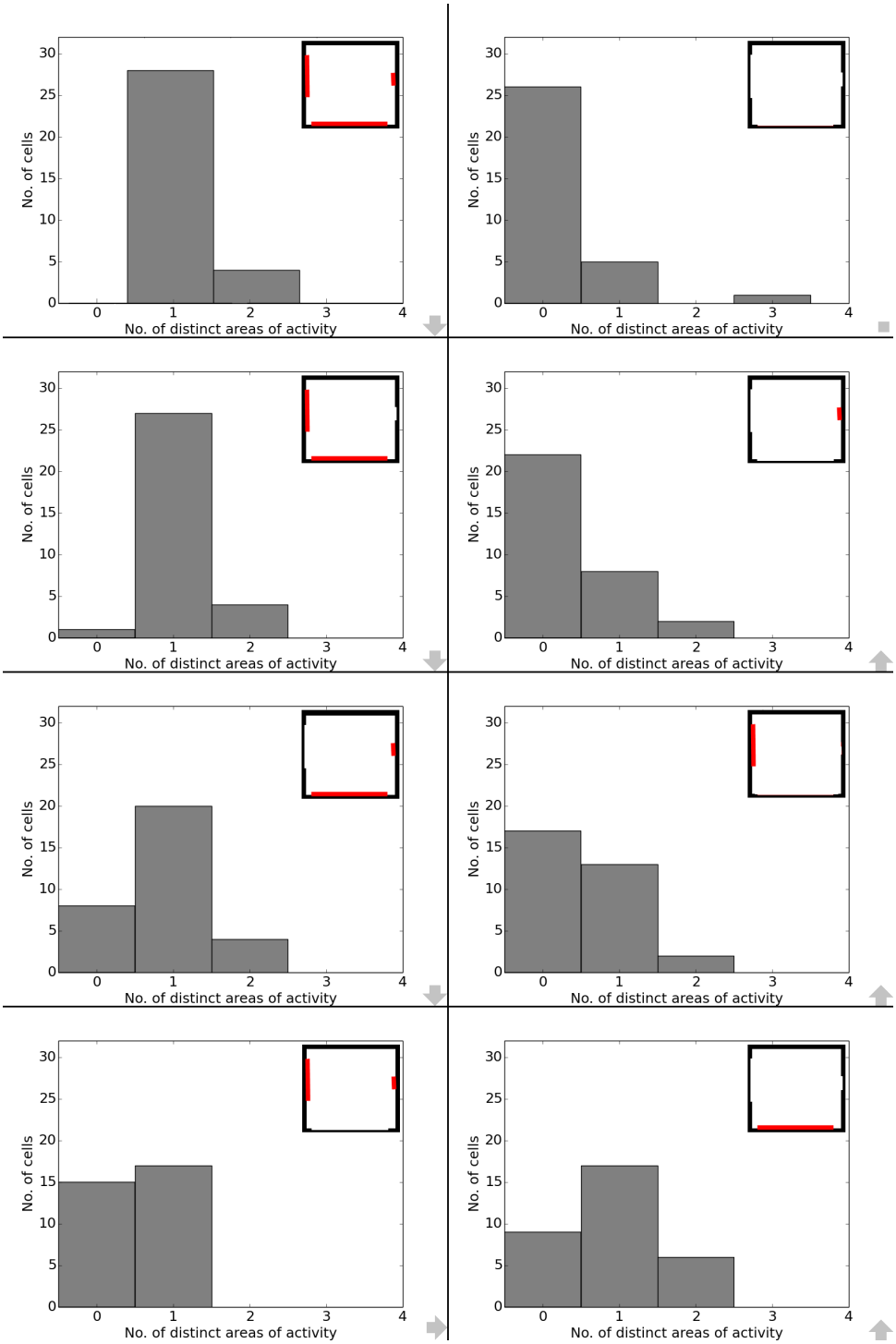

Figure 19: Quantitative analysis for cue card removal, II. Spatial consistency of the SFA-computed spatial representation during cue card removal. Plots follow the same arrangement as in the previous figure.

3.2.2 Directional firing in the linear track and open field

Besides experiments in the open field, where animals are allowed to roam freely, other experiments often restrict movement by placing animals within mazes featuring – or solely consisting of – narrow corridors. In such linear tracks animals are known to develop a spatial representation that is directional dependent, i.e., place cell activity depends on the direction the associated field is traversed in. While animals may be trained to follow such a track, this property of place cell activity does not rely on higher level functionality (such as planning) and can be measured in rats that merely explore the available space. In (McNaughton et al., 1983) animals explored the various arms of a multiple arm maze in a random sequence, while (Dombeck et al., 2010) presents measurements from animals that ran on top of an air suspended ball in front of a screen depicting a virtual linear track. In both cases a spatial representation was established that shows clear direction dependence.

To examine whether this property is found in the spatial representation learned by a hierarchical SFA network, a uniformly textured, 80 by 12 cm linear track was constructed. Two cue cards were present at both ends of the track, and wall height was set to a value low enough to allow the virtual agent to look over the surrounding walls and orient itself using a background depicting the inside of a laboratory. The agent was configured to traverse the corridor from one end to the other for 5 min. Note that when following a set or predefined waypoints, such as in this case, the software automatically adds a noise term. This makes the agent randomly deviate from the ideal linear path between two waypoints and ensures the training data from each iteration of traversal is not identical.

Fig. 20 shows both the real life recordings presented in (McNaughton et al., 1983) and the sampling results from a hierarchical SFA trained with the data generated in simulation. In both cases the spatial representation is **(a)** highly dependent on movement direction, and **(b)** place fields cover the full length of the track. Note that some cells did exhibit firing activity in both directions, a phenomenon that is also reported in (McNaughton et al., 1983) and captured by the SFA-based representation. Over the full population of 32 spatial signals, 11 remained largely silent, 9 exhibited the clear directional dependency presented in fig. 20, and 12

showed bi-directional activity (of which two signals showed multiple firing fields). To compare these numbers with real life data, (McNaughton et al., 1983) reports a percentage of 34% cells to be inactive and from 25 cells "... 14 were subjectively classified as highly directional, 6 relatively non-directional, and 5 were ambiguous." Fig. 21 shows the directional consistency of the SFA-based representation, which peaks around zero as expected from these results.

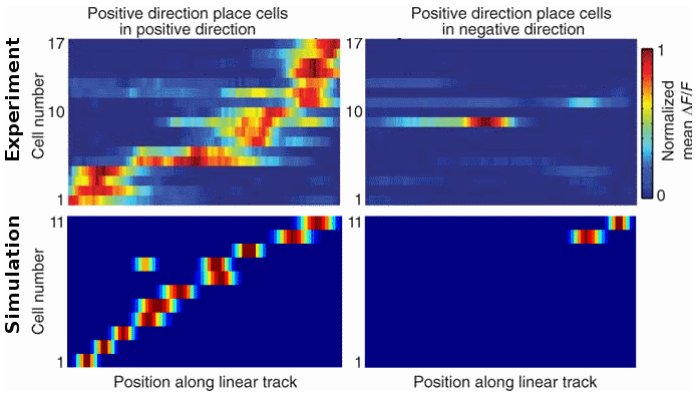

Figure 20: Place fields along a linear track. Spatial activity in the linear track of both real life cells and sampled from an SFA-based simulation. **Top row:** Place cell activity as measured and reported in [Reprinted by permission from Nature Neuroscience (Dombeck et al., 2010)]. **Bottom row:** Sampled network output signals after training in the linear track simulation. In both rows the left image depicts the activity while the respective agents traverse the linear track from left to right; the right image depicts the opposite direction.

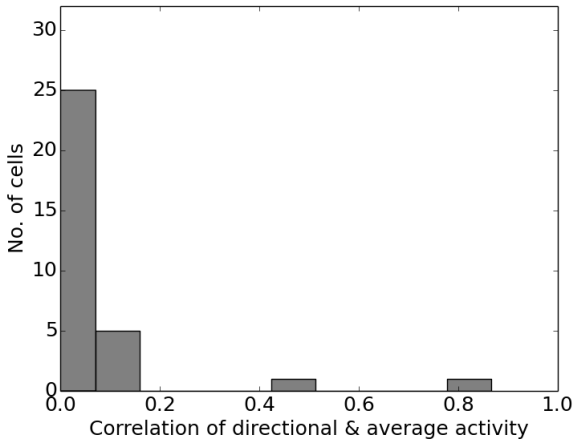

Figure 21: Quantitative analysis for the linear track. Directional consistency of the virtual place cell population within the linear track simulation.

Interestingly, the direction sensitivity of place cells can also be observed in the open field. Again, experimental protocols are not required to rely on higher level functionality, such as planning or remembering object relationships, to demonstrate this property. In (Markus et al., 1995) animals were trained to follow a rhombus-shaped path around the center of a circular environment in both a clockwise and a counterclockwise fashion. Animals were shown to establish stable place fields that were active depending on the direction the animal did traverse the familiar route. Such modulated cells were shown to either exhibit place fields in one traveling direction and remain silent in the other, or be active in both cases but at different locations.

To replicate this experiment a circular environment with a radius of 38 cm was specified and traversed by a virtual agent for 6.5 min. The agent followed a set of predefined waypoints and only explored the environment along the rhombus shaped path specified in the original protocol. Just like in the previous simulation, the software automatically adds a noise term to make the agent randomly deviate from the shortest path between to successive waypoints. As in the original experiment the agent was allowed to look over the walls of the enclosure and use the backdrop of a virtual laboratory to orient itself. Due to the nature of the simulation protocol, sampling of the network output after training was done in three steps. First, all output signals were sampled with the camera pointed in the four directions (ne, se, sw, nw) the agents faced while traveling between the four waypoints (n, e, s, w). Second, for each signal the four corresponding directional plots were cut together into a single plot consisting of the four legs of the path and with the camera pointed in the direction the agent was facing during each leg respectively. Finally, the overall traversed area of the environment was turned into a mask overlay and used to exclude those parts of the sampling plots that were not actually accessed by the agent during training.

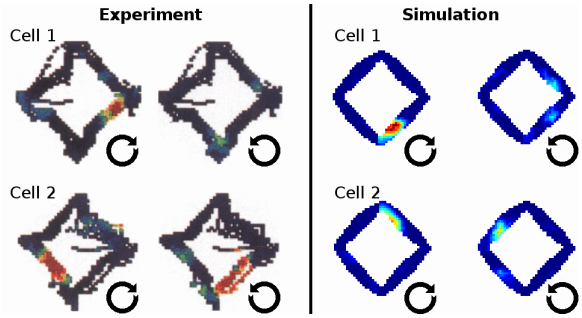

Figure 22: Place fields along the rhombus-track. Comparison of the spatial activity of cells and signals while the agent traverses a circular arena along a rhombus-shaped path in both a clockwise and a counterclockwise fashion (traveling direction indicated by black arrows). **Left:** Measurements of two real life cells recorded in animals. [Reprinted by permission from the Journal of Neuroscience (Markus et al., 1995)] **Right:** Two handpicked signals produced by a hierarchical SFA network.

Fig. 22 shows a direct comparison between the results presented in (Markus et al., 1995) and the sampling process of the trained SFA network. Fig. 23 shows seven additional examples of the spatial activity of the SFA-based representation along the rhombus path. The spatial activity of the different network output signals shows clear directional sensitivity as firing fields differ significantly depending on whether the center of the arena is encircled in a clockwise or counterclockwise fashion. It can also be seen that, as reported in the original experiment, signals do not react to the change in direction in a uniform manner. Firing fields may vanish, remap, and even be active in a different place, depending on direction. Of the 32 output signals of the hierarchical network, 16 showed clear spatial activity in one direction, and little to no activity in the other; 11 showed clear fields in one direction and significant, but unstructured, activity in the other; and the remaining 5 stayed silent. Fig. 24 shows the directional consistency of the population, which for the direction sensitive representation seen in the sampling of the network is again centered around zero.

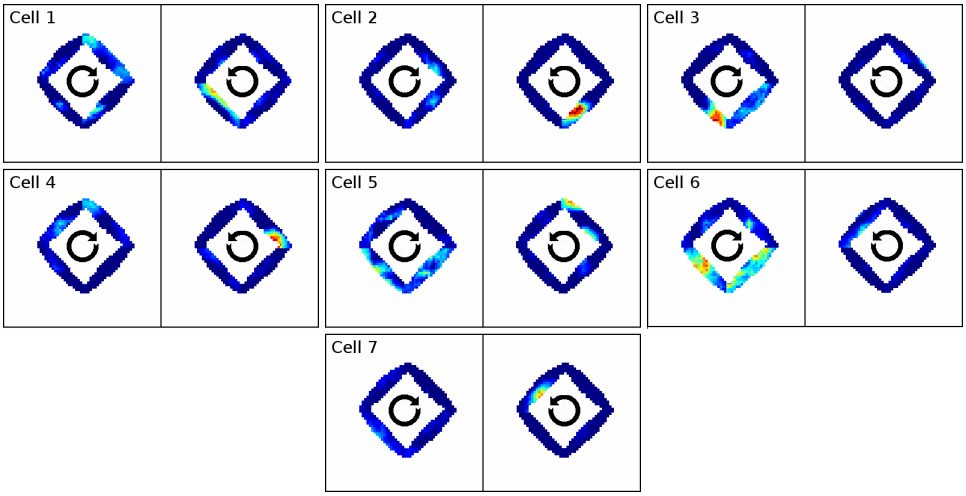

Figure 23: Rhombus-track sampling. Example set of different network output signals depicting the spatial representation as computed by hierarchical SFA after training along the rhombus-shaped path. Samples are shown in both clockwise and counterclockwise directions (indicated by black arrows).

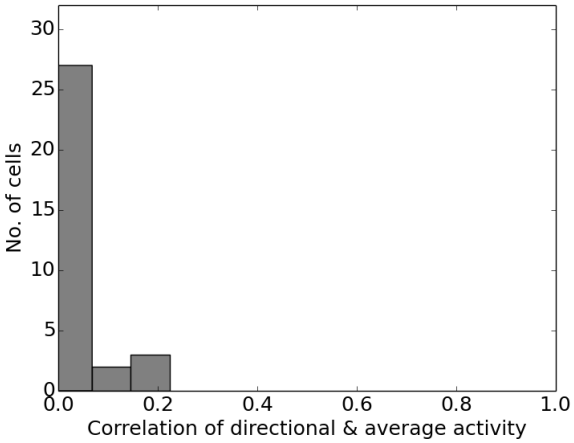

Figure 24: Quantitative analysis for the rhombus-track. Directional consistency of the spatial representation computed by hierarchical SFA in the rhombus-path simulation.

3.2.3 Adaptation to changes in wall configuration

The third aspect of hippocampal spatial encoding examined in this chapter is the sensitivity and adaptation of the internal representation of space to the overall context. Note that while the term "context" is routinely used throughout the literature, it is rarely stated in a clear and unambiguous way what it actually refers to. Thus

the phrase carries a different meaning throughout the literature and needs to be defined before use. In this work "context" refers to "spatial context" and a change in context refers to a specific type of spatial change, where not (only) the elements within a space – such as cue cards or other objects – change, but the overall environment changes as well. More specifically, in this work, for a change to be referred to as a change in context, the floor plan of the experimental apparatus needs to change.

Taking an animal from one environment into a completely different one can be seen as the most straightforward example of a context change. It would, however, simply result in the global remapping of a spatial representation, which is an already well established property of the hippocampal space code. A more subtle change in context is, among others, presented in (Wills et al., 2005) and (O'Keefe and Burgess, 1996). In the former, animals explored two similar environments that differ only in their floor plan: One is a circular arena, while the other is a rectangular one; both were constructed from the same set of modular wall segments. Once animals established a stable spatial representation in both of these, they were placed within a number of intermediate arenas, each a different in-between stage between the familiar circular and rectangular floor plans. During these trials, the animals' internal place cell representations were shown to abruptly switch from their rectangular arena mapping to their circular arena mapping. In other words, while the environment changes smoothly from one familiar setting into another, animals' representations seemed to stay stable during the first leg of the transition, then exhibit a sudden switch, and remain stable for the remainder of the transition (Wills et al., 2005).

To examine this behavior in an SFA-based spatial representation, the modular apparatus used in the original experiment was reproduced according to the exact specifications listed in (Wills et al., 2005). The environments consisted of small wall modules that were arranged to form the initial 62 cm square box and a circular space with a 39 cm radius, both of which were explored by the virtual agent for 10 min. Wall segments were textured uniformly but low enough to allow the agent to see a laboratory backdrop and use it for orientation. After training the network with the data collected during the two random exploration phases, it was sampled over both the original environments, as well as the four intermediate stages. The latter are built from the same wall modules as the the initial two mazes and their

construction in the simulation likewise adhered to the original specifications given in (Wills et al., 2005).

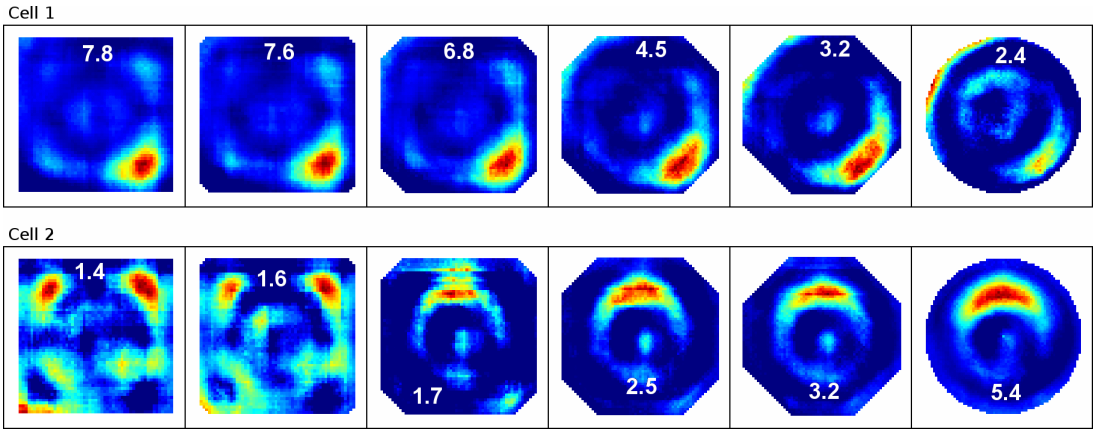

Figure 25: Morphing environment. Two examples of SFA network activity after training in both a rectangular and circular environment. Signals are sampled over both the two familiar settings as well as four "morphed", intermediate stages of the environments. Note that activity is scaled to the local maximum to clearly depict firing fields, while actual activity values are given as numbers and decrease as signals are sampled over non-preferred environments.

Fig. 25 shows two representative output signals of the trained SFA network to demonstrate the qualitative behavior of the SFA-based spatial representation: Each of the 32 network signals exhibited a distinct and stable place field in either the rectangular or the circular environment. When sampled over the various intermediate stages, place fields stayed in the same location (as far as the particular geometry allowed it) but became less and less active the further the environment morphed away from their preferred state. Fig. 26 shows the directional consistency of the population, where values can be seen to stay close to 1.0 in each environment, which documents that the network does not dissolve into erratic behavior. While place fields dissolved spatially (fig. 25), the representation still remained invariant to viewing direction. Fig. 27 shows the spatial consistency of all 32 signals and confirms the behavior shown in fig. 25 to be consistent over the whole population. Note how about half of the population features a single firing field in the rectangular environment (colored dark grey in fig. 27), while the other half features almost no place fields that satisfy the criteria to be counted. In the circular environment on the other hand, roles are reversed, and the population with no active fields in

the rectangular environment (light grey in fig. 27) now features a single place field and vice versa. During the initial morph stages, resembling a rectangle with cut off corners, the overall representation remained stable. Firing activity for most cells did fall below the 50% threshold, however, when sampled over the two morph states that resemble an octagonal polygon.

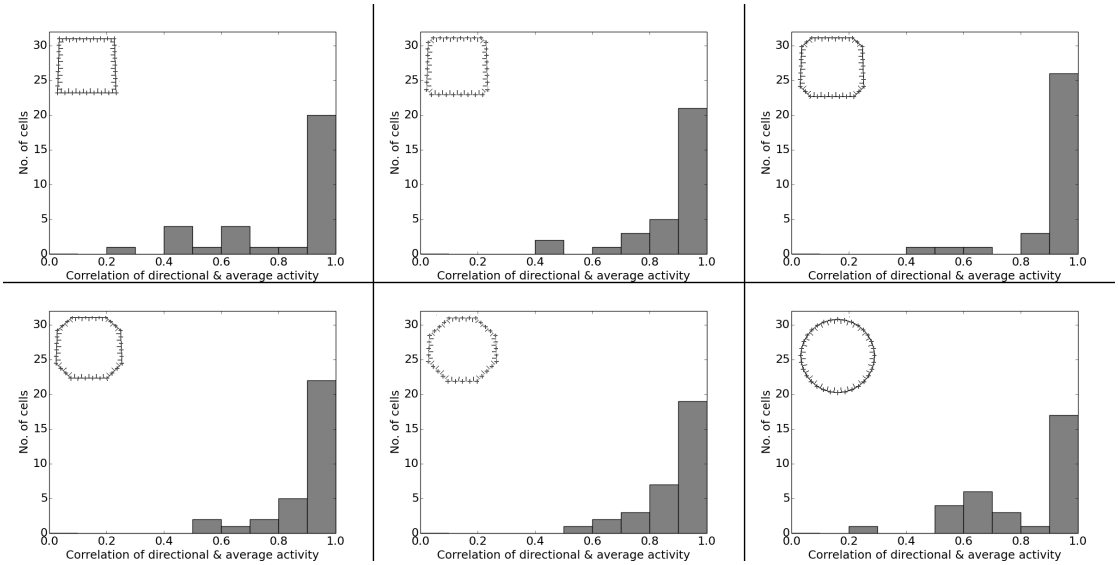

Figure 26: Quantitative analysis for the morphing environment, I. Directional consistency of the signal population after training in two environments and sampling in both the original and four additional in-between, "morphed" arenas. Subplots within each histogram depict the different environments the network is sampled in. Each inward pointing line denotes one of the slice-like wall segments that were used to form the overall environment according to the specifications given in (Wills et al., 2005).

A second experiment investigating the properties of spatial context changes in the hippocampus is presented in (O'Keefe and Burgess, 1996): Rats explored four different, stretched versions of an otherwise identical, rectangular environment. The animals were given 8 min to establish a spatial representation, during which their place cell activity was recorded. Rather than homogeneous adaptations to the different variations of the environment, however, the authors report a range of different behaviors as the individual place cells reacted to the changes in spatial context. This includes keeping their firing fields at the same relative position, stretching along the walls to mirror the change in setting, or falling silent and stop responding (O'Keefe and Burgess, 1996).

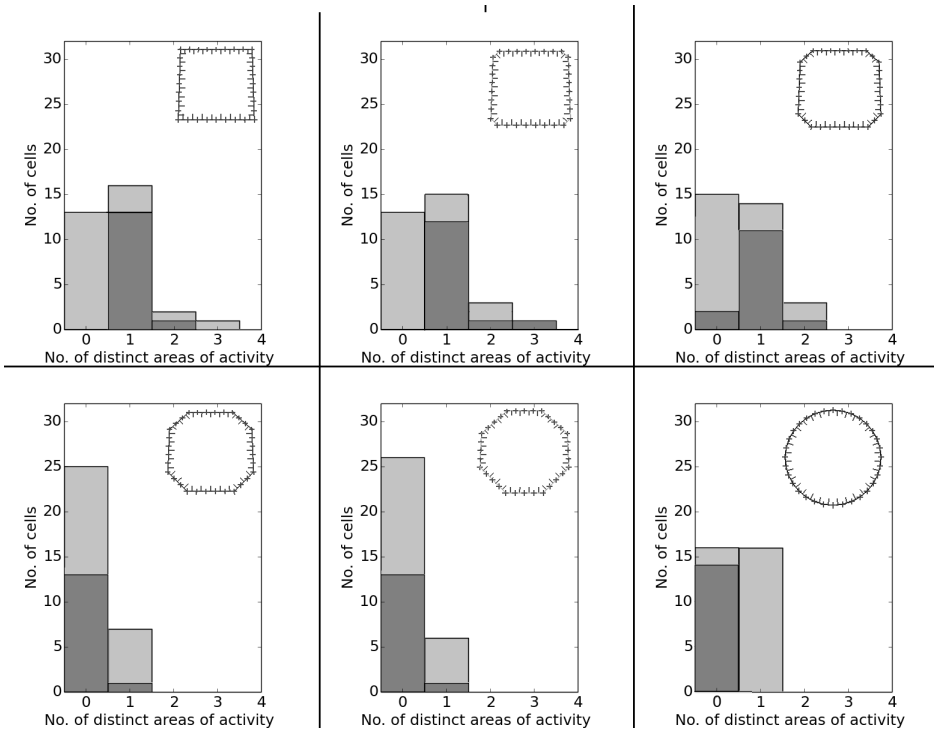

Figure 27: Quantitative analysis for the morphing environment, II. Spatial consistency of the signal population after training in two environments and sampling in both the original and four additional in-between, "morphed" arenas. Dark/light grey parts of columns denote cells predominantly active in the original rectangular/circular environment respectively. Subplot inlets are used as described above.

To examine this behavior in the SFA-based representation, the virtual agent was placed within a 120 by 60 cm box and randomly explores that space for 10 min. Walls were textured uniformly, but low enough to allow the agent to look over them and navigate via the available cues within the laboratory backdrop. The network was trained with the recorded visual data and then sampled in the original environment, as well as the three stretched versions of dimensions 60 by 120, 60 by 60, and 120 by 120 cm respectively.

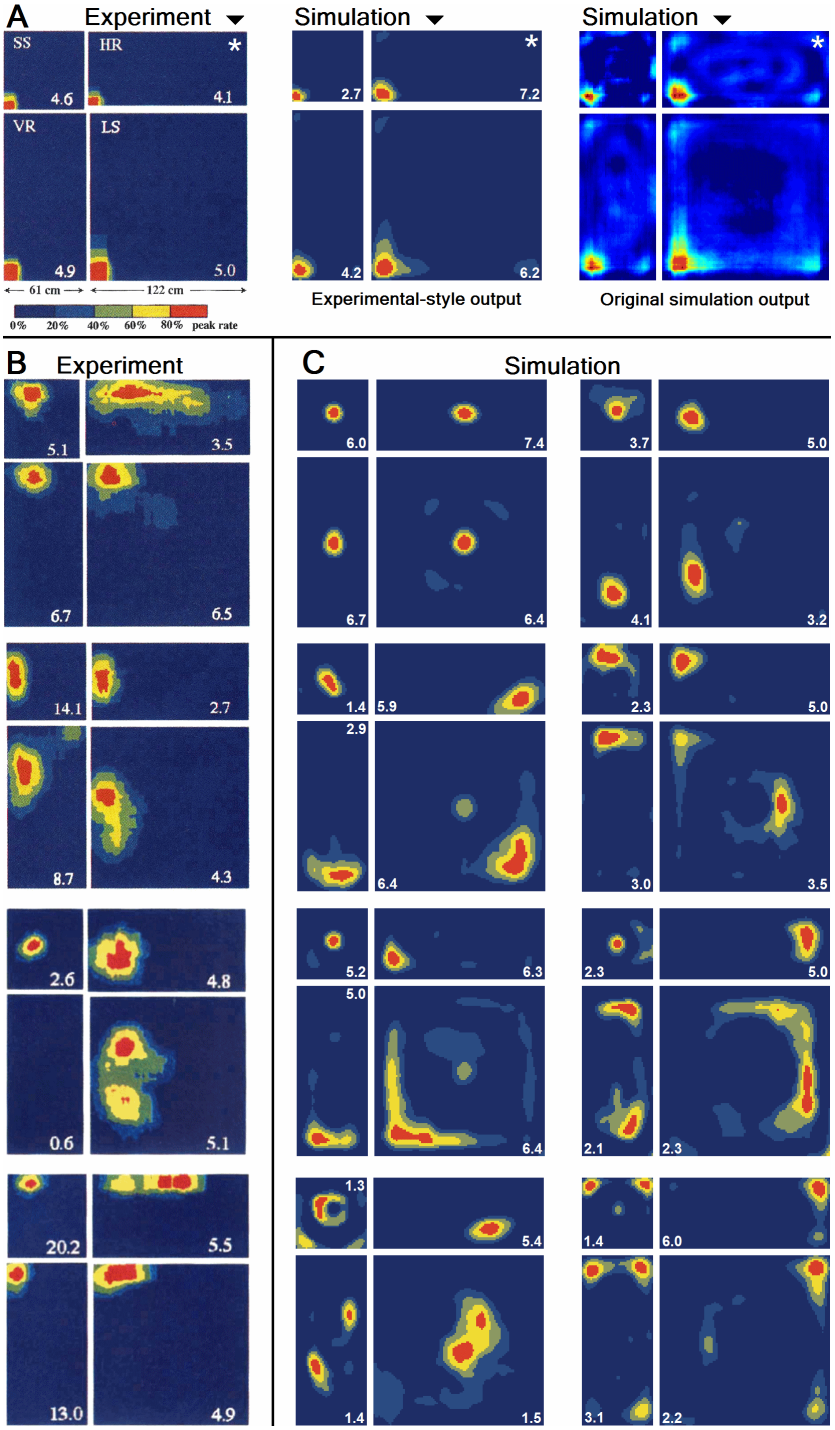

Figure 28: Stretching of the environment. Cell/signal activity in experiment and simulation after training in a rectangular 120 by 60 cm environment (star label) and sampling in three additional, stretched versions. **(A)** From left to right: Direct comparison between a real life cell recording presented in (O'Keefe and Burgess, 1996) and the best fitting signal as computed in simulation. Simulation results are plotted in the same format chosen by the authors in (O'Keefe and Burgess, 1996) to ease visual comparison of the data; third image shows the same simulation data plotted along the jet scale to allow a direct comparison with the other plots throughout this work. **(B)** Four cell recordings from the original real life experiment to demonstrate the range of individual behavior as place cells adapt their firing fields to the change in scenery. [Experimental data reprinted by permission from Macmillan Publishers Ltd: Nature (O'Keefe and Burgess, 1996)] **(C)** Eight examples taken from the virtual cell population to depict the adaptation of the SFA-based representation to the unfamiliar environments. Signals can be seen to follow the real world example and do not remap uniformly but rather adapt individually to the change.

Fig. 28 shows a direct comparison between a single place cell measured in simulation and (O'Keefe and Burgess, 1996) which happens to match exceptionally well, as well as additional examples provided in (O'Keefe and Burgess, 1996) and sampled from the network. The latter are not presented as a 1:1 comparison, but to showcase the range of individual cell/signal responses to the different environments in both the experiment and simulation. During all trials the sampled network signals showed clear spacial activity and none broke down into incoherent, random firing, as is usually the case if the model becomes unable to properly process visual input. Further, as the model adapts to the new environments, almost all of the signal patterns can be labeled as either belonging to one of the categories described in (O'Keefe and Burgess, 1996), or to one of the additional classes observed in the simulation: fields that rotate in order to keep their relative position in relation to landmarks such as corners; fields splitting into two, usually mirrored, fields; and fields that relocated (often to the center of a new environment and usually observed in the square variants). While firing patterns stayed coherent in almost all cases, two signals displayed an overall loss of localized firing, with different "mutations" of their original place fields in each of the stretched variations.

Figures 29 and 30 depict the statistics of this behavior for the overall population. While the directional consistency is close to 1.0 for the familiar 120 by 60 cm environment, the value drops for most cells in the scaled versions of the arena, with the large 120 by 120 cm variant being the most confusing to the network. The number of firing fields behaves in a similar fashion and cells can be seen to fall below 50% of their familiar-setting peak activity in the unfamiliar settings.

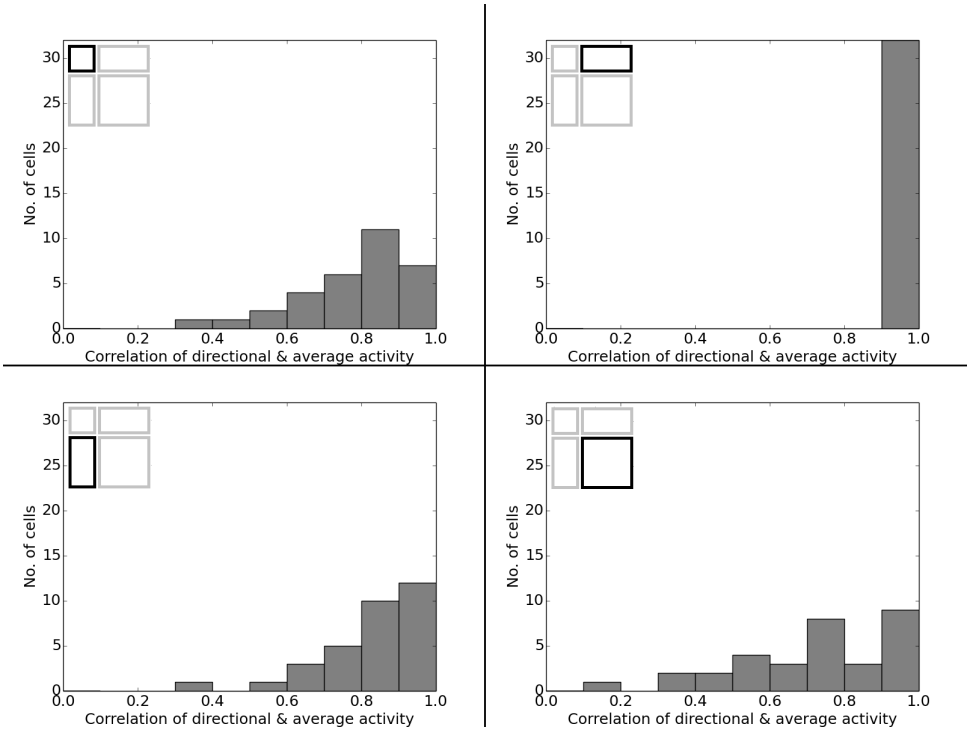

Figure 29: Quantitative analysis stretched environment, I. Directional conistency values for the SFA-based spatial representation over the original 120 by 60 cm environment and the four differently stretched variations (indicated by insets).

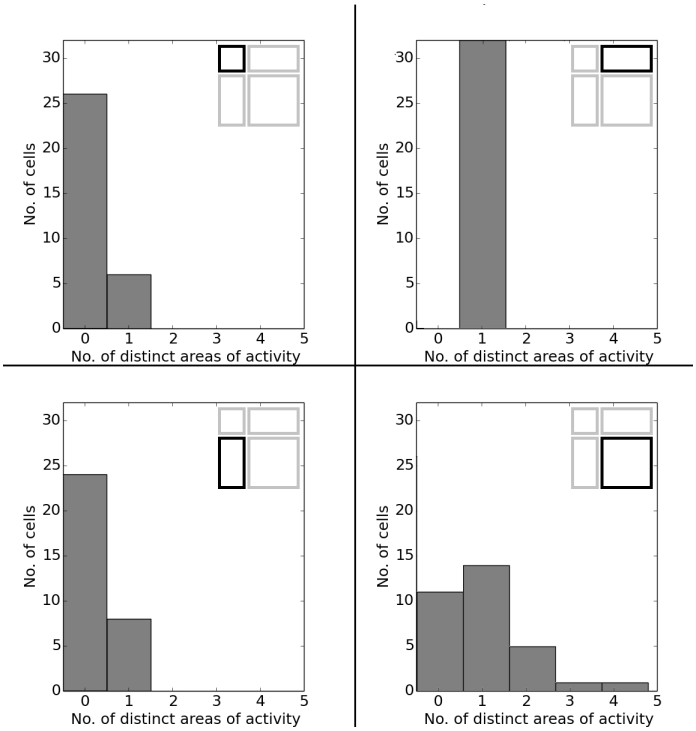

Figure 30: Quantitative analysis stretched environment, II. Spatial consistency values for the SFA-based spatial representation over the original 120 by 60 cm environment and the four differently stretched variations (indicated by insets).

3.3 A model to bridge spatial encoding and episodic memory

Note: *In the following, a "percept" denotes one "unit of perception" and is defined by its type and its content – the former of which is determined by the relevant input modality. Percepts are implemented as simple vectors holding a set of values, e.g., the signal activity of a hierarchical SFA network to form a visual percept.*

One fundamental problem with bridging the gap between episodic memory in humans and spatial encoding in rodents is that experiments addressing the two issues take place under markedly different conditions. Not only can human subjects be asked to perform a specific memory task, the person can also be directly queried about what he or she remembers about a specific past occurrence. In addition to any experimental apparatus that might be used, the "input" in the human case would typically comprise an instruction and a question, i.e., verbal constructs, that only relate to the targeted memory because of the cognitive abilities of that

particular person. Language is a highly developed concept; it requires several years to acquire and is supported by two areas of the cerebral cortex, namely Broca's area and Wernicke's area. The output in human trials consists of either a clear verbal response, or non-invasive measurements such as fMRI or EEG recordings, which tend to suffer from much lower resolutions when compared to cellular recordings. In rodent studies, on the other hand, this situation is reversed: while cellular recordings are readily available, animals can neither be verbally instructed about a task, nor are they able to verbally articulate their experience during an experiment. Consequently, animals are trained to perform specific tasks and the "input," in this case, is an animal's overall experience of the environment during a designated learning phase. The output generally consists of any observed behavior and/or measurements made during the experiment. In short, experiments that examine and differentiate the aspects of episodic memory are straightforward for humans, but both complicated to perform and difficult to interpret in rodents – and vice versa.

How can this disparity be addressed in a computational model? The initial approach to extend hierarchical SFA and incorporate additional input modalities was to add information to the visual input and feed it into the established network, possibly at different stages of the hierarchy. However, justifying the reasons for inserting new information at various points of the original model has proven problematic. Consequently, instead of extending hierarchical SFA, it is embedded as one of several input modules into a new architecture implementing hippocampal functionality. More precisely, a given set of features produced by hierarchical SFA is used as the content of visual percepts that, among the percepts of other input modalities, are processed by a more general architecture. In this expanded model hierarchical SFA fulfills a role more akin to the visual system than the overall hippocampus. Consequently the optional ICA layer of the full SFA network is dropped and only SFA components are used.

This new architecture is presented in the following. It is designed to capture the fundamental principles of hippocampal processing: **(a)** ability to associate and integrate information across different input modalities and time, and **(b)** access to a memory system to store and retrieve such amalgamations of information. In fact, the original question that lead to the development of this new system was:

Given only the central functionality of the hippocampus – association across input modalities and memory access – what may a hippocampal model be able to account for?

As this question already suggests, the model operates on a high level of abstraction. The proposed architecture (algorithm 1) receives a set of new percepts each time step and decides whether they contain new information that should be stored. If a set of percepts is already fully known, it does not contain new information and is discarded. If only some percepts are known while others are not, the known percepts are discarded and the new percepts are linked with the known percepts already stored in memory – thus expanding an existing set of percepts and forming a cluster of percepts. If all of the percepts are considered new, they are committed to memory to start a new cluster (initially only consisting of the newly acquired set of percepts); each such newly created cluster is further linked to the cluster that was added last.

The difference between known and unknown percepts is determined by two threshold values: If two percepts of the same type are similar enough, they are treated as equal and the new percept is discarded since it is already considered to be part of the internal memory. On the other hand, in order to be considered new, a percept has to be different enough from the most similar percept currently available in memory in order to be eligible for storage. Note that each input modality has both its own threshold values as well as its own function to compute the similarity between two percepts. The current implementation works with three types of percepts: visual information, grid cell activity, and words; the functions computing similarity are euclidean distance for both of the former and Levenshtein distance for the latter.[13] The similarity between two percepts of a different type is defined as not a number (`NaN`); this results in the boolean operations in lines (6) and (9) of algorithm 1 to always yield `False` if the compared percepts are of different types.

[13] In computer science, the Levensthein distance is the number of deletions and/or substitutions required to convert one word into another.

Algorithm 1: General percept handling architecture to model hippocampal function

```
input : new set of percepts new_percepts; existing percept clusters
        stored_percept_clusters
diff_count ← 0;
foreach cluster in stored_percept_clusters do
    match_count ← 0;
    foreach cluster_percept in cluster do
        foreach percept in new_percepts do
            if difference(cluster_percept,percept) < threshold value for
            "known" then
                match_count ← match_count +1;
            end
            if difference(cluster_percept,percept) > threshold value for "new"
            then
                diff_count ← diff_count +1;
            end
        end
    end
    if match_count = no. of percepts in cluster then // all percepts are already known
        return
    end
    if match_count > 0 then                          // some percepts known, others are new
        cluster ← cluster + unknown percepts;
        return
    end
end
if diff_count = no. of percepts in memory then       // all percepts are new
    new_cluster ← createNewCluster(new_percepts);
    link(last created cluster in stored_percept_clusters, new_cluster);
    stored_percept_clusters ← stored_percept_clusters + new_cluster;
end
```

As can be seen, the model continuously accepts new percepts and incorporates them into a network of percept clusters that grows as new input is fed into the model.

Note that percepts may carry virtually any information: visual features that result from plain exploration or are part of an episodic memory trial are accepted just like more abstract features like words or facial expressions. Since the hippocampus does not produce high level features from raw input but, presumably, merely processes them, any input may be generated outside of the model and then fed into the architecture as a percept of the appropriate type. New clusters of information are formed whenever novel percepts are encountered that cannot be matched to what is already known to the systems. While percepts are linked to form clusters, clusters, in turn, are linked to document the temporal sequence of their activation. Note that algorithm 1 does nothing more than implementing the two central elements of hippocampal processing noted above: **(a)** the ability to associate and integrate different percepts, and **(b)** access to a memory system to store and retrieve such sets of interlinked percepts. It does not make any assumptions about the incoming data, nor does it contain subroutines geared to any specific problem domain (e.g., mapping).

Once percepts have been incorporated into the memory system, clusters can be asked for their response to any given set of percepts. These responses denote the familiarity of the individual clusters with the input in question; the more similar the input to the percepts stored within the cluster, the higher the response value. These values serve as the output of the system and may be gathered either while a simulation is running or afterwards, i.e., the state of the system may be examined while it develops over time or after information gathering is considered complete, depending on the goal of the simulation. Implementation-wise, there are numerous possibilities to compute such a familiarity score. During development, several methods were tested, such as the average similarity of each percept of a cluster with the input set, or the maximum or minimum similarity value (using the same distance function used to determine new input as already known or new). The current version of the algorithm uses the average response of each percept within each cluster to denote how familiar the individual clusters are with the current input.

This architecture satisfies the following criteria:

- It does not require a dedicated learning phase; the model runs continuously

and processes new input as it arrives. The internal representation of the input may be queried at any time.

- Each percept is processed the same way, regardless of input modality.
- The model is not reset in between trials; a memory task may follow an exploration task (and vice versa) without the need to reset or change any internal data structures.
- The model is capable of both establishing a spatial representation and performing a simple episodic memory task. Hereby, the reaction of the model to the input at any given location, i.e., its spatial activity, emerges naturally from the application of algorithm 1 and is not the result of a purpose-built mechanism.

3.3.1 Sample trials: spatial exploration and word list recall

To provide a proof of concept of this architecture, two basic simulations were chosen: a random exploration task with no other requirement than to develop a spatial representation, and a plain memory task to recall words from a provided list. While the former is generic and not modeled after any specific experiment, the latter was taken from (Heit et al., 1988). In their work, the authors present recordings from individual hippocampal neurons in humans and show how different units display a preferred reaction to the individual words of a previously learned list (Heit et al., 1988).

- **Random exploration:** Similar to the simulations in the previous chapter, a virtual agent is placed within a cylindrical environment (80 cm radius) and randomly explores the environment. During each time step hierarchical SFA takes the current image of the field of view, computes 32 features, and packages them in a visual percept data structure that is fed into the new architecture. In contrast to previous simulations, however, the hierarchical SFA network is not trained in this setting specifically. The top ICA layer is removed, and the remaining network is merely trained with a batch of $100,000$ images of the generic training set used above (fig. 10, p. 42). Additionally, the model receives the input from an artificial grid cell system: At each position a grid

percept is generated that contains the activity of 32 grid cell-like grids of varying sizes.

- **World list recall:** The model is given a list of ten words and then queried with each of them. The activity of all clusters is measured as a response to the query and, similar to the results reported in (Heit et al., 1988), each cluster is expected to react to all words, but show a clear preference for one of them.

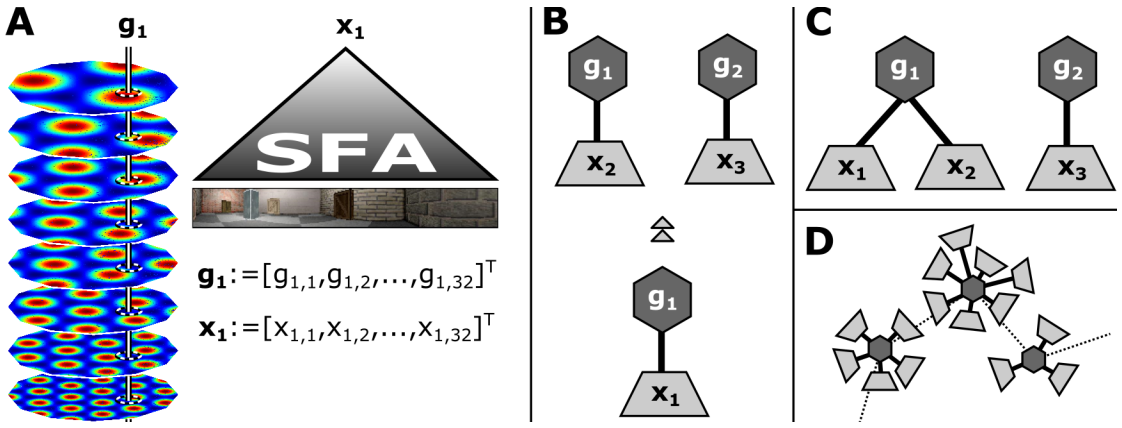

Figure 31: Spatial representation in the new model. A: During each time step the activity of 32 grid cells (left) at the current location is stored in a vector ($\mathbf{g}_1$) and paired with the the responses of a generically trained SFA network (right) reacting to the current visual input ($\mathbf{x}_1$). **B:** A new pair of spatial percepts is added to the memory which contains two clusters. One cluster already contains the grid activity at the same location ($\mathbf{g}_1$) but associated with a different view ($\mathbf{x}_2$). **C:** The already known grid percept $\mathbf{g}_1$ is discarded and the unknown visual percept added is to the cluster. **D:** After some time of exploration the internal memory holds several clusters encoding locations by associating grid percepts with different visual percepts. In addition, each cluster includes a (temporal) link pointing to the previously created cluster.

In the first experiment, the agent randomly explores the environment; the initial memory state is empty. Consequently, the grid and visual percepts received initially are recognized as new and stored in memory. The percepts immediately following will be similar to the already stored ones and thus be discarded as already known. Once the agent receives any input that is different enough from any known percepts, the new information will be added to the network. The percepts will either be associated with already learned percepts – thus growing a cluster of percepts – or be stored by themselves as the roots for future percept clusters. Fig. 31 depicts this process and shows how, after a phase of exploration, the internal memory contains a number of temporally linked clusters of varying degrees of sophistication. Each

cluster consists of one grid percept associated with several different visual percepts. This arrangement emerges naturally from the rules of algorithm 1: when the agent arrives at a previously learned location, the generated grid percept will already be known and thus discarded. The visual percept, however, might be different if the agent faces a different direction than before and thus does receive a visual input that may be considered new. Since the grid percept is known, but the visual percept is not, the visual percept will be associated with the known grid percept in memory, and the cluster encoding this location grows. If each cluster is queried about the familiarity with the input of the current time step, it can be seen how, over time, a freely roaming agent establishes place field-like activity (fig. 32).

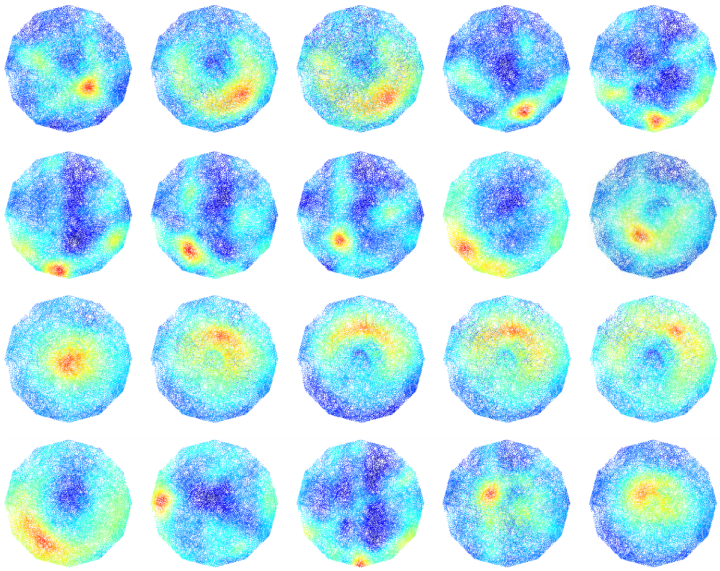

Figure 32: Place field-like activity in the general architecture. Spatial activity of 20 randomly selected percept clusters while the agent explored a circular environment for 25 min and developed 111 percept clusters.

Remembering the words from a given list is not a difficult task, especially when (human) subjects are explained the task at hand. Not only do subjects know that the following events comprise a memory task, but also that the presented words are individual entities to be learned for recall (in contrast to forming a sentence, for example). Therefore the model functions in a straightforward manner: for each word a percept containing the word is generated and handed to algorithm 1. Since words are expected to be individual entities, the threshold for new words is set

low. As a result, each received word is added to memory as an individual item and, while linked temporally with the preceding percept, not associated to form a cluster.

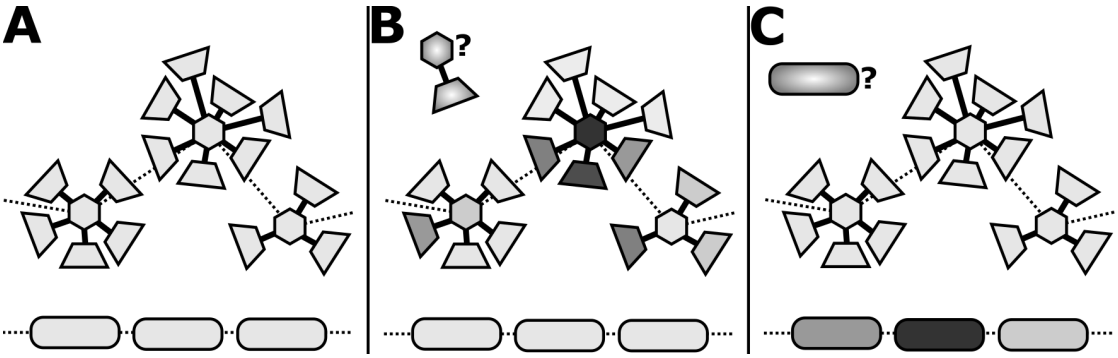

Figure 33: Querying the new model with different input. Clusters stored within memory can be asked to respond to any set of percepts to indicate their familiarity with the set. These responses are used as the output of the model and can be interpreted as cellular recordings. **A:** The memory network at rest; different shapes indicate type of percept: hexagons represent one percept of grid cell activity, trapezoids represent visual activity, and rounded rectangles represent words. **B:** The network is queried with a visual percept and a grid percept (indicated with a '?'). Shades of grey indicate cluster familiarity with the percepts. **C:** The network is queried with a word. As before, shades of grey indicate cluster responses. Clusters storing a different type of percept do not respond at all.

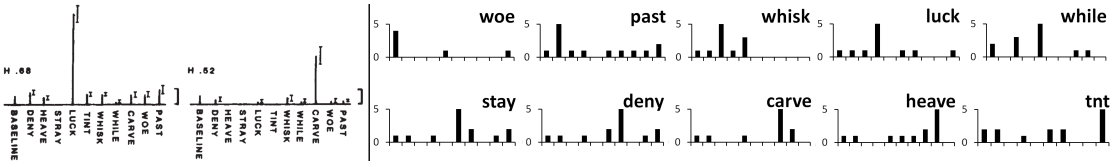

Figure 34: Reaction of the model to previously learned words. Right: Responses to the same words from different clusters as produced by the general architecture after learning the same list. Each subplot depicts the responses of all clusters to one word that is presented to the model (top right of each plot). **Left:** Recordings from two cells in the human hippocampus as subjects are presented ten previously learned words. [Experimental data originally presented in (Heit et al., 1988)]

As before, the individual clusters can be queried about the familiarity with different items (fig. 33). This is independent of any previous tasks, i.e., if the internal memory also contains information acquired during a spatial task, it may still be asked for a response to individual words. Fig. 34 depicts cluster activity when the system is queried with the learned words and compares it with the results presented in (Heit et al., 1988).

4 Discussion

4.1 Summary

There are two fundamental motivations for employing computational models in a natural science. One is an attempt to extend the field and ask: *"Based on the existing body of experimental results, what can a hypothesis and its implementation predict that can be, or has not yet been, confirmed by a real life experiment?"* The other is to look at the results acquired by experiments, and ask: *"Is there a way to unify this set of observations? Are there underlying principles that we may find to help explain the patterns behind the observations that we have accumulated?"* The former approach aims primarily to adding new data to the contemporary pool of results. The latter, in contrast, does not immediately seek to enlarge this set, but rather to gain insight into it by searching for underlying principles of organization and computation. As this work follows the latter approach it does not, primarily, provide a selection of new predictions. Instead, it presents an example of how modeling on a high level of abstraction can serve as an effective approach to identify and analyze the fundamental processes within a field of study – in this case, the slowness principle as a candidate process to explain hippocampal function.

In order to do so, three goals were set and presented in this work. First, a comprehensive software package was developed that allows other researchers to use hierarchical SFA and replicate experiments performed with real animals and compare modeling results with recordings from real cells. The software was purposely designed to be usable by researchers from different backgrounds and to allow scientists from a variety of fields to perform their own simulations based on the slowness principle. The available parameters enable the construction and implementation of a wide range of virtual environments and experimental protocols. The complete toolkit is published as free software under the GNU General Public License (v3) and includes a range of small scripts and examples that encourage modification and experimentation. In summary, the toolkit invites anyone interested in experimenting with the slowness principle to do so, and offers a software environment that provides everything needed to setup and simulate a desired experimental protocol.

Second, using the developed software, the slowness principle is shown to be capable

of developing a spatial representation of the environment that accounts for a multitude of observations reported in the experimental literature. A combination of hierarchical slow feature analysis (SFA) and independent component analysis (ICA) was applied to the (visual) output produced by computer simulations replicating published experimental protocols. Using this technique, the spatial representations developed by hierarchical SFA have been shown to bind to visual cues and an react appropriately to their removal, exhibit directional activity in both constricted and an open environments, and adapt plausibly to changes of the spatial context. These results demonstrate how a spatial representation may be based on visual information alone – as well as the limitations thereof – and how computational modeling may further our understanding of information processing in the brain as well as help unify previously reported experimental observations.

Third, a new abstract architecture has been introduced that implements the fundamentals of hippocampal processing and embeds hierarchical SFA to compute one of several possible input modalities. This new model operates on a high level of abstraction and is intended to start the process of merging the two primary areas in contemporary hippocampal research: episodic memory and spatial encoding as it is observed in the hippocampus of humans and rodents respectively. The proposed architecture is designed to continuously accept the input from any chosen input modality and process each received "percept" equally: Data is either discarded if already familiar or incorporated into an expanding memory network if assessed as new. The underlying algorithm implements basic hippocampal functionality – ability to store and retrieve memories and associate data across both input modalities and time – and asks what range of observations a model based on these principles alone may be able to account for. The architecture does not require a dedicated training phase, nor a reset in between different trials, and is shown to perform both spatial mapping as well as a basic memory task.

4.2 Detailed discussion

4.2.1 The ratlab software package

Instead of merely writing dedicated Python scripts to only perform a specific set of simulations without the ability to perform any other experiments, an open, flexible,

and freely accessible set of tools has been developed. The software package fulfills two objectives: first, it provides the tools necessary to set up and perform a multitude of simulations using hierarchical SFA, and second, due to accessibility being a design goal from the start, this functionality is made available to anyone interested in the subject. It requires no programming knowledge and includes a hand-drawn user interface that may be used to set up basic simulations and examine spatial representations based on the slowness principle. The user interface generates a command line script which, in conjunction with the provided documentation and examples (see appendix A) encourages a transition to more advanced simulations. Such advanced simulations may include custom environments of arbitrary size or layout and are not restricted to simple variations of one predefined protocol. The different modules may be interchanged, repeated, or outright modified to suit the needs of any one particular simulation.

As neither memory nor planning are part of hierarchical SFA, the model is best suited to simulations that allow an agent to first establish a spatial representation, introduce a change to the environment, and record the adaptation of the spatial representation to the modification. However, it can also be used to perform simulations of a different kind and suggest new lines of inquiry. For instance, the mathematics of SFA predict elongated place fields if the movement speed of an agent is not uniform in all directions (fig. 35). To show this effect in real animals, a sophisticated apparatus would be required, such as a virtual reality setup (Ravassard et al., 2013), or a moving floor plate similar to (Goodridge et al., 1998). Without a model with a proven track record to suggest it, there would be little reason to perform such an experiment.

The principle of slowness and the slow feature analysis algorithm offer a powerful model of hippocampal spatial encoding on an unusual level of abstraction. Without an accessible tool wrapping the access to hierarchical SFA, scientists interested in this principle and its application have to invest a significant amount of work and expertise: the input/output specifications of SFA need to be understood; a hierarchical network needs to be set up using an assortment of building blocks provided by (for example) the MDP library (Zito et al., 2009); the (visual) training data somehow needs to be generated; a computationally expensive training phase

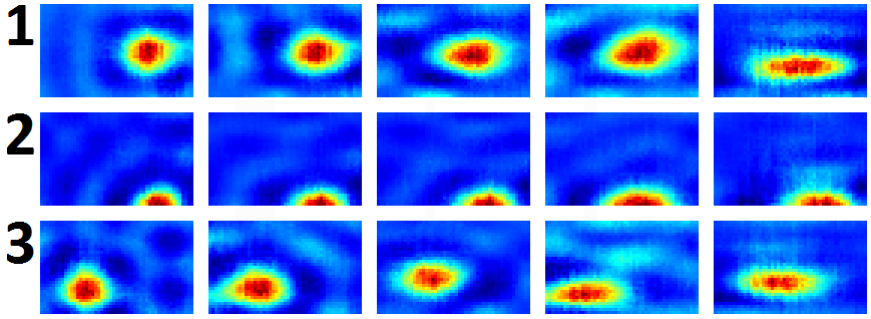

Figure 35: Translation bias effect on place fields. Sampling of SFA signals after training phases during which the agent was placed further along its trajectory when moving in the direction of the x-axis. Rows depict the activity of three different signals. Translation bias is absent in the left-most column and increases up to 1.4 towards the right-most column.

needs to be managed;[14] and, lastly, the trained networks need to be sampled properly to account for directional invariance (or not). With the development of the tools described in this work, however, researchers merely need to understand the idea of the slowness principle, install the software, and evaluate the finished plots depicting an SFA-based spatial representation an hour later.

In most modeling work the methods sections focus on the mathematical background the original software development was based on. While this normally includes enough information to replicate an externally developed program, this is a laborious and tedious process. As an example, while it is straightforward to understand the purpose of the building blocks offered by the `hinet` (hierarchical network) module of MDP – nodes, layers, and in-between wiring switchboards – actually using them requires the user to correctly input the data in terms of number of input channels, number of field channels, field spacing, and channel dimension. While these are by no means insurmountable obstacles, they do, at the very least, require a small measure of genuine programming experience, which, unfortunately, excludes outright large parts of the neuroscience community. Alternatively, if the originally used software is made available (or readily provided upon requested), it is usually not designed to be used by anyone else than the original programmer. Not only does this introduce an additional delay, but also does not eliminate the need for knowledge about programming. If an active and tight collaboration between

[14] The results published in (Franzius et al., 2007) were taken from parallelized simulations that ran on a local computer cluster.

experimental computational neuroscience is desired, then such an approach has to be labeled as inefficient.

While describing implementation details may not be the norm for most publications, there are of course other software packages available that offer access to different approaches of modeling. Brian (Goodman and Brette, 2008; Stimberg et al., 2014) and NEST (Gewaltig and Diesmann, 2007) are well known frameworks to develop simulations of spiking neural networks. Both are software tools that allow the user to set up a neural network and "record" from individual units. While Brian allows the user to specify neural behavior using mathematical equations, NEST provides more than 50 different preset types of neuronal models. The latter also puts a focus on modeling large scale networks and their neural dynamics. At the time of writing, Brian has been in development for about seven years, while NEST was first released in 2004; both are still in active development. Alternatives can be found in GENESIS (Bower and Beeman, 1994) and NEURON (Migliore et al., 2006), two well established simulation environments for single neurons and neural networks. GENESIS was initially released in 1988 and emphasizes its use as a teaching tool; NEURON is supported since 1976 and since 2007 also, in part, funded by the BlueBrain project. There are many more tools available, but as this limited selection already suggests, published software packages focus on the simulation of neural networks and implement different models of neurons to a varying degree of accuracy. In contrast to the software developed in this work, available software solutions also have been developed for much longer and by teams of people instead of a singular developer. As such it is difficult to compare the software developed in this work with other available simulation packages: established software has seen much more development time, but also targets an entirely different type of simulation.

While the ratlab software fulfills the role it was designed for, like any piece of software, it does not come without issues. First, hierarchical SFA is a model based purely on vision. While different input modalities may be added at will in theory, in practice it is difficult to determine what kind of input should be added to the hierarchy at which specific point. Furthermore, the model, as it is made available via the ratlab interface, does neither include any memory processes nor does it allow

for goal oriented behavior. As for the software itself, the main drawback in terms of development is the lack of user testing on any meaningful scale. While the code was developed with usability in mind, and limited user feedback was taken into account, this does, unfortunately, not guarantee a hassle-free user experience. In particular, the code has no way of making sure that a user-provided configuration is physically possible. A user may, for example, construct walls that intersect other walls, or an environment that does not actually leave any space for the virtual agent to navigate within. While such a setup does not make any actual sense, it may be produced by accident in which case users will be presented with rather unhelpful error messages. To alleviate this issue, an FAQ file is included in the software package that lists the most commonly encountered error messages and explains how users may fix the problem.

The software package presented in this work does not, in itself, answer the research questions posed in the introduction of this manuscript. However, it provides the necessary tools to not only address the questions posed here, but also the questions of other researchers about the slowness principle and its role in hippocampal processing. The software meets all the requirements that were set during development and offers a flexible tool to setup and perform simulations examining the spatial representation computed by slow feature analysis. At the time of writing, several students working at different institutions[15] have inquired about the software and the fully annotated source code was gladly received each time (fig. 36).

4.2.2 Behavior of SFA-based spatial representations

The ability of the slow feature analysis algorithm to compute spatial representations that behave similar to what is measured in rodents (O'Keefe and Dostrovsky, 1971) was first demonstrated by (Franzius et al., 2007). The authors focus on the mathematical background of SFA and present the results of a simulation within a simple rectangular environment and a linear track. With both the mathematics of SFA thoroughly examined (Franzius et al., 2007), and the development of the ratlab software package (Schönfeld and Wiskott, 2013), the next step in the process

[15] Chinese Academy of Sciences (China), Dalian University of Technology (China), Jilin University (China), Meiji University (Japan), Technion Israel Institute of Technology (Israel), Technische Universität München (Germany), Zhengzhou University (China).

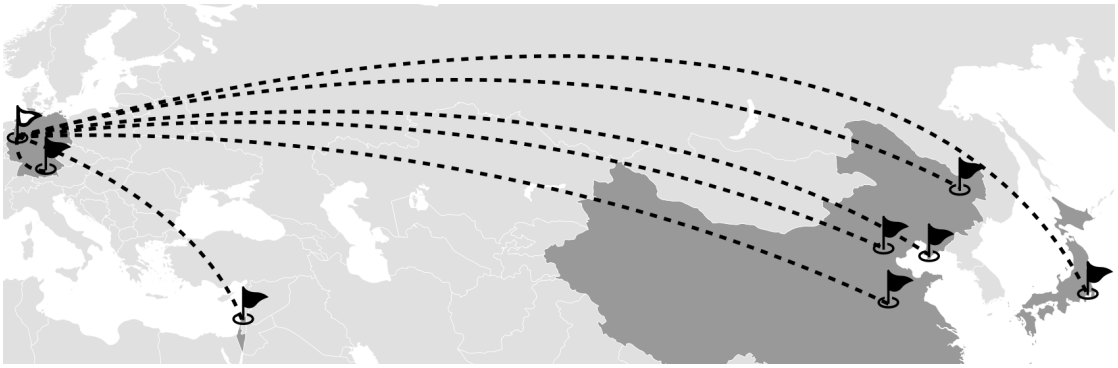

Figure 36: The ratlab software in international use. Origins of several explicitly made requests to use the developed software package.

of establishing the slowness principle as a computational building block in the hippocampus is to compare the behavior of an SFA-based spatial representation with the measurements recorded in experiments with real animals.

Chapters 3.1 and 3.2 present how an SFA-based representation forms over time, and how such a representation adapts to changes in the environment compared to recordings taken from real animals. Six results previously published in the experimental literature were chosen and their protocols replicated in simulation. The selected experiments cover many of the central elements of the hippocampal space code that do not require a higher level of cognition (such as goal oriented behavior): The control exerted by prevalent visual cues, the variable directional sensitivity of place fields depending on prior behavior, and the reaction to changes in an otherwise familiar spatial context. The results presented will be discussed in detail below.

Development time of spatial representations. When training different hierarchical SFA networks with successively longer parts of the same overall trajectory, it can be seen that a stable spatial representation is reached as quickly as within 2-4 minutes of simulated time (fig. 12). To compare this time frame to the experimental literature, the time for place fields to stabilize in an open field is usually given as around 8-12 minutes (Wilson and McNaughton, 1993; Yan et al., 2003). However, recordings from animals exploring a three-arm maze show that fields develop within only 2 minutes and stabilize after about 5-6 minutes (Frank et al., 2004). Can these differences in time scale be reconciled? Fig. 13 (p. 46) shows the

activity produced by a trained network, i.e., a network encoding a fully stabilized spatial representation. However, in contrast to the usual sampling procedure, which gathers network activity over the complete environment, network activity is now only accumulated along the original trajectory of the virtual agent. As can be seen, even a stable representation leads to seemingly unstable place fields, depending on how quickly the agent samples the full extent of the environment. In fact, several commonly reported properties of an "emerging" spatial representation can be observed, such as the apparent merging of multiple subfields, or fields moving around before settling on a final location. Consequently, it would be difficult to confirm that animals establish stable firing fields within 2-4 minutes in the open field, as it takes animals more time than that to adequately sample the environment and provide the data required to support that claim. Note that for place fields to become invariant to direction, it is not enough for animals to pass through a given location only once. Fields have to be traversed several times from more than one direction in order for the animal to be able to gather the required visual input from different angles. In more confined environments, however, such as the three-arm maze used in (Frank et al., 2004), the available space can be sampled faster and thus a spatial representation may be shown to be stable using data acquired over a smaller amount of time. Contrary to living animals, the internal state of a computational model can be frozen and a network may be sampled over a given environment without the visual input passing through the model during that process influencing the stored spatial representation. This may explain the phenomena observed during place field formation as a consequence of the sampling process without the need of introducing an additional mechanism. These results support the hypothesis that a hippocampal spatial representation stabilizes within only 2-4 minutes (fig. 12, p. 45) even though it might take more time than that to actually confirm that with cellular recordings, depending on the shape and size of the environment.

Cue card rotation. The effect of visual cues on hippocampal spatial representations is well documented; their position and relation to each other is a defining feature of place cell activity (Andersen et al., 2006, p. 499). This is not surprising, as animals have to base their internal representation on external cues, and stable visual cues are prime indicators of location – they are not as temporary as (natural)

sounds, not as diffuse and fading as smells, more varied than the floor texture, and not as fleeting as a localized taste. That is not to disregard these input modalities, however, as, after numerous experiments, it has been concluded that "many place cells receive information from more than one sensory input; they can use subsets of this total input to identify correctly the preferred place" (Andersen et al., 2006, p. 499), and studies have shown that other input modalities may take over in the absence of salient visual cues (Zhang and Manahan-Vaughan, 2013). Nevertheless, the spatial representation is strongly bound to visual cues, as demonstrated in (Knierim et al., 1995): Animals were to explore a circular environment with a white cue card as the only available landmark. When the card was rotated in the absence of the animals, their spatial representation followed that rotation upon re-entering the environment. This result should come as no surprise, as the animal can only orient itself using the cues available in environment. It should be noted, however, that visual inspection of the results reported in (Knierim et al., 1995) shows that the place fields clearly undergo more than just a rotation by ninety degrees as their shape does not remain unchanged over the course of the experiment. As the available visual input is strictly controlled and advocates a plain rotation by 90°, this indicates that additional information is taken into account – information that, while keeping the representation from performing a clean rotation, is also overridden by the visual input as the representation ultimately does follow the available visual cues. When replicating this protocol *in silico*, the results are again predictable: spatial activity follows the rotation of the only available visual cue. However, being an entirely artificial simulation, one would assume this rotation to be close to perfect: rotating the original plots should essentially yield the same image than sampling the network after cue rotation. Surprisingly this is not the case though: fig. 14 (p. 49) shows that network activity is not identical before and after cue rotation. While this shows that an SFA-based spatial representation is sensitive to even minuscule changes – a plus – this fortunately does not hint at a trend: representations are remarkably robust in the face of changes in the environments. Additionally, these results indicate that the spatial representations reported in (Knierim et al., 1995) are not based on visual input alone but also take other information into account.

Cue card removal. With the link between spatial representations and the land-

scape of available visual cues established, the immediate follow-up question is to ask how the former adapts to the removal of the latter. In (Hetherington and Shapiro, 1997) rats explore a rectangular environment featuring three cue cards of varying size at different positions along the available wall segments. Upon removal of individual cue cards, the authors report a diminished firing rate in the place cells of their animals and interpret this finding as a measure of distance. That is, the firing rate of a given place cell seems to be lowered by a greater amount the closer the associated place field is located to a now-missing cue card (Hetherington and Shapiro, 1997). Fig. 16 and 17 (p. 51f) show the results of replicating this experimental protocol in simulation. While the first presents a handpicked example that happened to closely match a result presented in the original experimental publication (Hetherington and Shapiro, 1997), the second represents the full range of the behavior of the SFA-based spatial representation – including the reaction of the model to the removal of more than one cue card at a time, which was not done in the original experiment. The latter was included to examine how much an environment can change before the computed spatial representation collapses. As it turns out, the model reacts surprisingly robust even when only the smallest of the three cue cards remains. While the developed place fields are mostly lost at this point, the network reacts to the unfamiliar scene by simply calming down instead of resorting to random or erratic patterns of activity.

Consistent with the hypothesis put forward in (Hetherington and Shapiro, 1997), some of the signals produced by hierarchical SFA do seem to react to the changes in a distance-dependent manner, while others, however, do not: Different place fields can be seen to adapt in ways that do not seem to be based on the distance to a missing cue (e.g., cell 6 in fig. 17). Unfortunately, these observations can only be compared poorly to the full extent of the experimental results, as the original publication only presents a small number of examples of the recorded activity. Nevertheless, the simulation provides the data to offer an alternative explanation: Instead of introducing a new variable – distance – in order to explain the observed results, the elemental functionality of association in the hippocampus may hold the answer. With only visual information available, a "place" can be thought of as a number of different views that can be seen from one location. When standing close to a familiar cue and looking directly at it, the view of an

agent will be dominated by that cue. When standing at the same position but looking away from the cue, however, the visual input will contain only small parts of the cue in question. Consequently, upon removal of that cue, familiarity of the new view with the original one should depend on how much the cue was part of the latter. A place field located directly next to a missing cue should thus be significantly impacted when the agent is facing the position of the cue, but *not* be affected when facing away from it – thus driving down the *average* firing activity of the field over all view directions. This is supported by the data provided by the simulation (fig. 37): When the agent is facing a missing cue, firing fields close to it are left in disarray. When looking away, however, fields can still be seen to encode the same location as before, even if their shape can become noticeably disturbed. The latter can be explained by the wide 320 degree field of view of rats, which implies that, if there are no obstructions, every visual cue will, to some extend, be perceived almost all of the time. As the distance of a place field increases from the nearest cue card – and the cue thus naturally becomes an ever smaller part of the overall view – the effect of the cue card on the field should gradually diminish.

This explanation not only covers the behavior originally observed in (Hetherington and Shapiro, 1997) but can also account for the variety of results presented in fig. 17. In addition, while the distances between cue cards and place fields do play a role in the observed results, they are not required to be actively measured in order to model and explain the observed behavior. This hypothesis can also account for the impact of the the removal of the largest cue card: When compared to firing activity in the original environment, the removal of this cue has a bigger effect on the individual place fields than the removal of any other single cue card. Considering its size, and thus its appearance in virtually any view from within the environment, its removal can be expected to significantly alter the general appearance of the environment. Notably though, place fields located directly next to one of the other cue cards are less affected by this effect; the view from very close to any particular cue is dominated by that card instead.

Spatial activity invariant to direction. Directional invariant place fields, i.e., firing activity sensitive to location but independent of view direction, are commonly observed when rats are allowed to freely explore an open field and are

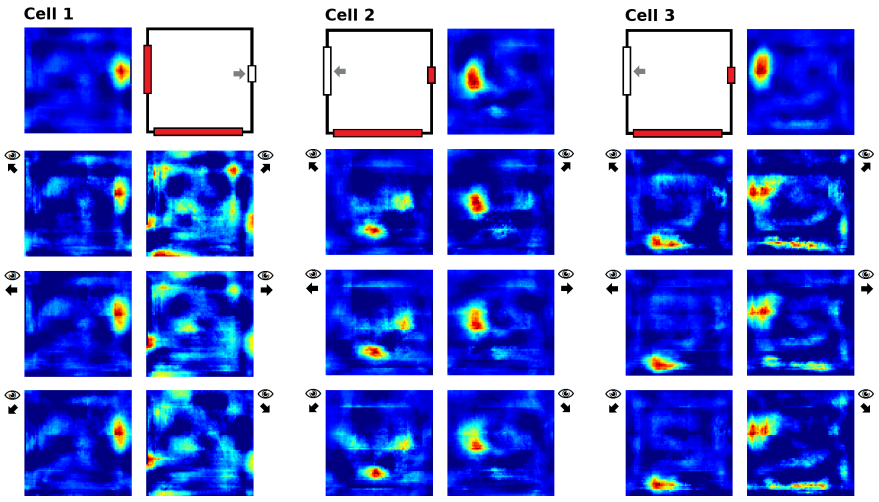

Figure 37: Directional activity after cue removal. Each of the three plot sets (cell 1, cell 2, and cell 3) depicts the activity of one SFA-computed signal sampled over the environment. The black and white diagram shows the position of all cues and indicates which one was removed in each case (arrow). Plots directly next to this diagram show the activity of each cell with the cue cards not removed. Below each of these pairs, six plots depict signal activity over the environment while the agent is either looking in the direction of the missing cue, or looking away from it (black arrows indicate view direction during the sampling process).

a feature of SFA-based spatial representations as well (fig. 7, p. 35). Computing a representation with such a property is not trivial, however. It requires the association of potentially drastically different visual input at the same location, depending on view direction. In rats, this might be explained with their wide field of view, which easily facilitates overlapping views at individual locations. If we presume the hippocampus to perform the same computations across different species, however, then a spatial representation invariant to direction must either not rely on such a wide field of view[16], or simply is not computed in mammals lacking such a feature. Supporting the former is the ability of rats that are born blind, and thus without any field of view to begin with, to establish a perfectly functional spatial representation (Save et al., 1998). Place cells have also been reported in monkeys (Ono et al., 1993), who do not posses a field of view as wide as that of rats. These findings, however, are disputed (Las and Ulanovsky, 2014) and it has been suggested that all measurements of spatial activity in monkeys actually stem from spatial view cells instead of place cells (Georges-Franois et al., 1999). In

[16]The human field of view covers an arc of about 105 degrees of depth perception-enabled vision and about 20-30 additional degrees of peripheral vision on each side.

(Franzius et al., 2007) the authors used a field of view of only 60 degree and were still able to show hierarchical SFA capable of producing a spatial representation and note that "especially rotation invariance is not an effect of a large FOV [field of view]. However, the views have to contain enough visual information." Overall it is currently unknown how widely spread directional invariant place cells actually are among mammals.

Directional activity in the linear track. In contrast to the activity measured when rats freely roam an open field, restricting the animals to run along a narrow corridor results in spatial activity that does, in part, depend on the direction the animal is traveling. This is shown in, for example, (McNaughton et al., 1983), where rats navigated along the corridors of a multiple-arm maze, or (Dombeck et al., 2010), where mice were held in place over an air-cushioned ball, functioning as a treadmill, to navigate a virtual corridor. In both cases the movement of the animal was restricted: the width of the individual corridors of the multiple-arm maze was a mere 5.5 cm, while the virtual reality apparatus fixed the heads of the animals in place during trials. In both studies the authors were able to report place fields that were only active if the animals traversed the corridors in one specific direction. When a virtual animal is placed within a corridor in simulation, and hierarchical SFA is trained with the data resulting from the agent moving back and forth, it can be seen that the model does account for directional spatial activity along a linear track as well (fig. 20, p. 57). Similarly to the results presented in (Dombeck et al., 2010), the SFA-based place fields in the simulation also cover the whole range of the linear track – except for both ends since the agent is not allowed closer than 5 cm to the walls – and about $\frac{1}{3}$ (9 of 32) of the signals are only active when the rat traverses the corridor along one direction, which happens to roughly match the ratio reported in (McNaughton et al., 1983). It should be noted, however, that even though the percentages of silent signals, signals sensitive to direction, and signals invariant to direction match those reported in (McNaughton et al., 1983) surprisingly well, this probably has to be viewed as happenstance. What can be argued though, is that the model does actually account for all three of these classes of signals, which was not entirely predicted. Furthermore, it should be mentioned that the signals depicted in fig. 20 represent all the signals sensitive to direction in the set of signals of this particular simulation, i.e., all signals that

were sensitive to direction were, in this case, sensitive to the same direction.

Directional activity in the open field. The most obvious difference when moving along a narrow corridor as compared to freely roaming an open field is the restrictive geometry and the limitations to the visual field that go hand in hand with it. As it turns out though, those are not the deciding factors that determine place fields to become directional selective. In (Markus et al., 1995) rats were trained to run along a rhombus-shaped path in different open environments. In all of those the authors report direction selective firing fields when analysing recordings from rats running along this familiar path in both a clockwise and counterclockwise fashion (Markus et al., 1995). Within the most featureless of the tested environments, the percentage of directional fields was between 14-18%. In the environment that was replicated in this work, this ratio increased to 20-30%. When performing this experiment in simulation, directional activity along the trained track was found in almost all of the cells to a varying degree (fig. 23, p. 60). Directional activity in clearly defined firing fields was found in 11 of the available 32 signals, 5 signals stayed silent, and the remaining signals showed directional, but largely unstructured activity. These numbers indicate a ratio about 34% of clearly defined spatial activity along the track. However, as in the previous case, the surprising match of these numbers in the simulation and the original experiment are probably nothing an actual argument should be based on. More important than matching percentages is the fact that the simulation is able to account for the different classes of firing behavior. Moreover, the ratio of signals sensitive to direction in the simulation increases with a more visually stimulating environment, which also matches the findings reported in (Markus et al., 1995). While directional activity can clearly be observed within an environment much less restrictive than a narrow linear track, the authors also note that, when comparing their results from an radial 8-arm maze and the open field, "this suggested the possibility that place field directionality is also affected by the physical constraints arising from the geometry of the apparatus" (Markus et al., 1995). If true, it is unclear whether this stems from the constraint in physical movement, or the effect such a restrictive environment has on the visual field: With large portions of the visual field being obscured by the surrounding walls, the visual input along a narrow corridor will be mostly identical. This may lead to any remaining available cues to become

more effective, especially in the case of cue cards mounted at the ends of the corridor. In simulations such cue cards have a marked effect, which does match the reported results that a more visually rich environment facilitates directional activity.

Morphing the environment. The activity of place cells in morphed environments has been the subject of several experiments. The original experiment reported in (Wills et al., 2005) familiarizes rats with two different environments: an open circular arena, and an open rectangular arena. The original goal was to present evidence for the theoretical idea of attractor dynamics, a common approach in modeling memory (and described throughout sections 1.3.1 and 1.3.2). The two familiar environments were suggested to form two distinct attractor states within the memory of the animal. With such attractor representations established, the animals were put in a number of in-between states of the two familiar environments, i.e., environments that were somewhere along the spectrum between the familiar rectangular and circular shapes. The authors recorded from place cells found in the CA1 area of the rat hippocampus and were able to confirm their assumption: along the morph-spectrum, spatial representations switched abruptly from one distinct in-between state to the next – indicating that the spatial representation is caught in one basin of attraction[17] until the environment becomes different enough for the representation to fall into the second basin of attraction. This was the experiment originally replicated in simulation. However, the spatial activity produced by hierarchical SFA does not match the firing pattern reported in (Wills et al., 2005). Instead, in the SFA-based representation, place fields stay in their position as much as the changing geometry along the morph spectrum permits it (fig. 25, p. 62). Signals also have a preferred representation, and are much more active in one of the two familiar environments. The further away the state of the environment gets from their preferred state, the more the overall activity decreases (but does not quite reach silent levels). In other words the SFA-based representation shifts gradually instead of abruptly in between the different states of the environment. This difference might be explained by the fact that hierarchical SFA does not include any recurrent connections – the very element crucial for attractor dynamics. If a gradually shifting representation were to be fed into a

[17] If a network state finds itself within a "basin of attraction" of an attractor, it will converge on it and then stabilize there.

recurrent network a sudden shift could be observed.

However, just shortly after the publication of the original morph study, the experiment was performed again in (Leutgeb et al., 2005a), and this time the authors reported a contradictory finding: instead of the originally reported, sudden switch pointing to attractor dynamics, the spatial representation was shown to gradually shift. Despite an in-depth discussion about the experimental protocols employed in both studies – and pointing out various differences – the authors were unable to pinpoint one particular distinguishing feature that would explain the conflicting observations. One aspect that is not mentioned though, is the makeup of the used apparatus in both studies. While it is noted in (Leutgeb et al., 2005a) that the "nature of the input stimuli" may differ, this does refer to the details of the experimental protocol and not the different apparatuses. In (Wills et al., 2005) the last morph stage before arriving at the fully circular environment is not "almost circular," but a regular octagon. In contrast, the apparatus used in (Leutgeb et al., 2005a) does actually produce a layout that is best described as a somewhat wobbly almost-circle. This is interesting since the signals produced by hierarchical SFA do exhibit their biggest jump in activation from this second to last to the final stage of the morph process – and it is noteworthy that the shape of the environment is, in the end, everything the animals have to distinguish between their potential attractor states. However, while the simulation results match the findings reported in (Leutgeb et al., 2005a) quite well, it does actually emulate the apparatus used in (Wills et al., 2005).

Stretching the environment. In the experiment described in (O'Keefe and Burgess, 1996) the spatial context is modified in a different way. An apparatus consisting of four movable walls is used to record place field activity of rats in four rectangular environments that only differ in their dimensions: 60 by 60 cm, 120 by 60 cm, 60 by 120 cm, and 120 by 120 cm. Place cell recordings in these settings reveal firing fields that react to these changes in a variety of ways: some keep their relative position, some relocate to different positions, while others stretch out to seemingly follow the change in the environment. The remaining cells simply become silent and stop responding altogether. This differs from pure rate or global remapping, as some cells appear to apply the former, while others choose to go with the latter. The firing rate of the cells does also not change uniformly: Some cells

become more active in different variations of the arena, while others decrease their firing rate when comparing the same pair of boxes (O'Keefe and Burgess, 1996). In the original experiment, rats have eight minutes to establish a stable representation in each variation of the base environment. In the simulation, the agent was instead trained with 8 min worth of data in one environment and afterwards sampled in all the variations. This was done since the alternative – training four networks independently of each other in the four environments – would require to manually match the resulting signals of the four simulations. Since the ICA layer at the top of the network does rearrange the signals in a random order, however, and some of the cells were expected to remap to new locations, such a mapping would be impossible to verify. Even with this disadvantage though, the SFA-based spatial representation can still be seen to capture the variety in behavior reported in (O'Keefe and Burgess, 1996): place fields stay in their relative positions, relocate to different positions, and/or stretch in accordance with the environment. In contrast to the original data, cells do not become silent, however, and the firing rate does uniformly decrease when sampled in one of the unfamiliar variations of the environment.

4.2.3 A new general architecture

The architecture presented in section 3.3 embeds hierarchical SFA in a more general framework that is designed to employ the fundamental principles of hippocampal processing and address the shortcomings of the original model. The latter primarily consist of the lack of memory and the difficulty of expanding the model to integrate different input modalities. The architecture operates on a high level of abstraction and is implemented in an algorithmic fashion rather than being based on more low level abstractions such as artificial neurons. At the time of writing the architecture and the results presented provide a proof of concept and demonstrate how a unifying process may be able to support simulations regarding both episodic memory and spatial representations. At this point in development it is irrelevant how the biological hippocampus implements what is here assumed to be its core functionality; the model first needs to demonstrate that *any* implementation of that proposed functionality is able to account for the observations accumulated in the literature. Once this is shown, the different parts of the model may be replaced by more biologically plausible modules and matched to the various areas

of the hippocampal anatomy. However, when introducing such an encompassing model it is not expedient to start at a lower level – say neural networks – if the implemented computational principles have yet to be shown to be of use in the first place. Modeling on a higher level of abstraction may answer this question and allow for an informed decision of how to move forward without having to advance large amounts of resources first. In this way a more abstract modeling approach enables an efficient process of iteration that allows developers to converge on the most capable model possible.

Interestingly, researchers from different backgrounds have arrived at a similar approach. Working in robotics, Micheal Milford developed the hippocampus-inspired RatSLAM algorithm to solve the SLAM (Simultaneous Location And Mapping) problem (Milford et al., 2004). At its core the model records different "poses" – the current view and a path integration-based position estimate – of a robot during navigation. A competitive attractor network of "pose cells" encodes the currently held belief of the position and orientation of the robot at any time. Local view cells encode different views encountered during navigation and are associated with the pose cells to enable navigation. While RatSLAM was never intended to explain hippocampal functionality but to facilitate efficient robot navigation, the basic principle of linking views with pose estimates proved highly successful and lead to the successor CAT-SLAM[18] algorithm (Maddern et al., 2012) which was demonstrated to enable loop closure at large scales (Maddern et al., 2013). In another line of work visual information is also used to correct the accumulated error of an internal path integration system to facilitate self localization (Steinhage and Schöner, 1997, 1998). More precisely, robots are given an out of sight target and use the proposed system to navigate towards it. The original system described in (Steinhage and Schöner, 1997) makes use of path integration as well as predefined visual data to recognize locations encountered on the way. The system is required to be able to make use of such stored views invariant under rotation – an issue that filtering raw visual data through a trained SFA network solves of in the simulations of chapter 3. To integrate path integration and visual information, neural fields (Beurle, 1956; Sandamirskaya, 2014) are used to maintain an estimate of the agents current position. If the current view is unfamiliar, this estimate is based on path

[18] Continuous Appearance-based Trajectory SLAM

integration. Once the robot recognizes the visual input the path integration process is reset and the system estimates location and orientation using its knowledge of the (visual) world. As before, this approach does not aim at explaining hippocampal function but instead attempts to provide the hippocampal-inspired means for a robot agent to navigate the world. Nevertheless, the principles of associating views and integrating the information from different sources/modalities are again shown to enable successful localization.

In contrast to the approaches in robotics, the goal of the architecture presented in this work is to converge on a unifying process in order to help explain hippocampal processing: The establishment and maintenance of spatial representations, association and integration of different input modalities, and formation of episodic memories over time. As a proof of concept, the architecture is shown to solve two fundamentally different tasks: developing a spatial representation during random exploration, and recalling the items of a given list of words. In its current version, the model works with three different kinds of input: visual data processed by hierarchical SFA, path integration information in the form of grid cell activity, and plain words. While the former are implemented to at least yield cell-like output in the form of an activity value, the latter are simple string variables within a computer program. This might seem like an oversimplification, but, as language processing in the human brain is tremendously complex, there are few alternatives other than either a very simple or a very complicated model. In theory, words could be presented as images and filtered through another trained SFA network to simulate a cell population that produced different patterns for individual words. But that is not how words are perceived: They are not a collection of lines, but distinct items learned over time. A more sophisticated model might employ a more plausible process, but ultimately end up working with similar unique identifiers such as plain words.

Similarly to hierarchical SFA, the spatial representation developed by the new architecture (fig. 32) is established within minutes of simulated time. Once the agent crosses the environment for the first time, new grid cell activity is encountered and stored in the form of new percept clusters. Over time, these clusters grow as additional visual input is gathered and attached to them. In the current version

of the model, a multitude of such clusters if formed; when exploring a circular environment for 8 min of simulated time, the model produced a total of 111 percept clusters. Not all of these exhibit the same grade of localized firing. While some clusters react only from the input from within a small area, others react to the input encountered throughout much larger areas of the environment. The exact behavior depends on the chosen threshold values for how similar or different new percepts need to be in order to be accepted or discarded. At the time of writing, these values are set by hand, but for future work it would be desirable for these to be set dynamically and automatically. When replicating the word-list task presented in (Heit et al., 1988), the goal is not to demonstrate the capability of the model to retain string variables – that is a mere consequence of using a computational model. Instead, the model aims to reproduce the firing behavior recorded in the human hippocampus using the same algorithm that developed a spatial representation based on realistic visual input just moments before. Fig. 34 (p. 76) compares the cellular recordings presented in (Heit et al., 1988) with the response values of the new architecture. In both cases, the individual cells/clusters can be seen to have some reaction to most of the words, but clearly exhibit a preferences for one distinct word.

The underlying approach of viewing the hippocampus as a structure to associate and integrate the different input streams from the available input modalities is not a new idea; in fact, it is regularly brought up throughout the experimental literature. It is less of a question of whether this happens, but rather how: Are some input modalities preferred over others? How are conflicts between the input streams handled? Is the priority of each input modality predefined, or based on its statistics, and thus amenable to change over time? And, of course, how can the processes responsible for all this be modeled?

4.3 Open questions and future work

4.3.1 Software development

A common saying in software development is “software isn’t finished but abandoned.” This refers to the fact that virtually every piece of software comes with an open list of possible improvements, additional features, and ways to make it more accessible.

The software toolkit presented in this work is no different and there are numerous avenues for future development:

- **Accessibility:** Even though this has been a concern from the start, accessibility can always be improved further. A more sophisticated user interface to access all of the available parameters, or even a graphical editor to drag and chain together the different modules would be major improvements. The overhead of such an addition should not be underestimated though, and the immediate trade-off is to not spend that development time working on actual features.

- **Error handling:** In its current form, the software is stable and performs reliably. However, if it does fail, error messages are not always descriptive, since much of the error handling is performed by Python itself rather than the actual software. Custom exception handling would be a step towards a more comfortable user experience.

- **Flexibility:** There are several possible features that could be added to the virtual environments to allow for even more flexibility when setting up (or replicating) an experimental protocol in simulation. This includes moving elements during runtime, adding links and triggers to make elements interactive, adding ramps and other elements to model different levels within an environment, or matching the movement of the virtual agent more closely to the movement observed in real animals. Another useful feature would be offscreen rendering, which would allow users to start and perform simulations remotely. As of right now, generating the data does require a local user, as the used OpenGL libraries require access to the physical screen.[19]

- **Information:** In its current version, an agent essentially asks the world what it perceives visually given its current position and direction. In addition to just visuals, the world could also provide the smell at the current position, as well as sounds, the feeling of a ground texture, and other sensory information. While these would actually be straightforward additions to implement, the currently embedded model – hierarchical SFA – has no way of handling them.

[19] In theory, this can be circumvented by rendering the frame buffer to a texture instead of the screen buffer and catching the exception thrown by OpenGL that warns about the missing screen. In practice, this is a hack and should not be part of a software that is distributed to a user base of different backgrounds.

Such features would become relevant if the model would be replaced by the newly introduced architecture, however, which would be able to integrate such information.

4.3.2 Modelling

There is a small number of open questions regarding SFA-based spatial representations. The model seems to have problems when it comes to global remapping and maintaining two distinct spatial representations, it does not include any memory other than the network adaptations developed during training, and is incapable of integrating multiple input modalities in its current form. One way to address these issues would be to extend the original SFA network and add more layers, input channels, and potentially even whole sub-networks. However, since there is no apparent and straightforward way to solve the existing problems in such a fashion, the newly proposed architecture embeds hierarchical SFA within a more general network. This new model is easier to extend and designed to quickly iterate possible solutions to the new questions. Of those, the highest priority is to implement a unifying process that is not only able to integrate different input modalities, but also to accept and perform trials aimed to probe episodic memory function. The results presented in this work show what such a process might look like, and the next steps will be to extend this framework by adding more input modalities (e.g., scent, sound, somatosensation, and possibly taste) in an abstract, yet plausible fashion (e.g., by adding drift to the grid cell system, diffusion to scents, etc.). In addition to such questions of implementation, a number of experiments need to be selected to confirm the model to behave correctly. This means spatial trials that involve multiple input modalities on the one hand, and episodic memory trials that can be replicated in an algorithmic fashion on the other. An example of the former might be the experiment presented in (Zhang and Manahan-Vaughan, 2013), where differently localized scents are shown to take control over the spatial representation in an otherwise featureless environment. The latter – finding appropriate episodic memory trials – might be more of a challenge. First, common trials aimed at humans largely consist of simply asking a person about past experiences and listening to their answer; an approach that would be difficult to mimic *in silico*. Second, trials that map to an algorithmic approach – often because they themselves follow an algorithm already – tend not to actually

target episodic memory, but rather target working memory.[20] One ambitious goal for the architecture would be the ability to replicate experiments performed with animals that are, in turn, aimed at demonstrating the different aspect of episodic memory found in humans (Babb and Crystal, 2005, 2006).

4.4 Conclusions

This work set out to present the slowness principle as a valid candidate for a general principle to be employed in the computation of internal spatial representations as they are found in the rodent hippocampus. To do so, three steps were taken and are presented here:

1. A software framework was developed to allow for a wide variety of spatial experiments to be performed in simulations based on the hierarchical application of the slowness principle to raw visual input data.

2. Slowness-based spatial representations are shown to account for a multitude of observations reported in the experimental literature. This includes the relationship between representations and visual cues; the influence of movement patterns during training on firing properties of cells; and the adaptation of representations to changes in the spatial context.

3. Hierarchical SFA was embedded within a prototype of a general architecture that implements the fundamental hippocampal functions of association, integration, and memory access. The new model is shown to form a spatial representation with the help of the slowness principle, while at the same time being able to replicate cellular activity during a basic episodic memory task.

The results presented here form a strong case for the slowness principle to be involved in the computations forming spatial representations as they are found in the hippocampus. Using nothing but raw visual information, it is possible to establish a stable encoding of the environment and react to environmental modifications in a manner highly similar to that observed in real cells. The main disadvantages of the model can readily be explained by its inherent limitations – such as its inability to function in darkness due to its dependence on visual input.

[20] Remembering an ever increasing set of numbers, for example, or the commonly used n-back task, which requires subjects to remark on the occurrence of a familiar item shown n items earlier.

On the other hand, comparing purely vision-based results with recordings from real animals allows for new conclusions to be drawn about the role of the remaining, non-visual sensory modalities: When rotating cue cards in an otherwise featureless environment, the noisy rotation of the spatial representation demonstrates significant crosstalk from other modalities – even when the experiment is specifically designed to exclude their influence (Knierim et al., 1995). The directional sensitivity of spatial representations seems to be routed as much in behavior as in the surrounding geometry and available cues. And results from experimenting with the spatial context suggest that higher level processes are required to sharply distinguish between similar (across all sensory modalities) but distinctly different environments.

Nevertheless a model able to integrate more than just visual information is desirable. Consequently, the original model was embedded within a more general architecture designed to make use of hierarchical SFA to process visual input and offer the additionally required functionality to address the issues the SFA network is unable to address by itself. Specifically, that means association and integration of several input modalities and bridging the gap between the spatial representations observed in rodents and the episodic memory capabilities examined in humans.

Overall, this work is an example of how theoretical work can fit into the larger, multidisciplinary field of neuroscience: In order to perform any modeling at all, theoreticians rely on the different groups of experimentalists to gather a set of observations that can be studied to identify patterns and access points for modeling work – the spatial activity in the hippocampus is a prime example of this, as the activity of the cells can directly be linked to the performed experiment. Once a hypothesis is formed, the theoretician's tool set is well suited to quickly iterate suitable prototypes until simulation results match experimental observations. Such a confirmed model may then, in return, provide suggestions to experimentalists. Usually these come in the form of promising avenues for future experiments to confirm specific properties that became apparent during the theoretical work – spawning a new wave of experimental work. This continuous cycle means that neither theoretical, nor experimental work is an isolated process; their intertwined nature should not be avoided but embraced if we are to be efficient in our endeavor.

Epilog: ***Cooperating on path integration.*** While an SFA-based spatial representation does not function in the absence of visual input, real place cells still exhibit localized activity in darkness. In this case, animals presumably make use of path integration to update and maintain an estimate of their current position and direction. This process depends on unreliable motor sensors though and the animal's estimate consequently suffers from an accumulating error. Since hierarchical SFA, as used throughout this work, lacks the capability for path integration, the implications of relying on such a process could not be examined directly. However, as path integration is an elemental process in navigation, it still was a topic of some debate. During the course of this work, this lead to a fruitful cooperation with the neighboring department of neurophysiology and Dr. Sijie Zhang. Sijie and myself met several times to discuss the issue of path integration. Using his experimental expertise, our knowledge of the literature, and my theoretical considerations, we developed two experimental protocols to explore our hypotheses: In the first, we examined the diffusion of place fields within a circular environment in darkness and showed how the borders of the environment help partially reset the internal spatial representation of animals (Zhang et al., 2014). The second protocol uses a more elaborate apparatus to examine how a conflict between visual and path information is resolved (at the time of writing, the results of this experiment are still in submission). This exchange lead to a much better understanding of our respective approaches and how we viewed the same problems through the lenses of two very different sets of tools.

Acknowledgements

First of all I need to thank my supervisor Prof. Laurenz Wiskott for his willingness to entertain (and ability to bear) the constant stream of tangents, ideas, and complaints I put out over the years. I am also grateful for his guidance in non-work related matters and will always remember that sometimes risks are just a part of the path. He also flies a mean Firespray-31, which you wouldn't expect when you first met him!

Next have to be the silly people who thought that befriending me would be a worthwhile use of their time: Bianca, I will always wish you and your family only the best and will always be proud of what you have achieved. Arthur, there's nothing I could write here that would be adequate and still be acceptable to publish as a part of my PhD thesis – but luckily I don't have to, since you already know what I was gonna say anyways.

I would also like to explicitly thank the administrative staff for their help regarding all things bureaucratic, as well as their inhuman patience: Angelika Wille (INI), Arno Berg (INI), Frau von Erd (INI), Kathleen Schmidt (INI), Dr. Sabine Dannenberg (SFB), Ursula Heiler (IGSN), Gisela Stephan (IGSN). Also Michael Ziesmer for helping me fix my gaming console (it still works!).

References

Andersen, P., Morris, R., Amaral, D., Bliss, T. V. P., and O'Keefe, J., editors (2006). *The Hippocampus Book*. Oxford University Press.

Babb, S. J. and Crystal, J. D. (2005). Discrimination of what, when, and where: Implications for episodic-like memory in rats. *Learning and Motivation*, 36(2):177–189.

Babb, S. J. and Crystal, J. D. (2006). Episodic-like Memory in the Rat. *Current Biology*, 16(13):1317–1321.

Barrachina, S., Castillo, M., Igual, F., Mayo, R., and Quintana-Ort, E. (2008). Evaluation and tuning of the Level 3 CUBLAS for graphics processors. In *IEEE International Symposium on Parallel and Distributed Processing, 2008. IPDPS 2008*, pages 1–8.

Barry, C., Lever, C., Hayman, R., Hartley, T., Burton, S., O'Keefe, J., Jeffery, K., and Burgess, N. (2006). The boundary vector cell model of place cell firing and spatial memory. *Reviews in the Neurosciences*, 17(1-2):71–97.

Bear, M. F., Connors, B. W., and Paradiso, M. A. (2007). *Neuroscience Exploring the brain*. Lippincott Williams & Wilkins, 3. ed edition.

Berkes, P. (2005). Pattern Recognition with Slow Feature Analysis.

Berkes, P. and Wiskott, L. (2005). Slow feature analysis yields a rich repertoire of complex cell properties. *Journal of Vision*, 5(6):9.

Beurle, R. L. (1956). Properties of a Mass of Cells Capable of Regenerating Pulses. *Philosophical Transactions of the Royal Society of London B: Biological Sciences*, 240(669):55–94.

Bienenstock, E. L., Cooper, L. N., and Munro, P. W. (1982). Theory for the development of neuron selectivity: orientation specificity and binocular interaction in visual cortex. *The Journal of Neuroscience: The Official Journal of the Society for Neuroscience*, 2(1):32–48.

Blackstad, T. W. and Kjaerheim, A. (1961). Special axo-dendritic synapses in the hippocampal cortex: Electron and light microscopic studies on the layer of mossy fibers. *The Journal of Comparative Neurology*, 117(2):133–159.

Blair, H. T. and Sharp, P. E. (1996). Visual and vestibular influences on head-direction cells in the anterior thalamus of the rat. *Behavioral Neuroscience*, 110(4):643–660.

Blaschke, T. and Wiskott, L. (2004). Independent Slow Feature Analysis and Nonlinear Blind Source Separation. In Puntonet, C. G. and Prieto, A., editors, *Independent Component Analysis and Blind Signal Separation*, number 3195 in Lecture Notes in Computer Science, pages 742–749. Springer Berlin Heidelberg.

Bliss, T. V. P. and Collingridge, G. L. (1993). A synaptic model of memory: long-term potentiation in the hippocampus. *Nature*, 361(6407):31–39.

Bliss, T. V. P. and Lomo, T. (1973). Long-lasting potentiation of synaptic transmission in the dentate area of the anaesthetized rabbit following stimulation of the perforant path. *The Journal of Physiology*, 232(2):331–356.

Bower, J. and Beeman, D. (1994). *The Book of GENESIS: Exploring Realistic Neural Models with the GEneral NEural SImulations System*. Springer.

Brown, J. E., Yates, B. J., and Taube, J. S. (2002). Does the vestibular system contribute to head direction cell activity in the rat? *Physiology & Behavior*, 77(4-5):743–748.

Brun, V. H., Solstad, T., Kjelstrup, K. B., Fyhn, M., Witter, M. P., Moser, E. I., and Moser, M.-B. (2008). Progressive increase in grid scale from dorsal to ventral medial entorhinal cortex. *Hippocampus*, 18(12):1200–1212.

Burgess, N., Becker, S., King, J. A., and O'Keefe, J. (2001). Memory for events and their spatial context: models and experiments. *Philosophical Transactions of the Royal Society of London. Series B, Biological Sciences*, 356(1413):1493–1503.

Burn, C. C. (2008). What is it like to be a rat? Rat sensory perception and its implications for experimental design and rat welfare. *Applied Animal Behaviour Science*, 112(12):1–32.

Buzsaki, G. (2005). Theta rhythm of navigation: Link between path integration and landmark navigation, episodic and semantic memory. *Hippocampus*, 15(7):827–840.

Carew, T. J., Castellucci, V. F., and Kandel, E. R. (1971). An Analysis of Dishabituation and Sensitization of The Gill-Withdrawal Reflex In Aplysia. *International Journal of Neuroscience*, 2(2):79–98.

Cheng, S. (2013). The CRISP theory of hippocampal function in episodic memory. *Frontiers in Neural Circuits*, 7:88.

Cl., M. M. and C., N. J. (1989). Catastrophic Interference in Connectionist Networks: The Sequential Learning Problem. *Psychology of Learning and Motivation*, 24:109–164.

Dayan, P. and Willshaw, D. J. (1991). Optimising synaptic learning rules in linear associative memories. *Biological Cybernetics*, 65(4):253–265.

Derdikman, D., Whitlock, J. R., Tsao, A., Fyhn, M., Hafting, T., Moser, M.-B., and Moser, E. I. (2009). Fragmentation of grid cell maps in a multicompartment environment. *Nature Neuroscience*, 12(10):1325–1332.

Dere, E., Kart-Teke, E., Huston, J. P., and De Souza Silva, M. A. (2006). The case for episodic memory in animals. *Neuroscience and Biobehavioral Reviews*, 30(8):1206–1224.

Desjardins, C., Maruniak, J. A., and Bronson, F. H. (1973). Social rank in house mice: differentiation revealed by ultraviolet visualization of urinary marking patterns. *Science*, 182(4115):939–941.

D'Hooge, R. and De Deyn, P. P. (2001). Applications of the Morris water maze in the study of learning and memory. *Brain Research. Brain Research Reviews*, 36(1):60–90.

Dombeck, D. A., Harvey, C. D., Tian, L., Looger, L. L., and Tank, D. W. (2010). Functional imaging of hippocampal place cells at cellular resolution during virtual navigation. *Nature Neuroscience*, 13(11):1433–1440.

Eichenbaum, H. (2014). Time cells in the hippocampus: a new dimension for mapping memories. *Nature Reviews. Neuroscience*, 15(11):732–744.

Ekstrom, A. D., Kahana, M. J., Caplan, J. B., Fields, T. A., Isham, E. A., Newman, E. L., and Fried, I. (2003). Cellular networks underlying human spatial navigation. *Nature*, 425(6954):184–188.

Escalante-B, A. N. and Wiskott, L. (2010). Gender and Age Estimation from Synthetic Face Images. In *Computational Intelligence for Knowledge-Based Systems Design*, number 6178 in Lecture Notes in Computer Science, pages 240–249. Springer Berlin Heidelberg.

Escalante-B, A. N. and Wiskott, L. (2011). Heuristic Evaluation of Expansions for Non-linear Hierarchical Slow Feature Analysis. In *2011 10th International Conference on Machine Learning and Applications and Workshops (ICMLA)*, volume 1, pages 133–138.

Escalante-B, A. N. and Wiskott, L. (2013). How to Solve Classification and Regression Problems on High-Dimensional Data with a Supervised Extension of Slow Feature Analysis. *Journal of Machine Learning Research*, 14:36833719.

Etienne, A. S. and Jeffery, K. J. (2004). Path integration in mammals. *Hippocampus*, 14(2):180–192.

Fletcher, M. C. (2012). PyOpenGL 3.x the Python OpenGL binding. [Online; accessed 5th February 2016].

Földiak, P. (1991). Learning Invariance from Transformation Sequences. *Neural Computation*, 3(2):194–200.

Fouquet, C., Tobin, C., and Rondi-Reig, L. (2010). A new approach for modeling episodic memory from rodents to humans: the temporal order memory. *Behavioural Brain Research*, 215(2):172–179.

Frank, L. M., Stanley, G. B., and Brown, E. N. (2004). Hippocampal plasticity across multiple days of exposure to novel environments. *The Journal of Neuroscience: The Official Journal of the Society for Neuroscience*, 24(35):7681–7689.

Franzius, M., Sprekeler, H., and Wiskott, L. (2007). Slowness and Sparseness Lead to Place, Head-Direction, and Spatial-View Cells. *PLoS Comput Biol*, 3(8):e166.

Franzius, M., Wilbert, N., and Wiskott, L. (2011). Invariant object recognition and pose estimation with slow feature analysis. *Neural Computation*, 23(9):2289–2323.

Fuhs, M. C. and Touretzky, D. S. (2000). Synaptic learning models of map separation in the hippocampus. *Neurocomputing*, 3233:379–384.

Fyhn, M., Hafting, T., Treves, A., Moser, M.-B., and Moser, E. I. (2007). Hippocampal remapping and grid realignment in entorhinal cortex. *Nature*, 446(7132):190–194.

Georges-Franois, P., Rolls, E. T., and Robertson, R. G. (1999). Spatial view cells in the primate hippocampus: allocentric view not head direction or eye position or place. *Cerebral Cortex (New York, N.Y.: 1991)*, 9(3):197–212.

Gewaltig, M.-O. and Diesmann, M. (2007). Nest (neural simulation tool). *Scholarpedia*, 2(4):1430.

Giocomo, L. M., Stensola, T., Bonnevie, T., Van Cauter, T., Moser, M.-B., and Moser, E. I. (2014). Topography of Head Direction Cells in Medial Entorhinal Cortex. *Current Biology*, 24(3):252–262.

Goodman, D. F. M. and Brette, R. (2008). Brian: a simulator for spiking neural networks in Python. *Frontiers in Neuroinformatics*, 2:5.

Goodridge, J. P., Dudchenko, P. A., Worboys, K. A., Golob, E. J., and Taube, J. S. (1998). Cue control and head direction cells. *Behavioral Neuroscience*, 112(4):749–761.

Gross, C. G. (2002). Genealogy of the "grandmother cell". *The Neuroscientist: A Review Journal Bringing Neurobiology, Neurology and Psychiatry*, 8(5):512–518.

Hafting, T., Fyhn, M., Molden, S., Moser, M.-B., and Moser, E. I. (2005). Microstructure of a spatial map in the entorhinal cortex. *Nature*, 436(7052):801–806.

Hampton, R. R. and Schwartz, B. L. (2004). Episodic memory in nonhumans: what, and where, is when? *Current Opinion in Neurobiology*, 14(2):192–197.

Hayman, R., Verriotis, M. A., Jovalekic, A., Fenton, A. A., and Jeffery, K. J. (2011). Anisotropic encoding of three-dimensional space by place cells and grid cells. *Nature Neuroscience*, 14(9):1182–1188.

Heit, G., Smith, M. E., and Halgren, E. (1988). Neural encoding of individual words and faces by the human hippocampus and amygdala. *Nature*, 333(6175):773–775.

Hertz, J., Palmer, R. G., and Krogh, A. S. (1991). *Introduction to the Theory of Neural Computation*. Perseus Publishing, 1st edition.

Hetherington, P. A. and Shapiro, M. L. (1997). Hippocampal place fields are altered by the removal of single visual cues in a distance-dependent manner. *Behavioral Neuroscience*, 111(1):20–34.

Hinton, G. E. (1989). Connectionist learning procedures. *Artificial Intelligence*, 40(13):185–234.

Hughes, A. (1979). A schematic eye for the rat. *Vision Research*, 19(5):569–588.

Hunter, J. D. (2007). Matplotlib: A 2d graphics environment. *Computing in Science & Engineering*, 9(3):90–95.

Hyvarinen, A. (1999). Fast and robust fixed-point algorithms for independent component analysis. *IEEE Transactions on Neural Networks*, 10(3):626–634.

Izhikevich, E. M. (2005). Large-Scale Simulation of the Human Brain.

Jacobs, J., Weidemann, C. T., Miller, J. F., Solway, A., Burke, J. F., Wei, X.-X., Suthana, N., Sperling, M. R., Sharan, A. D., Fried, I., and Kahana, M. J. (2013). Direct recordings of grid-like neuronal activity in human spatial navigation. *Nature Neuroscience*, 16(9):1188–1190.

Jensen, O. and Lisman, J. E. (2005). Hippocampal sequence-encoding driven by a cortical multi-item working memory buffer. *Trends in Neurosciences*, 28(2):67–72.

Jonas, P., Major, G., and Sakmann, B. (1993). Quantal components of unitary EPSCs at the mossy fibre synapse on CA3 pyramidal cells of rat hippocampus. *The Journal of Physiology*, 472:615–663.

Jones, E., Oliphant, T. E., Peterson, P., et al. (2001). SciPy: Open source scientific tools for Python. [Online; accessed 5th February 2016].

Kelly, J. B. and Masterton, B. (1977). Auditory sensitivity of the albino rat. *Journal of Comparative and Physiological Psychology*, 91(4):930–936.

Kilgard, M. J. (1996). The OpenGL Utility Toolkit (GLUT) Programming Interface.

Knierim, J. J., Kudrimoti, H. S., and McNaughton, B. L. (1995). Place cells, head direction cells, and the learning of landmark stability. *The Journal of Neuroscience: The Official Journal of the Society for Neuroscience*, 15(3 Pt 1):1648–1659.

Kropff, E., Carmichael, J. E., Moser, M.-B., and Moser, E. I. (2015). Speed cells in the medial entorhinal cortex. *Nature*, 523(7561):419–424.

Kli, S. and Dayan, P. (2000). The Involvement of Recurrent Connections in Area CA3 in Establishing the Properties of Place Fields: a Model. *The Journal of Neuroscience*, 20(19):7463–7477.

Las, L. and Ulanovsky, N. (2014). Hippocampal Neurophysiology Across Species. In Derdikman, D. and Knierim, J. J., editors, *Space, Time and Memory in the Hippocampal Formation*, pages 431–461. Springer Vienna.

Le, Q., Ranzato, M. A., Monga, R., Devin, M., Chen, K., Corrado, G., Dean, J., and Ng, A. (2012). Building high-level features using large scale unsupervised learning. In *Proceedings of the 29th International Conference on Machine Learning*, pages 81–88. ACM.

Leutgeb, J. K., Leutgeb, S., Treves, A., Meyer, R., Barnes, C. A., McNaughton, B. L., Moser, M.-B., and Moser, E. I. (2005a). Progressive Transformation of Hippocampal Neuronal Representations in Morphed Environments. *Neuron*, 48(2):345–358.

Leutgeb, S., Leutgeb, J. K., Barnes, C. A., Moser, E. I., McNaughton, B. L., and Moser, M.-B. (2005b). Independent Codes for Spatial and Episodic Memory in Hippocampal Neuronal Ensembles. *Science*, 309(5734):619–623.

Lever, C., Burton, S., Jeewajee, A., O'Keefe, J., and Burgess, N. (2009). Boundary Vector Cells in the Subiculum of the Hippocampal Formation. *The Journal of Neuroscience*, 29(31):9771–9777.

Lomo, T. (2003). The discovery of long-term potentiation. *Philosophical Transactions of the Royal Society B: Biological Sciences*, 358(1432):617–620.

Löwel, S. and Singer, W. (1992). Selection of intrinsic horizontal connections in the visual cortex by correlated neuronal activity. *Science*, 255(5041):209–212.

Lubenov, E. V. and Siapas, A. G. (2009). Hippocampal theta oscillations are travelling waves. *Nature*, 459(7246):534–539.

Lynch, G. S., Dunwiddie, T., and Gribkoff, V. (1977). Heterosynaptic depression: a postsynaptic correlate of long-term potentiation. *Nature*, 266(5604):737–739.

MacDonald, C. J., Lepage, K. Q., Eden, U. T., and Eichenbaum, H. (2011). Hippocampal Time Cells Bridge the Gap in Memory for Discontiguous Events. *Neuron*, 71(4):737–749.

Maddern, W., Milford, M., and Wyeth, G. (2012). CAT-SLAM: probabilistic localisation and mapping using a continuous appearance-based trajectory. *The International Journal of Robotics Research*, 31(4):429–451.

Maddern, W., Milford, M., and Wyeth, G. (2013). Towards persistent localization and mapping with a continuous appearance-based topology. In Roy, N., Newman, P., and Srinivasa, S., editors, *Robotics: Science and Systems VIII*, pages 281–288. MIT Press.

Markus, E. J., Barnes, C. A., McNaughton, B. L., Gladden, V. L., and Skaggs, W. E. (1994). Spatial information content and reliability of hippocampal CA1 neurons: effects of visual input. *Hippocampus*, 4(4):410–421.

Markus, E. J., Qin, Y. L., Leonard, B., Skaggs, W. E., McNaughton, B. L., and Barnes, C. A. (1995). Interactions between location and task affect the spatial and directional firing of hippocampal neurons. *The Journal of Neuroscience: The Official Journal of the Society for Neuroscience*, 15(11):7079–7094.

Marr, D. (1971). Simple memory: a theory for archicortex. *Philosophical Transactions of the Royal Society of London. Series B, Biological Sciences*, 262(841):23–81.

Mather, J. G. and Baker, R. R. (1981). Magnetic sense of direction in woodmice for route-based navigation. *Nature*, 291(5811):152–155.

McClelland, J. L., McNaughton, B. L., and O'Reilly, R. C. (1995). Why there are complementary learning systems in the hippocampus and neocortex: insights from the successes and failures of connectionist models of learning and memory. *Psychological Review*, 102(3):419–457.

McNaughton, B. L., Barnes, C. A., Gerrard, J. L., Gothard, K., Jung, M. W., Knierim, J. J., Kudrimoti, H., Qin, Y., Skaggs, W. E., Suster, M., and Weaver, K. L. (1996). Deciphering the hippocampal polyglot: the hippocampus as a path integration system. *Journal of Experimental Biology*, 199(1):173–185.

McNaughton, B. L., Barnes, C. A., and O'Keefe, J. (1983). The contributions of position, direction, and velocity to single unit activity in the hippocampus of freely-moving rats. *Experimental Brain Research*, 52(1):41–49.

McNaughton, B. L., Battaglia, F. P., Jensen, O., Moser, E. I., and Moser, M.-B. (2006). Path integration and the neural basis of the 'cognitive map'. *Nature Reviews Neuroscience*, 7(8):663–678.

McNaughton, B. L. and Morris, R. G. M. (1987). Hippocampal synaptic enhancement and information storage within a distributed memory system. *Trends in Neurosciences*, 10(10):408–415.

Migliore, M., Cannia, C., Lytton, W., Markram, H., and Hines, M. L. (2006). Parallel Network Simulations with NEURON. *Journal of computational neuroscience*, 21(2):119–129.

Milford, M. J., Wyeth, G. F., and Prasser, D. (2004). RatSLAM: a hippocampal model for simultaneous localization and mapping. In *2004 IEEE International Conference on Robotics and Automation, 2004. Proceedings. ICRA '04*, volume 1, pages 403–408 Vol.1.

Mizuseki, K., Royer, S., Diba, K., and Buzsaki, G. (2012). Activity dynamics and behavioral correlates of CA3 and CA1 hippocampal pyramidal neurons. *Hippocampus*, 22(8):1659–1680.

Morton, J., Hammersley, R. H., and Bekerian, D. A. (1985). Headed records: A model for memory and its failures. *Cognition*, 20(1):1–23.

Müller, M. and Wehner, R. (1988). Path integration in desert ants, Cataglyphis fortis. *Proceedings of the National Academy of Sciences of the United States of America*, 85(14):5287–5290.

Nadel, L. and Moscovitch, M. (1997). Memory consolidation, retrograde amnesia and the hippocampal complex. *Current Opinion in Neurobiology*, 7(2):217–227.

Nickolls, J., Buck, I., Garland, M., and Skadron, K. (2008). Scalable Parallel Programming with CUDA. *Queue*, 6(2):40–53.

O'Keefe, J. and Burgess, N. (1996). Geometric determinants of the place fields of hippocampal neurons. *Nature*, 381(6581):425–428.

O'Keefe, J. and Dostrovsky, J. (1971). The hippocampus as a spatial map. Preliminary evidence from unit activity in the freely-moving rat. *Brain Research*, 34(1):171–175.

Oliphant, T. E. (2007). Python for scientific computing. *Computing in Science & Engineering*, 9(3):10–20.

Ono, T., Nakamura, K., Nishijo, H., and Eifuku, S. (1993). Monkey hippocampal neurons related to spatial and nonspatial functions. *Journal of Neurophysiology*, 70(4):1516–1529.

Pause, B. M., Zlomuzica, A., Kinugawa, K., Mariani, J., Pietrowsky, R., and Dere, E. (2013). Perspectives on episodic-like and episodic memory. *Frontiers in Behavioral Neuroscience*, 7:33.

Preston, A. R. and Eichenbaum, H. (2013). Interplay of Hippocampus and Prefrontal Cortex in Memory. *Current Biology*, 23(17):R764–R773.

Quirk, G. J., Muller, R. U., and Kubie, J. L. (1990). The firing of hippocampal place cells in the dark depends on the rat's recent experience. *The Journal of Neuroscience: The Official Journal of the Society for Neuroscience*, 10(6):2008–2017.

Quiroga, R. Q., Reddy, L., Kreiman, G., Koch, C., and Fried, I. (2005). Invariant visual representation by single neurons in the human brain. *Nature*, 435(7045):1102–1107.

Raaijmakers, J. G. and Shiffrin, R. M. (1981). Search of associative memory. *Psychological Review*, 88(2):93–134.

Ravassard, P., Kees, A., Willers, B., Ho, D., Aharoni, D., Cushman, J., Aghajan, Z. M., and Mehta, M. R. (2013). Multisensory Control of Hippocampal Spatiotemporal Selectivity. *Science*, 340(6138):1342–1346.

Redish, A. D. and Touretztky, D. S. (1998). The Role of the Hippocampus in the Morris Water Maze. In Bower, J. M., editor, *Computational Neuroscience*, pages 101–106. Springer US.

Rodriguez, F., Lpez, J. C., Vargas, J. P., Broglio, C., Gmez, Y., and Salas, C. (2002). Spatial memory and hippocampal pallium through vertebrate evolution: insights from reptiles and teleost fish. *Brain Research Bulletin*, 57(3-4):499–503.

Rolls, E. T. (1996). A theory of hippocampal function in memory. *Hippocampus*, 6(6):601–620.

Rossum, G. (1995). Python reference manual. Technical report, CWI (Centre for Mathematics and Computer Science), Amsterdam, Netherlands.

Samsonovich, A. and McNaughton, B. L. (1997). Path Integration and Cognitive Mapping in a Continuous Attractor Neural Network Model. *The Journal of Neuroscience*, 17(15):5900–5920.

Sandamirskaya, Y. (2014). Dynamic neural fields as a step toward cognitive neuromorphic architectures. *Neuromorphic Engineering*, 7:276.

Save, E., Cressant, A., Thinus-Blanc, C., and Poucet, B. (1998). Spatial firing of hippocampal place cells in blind rats. *The Journal of Neuroscience: The Official Journal of the Society for Neuroscience*, 18(5):1818–1826.

Schönfeld, F. and Wiskott, L. (2013). RatLab: an easy to use tool for place code simulations. *Frontiers in Computational Neuroscience*, 7:104.

Sharp, P. E. (2013). Computer simulation of hippocampal place cells. *Psychobiology*, 19(2):103–115.

Sharp, P. E., Tinkelman, A., and Cho, J. (2001). Angular velocity and head direction signals recorded from the dorsal tegmental nucleus of gudden in the rat: implications for path integration in the head direction cell circuit. *Behavioral Neuroscience*, 115(3):571–588.

Shreiner, D. and Group, T. K. O. A. W. (2009). *OpenGL Programming Guide: The Official Guide to Learning OpenGL, Versions 3.0 and 3.1.* Addison-Wesley Professional, 7th edition.

Skaggs, W. E., Knierim, J. J., Kudrimoti, H. S., and McNaughton, B. L. (1995). A Model of the Neural Basis of the Rat\textquotesingle s Sense of Direction. In Tesauro, G., Touretzky, D. S., and Leen, T. K., editors, *Advances in Neural Information Processing Systems 7*, pages 173–180. MIT Press.

Stackman, R. W. and Taube, J. S. (1998). Firing properties of rat lateral mammillary single units: head direction, head pitch, and angular head velocity. *The Journal of Neuroscience: The Official Journal of the Society for Neuroscience*, 18(21):9020–9037.

Steinhage, A. and Schöner, G. (1997). Self-calibration based on invariant view recognition: Dynamic approach to navigation. *Robotics and Autonomous Systems*, 20(24):133–156.

Steinhage, A. and Schöner, G. (1998). Dynamical systems for the behavioral organization of autonomous robot navigation. volume 3523, pages 169–180.

Stimberg, M., Goodman, D. F. M., Benichoux, V., and Brette, R. (2014). Equation-oriented specification of neural models for simulations. *Frontiers in Neuroinformatics*, 8(6).

Taube, J. S. (1995). Head direction cells recorded in the anterior thalamic nuclei of freely moving rats. *The Journal of Neuroscience: The Official Journal of the Society for Neuroscience*, 15(1 Pt 1):70–86.

Taube, J. S., Muller, R. U., and Ranck, J. B. (1990a). Head-direction cells recorded from the postsubiculum in freely moving rats. I. Description and quantitative analysis. *The Journal of Neuroscience*, 10(2):420–435.

Taube, J. S., Muller, R. U., and Ranck, J. B. (1990b). Head-direction cells recorded from the postsubiculum in freely moving rats. II. Effects of environmental manipulations. *The Journal of Neuroscience*, 10(2):436–447.

Templer, V. L. and Hampton, R. R. (2013). Episodic memory in nonhuman animals. *Current biology: CB*, 23(17):R801–806.

Treves, A. and Rolls, E. T. (1992). Computational constraints suggest the need for two distinct input systems to the hippocampal CA3 network. *Hippocampus*, 2(2):189–199.

Treves, A. and Rolls, E. T. (1994). Computational analysis of the role of the hippocampus in memory. *Hippocampus*, 4(3):374–391.

Tulving, E. (1984). Precis of Elements of episodic memory. *Behavioral and Brain Sciences*, 7(02):223–238.

van der Walt, S. (2011). The numpy array: A structure for efficient numerical computation. *Computing in Science & Engineering*, 13(2):22–30.

Vorhees, C. V. and Williams, M. T. (2006). Morris water maze: procedures for assessing spatial and related forms of learning and memory. *Nature Protocols*, 1(2):848–858.

Warren, M., Salmon, J., Becker, D., Goda, M., Sterling, T., and Winckelmans, W. (1997). Pentium Pro Inside: I. A Treecode at 430 Gigaflops on ASCI Red, II. Price/Performance of $50/Mflop on Loki and Hyglac. In *Supercomputing, ACM/IEEE 1997 Conference*, pages 61–61.

West, M. J. (1990). Stereological studies of the hippocampus: a comparison of the hippocampal subdivisions of diverse species including hedgehogs, laboratory rodents, wild mice and men. In *Progress in Brain Research*, volume 83, pages 13–36. Elsevier.

Wiener, S. I. (1993). Spatial and behavioral correlates of striatal neurons in rats performing a self-initiated navigation task. *The Journal of Neuroscience: The Official Journal of the Society for Neuroscience*, 13(9):3802–3817.

Wills, T. J., Lever, C., Cacucci, F., Burgess, N., and O'Keefe, J. (2005). Attractor dynamics in the hippocampal representation of the local environment. *Science*, 308(5723):873–876.

Willshaw, D. J., Dayan, P., and Morris, R. G. M. (2015). Memory, modelling and Marr: a commentary on Marr (1971) Simple memory: a theory of archicortex. *Phil. Trans. R. Soc. B*, 370(1666):20140383.

Wilson, M. A. and McNaughton, B. L. (1993). Dynamics of the hippocampal ensemble code for space. *Science*, 261(5124):1055–1058.

Wiskott, L. and Sejnowski, T. J. (2002). Slow feature analysis: unsupervised learning of invariances. *Neural Computation*, 14(4):715–770.

Wyss, R., König, P., and Verschure, P. F. M. J. (2006). A Model of the Ventral Visual System Based on Temporal Stability and Local Memory. *PLoS Biol*, 4(5):e120.

Yan, J., Zhang, Y., Roder, J., and McDonald, R. J. (2003). Aging effects on spatial tuning of hippocampal place cells in mice. *Experimental Brain Research*, 150(2):184–193.

Yartsev, M. M. and Ulanovsky, N. (2013). Representation of Three-Dimensional Space in the Hippocampus of Flying Bats. *Science*, 340(6130):367–372.

Yartsev, M. M., Witter, M. P., and Ulanovsky, N. (2011). Grid cells without theta oscillations in the entorhinal cortex of bats. *Nature*, 479(7371):103–107.

Zhang, K. (1996). Representation of spatial orientation by the intrinsic dynamics of the head-direction cell ensemble: a theory. *The Journal of Neuroscience: The Official Journal of the Society for Neuroscience*, 16(6):2112–2126.

Zhang, S. and Manahan-Vaughan, D. (2013). Spatial Olfactory Learning Contributes to Place Field Formation in the Hippocampus. *Cerebral Cortex*, page bht239.

Zhang, S., Schönfeld, F., Wiskott, L., and Manahan-Vaughan, D. (2014). Spatial representations of place cells in darkness are supported by path integration and border information. *Frontiers in Behavioral Neuroscience*, 8:222.

Zito, T., Wilbert, N., Wiskott, L., and Berkes, P. (2009). Modular toolkit for Data Processing (MDP): a Python data processing framework. *Frontiers in Neuroinformatics*, 2:8.

Ziv, Y., Burns, L. D., Cocker, E. D., Hamel, E. O., Ghosh, K. K., Kitch, L. J., El Gamal, A., and Schnitzer, M. J. (2013). Long-term dynamics of CA1 hippocampal place codes. *Nature neuroscience*, 16(3):264–266.

Curriculum Vitae

Name: Fabian Schoenfeld

Born: Nuremberg, July 25th, 1984

Email: fabian.schoenfeld@ini.ruhr-uni-bochum.de

Education

Ruhr-University Bochum — **PhD student** — Bochum, Germany
July 2010 – Present
PhD thesis at the *Institute for Neural Computation* and the *International Graduate School of Neuroscience* (IGSN).

FAU Chair for Computer Graphics — **Research assistant** — Erlangen, Germany
October 2009 – February 2010

Friedrich-Alexander University (FAU) — **Diploma student** — Erlangen, Germany
October 2004 – October 2009
Computer science with secondary subject engineering.

Diploma (grade: 1.1)
Thesis: *"Interactive debugging tool for the PE physics engine"* (grade: 1.0)
Oral exams: Artificial intelligence (grade: 1.3); Computer graphics (grade: 1.0); Programming languages and methods (grade: 1.3), Engineering (grade: 1.0)
Studienarbeit: *"A parallel 3-SAT solver on CUDA"* (grade: 1.0)

Teaching experience

Ruhr-University Bochum — **PhD student** — Bochum, Germany
September 2012 – June 2013
Bachelor thesis supervision (x3)

Schüllerkolleg Schwabach — **Tutor** — Schwabach, Germany
March 2007 – June 2010
(Mathematics, Physics)

Friedrich-Alexander University — **Instructor** — Erlangen, Germany
October 2007 – February 2008
(Algorithms, Java)

FAU Engineering Department — **Advisor** — Erlangen, Germany
April 2008 – July 2008
(Java programming)

Skills

Languages German (native), English (fluent), French (elementary)

Programming
Python, C++ (fluent)
C, CUDA, Java, VHDL (proficient)
Assembler, Scheme, Prolog (familiar)

Education 3D-CAD, Algorithm design, artificial intelligence, computer graphics, GPU programming with CUDA, hardware (FPGA) programming, LaTeX, machine learning, object-oriented programming, parallel computing, software development.

Soft skills Activity management, sociable in professional settings, problem analysis, task organization & coordination, research & information gathering.

Awards & Certificates

- Outstanding performance in Algorithm Design (award, Erlangen)
- Advanced scientific programming in Python (Summer school, Zurich)
- Scientific writing DAAD spring school (workshop, Bochum)
- Science communication and media skills training (workshop, Bochum)
- Intercultural communication: How to work together in an inter. team (workshop, Bochum)

Publications

Fabian Schoenfeld and Laurenz Wiskott (2015) Modeling place field activity with hierarchical slow feature analysis. *Frontiers in Computational Neuroscience*, 9:51.

Sijie Zhang, Fabian Schoenfeld, Laurenz Wiskott, and Denise Manahan-Vaughan (2014) Spatial representations of place cells in darkness are supported by path integration and border information, *Frontiers in Behavioral Neuroscience*, 8:222.

Fabian Schoenfeld and Laurenz Wiskott (2013) RatLab: an easy to use tool for place code simulations. *Frontiers in Computational Neuroscience*, 7:104.

Fabian Schoenfeld, Quirin Meyer, Marc Stamminger and Rolf Wanka (2010) 3-SAT on CUDA: Towards a massively parallel SAT solver. *High Performance Computing and Simulation (HPCS)*, Caen, France, 306-313.

Abstracts / posters:

Fabian Schoenfeld and Laurenz Wiskott (2012) Sensory integration of place and head-direction cells in a virtual environment. *NeuroVisionen 8*, Aachen, Germany.

Fabian Schoenfeld and Laurenz Wiskott (2012) Sensory integration of place and head-direction cells in a virtual environment. *8th FENS Forum of Neuroscience*, Barcelona, Spain.

Fabian Schoenfeld (2010) Der Physik-Engine Editor ped. *Informatiktage 2010*, Bonn, 97-100.

In submission:

Sijie Zhang, Fabian Draht, Fabian Schoenfeld, Laurenz Wiskott, and Denise Manahan-Vaughan (2016) Spatial representations by place fields in darkness in the absence of reliable external sensory cues is supported by path integration.

Transcript of records

(Lectures with no. of semesters and lecturers)

Programming **Algorithms** (I-III, Stoyan, Wilke, and Greiner), **Cluster computing** (Veldema), **Computer graphics** (Grosso), **Interactive computer graphics** (Stamminger), **Parallel algorithms** (Valdema), **Software systems** (I-III, Kleinöder, Meyer-Wegener, and Saglietti).

Theoretical **Advanced graphics algorithms** (Dachsbacher), **Artificial intelligence** (I-II, Stoyan), **Computer engineering** (I-IV, Teich, Huemer, and German), **Geometric modeling** (Greiner), **Mathematics for engineers** (I-IV, Klamroth, Strauß and Graef), **Theoretical computer science** (I-III, Pflaum, Wanka, and Strehl).

Engineering **3D-CAD introduction and product development** (Meerkamm), **Integrated product development** (Meerkamm), **Methods and computer-aided design** (Meerkam), **Production-ready design** (Meerkamm), **Technical drawing** (Paetzold), **Design theory** (Paetzold).

Seminars **Exploration and path planning with robots** (Teich), **GameAI** (Ludwig), G**raPa: Graphics programming and application** (Stamminger and Enders), **Modelling and simulation** (Moor), **Multicore processing** (Dutta).

Neuroscience **Computational neuroscience: neural dynamics** (Schöner), **Computational neuroscience: vision and Memory** (Wiskott), **From Molecules to Cognition** (IGSN faculty members), **Machine Learning** (Wiskott), **Mathematics for modeling and data analysis** (Wiskott), **Spatial navigation and memory** (Yoshida), as well as 35 symposia and colloquia.

A Using ratlab

Listing 5: Folder structure reference

```
/ratlab/
+-/current_experiment/
|   /sequence/
|     frame_*.png                 // image sequence along path
|   /sampling/
|     /average/
|       avg_sample_*.png          // averaged sampling plots     *
|     /local/
|       sample_*.png              // sampling plots             **
|   exp_finish.png                // final simulation state
|   exp_setup                     // simulation parameter log
|   exp_trajectory.txt            // trajectory information
|   *.tsn                         // (t)rained (s)fa (n)etwork
|
+-/textures/
|   crate_*.bmp                   // textures for obstacles
|   floor_*.bmp                   // floor texture (x1)
|   skybox.bmp                    // global environment (x1)
|   wall_*.bmp                    // textures for wall segments
|
+-/tools/
|   /alternative_texture_sets/*   // different textures
|   /GUI/*                        // GUI files
|   /legal/gpl.txt                // gpl license
|   custom_example_world.txt      // example of a freeform setting
|   *.py                          // additional tools (see below)
|
+-/util/
|   setup.py                      // configuration parameters
|   ratbot.py                     // code controlling the agent
|   world.py                      // code setting up the 3D world
|   *.py                          // additional source code
|
+ __readme__.txt                  // ratlab readme file
+ *.py                            // ratlab core modules
+ *.sh                            // example shell scripts

*  e.g., avg_sample_8[-0.5;6.8].png (signal id and min/max values)
** e.g.,  sample_8_n[-8.5;6.0].png (id, direction, min/max values)
```

A.1 Overview

This appendix serves as a documentation of the ratlab toolkit. It contains all the information necessary to get started using the software as well as an overview of all available parameters. When using the software, the user is assumed to be looking at the `/ratlab/` base folder, and all paths in this appendix are relative to that folder. It contains the core modules to run simulations (cf. chapter 2.2.2), a readme `txt` file, and folders holding the rest of the source code, the files of the current simulation, and additional utilities. Of these, the `/current_experiment/` folder is the most important when using ratlab. It holds all the files that are generated when running a simulation: the image sequence created during the actual experiment, all generated sampling plots, a file holding the trajectory information (position and direction over time), as well as an overview of the used parameters (`exp_setup`), and a screenshot of the final state of the simulation. Next to the `/current_experiment/` folder, the `/textures/` folder stores all the textures used in the current simulation, the `/util/` folder stores the rest of the source code, and the `/tools/` folder stores a selection of additional elements to further customize a simulation or process its results.

Listing 5 provides an index of the complete ratlab folder structure. Fig. 38 provides an overview of a ratlab simulation. Listing 6 lists the available keyboard commands when running `ratlab.py`.

Listing 6: Available keyboard shortcuts in `ratlab.py`

```
ESC: Abort a running simulation and quit the program.
F1:  Show/hide the map overview (switch off to speed up simulation).
F2:  Show/hide rat view.
F3:  Show/hide progress bar.
F4:  Double the size the rat view is drawn (for demo purposes).
F12: Take screenshot (saved in /current\_experiment/ folder).
```

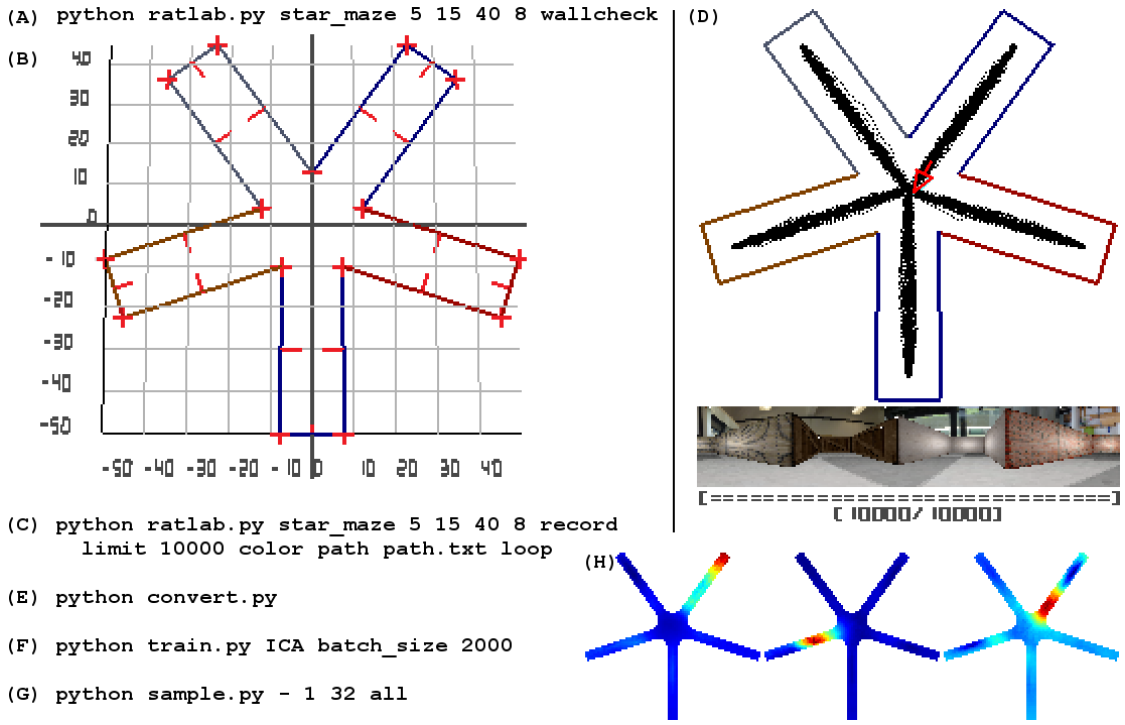

Figure 38: Ratlab simulation process. Example of performing a simulation using the ratlab tools. **(A)** Calling `ratlab.py` with the `wallcheck` option plots a blueprint of the simulation setup. **(B)** The wallcheck image shows all wall segments and their orientation (red lines). All wall segments need to be oriented correctly (red lines pointing inwards) for the simulation to complete successfully. The plot also includes a raster that can be used as a reference to determine coordinates for a predefined path the agent has to follow. **(C)** The Python call to execute `ratlab.py` and run and document the simulation. Parameters define a five-arm star maze, a number of $10,000$ time steps to be performed, and to loop around the waypoints provided in a file called `path.txt`. **(D)** A screenshot of the state of the simulation upon completion. The trajectory of the agent is plotted, the final view of the agent is shown, and the filled progress bar confirms the number if time steps of the simulation. **(E)** The call to `convert.py` collects all images and stored their data inside a single file (`exp_trajectory.txt`). **(F)** The call to run `train.py` and train the network with the generated image data. The network is set to include an additional ICA node at its top and divides the training data into batches of 2000 images each. The latter adjusts the memory footprint of the program and allows the user to continue to use the machine during the training phase. **(G)** The call to run `sample.py` and sample the trained SFA network over the original environment. The `all` parameter tells the sampling procedure to sample SFA activity and average over eight different directions. `sample.py` archives plots of each signal over each direction, as well as the average activity in the `/sampling/` folder. **(H)** Three examples of SFA signals sampled over the complete environment and averaged over eight directions. The whole process from start to finish was completed in about 50 minutes.

A.2 Command line interface

While the ratlab toolkit includes a graphical user interface (cf. fig. 3) to set up basic simulations, the default way to make use of the tools and all available options is to use the command line interface. Most simulations consist of each core module to be called once to generate new data, train an SFA network, and sample the resulting features over the used environment (listing 7).

Listing 7: Running a ratlab simulation using only default parameters

```
$ ratlab.py record limit 20000
$ convert.py
$ train.py -
$ sample.py 32 1 all
```

As can be seen, already in this case a small number of parameters is required. For this basic simulation the chosen values tell the software to run a simulation for $20,000$ time steps and sample all 32 slow features computed by the trained network in all (eight) directions. The following gives an overview of all available parameters per module.

A.2.1 Command line parameters

`ratlab.py`

- `-h, h, --help, help`: Print an overview of the ratlab module, the available parameters, and a number of examples to the command line.
- `limit <x>`: Run the simulation for `x` time steps, then quit.
- `record`: Store an image of the visual field during every time step. This generates the training data required for the remaining python modules.
- `grey`: By default, images are stored in RGB color format. By setting this flag the color palette is switched to greyscale instead (rats, especially albino rats, have only reduced color vision compared to humans).
- `duplex`: Setting this flag stores two image sequences, one in RGB color format, and a second one in greyscale format (to compare the impact of color information for a given simulation).
- `dim <x> <y> <z>`: The default simulation environment is a rectangular box. The dimensions of this box can be set using this parameter and values for the length, width, and height.
- `star_maze <no_of_arms> <width> <length> <height>`: This parameter specifies a star maze template environment. The star maze consists of `no_of_arms` arms of given width, length, and height (e.g., fig. 38). Alternatively to the full parameter the shortcut `'S'` may be used.

- `t_maze <v_length> <v_width> <h_length> <h_width> <height>`: This parameter specifies a T-maze template environment. Width and length of the vertical bar are specified by the values for `v_width` and `v_length`, respectively; width and length of the horizontal bar by `h_width` and `h_length`, respectively. The height of all walls within the maze is set by `height` (shortcut: `'T'`).
- `circle_maze <radius> <segments> <height>`: This parameter specifies a circular template environment. The radius of the environment is given by `radius`, the number of straight wall segments forming the circle by `segments`, and the height of the enclosure by `height` (shortcut: `'o'`).
- `custom_maze <file>`: This parameter specifies a user defined free form environment using the wall segments given in the `.txt` file `file`. See section A.2.3 below for details.
- `wallcheck`: If this flag is set the software does not run a simulation. Instead it draws an overview of the specified environment, including additional debug information. This is mainly used in conjunction with the `custom_maze` parameter to confirm the validity of a manually specified environment. An enclosure can be considered valid if the normal vector for each wall segment points INWARDS, and there are no gaps in between two neighboring wall segments.
- `no_wallmix`: By default, each wall segment within the environment is drawn with one of the different textures in the `/textures/` folder (all wall textures have a `_wall` tag in their name). If this flag is set, all wall segments share the same texture.
- `box <ll_x> <ll_y> <ur_x> <ur_y>`: Use this to place a rectangular obstacle (textured to look like a wooden crate) within the environment. The box is defined by the coordinates of its lower left (`ll_x`,`ll_y`) and upper right (`ur_x`,`ur_y`) corner.
- `box <ll_x> <ll_y>`: This is used to place a default-sized (eight by eight centimeters) box within the environment. Coordinates of its lower left corner are given by (`ll_x`,`ll_y`)
- `boxmix`: Set this flag to apply a different (wooden crate) texture to the different obstacle boxes added to the environment. (Obstacle textures can be found in the `/textures/` folder; their name contains the `'crate_'` tag.)
- `small_fov`: This flag changes the default viewport of 320 by 40 pixels to 55 by 35 pixels. This can be used to simulate an agent with a much narrower field of view, similar to available cameras that can be mounted on a robot, for example.
- `speed <s>`: A speed multiplier for the movement of the agent. A value of 2.0 will make the agent move twice as fast.
- `arc <x>`: This value defines the decision arc of the agent. A value of 360 (degrees) means the agent is allowed to pick a completely random direction during each time step.
- `mom <x>`: This value defines the momentum of the agent, and has to lie between zero and one. See listing 2 for how the momentum value factors into the calculation of the next randomized step.
- `path <file_name>`: The rat will follow a series of waypoints specified by the user. The waypoints can be found in the file `file_name` which in turn has to be found in the `/current_experiment/` folder. Note that the simulation will assume all waypoints are represented by valid coordinates and does not perform a check to confirm that the path of the agent will not cross a wall segment.
- `loop`: This flag only works in conjunction with a user-specified path (see above). It tells the rat to run back to the first waypoint once the last one has been reached. If this flag is not set, the agent will "teleport" back to the first waypoint and continue running along the specified path from there.

- `bias <x> <y> <s>`: This introduces a movement bias, i.e., makes the rat move faster if its movement direction aligns with the vector given by (`x`,`y`). The scaling factor `s` determines how much speed is added to the normal movement speed if the agent's movement direction aligns perfectly with the specified bias vector; a value of 1.0 doubles the agent's movement speed).

train.py

- `batch_size <x>`: If specified the network will not be trained will all available image frames in one go. Instead, the data will be split into batches of the given size and the network trained with one after the other. This is helpful when working with large amounts of files ($40,000+$) as it is much faster to train hierarchical SFA with a number of smaller batches instead of one large chunk of data.
- `frames <x>`: If specified, the network will be trained with `x` frames only, even if the full data set includes more images. This can be used to create one data set and then train SFA with different amounts of images to see how the spatial representation develops with additional image data.
- `ICA`: This flag specifies whether the network should include an top level ICA layer. This is necessary to produce the sparse, place field like spatial representation presented throughout this work. Note that this may also be used in conjunction with the `file` parameter (see below) to add an ICA layer to an already trained network that is missing an ICA layer. In this case only the the untrained layers of the network will be trained, i.e., either the whole network or merely the newly added ICA layer.
- `file <file_name>`: This parameter indicates that an already existing network is to be trained. This is usually used when using a network `.tsn` file that underwent generic training (see below) before and is now to be trained with the actual data of the current simulation. The name of the `.tsn` file – expected to be found in the `/current_experiment/` folder – is given by the `file_name` parameter. If `file_name` is simply `'-'`, `train.py` looks through the `/current_experiment/` folder and uses the first `.tsn` file it can find (this is a handy shortcut if there only exists one `.tsn` file).
- `noise`: This flag adds noise to the individual SFA nodes (cf. fig. 6). This may sometimes be required to successfully execute the training procedure.
- `generic`: This flag indicates that only the lower two SFA layers of the network are to be trained while the upper layers are left untrained. The data for generic training is expected to be found in the `/current_experimen/sequence_data_generic` file, in contrast to the regular training data which should be stored in the automatically generated `/current_experiment/sequence_data` file (cf. section A.2.4 below for more information about how to use generic training).

sample.py

- `<tsn_file>`: When performing spatial sampling, the first parameter always has to name a `.tsn` file that is found in the `/current_experiment/` folder and contains the trained network that is to be sampled.
- `<frequency>`: The second parameter indicates the sampling frequency. A value of n (usually 1) indicates that every n-th (integer) position is to be sampled.
- `<no_of_signals>`: The third parameter indicates how many of the 32 available features computed by the default SFA network are to be sampled.

- `<dir>`: The fourth parameter(s) indicate what directions should be sampled. This may include any of the following: `n`, `ne`, `e`, `se`, `s`, `sw`, `w`, `nw` (denoting "north", "north-east", etc., respectively). Alternatively the shortcut `all` indicates that the spatial activity of the network should be sampled while looking in all of the available directions. If more than one direction is chosen, the average activity of all signals over all chosen directions is also computed and stored separately.
- `<tsn_file> <no_of_signals> dir`: Alternatively to spatial sampling, a network may also be sampled over all directions. While the former plots the activity of a network over all positions within the environment, the latter plots the activity of the network while pointing at any direction. To use this functionality the parameter `dir` has to be included. In this case the first parameter specifies the `.tsn` file to be used; the second parameter the number of signals to be sampled (usually all 32 available); and the third parameter consists of the mandatory `dir` flag (cf. section A.2.5 below for more information about how to use directional sampling).
- `grid <s>`: This parameter is mandatory for directional sampling and defines the sampling frequency (cf. section A.2.5 for details).
- `map`: If this flag is set, no sampling takes place. Instead, an overview of all valid positions of the currently used environment is plotted. Its primary use is to verify the simulation setup and to manually define points for directional sampling (cf. section A.2.5 for details).

A.2.2 Internal parameters

This following listing covers the internal parameters of the software framework. This includes parameters that are usually not changed and thus are not included in the set of command line parameters. They are collected in the `/util/setup.py` file, which consists of all the available parameters that are accessible from within the software during runtime (listing 8). Note that some of these variables were already described above, as they they may be modified by command line parameters. Changing such value in the `setup.py` file means redefining their default values that will be used if no command line parameters are given.

Listing 8: List of all ratlab parameters available during runtime

```
# +--+world
# |  |
# |  +->type:        world type: 'box', 'star', 'T', 'circle', 'file' [def: 'box']
# |  +->dim:         tuple defining dimensions of the respective world type
# |  |               +->box:    (<length>, <width>, <height>)
# |  |               +->star:   (<arms>, <arm_width>, <arm_length>, <arm_height>)
# |  |               +->T:      (<v_len>, <v_width>, <h_len>, <h_width>, <height>)
# |  |               +->circle: (<radius>, <segments>, <wall_height>)
# |  |               +->file    (<file_name>)
# |  +->box_dim:     dimensions of a default box
# |  +->boxmix:      choose textures for obstacles randomly?
# |  +->wallmix:     uniform or mixed wall texturing?
# |  +->wall_offset: agent will not come closer to walls than this distance
# |  +->cam_height:  height of the agent camera above ground level
# |  +->obstacles:   list of obstacles (empty by default)
# |  +->limits:      min/max coordinates of the world (computed by world.py)
# |  |
# |  +->color_background:     background color (default: black)
# |  +->color_rat_marker:     color used to draw the rat marker (default: red)
# |  +->color_rat_path:       color used to draw the rat's path (default: blue)
# |  +->color_sketch_default: color used for uniform sketching (default: blue)
# |
# +--+opengl
# |  |
# |  +->clip_near: near clipping plane (objects closer than this are not rendered)
# |  +->clip_far:  far clipping plane (objects farther than this are not rendered)
# |
# +--+rat
#    |
#    +->color:    output color mode: 'greyscale', 'RGB', or 'duplex'
#    +->fov:      rat field of view
#    +->arc:      arc defining rat mobility
#    +->path_dev: deviation from straight line when following a given path
#    +->path_mom: momentum/smoothness of the rat's path
#    +->bias:     optional movement bias
#    +->bias_s:   strength factor of the optional movement bias
#    +->speed:    speed multiplier for the movement of the rat
```

A.2.3 Freeform environments

Instead of using one of the template environments offered by the software, users may choose to define their own setting. To this end, `ratlab.py` can be called with the `custom_maze <file_name>` parameter option. The custom environment is defined by specifying the individual wall segments in a plain `.txt` file. The first line of the file gives the floor texture to be used; each line after that describes

one wall segment by listing two pairs of coordinates and one texture identifier to be used for this wall. Each wall segment needs to be specified "from left to right", i.e., if the agent was to stand in front of it, the coordinates of the left corner need to be specified first, followed by the coordinates of the right corner. This is important as the collision detection algorithm of the software relies on the correct alignment of all walls. To help users with this step, running `ratlab.py` with the `wallcheck` flag set plots an overview of the environment that renders a normal vector for each wall segment; if all normals point INWARDS, the environment is set up correctly. The file containing the newly specified environment can be named anything, but, like every other component of the simulation, needs to be placed in the `/current_simulation/` folder. The folder `/tools/` contains the file `custom_example_world.txt` which contains a free form environment example for users to use and modify; a small section of this file is shown here (listing 9) as an example for the file format.

Listing 9: Setting up a custom maze

```
floor floor_concrete_chequered_[accent_lighting]
  0   0    0 160   wall_01_[accent_lighting]
  0 160   40 160   wall_01_[accent_lighting]
 40 160   40 120   wall_01_[accent_lighting]
  5   5   10   5   wall_cue_card_front
 10   5    5   5   wall_cue_card_back
                ...
```

Note that walls do not have to be connected, which allows users to add cue cards to the environment as well. A cue card consists of two different, back to back wall segments that may be located anywhere within the environment; the last two lines in listing 9 show an example of this. Note that cue cards *have to* consist of a front and back side so as to not confuse the collision detection algorithm.

A.2.4 Generic training

Generic training means training an SFA network in two phases: first, the lower SFA layers are trained with a (large) image set of (for example) a multitude of different environments ; second, the upper layers (SFA and possibly ICA) are trained using an imagine sequence taken from the actual setting of the current simulation. The motivation hereby is that a network trained with phase one only may receive its second-stage training from any other simulation. In other words, a base-level

trained network may be stored and reused for later simulations, thus cutting down the required training time significantly. To perform generic training users first need to collect the data to be used for the base-level training stage. This may be done by running a number of different simulations and collecting the produced images, or any other method that comes to mind. Once this data is collected or generated it is to be placed in the `/current_experiment/sequence_generic/` folder (has to be created manually). Running `convert.py` automatically converts any image sequence found in the `/current_experiment/sequence/` folder into a file labeled `sequence_data`. Likewise, any data found in the `/current_experiment/sequence_generic/` folder will be converted into a file labeled `sequence_data_generic`. When running `train.py` later, the former will be used to train the final stages of the network, while the latter will be used to perform the general training phase. The predefined directory and naming structure was put in place to ensure that all files required for one particular simulation are always clearly labeled and found in one place only, namely the `/current_experiment/` folder of the simulation in question. Listing 10 shows an example of generic training: the data is converted using `convert.py`; base-level training is executed, creating a `.tsn` file of the partly trained network; second stage training is performed using the only available `.tsn` in the `/current_experiment/` folder (also adds a top level ICA layer to the network).

Listing 10: Performing generic training

```
convert.py
train.py generic
train.py file - ICA
```

A.2.5 Directional sampling

In order to sample the direction sensitivity of a trained network the agent is placed at various positions throughout the environment. At each position the camera is rotated by 360 degrees (one degree at a time), and a screenshot of the current view of the environment is taken and fed into the SFA network. The responses of the network over all positions and directions are collected and used to plot the average activity of the network while pointing at all sampled directions over each location used during sampling. The resulting polar plots (cf. fig. 8) show the sensitivity of the different SFA features to direction, irrespective of location. The software also plots the directional sensitivity for each signal at each location separately, so that

users can check whether the network reacts uniformly over the whole environment. To use this functionality, `sample.py` is called in the following manner:

```
sample.py <file_name> <no_of_signals> dir grid <grid_spacing>
```

The first parameter indicates the network file to be sampled; the second how many of the set of 32 slow signals are to be sampled (usually all of them); the third flag tells `sample.py` to perform directional sampling instead of spatial sampling; and the `grid_spacing` parameter defines the spacing of the sampling grid. The sampling grid contains all the positions the camera will be placed on and rotated around during the sampling process; a value of n means that every n-th x and y integer position that lies within the environment is going to be used. Since directional sampling gathers a lot more data than spatial sampling, the spacing for the sampling grid should not be chosen too low.

Alternatively, users may define their own sampling positions. To this end, `sample.py` can be called with the `map` flag set. Instead of sampling a network, `sample.py` will create a plot of the environment that shows all valid positions for the agent to be located in. This plot can be opened in any basic drawing tool and the user may draw single red dots[21] within the accessible space to indicate the locations where the camera is to be put during directional sampling. To use such custom sampling positions the user may simply forego the `grid <grid_spacing>` part of the above call to `sample.py`. Upon execution, the code will then look for the `exp_map.png` file, created by calling `samply.py` with the `map` parameter and modified by the user, to extract the desired sampling positions.

A.2.6 Example scripts

The following presents a number of small shell scripts to execute all software modules and complete various simulations. These are designed for users to modify and experiment with in order to encourage experimentation and users writing their own simulation scripts. Some of these examples are also included in the documentation for `ratlab.py` which is shown when displaying the help text of the module. Note that, in contrast to above, the listings in the following show directly

[21] Each red-colored pixel will be translated to one sampling position. The used red has to be pure red, i.e., (255,0,0) in the commonly used RGB color space.

usable shell scripts, i.e., also include the required calls to Python itself as well as other basic shell commands.

Listing 11: Basic shell script example no.I

```
# This script executes a basic simulation of the rat exploring a circular
# environment for a simulated time of 8 minutes. Note that the network does not
# include the top ICA layer.
python ratlab.py record limit 9600 circle_maze 80 24 40
python convert.py
python train.py
python sample.py - 32 1 all
```

Listing 12: Basic shell script example no.II

```
# Script to run a longer simulation within the custom maze found in the /tools/
# folder. Note that in order to train the network with 100000 images, the data is
# split into four chunks of 25000 images each. Also, a previously generically
# trained network is being used (the software includes a number of networks
# pre-trained with generic data).
python ratlab.by record limit 100000 custom_maze custom_example_world.txt
python convert.py
python train.py file full_generic_100k_sim.tsn batch_size 25000 ICA
python sample.py - 32 1 all
```

Listing 13: Basic shell script example no.III

```
# This script creates 10000 images that will be used for generic training later.
# The 'mv' command moves the data into a dedicated folder for later use.
python ratlab.py record limit 10000 dim 60 40 12 box 26 38 34 40 box 26 0 34 2
                box 0 16 2 24 box 58 16 60 24 box 28 18 32 22 boxmix
mv ./current_experiment/sequence ./current_experiment/sequence_1_box_maze
mv ./current_experiment/exp_finish.png ./current_experiment/box_maze_finish.png
```

Listing 14: Basic shell script example no.IV

```
# This script is part of a larger trial. Different simulations were performed,
# using a differently sized environment for each. This example shows how the
# information stored in the automatically created exp_setup files can be used to
# restore the current simulation to a previous setting, sample the network using
# these parameters, storing the resulting plots, and restoring the state of the
# /current_experiment/ folder to its neutral state.
# The used 'mkdir' command is used to create a new folder ("make directory").
mv ./current_experiment/exp_setup_70x60 ./current_experiment/exp_setup
python sample.py - 1 32 all
mkdir ./current_experiment/sampling_70x60
mv ./current_experiment/sampling_plots      ./current_experiment/sampling_70x60/
mv ./current_experiment/sampling_values     ./current_experiment/sampling_70x60/
mv ./current_experiment/sampling_values_raw ./current_experiment/sampling_70x60/
mv ./current_experiment/exp_setup ./current_experiment/exp_setup_70x60
```

Listing 15: Basic shell script example no.V

```
# This script parallelizes the sampling process on a contemporary multicore
# system. The sample_collector module used in the end is called after the
# individual sampling processes have terminated and creates the averaged plots
# from the accumulated data (see the following section for details).
python sample.py generic_100k_sim_ICA.tsn 1 32 n  &
python sample.py generic_100k_sim_ICA.tsn 1 32 e  &
python sample.py generic_100k_sim_ICA.tsn 1 32 s  &
python sample.py generic_100k_sim_ICA.tsn 1 32 w  &
python sample.py generic_100k_sim_ICA.tsn 1 32 ne &
python sample.py generic_100k_sim_ICA.tsn 1 32 se &
python sample.py generic_100k_sim_ICA.tsn 1 32 sw &
python sample.py generic_100k_sim_ICA.tsn 1 32 nw &
wait
python sample_collector.py
```

A.3 Utility package

Included in the overall software package is a number of small Python scripts that proved helpful during development. These are neither overly complicated nor claim to to perform sophisticated computations. They are, however, building blocks of source code that has be reused several times and are thus included in the overall package. They should be seen as a collection of examples that may be disassembled, modified, and merged in order to develop new tools as required. All of these can be found in the `/tools/` folder and are listed in the following:

- `sys_check.py`: This small script simply imports all libraries used throughout the software and prints out their used version. This yields potential debugging information and can be used to make sure that all required libraries are installed on the currently used system – thus making sure that any later occurring errors are not cause by missing libraries.
- `circle_maze.py`: This script creates a file that may be used in conjunction with the `custom_maze` parameter when calling `ratlab.py`. The created environment is a circular area with a given arc of the wall covered by a cue card with a given texture. In other words, this script adds an additional template environment to the software.
- `mask.py`: This script allows the user to name a file to be used as a mask and a directory containing result plots. Each white pixel in the masking image

will be overwritten as white in each of the result plots. This can be used to use the environment from one experiment as a masking overlay for another one to see how signals relate to each other.

- `plotfix.py`: This script may be used to, retrospectively, change the background color (default black) to any other color (usually white).

- `rename.py`: This script renames the files within a given folder to form a numerical sequence, starting with a given starting value. Since `ratlab.py` starts numbering each created image at zero, this script can be used to merge different sets of image sequences into one. This enables the creation of data sets used for generalized training by merging several `ratlab.py`-produced image sequences.

- `scale_equalizer.py`: When plotting the results from different simulations, the signals computed by SFA are scaled to their local maximum, i.e., the warmest color in one series of plots might correspond to a different value of activity than in another. This script corrects this; it is provided with a number of directories holding finished plots and re-scales all plots to the min/max values of the complete set. This is particularly useful when making use of a multicore CPU to split up the plotting procedure (e.g., lis. 15) into multiple, independent processes that do not communicate min/max values to each other.

- `snip.py`: This script takes a given image sequence and cuts out the center part of each image, thus creating a new sequence. It can be used to extract data of the correct format from any kind of input stream. (Note, however, that images are not scaled before their center is cut out.)

- `trajectory_plot.py`: While `ratlab.py` provides a plot of the trajectory of the virtual agent upon completion of each simulation, that plot is rather functional and only offers a low resolution. If users wish to present the results of a simulation, this script can be used to draw a plot of the same data in a much higher resolution (cf. fig. 11).

- `ratlab.faq`: This file is not a script but rather contains some common error messages that new users might happen upon. (As an example: if the environment does not offer enough space for the agent to navigate within, this is

communicated with the unhelpful wording of `"invalid value encountered in double_scalars"`.)

Besides this script collection, the `/tools/` folder also contains alternative texture sets – ranging from entirely artificial color cues to photos from real brick walls – to be used in simulations. This includes alternative textures for the so called "skybox" as well, the texture that can be seen beyond the wall of the enclosure at a longer distance (affecting the parallax of the cues). Options include an office view, and two differently colored curtains (black and red, with the latter displaying visible creases).

Lastly, the `/tools/` folder contains a copy of the license the software is published under, namely the GNU General Public License (version 3, 29th of June, 2007). It allows anyone to make use of the software as they see fit, as long as the original authorship is made clear.

A.4 CUDA link

This sections describes how to modify the python MDP library in order to make use of the CUDA link included in the software package. This allows a central matrix multiplication step within the SFA algorithm to be performed on a CUDA-enabled graphics card instead of the local CPU. All required files can be found on the disk accompanying this work in the `/cuda_patch/` folder (this includes a readme `.txt` file that allows for an easy copy&paste of the required compiler commands). Provided a working CUDA installation on the system, the following steps need to be performed:

1. Build a shared library object from the provided `.c` source code files (see listing 16 for the required commands).
2. Copy shared library `.so` file into the `/mdp/utils/` folder of your local MDP installation.
3. Copy the `covariance.py` file found in the `/cuda_patch/` folder of the attached disc into the same `/mdp/utils/` folder; overwrite/backup the already existing file.
4. Optional: copy the `layer.py` file into the `/mdp/hinet/` folder. This adds a progress bar so the state of a longer training phase can be checked.

Listing 16: Compiling the shared library object (`libmodule.so`) to link CUDA with Python

```
gcc -I/usr/local/cuda/include -L/usr/local/cuda/lib64 module.c -o cudamult -lcuda
    -lcudart -lcublas
gcc -I/usr/local/cuda/include -L/usr/local/cuda/lib64 -fPIC -c module.c
    -o libmodule.o -lcublas -lcuda -lcudart
gcc -shared -Wl,-soname,libcudamult.so -L/usr/local/cuda/lib64 -o libmodule.so
    libmodule.o -lcublas -lcuda -lcudart
```