Chair of Robotics, Artificial Intelligence and Real-time Systems
Department of Informatics
Technical University of Munich

Neuromorphic Implementation of a 3D Attractor Network for Maintaining and Updating Multiple Simultaneous Pose Estimates During Path Integration

Interdisciplinary Project Report

Martin Schonger

martin.schonger@tum.de

Supervisor:
Prof. Dr.-Ing. habil. Alois Christian Knoll

Advisor:
Genghang Zhuang, M.Eng.

Application Subject:
Electrical and Computer Engineering

Submitted:
Garching, 07.02.2023

Abstract

Mammalian path integration, i.e. the integration of self-motion information over time to update an internal position estimate, revolves around the interaction of spatially selective neurons, including grid cells, head direction cells, and conjunctive cells. Based on that, we propose a neuromorphic model for representing and updating multiple simultaneous pose estimates. More than one estimate may be feasible after vague visual input or loop closure events. To avoid ambiguity, we work in the joint space of 2D position and orientation. Our unit domain resembles a right rhombic prism with periodic boundary conditions. Poses are treated as 3D Gaussians and represented with a 3D continuous attractor network of spiking neurons tuned to one pose each, i.e. conjunctive grid-by-orientation cells. The neurons are topographically arranged and form a hexagonal-orthogonal grid. The recurrent connectivity features manually-tuned local center-surround excitation-inhibition and a skewed wraparound that gives rise to a stacked, twisted toroidal manifold. This corresponds to a periodic representation of planar space with optimal coverage. Slight constant global inhibition regulates the overall activity. Per pose estimate, the single-neuron and population responses exhibit three proper axes of symmetry. We implement the attractor as a Nengo ensemble, which motivates switching to reciprocal space. The periodic basis functions are well-suited for our domain and convenient to work with.

Pose estimates are moved around on the attractor manifold by three shifting ensembles with offset feedback connections to the main attractor network. These ensembles essentially operate on a delayed replication of the attractor state. For translation, one ensemble shifts activity within each horizontal 2D neural sheet into the associated heading direction. The other two ensembles shift upward/downward along the θ-dimension, corresponding to counterclockwise/clockwise rotation. As with the attractor, connection weights are tuned manually. The shifting ensembles are selectively and proportionally inhibited based on tangential and angular velocity input. This setup enables smooth and independent movement of each individual pose estimate. So, in the presence of large uncertainty, multiple feasible pose hypotheses can be propagated until enough evidence is accumulated to establish one of them as the correct one, and the others vanish.

Overall, the system closely follows the biological model functionally and neurophysiologically. It is engineered to run on an Intel Loihi neuromorphic chip, exploiting its architecture and respecting the associated constraints. This pertains mainly to the number of neurons and the connectivity pattern between them.

We show the stable representation and correct propagation of up to three concurrent pose estimates. The system's trajectories on real and simulated robot velocity data demonstrate that topological features such as curves are captured and that the relative distances between them are accurate. Due to the discrete nature of the neuron grid, the stable attractor states coincide with the neurons' locations and combinations thereof. A preference for movement along grid axes is evident. As of now, the evaluation is purely simulation-based. It does not yet run in real time. Deployment on Loihi may fix this by eliminating the need for simulated neuron dynamics and enabling parallel execution.

Contents

1 Notation

Abbreviations	Artificial Neural Network	ANN
	Entorhinal Cortex	EC
	Excitatory Postsynaptic Potential	EPSP
	Field-Programmable Gate Array	FPGA
	[Inverse] (Discrete/Fast) Fourier Transform	[I](D/F)FT
	Grid Cell	GC
	Head Direction Cell	HDC
	Hippocampus	HIP
	Inertial Measurement Unit	IMU
	Inhibitory Postsynaptic Potential	IPSP
	Leaky Integrate-and-Fire (Model)	LIF
	Medial Entorhinal Cortex	MEC
	Neural Engineering Framework	NEF
	Place Cell	PC
	Path Integration	PI
	Simultaneous Localization and Mapping	SLAM
	Spiking Neural Network	SNN
	Spike-Timing-Dependent Plasticity	STDP
Math & Definitions	Vectors are typeset in bold and non-italic	$\mathbf{x}$
	Unit length	u
	Grid coordinates	4_g, $(4,4,0)_\mathrm{g}$
	Tangential velocity	v $(\mathrm{u/s})$
	Angular velocity	$\omega \coloneqq d\theta/dt$ $(\mathrm{rad/s})$
	Dot product	$\mathbf{a}^\intercal\mathbf{b}$, $\mathbf{a}\cdot\mathbf{b}$

2 Introduction

2.1 General Background

Autonomous mobile robots for applications such as passenger transport or parcel delivery need to navigate unknown environments precisely and reliably. A popular concept to achieve that is simultaneous localization and mapping (SLAM), where local visual cues are combined with self-motion cues to incrementally build a map of the environment and to estimate the current position therein. The traditional geometric approach to SLAM is based on optimization or probabilistic filters.[62] Recently, neuronal architectures inspired by the spatial navigation system of mammals have been proposed as a more robust and efficient alternative.[69]

Neuroscience research has identified several neuron types with characteristic spatial firing patterns: place cells (PC; encode the current position by firing at one or few locations in space), grid cells (GC; have multiple firing locations arranged in a regular hexagonal lattice), head direction cells (HDC; fire when facing a specific direction), speed cells, and others.[47] Interactions between the different cell types are important for generating coherent spatial representations, i.e. a cognitive map.[49]

By combining multiple grid cells with firing fields of different phases and varying spatial scales it is possible to obtain non-periodic firing fields corresponding to single locations in space. This facilitates efficient, precise and robust encoding of large spaces.

One widely accepted hypothesis is that grid cells are connected as a continuous attractor network (CAN). Together with input from head direction cells and speed cells, path integration is possible,[10] i.e. computing the current location by integrating self-motion information (e.g. relative tangential and angular velocity). Notably, this also works in the absence of sensory inputs (e.g. vision or olfaction).[38,58] During normal operation, sensory cues can be incorporated via place cells and can be utilized to correct path integration errors or even to reset the system to an entirely different position.[20]

For optimal performance of biologically-inspired software models, neuromorphic hardware can be employed. These are computers that imitate the neural structure and operation of the human brain, comprising millions of artificial silicon neurons that transfer electric spikes between one another. Functionality such as spike propagation or dynamically updating synaptic connection weights is directly supported at the hardware level, making these chips highly efficient.

2.2 Related Work

There have been multiple previous efforts to recreate various aspects of the biological spatial navigation system for robotic applications. There are computational approaches, which comprise algorithms and data structures that imitate functionality of the biological model, such as the concept of continuous attractor networks.[43] Other approaches more closely follow the neuronal components (GC, HDC, etc.),[4,69] account for their interplay, or consider the physiological aspects of spiking neurons[13] or synaptic plasticity. A few works even employ neuromorphic hardware,[22,35,37,38] which is more efficient for running densely connected networks than classical computing architectures. The approaches further differ in their comprehensiveness, i.e. which components of spatial navigation they consider: path integration,[38] map creation, or episodic memory for contextual information.[63] While Krishna et al.[38] developed a rather extensive model for path integration, they don't support multiple simultaneous position hypotheses. Only a few works address the issue of large uncertainty and multiple hypotheses, but these lack other features of the biological model.[44]

2.3 Significance

In contrast to existing work, we aim to develop a neuromorphic model for path integration which combines grid cells, head direction cells, continuous attractor dynamics, spiking neuron properties, and deployment on state-of-the-art neuromorphic hardware; while also explicitly accounting for large uncertainties by supporting multiple simultaneous

pose estimates. For example, due to ambiguous visual input or loop closure more than one pose may be feasible at any given time. For accurate path integration, all of them need to be represented unambiguously, and be properly and individually propagated until one pose has sufficient support to take dominance.

The exploitation of neuromorphic computing principles such as spiking dynamics and high parallelism, as well as the future deployment on Intel Loihi promise competitive performance with respect to power consumption, physical space requirements, and fast-paced learning.

2.4 Project Goal

The final system should be capable of performing accurate path integration given relative tangential and angular velocity inputs. Therefore, it needs to be able to stably represent position estimates in the absence of input, and to translate these estimates accordingly and individually when input is present. The system should be able to handle local uncertainty stemming from noisy velocity inputs, integration errors, and smaller disturbances; as well as non-local uncertainty due to larger errors caused by outside influence (e.g. repositioning the system) or ambiguous visual inputs. Hence, the system needs to be capable of simultaneously representing multiple hypotheses about the current pose until enough support for a single one accumulates. Furthermore, the system should be biologically realistic according to the current understanding of spatial navigation in the mammalian brain. In particular, it should incorporate the workings of grid cells and head direction cells. Neuromorphic computing principles such as using a large number of individual computing units (i.e. neurons), hierarchically organized computing layers, a high degree of parallelism, and the dynamics of spiking neurons should be considered and exploited.

2.5 Our Approach In a Nutshell

We propose a three-dimensional continuous attractor network, where the first two dimensions correspond to x and y position on a plane, and the third dimension corresponds to the orientation angle θ. This network is implemented as an ensemble of neurons, where the neurons are topographically arranged in a three-dimensional grid. The location in the grid directly determines a neuron's response curve in the representational space, which is specified via a 3D Gaussian in that space.

The representational space has periodic boundary conditions, that are additionally skewed when viewed in the x-y plane, resembling a twisted torus. This leads to a hexagonal firing pattern of the individual neurons, akin to grid cells in the mammalian entorhinal cortex. The recurrent connection between these neurons follows a three-dimensional Mexican hat shape with local excitation and inhibition. Constant global inhibition regulates the population activity. Individual pose estimates are treated as 3D Gaussians and, consequentially, correspond to packets of activity in the neuron ensemble. A meticulously tuned weight profile allows sustaining multiple activity packets.

Pose estimates are translated through the representational space via three distinct shifting ensembles: one for translation in positive θ-direction, i.e. counterclockwise rotation, one for translation in negative θ-direction, i.e. clockwise rotation, and the third one for in-plane translation, i.e. straight movement in the preferred direction per horizontal slice of the attractor network. The movement type and speed are regulated by selectively and proportionally inhibiting the shift networks.

Computation-wise, we use the Neural Engineering Framework (NEF), which centers around encoding and decoding from and to the representational space. In our case, this space is roughly defined as the set of Fourier coefficients of the three-dimensional space of x, y, and θ. We also call this the Fourier or reciprocal space. A neuron's instantaneous activity is proportional to the dot product of the to-be-represented pose estimates' combined coefficient vector and the coefficient vector of the neuron's preferred pose. Moving bumps around is done by rotations in reciprocal space.

2.6 Contributions

- Neuromorphic implementation of a 3D continuous attractor network to perform path integration, engineered to run on Intel Loihi.
- Sustain multiple simultaneous pose estimates in the case of uncertainty and visual ambiguity. In contrast to stacked grid and head direction cells, our approach represents position

and orientation conjunctively. The main attractor network essentially consists of conjunctive grid-by-orientation cells.

- Smooth, individual translation and rotation of each pose estimate via three selectively and proportionally inhibited shifting networks with offset feedback connections to the main attractor network. One network shifts activity within each horizontal 2D neural sheet into the associated heading direction. The other two networks shift upward and downward along the θ-dimension, respectively; corresponding to counterclockwise and clockwise rotation, respectively.

3 Background

3.1 Neural Computations and Spiking Neural Networks[7,22,31]

Computation in the brain involves the interaction of millions of neurons via even more synaptic connections. Thereby, the neurons produce precisely timed electrical spikes. In more detail, the signal transmission works as follows. Action potentials, originating from neuron cores, travel along axons toward synapses, which are connection points between axons and dendrites, efferent and afferent signal pathways, respectively. These synapses are activated and release neurotransmitters towards the postsynaptic neuron, thereby influencing its membrane potential. We differentiate between Excitatory Postsynaptic Potentials (EPSP) that increase the membrane potential, and Inhibitory Postsynaptic Potentials (IPSP) with an opposite effect. These two effects are referred to as depolarization and hyperpolarization, respectively. Once the membrane potential reaches some threshold, the neuron generates a spike, then enters the so-called refractory period, within which it cannot be reactivated, and finally restores its initial resting state. The generated spike meanwhile propagates along the axon and the conceptual cycle repeats. During periods of no incoming EPSPs, the membrane potential slowly approaches the resting state due to leakage. The synapses and dendrites are not just simple connections, but actually contribute to the computation by e.g. applying gains or more complex modifications to passing spikes. These contributions adapt depending on the relative spike-timing between their endpoints and other factors, which is known as synaptic plasticity and forms the physiological basis of learning. While the provided explanation simplifies reality, it is already evident that via all the timing and gain parameters many distinct spiking patterns can be achieved, both in terms of the shape of action potentials and their relative timing.

3.1.1 Spiking Neuron Models

As an at-least-as-powerful alternative to rate-based Artificial Neural Networks (ANN), which are a purely abstract model of computations in the brain, Spiking Neural Networks (SNN) try to capture the physiological aspects of real neural computations outlined above with various degrees of detail, particularly by explicitly modeling spikes. To this end, various neuron models have been proposed, including the Hodgkin-Huxley model and the Leaky-Integrate-and-Fire (LIF) model. When selecting an appropriate model for a use case, a trade-off between neuroscientific realism and computational complexity needs to be considered. These neuron models are typically specified as dynamical systems by differential equations.

LIF's predecessor, the Integrate-and-Fire model (IF), revolves around the concept of integrating incoming spikes until some threshold is reached, upon which an action potential is generated and the integrator is reset. Adding a term to account for the aforementioned leakage effect results in the LIF model. Both models focus on the timing aspect rather than the exact action potential shape.

3.1.2 Neural Codes

To facilitate neural computations, a relationship between information and spiking patterns needs to be established, known as a neural code. It is used to encode any quantities involved in the computation, e.g. abstract numbers or physical properties, as some population activity of neurons, i.e. particular sets of firing rates or relative spike timings. Then the neural network changes its state based on that input via a series of spiking interactions, i.e. it performs the computation. Finally, we can read out the state of the neurons and decode it back to the information space, which is more comprehensible for humans. The Neural Engineering Framework (NEF) centers around the encoding and decoding of information with spikes, and is discussed further in Section 3.4. Note, that the actual neural codes are yet to be fully discovered and understood. Still, a number of codings have been proposed based on the current knowledge. Fully temporal codes consider the timing of spikes. The following are special cases of them. Binary coding only cares about

whether a neuron is active or silent. This is simple but ignores individual spikes and their timings. Rate coding is all about the firing rate of neurons within particular time periods, thus, also abstracts from the precise timing of spikes. Latency coding, on the other hand, focuses on the timing information instead of the number of spikes. There are also predictive and probabilistic coding schemes.

3.1.3 Spike-based Learning

For populations of neurons to actually perform the encoding and computations, a certain network connectivity needs to be established, synaptic weights adjusted, and dendritic spines configured. Several learning methods try to explain these processes. Hebbian learning, one of the most basic theories, states that neurons that fire together wire together. Spike-timing-dependent plasticity (STDP) builds upon that: if the presynaptic neuron fires immediately before the postsynaptic neuron, the connection is strengthened (Long Term Potentiation (LTP)), whereas it is weakened if the presynaptic neuron fires immediately after the postsynaptic one (Long Term Depression (LTD)). Apart from unsupervised learning, of which the aforementioned concepts are instances, there are also supervised learning and reinforcement learning.

3.2 Neuromorphic Computing[7,22,35–37]

While running biologically-inspired software models, e.g. spiking neural networks, on conventional computing hardware is possible and often the only option, it is also quite inefficient and impractical for real-time applications. To unleash the full potential of these models, they can be paired up with specialized neuromorphic computing hardware.

Spiking neurons, as opposed to rate-based neurons, do not require updating from every incoming connection at each time step, but only from those that see spiking activity. Typically, the set of active presynaptic neurons will be considerably smaller than the full set of presynaptic neurons. This is one of the main reasons for SNNs' efficiency and could be exploited in hardware by reusing resources. On the other hand, updating the membrane potential of spiking neurons in each time step is more involved than the weighted sum computation for rate-based neurons, because of the nonlinear dynamics associated with spiking neuron models. These properties of reduced communication effort and increased computational complexity of individual units make SNNs well-suited for fine-grained parallelism and event-based asynchronous computation, which is precisely what neuromorphic hardware provides. These chips aim to mimic the physiology of real neural systems in terms of neuron models and synaptic connectivity. They comprise thousands or millions of artificial silicon neurons and synapses that transfer electric spikes between one another. Typically, the number of synapses is significantly larger than the number of neurons to cope with realistic communication-intensive neural workloads. Functionality such as spike propagation, feedback loops, recurrent connectivity, or dynamically updating synaptic connection weights is directly supported at the hardware level. Furthermore, in contrast to the classical von Neumann architecture, neuromorphic chips often feature coupled computing and memory units. They are designed to work well with irregular communication patterns, where classical cache hierarchies would lose their effectiveness. There are digital, analog, and mixed-signal models available. Particularly for analog parts, variability and imperfections in the silicon structures introduced during the production process translates to noisy behavior of the silicon neurons and synapses. This nicely mimics the randomness of their biological counterparts.

Neuromorphic hardware can outperform classical chips for appropriate workloads where flexibility, robustness, and scalability are required, and where power efficiency is paramount. For example, mobile autonomous robotic systems that carry out tasks such as perception and decision making may benefit greatly from neuromorphic computing.

3.2.1 Intel Loihi[16,22,36]

Intel's Loihi* is a current example of a neuromorphic chip, that targets research labs. Since our system is designed to eventually run on such a chip, we highlight it here as an example.

The chip is fully digital and deterministic. It features 128 neuromorphic cores, each of which comprises 1024 spiking neural units. This amounts to a total of 131072 neurons, while the total number of possible synapses is 130 million. The cores are

* Intel and the Intel logo are trademarks of Intel Corporation or its subsidiaries.

arranged in a so-called many-core mesh and communicate asynchronously via an appropriate mesh protocol. Multiple chips may be linked together to form a larger planar mesh of cores. Interestingly, the protocol messages can not only originate from any of the cores, but also from the host CPU. There are also three conventional computing cores for handling input and output, as well as performing classical algorithmic computations on-chip. In detail, the neurons per core are hierarchically grouped in trees, where the neurons within a group together with the associated fan-in's and fan-out's share configuration values as well as state variables.

The chip uses a variant of the current-based (CUBA) neuron model, which belongs to the LIF class. Each neuron group has two state variables: synaptic response current, which is the weighted sum of all low pass-filtered input spikes and the neuron's/group's bias; and the membrane potential, which integrates the synaptic response current in a leaky fashion. As soon as the membrane potential reaches the firing threshold, the neuron generates a spike and resets the membrane potential to 0. Optionally, despite the chip being digital, additive stochastic noise may be enabled on the state variables.

These dynamics are approximated by a discrete time-step model with fixed-size time steps. Within a time step the order in which spikes arrive is irrelevant. While the neurons operate independently during the steps, all neurons are locally barrier-synchronized at the end of each step to maintain global consistency. This synchronization first flushes any spike messages still in flight, and then, again via barrier messages, advances each core's time step.

Whenever a neuron spikes, the action potential is routed to all cores contained in the associated fan-out list entries. The corresponding entry in a target core's fan-in list specifies the target neuron groups. In more detail, a neuron's outgoing connections are mapped to a number of entries in the containing core's fan-out list, specifying one target core per entry. The other endpoint of these core-to-core edges is a single entry in the destination core's fan-in list, containing a list of all neurons targeted by the source neuron, along with the weight and time constant per incoming synapse, i.e. per target neuron. The memory for such information is limited to $128\,\mathrm{KB}$ per core, which roughly amounts to 2000 incoming synapses. The number of entries in each core's fan-in and fan-out cannot exceed 4096 per list. Note that even connections within the same core need to go through the corresponding outgoing and incoming arrays. All mentioned limits influenced our development and design decisions at some point. For example, the largest possible fully connected network can contain about 300 neurons in order to satisfy Loihi's resource constraints. Our main attractor network alone has 1728 neurons and, in its standard form, would be fully connected.

Apart from the classical dense connection pattern, Loihi also supports sparse localized connectivity as well as topological constructs like broadcast or core-to-core multicast to share weights and save resources. Synaptic weights can be configured with variable precision. So-called population-based hierarchical connectivity is another resource-sharing mechanism of the chip. Disjoint populations of neurons with repeating inter-population communication patterns can be defined. Then, only one spike per destination population needs to be propagated, instead of one spike per destination core as in the regular connectivity setup. This mechanism could be useful for our use case, e.g. for the communication from the main attractor network to the three shift networks. Unfortunately, however, it is not supported by the Nengo Loihi implementation.

Loihi features sophisticated and flexible on-chip learning. As per the so-called locality constraint, connection weights can only be modified by the postsynaptic neuron using locally available spiking information. This locality allows for large-scale parallelization of the learning process.

Intel provides an API for interacting with the Loihi, i.e. creating spiking neural networks with desired neuron and synapse properties, compiling them, configuring the chip, and running the compiled code on chip. We use Nengo Loihi, a Nengo backend, to simulate our neural models as if they were run on an Intel Loihi architecture. Future work will use Nengo Loihi and Intel's NxSDK to run the models on an actual hardware chip.

3.3 Representation of Space and Navigation in the Mammalian Brain[2,10,11,13–15,20–24,27–30,33,35 38,42,46,47,49,50,54–58,64,66]

A complete navigation system requires three subsystems to cooperate: one to provide a metric of

space and a representation in the form of a map, one to anchor the internal spatial representation to environmental landmarks, and one to perform route planning. In this project we focus on the spatial metric part and how to keep it up to date - known as path integration.

While the neural mechanisms underlying the representation of space and the workings of spatial navigation in the mammalian brain are not yet fully understood, we do at least have a pretty good general and intuitive idea of it.

The entorhinal cortex (EC) and the hippocampus (HIP), as well as surrounding brain regions, have been found to be highly involved in spatial processing and cognitive mapping. The latter refers to creating and maintaining an internal representation of real or abstract space, and associating information with these representations, such as time or object properties. Often, the term 'visuospatial associations' is used when discussing the matter. We specifically deal with the spatial aspect, rather than the abstract one.

At the core, there are various neuron types in these brain regions that feature distinct spatially-tuned firing characteristics. Together and on multiple scales they encode various spatiotemporal features. Most relevant for us are head direction cells (HDC), grid cells (GC), place cells (PC), speed cells, border cells, as well as conjunctive cells. The latter are responsive to a combination of more than one feature, e.g. direction-by-grid or direction-by-grid-by-speed cells. We refer the reader to e.g. Grieves and Jeffery[30] for a review on such spatially-relevant cells. Essentially, single-feature cells maintain a stable representation of some instantaneous spatial feature, and conjunctive cells together with appropriate connectivity incorporate input into the system and update the single-feature cells' representations.[54]

The precise anatomical organization is subject to current research. Certainly, some form of interaction between those neurons is critical for achieving a coherent spatial representation, i.e. a cognitive map.[49] It is further believed that shaping and stabilization of the involved activity patterns could happen via dopaminergic input or reward. For example, firing fields could emerge at or shift towards goal locations associated with reward. There is also a phenomenon called theta phase precession, where the spatial representation continuously cycles through a brief time window around the present. This repeats for every theta cycle. Essentially, this provides the brain access to the immediate past, the present, and even the hypothetical or likely future. That kind of information is very useful for path planning and navigation.[14]

3.3.1 Place Cells

Encode the relative position in the current environment. To this end, they fire maximally at one or few locations in planar space. Importantly, place cells' tuning is unaffected by the current heading direction, velocity, or time. The firing fields resemble 2D Gaussians, centered at the respective encoded x-y position.

Place cells are organized in modules based on their firing field size, i.e. the variance of the Gaussian, along the dorsoventral axis of the hippocampus. From dorsal to ventral the size increases in discrete steps. A single module contains cells such that the combined (distinct) firing fields roughly cover the represented space. The mapping of place cells within a module is nontopographic, i.e. physically close cells are not necessarily correlated. Without loss of generality, we assume topographic arrangement. Combining multiple scales can give equal or better performance than just using a single module, while still being feasible regarding the number of required neurons.

In a constant environment the place fields remain stable. If, however, parts of the environment change, then place cells experience so-called remapping.[9] Depending on the severity of the change, remapping comes in different fashions. When moving to an entirely different environment, a complete remapping occurs - firing rates and locations of most place cells are changed at random. There is no correlation to the previous spatial tuning. When significant parts of the environment are modified, partial remapping takes place - some place cells completely alter their firing behavior, but the majority of them does not. Finally, when only small environmental features change, then we might observe rate remapping - only some firing rates are adapted, but firing locations are unaffected. Crucially, upon returning to or restoring the original environment, the original firing characteristics are fully recovered. This is important for spatial memory. For example, place cells are believed to play a role in associating landmarks with positions in an environment. They are further hypothesized to provide reset input to grid cells.

As a potential mechanism underlying the place cell system, 2D attractor networks have been proposed - one network per module. Put simply, neurons are topographically arranged at the vertices of a two-dimensional grid. The position of a neuron directly corresponds to it's preferred firing location in space. At any time, there is exactly one activity bump representing the current position estimate. It is sustained through recurrent connectivity between the neurons.

3.3.2 Head Direction Cells

Encode the orientation relative to some reference direction in the current environment. They fire maximally when facing a specific direction. Head direction cells collectively cover roughly all possible angles from $0°$ to $360°$. The firing behavior is invariant to the position in space, velocity, or time. Single tuning curves resemble one-dimensional Gaussians, centered at the respective encoded orientation angle.

A ring attractor model is often used to explain the dynamics of head direction cells. Neurons are topographically arranged on a ring according to their respective preferred orientation angle. At any time, there is exactly one packet of activity representing the current heading direction estimate. The firing activity is sustained through recurrent connectivity between the neurons. Some works also propose two hidden layers or rings as a plausible mechanism for bump translation. These comprise neurons tuned to positive or negative angular velocity, respectively. They receive initializing input from the main ring, and connect back to it with a counterclockwise or clockwise offset, respectively. Their activity is modulated by angular velocity input. Alternatively, there could be multiple pairs of hidden rings, each pair being connected to the main attractor with a different absolute offset, thus, achieving a different translation speeds of the activity bump.

While we focus on planar movement, real movement often happens in volumetric space or is at least possible in such space. Head orientation can be specified with three Euler angles: azimuth, pitch, and roll. Correspondingly, there exist neurons that (conjunctively) code for various combinations thereof. When only the direction is needed, azimuth and pitch are sufficient. For example, azimuth-by-pitch cells have been recorded in bats. A toroidal model parameterized directly by the two circular quantities fits the observed behavior at upside-down movement, and even accommodates pure azimuth and pure pitch cells as horizontal and vertical rings on the torus, respectively. A cell's tuning can be described as a von Mises function with the peak at the preferred angle. In two dimensions, this becomes the product of two 1D von Mises with independent peaks at the preferred angles.[23] The next section contains more information on toroidal manifolds; and Sec. 3.3.5 deals with 3D space from the perspective of encoding position.

3.3.3 Grid Cells

Realize a metric[4] coding of local position information. A grid cell's firing locations lie at the vertices of a regular hexagonal grid, hence the name, and thereby periodically tile the two-dimensional space. This is actually the most efficient way to cover a 2D surface.[12] The individual firing locations are of a 2D Gaussian shape.

Based on the underlying grid, these cells are defined by three properties: spacing, orientation, and phase. The phase refers to an offset from some arbitrary reference point. Similar to place cells, grid cells are organized in independent[59] modules of discretely increasing spatial scale, i.e. grid spacing, along the dorsoventral axis of the medial entorhinal cortex (MEC). Within a module, the cells differ in their phase but typically have the same orientation.[58] The increase in spatial scale approximately follows a geometric progression with a factor of about 1.42. Interestingly, factor of $\sqrt{(2)}(\approx 1.42)$ would double the area of the grid's hexagons per step.[59] The smallest grid spacing is thought to be about $30\,\mathrm{cm}$, and the largest about $8\,\mathrm{m}$. Grid cell's firing behavior remains unaffected by environmental changes, behavioral changes, or sensory inputs. The arrangement of grid cells is nontopographic, i.e. the three defining properties of neighboring cells are not necessarily similar. Without loss of generality, we assume topographic arrangement.

Due to the periodic firing fields, single grid cells only provide an ambiguous position estimate. Considering multiple cells with different phases and spatial scales collectively, however, we can precisely, efficiently, and robustly encode unique positions in large spaces. This is an instance of population coding: values are encoded by many neuron's activities; linearly combining them yields a global readout. The activities of involved neurons are collectively referred to as population vector. Actually, already with only a handful of cells

such coding mechanism is rather powerful as it allows for a large number of combinations of neuron activities. Note that the multiscale aspect specifically improves spatial resolution of the entire representation while maintaining support for long-range navigation, and also adds the capability of error correction.[58]

Grid cells are believed to be highly involved in path integration, i.e. they might integrate velocity and direction information and, in turn, provide self-motion related and landmark-independent spatial input to place cells. These also potentially decode the representation of grid cells.[67] In the other direction input from place cells may be capable of resetting grid cell activity, correcting drift and anchoring the grid to landmarks.[4,67]

We next discuss a potential model for explaining grid cell activity and interaction. The population activity of a grid module intuitively resides on a two-dimensional toroidal manifold, with single cells tuned to exactly one position thereon. Such torus can be parameterized by two angular coordinates, also referred to as toroidal coordinates, which are measured along two directions that intersect at an angle of $60°$. This configuration yields a twisted torus and, as a consequence, results in the firing field grid of single neurons to be hexagonal. To see this, consider the unrolled toroidal surface: the equilateral parallelogram, i.e. rhombus, can be repeated to tessellate 2D space and, when plotting the firing location of a single grid cell, a hexagonal grid becomes visible. This tessellation further illustrates how real space can be mapped to the conceptual internal manifold, i.e. how the position in the environment can be mapped to toroidal coordinates. This mapping defines a grid module's scale and is specified by the so-called population code. Because the unrolled toroidal manifold is a rhombus, there are not just the two axes of periodicity corresponding to the directions of the angular coordinates, at 0 and $60°$, but there is actually a third one at $120°$. We also call these the three axes of symmetry. The periodic ambiguity motivates why multiple grid cells of different phase and scale need to be combined.

An interesting observation is that a toroidal topology is not intrinsic to the encoded two-dimensional space, but was still (likely) chosen by evolution since it seems to get the job done with limited resources. This is different for multidimensional head direction representation, as mentioned in the previous section, because the product of two circular quantities naturally forms a torus. For example, conjunctive pitch-by-azimuth cells in bats seem to follow a toroidal neural code.[23]

The toroidal structure of grid modules remains unaffected by environmental changes, behavior states, or sensory inputs. Still, deformations of the hexagonal firing grid have been observed. These two facts suggest that any distortions manifest themselves in the mapping from real space to the toroidal manifold, potentially via modulation of velocity and/or direction inputs.

A potential nonlinear dynamical model that explains all described behavior, properties, and toroidal structure of grid cells is the two-dimensional continuous attractor network (CAN) with periodic and offset boundary conditions. There is one CAN per grid module. The neurons are topographically arranged on a rhombus, or, in other words, on a torus. Thereby, the most efficient way to cover the entire rhombus would be to arrange the neurons themselves in a regular hexagonal grid. The cells are recurrently connected with local excitation and local inhibition. Specifically, the connection weights follow a center-surround shape, which can be obtained as difference of two zero-centered Gaussians. A Gaussian with larger variance is subtracted from a Gaussian with smaller variance. The result is an excitatory region in the center surrounded by a ring of inhibition. Hereby, the balance between excitation and inhibition strengths and extents is crucial.[64] Alternatively, there are proposals for purely inhibitory connectivity and excitation from outside, which have potentially favorable characteristics such as greater resistance to divergence. In any case, the network is initialized by some excitatory input at the to-be-represented position. The population activity then quickly converges to a single stable 2D Gaussian bump whose center represents the current position estimate. Ideally, the network connectivity and dynamics allow the center of activity to be kept stable at an arbitrary location on the attractor manifold, which is where the term 'continuous' in CAN originates.

The position estimate needs to be shifted around on the toroidal manifold according the actual movement in real space. Therefore, visual input or self-motion information, e.g. angular and tangential velocity, is required. One hypothesized way to incorporate the latter is via hidden layers of conjunctive grid-by-direction-by-speed cells, but the exact work-

ings in the mammalian brain are yet to be discovered. We propose an effective solution based on modulated inhibition of distinct shifting ensembles of conjunctive cells in Sec. 5.7.

As an alternative to continuous attractor networks, some works suggested the oscillatory interference model.[11] There, two sinusoids of different frequency interfere and, under the right conditions, a triangular grid is formed. Excitatory and inhibitory connectivity is used for stabilization. The performance of this mechanism is significantly reduced in the presence of noise.

3.3.4 Path Integration

When navigating in some environment, there are two sources of information that can be used to track of the traveled path and to estimate the current position. The first source are external stimuli, such as visual, auditory, or olfactory cues. The issue with e.g. visual cues is that they are often ambiguous. Navigating only by such allothetic cues may be referred to as landmark navigation. Secondly, it can be system-internal information, such as vestibular signals or motor efference copies. Solely relying on such idiothetic cues is likely to accumulate error and drift over time. Biological and technical systems typically use a combination of both information sources for best performance.[20] Idiothetic information gives an unambiguous but noisy estimate and allothetic information is used to correct this estimate.

However, it is still possible to use only one of them. For example, rodents are able to navigate mazes in complete darkness and deprived of olfactory cues; hence, only using self-motion information. Navigating only by such information is known as path integration (PI) in biology, and as dead reckoning in robotics. Be it from the vestibular system, from commands sent to actuators, from velocity encoders on motors, or from inertial measurement units (IMU); some information is needed to compute the current angular and tangential velocity. These can then be integrated over time to update an internal estimate of the current orientation and position.

Relating the concept of path integration to the previously discussed spatially-tuned neuron types, (noisy) speed and direction input is integrated via grid cells and various forms of conjunctive cells, receiving themselves input from head direction cells and speed cells; while visual input for drift correction and resetting is incorporated via place cells.

3.3.5 3D Grid Cells[12,28]

Mammals live and navigate not only on a single 2D plane, but in a 3D world. Hence, appropriate mechanisms and cells are needed in the brain to handle more than two spatial dimensions: 3D place cells, 3D head direction cells, and 3D grid cells; the neural basis of which have been found in mammals.[69] Here we discuss the latter type, 3D grid cells. While their exact workings are not yet fully understood, there exist very plausible proposals.

These cells' firing characteristics are not simply an extension of 2D grid cells' perfectly hexagonal and globally ordered firing patterns to three-dimensions. Achieving such regular and stable patterns in 3D, especially with attractor dynamics, would require a huge number of grid cells. Instead, it has been found in e.g. bats that such cells' display only local order in their firing. Specifically, the firing fields per cell are separated by some characteristic distance, but there is no underlying global lattice. The individual firing locations are of isotropic shape. Similarly to 2D grid cells, the spacing of 3D grid cells' firing activity increases along the dorsoventral axis of the medial entorhinal cortex (MEC). It has been shown that pairwise interactions between neurons together with noise can generate a global hexagonal ordering in 2D as well as a local ordering in 3D. These interactions may originate from synaptic plasticity on incoming connections to the grid cells. The absence of global ordering could then be explained as the dynamical system getting stuck in some suboptimal solution; the optimum corresponding to the densest packing of spheres in three dimensions.

Another suggested mechanism to achieve local ordering is that there could be several differently-angled planes of 2D grid cell activity, which intersect at various locations, and also some cells that combine such activity into a locally-ordered response. In favor of such theory is the fact that most mammals, be it rats or even bats, tend to stick to a set of preferred 2D planes when navigating 3D space.

The continuous attractor network model is hypothesized to be compatible with a local ordering in 3D after some modifications.

Note that biological 3D grid cells are not to be confused with our 3D grid-by-direction cells found e.g. in the main attractor network. Nonetheless, this section illustrates the difficulty of getting regular and stable global order of neuron activity in 3D.

3.4 Neural Engineering Framework[5,18,48]

The core idea of the Neural Engineering Framework (NEF) is that biologically-plausible and large-scale neural systems can be specified functionally in interpretable real vector spaces and/or as dynamical systems using control theory. Based on this specification, NEF creates neural ensembles with arbitrary neuron models and automatically computes the synaptic connection weights. The system and its underlying neural properties can then be simulated efficiently and analyzed in detail.

This is opposed to the significantly less-intuitive approach of constructing neural systems from the electrophysiological side, but can achieve the same results.

The framework consists of three basic principles. For ease of illustration we consider the example of computing $y = f(x)$, where $x \in X$ and $y \in Y$ are elements in the vector spaces X and Y, respectively, and the function $f\colon X \to Y$ projects from X to Y. This is inspired by the Appendix section in Bekolay et al.[5].

Representation. * Essentially, a form of population coding is used to represent a time-varying real-valued vector with a population of neurons. In other words, a specific vector space is associated with each population. It is important to note that usually only the unit hypersphere within some vector space is meant when referring to a vector space in connection with NEF/Nengo. To make that explicit we sometimes include the specifier 'normalized'. In the running example, we associate X with population N_1.

There is a nonlinear encoding phase, where the vectors are treated as input signals to the neural population. Each neuron has a tuning curve that determines how strongly it responds to a given input signal. Specifically, each neuron has an encoder, which is a unit vector in the population's associated vector space. The encoder can be interpreted as the neuron's preferred direction in that space. The raw input current

$$I_{raw}(x) = e_i \cdot x$$

to neuron i in N_1 is computed as the dot product between the neuron's encoder e_i and the input x. The encoders are collected in the matrix E_1. Note that I_{raw} is in the neuron current space. So, encoders project from vector space to current space. Based on the fact that usually only unit hyperspheres of vector spaces are considered, the current space typically concerns the range from -1 to 1. To this end, Nengo automatically normalizes encoders to unit length, i.e. they are actually points on the surface of said hyperspheres. In case the input vector is of length greater than 1, implicit normalization after projecting onto the encoder may be assumed. The actual input current I to the neuron is then computed as

$$I(I_{raw}) = \alpha I_{raw} + I_{bias} \quad ,$$

with the neuron's gain α and constant bias current I_{bias}. Finally, the neuron's nonlinear activation function determines whether a spike is generated. For our example,

$$a(x) = G(I(I_{raw}(x))) \quad .$$

The tuning curve of a neuron effectively corresponds to the function $a(x)$ from vector space to neural activity. The function $a(I_{raw})$ is called response curve, which projects from current space to neural activity. Note how the former is typically multidimensional, whereas the latter is always scalar. Intuitively, the response curve is a tuning curve along the neuron's preferred direction. Furthermore, since tuning curves are defined on abstract vector spaces, they can increase or decrease with increasing alignment to e; whereas response curves operate on electrical currents and, hence, realistically only increase for increasing input.

There are two more important properties. Intercepts denote the value in current space at which a neuron starts to fire. And the maximum firing rate corresponds to a neuron's firing rate when it receives an input current of 1, i.e. the largest value in current space. A better name for it would be nominal firing rate, because with an input current larger than 1 the firing rate can actually be higher.

The user can specify either gain and bias, or intercept and maximum firing rate, and Nengo computes the respective other pair of properties. This

* Xuan Choo's posts clarified some important details, https://forum.nengo.ai/t/reservoir-example-with-spiking-neurons/2111/10, https://forum.nengo.ai/t/nengo-nef-algorithm-finding-gain-and-bias/1706/2, and https://forum.nengo.ai/t/nengodl-simulator-provides-different-results-in-training-network/1913/4.

is completely interchangeable. Also, sometimes we might refer to scaled encoders, which means encoders that have already been scaled by the neuron gains.

Until now, we only discussed how real-valued vectors are encoded by a population of spiking neurons. The neural activity can be decoded to approximately get the original values back.

For decoding, the discrete-event spike trains are first temporally filtered with exponential decay in order to get a continuous signal. The filtering accounts for the biological process of a spike causing a nonzero postsynaptic current. These signals are then multiplied by appropriate decoder weight matrix D and summed up, resulting in an approximation $\hat{x}$ of the original values x, i.e.

$$\hat{x} = D_1 \cdot a(x) \quad .$$

The decoding weights per connection are automatically determined by Nengo based on the encoders, gains, biases, and the desired function by least squares optimization. Naturally, the more neurons are used in the population the more accurate the decoding. This minimization problem is usually solved using randomly sampled points from the population's associated vector space, making the decoders independent of any input.

Transformation. We can not only realize identity functions with this encoding-decoding scheme, but any linear or linearly-approximated nonlinear function or transformation of the input signal. This happens in the decoding phase by simply premultiplying the decoder matrix by some appropriate matrix.

In the example we actually want to approximate $f(x)$ which might be different from the identity function. So, we need to determine

$$D^f : X \to Y$$

as a linear approximation of f, and then obtain the composite decoding matrix as

$$D^f \cdot D \quad .$$

Conveniently, the NEF computes D^f for us given f. Since the decoding weights correspond to synaptic weights in the neural system, the arbitrary nonlinear function is effectively computed fully by the physiologically-realistic neural system.

With the presented methods, it is straightforward to realize connections between neural populations that compute any function. The decoded and transformed values from the first population are passed to the encoding logic of the second population. All three involved operations correspond to matrix multiplications and hence can be concatenated into a single matrix. This matrix represents the neuron-to-neuron connection weights between the two populations. During simulation, however, the matrices are usually stored separately to save space. This is also computationally more efficient and allows for large-scale models.

Dynamics. We mentioned dynamical systems multiple times in this section. These typically feature system-internal state variables that evolve over time. By recurrently connecting a neural population to itself, persistent activity can be achieved and the population's firing activity effectively becomes a state vector.

The advantage of using NEF for realizing dynamical systems is that the analysis and design can happen entirely in vector space using classical control theory, while the model is automatically translated to and from a neural substrate using the first two principles. The simulation is done with neural dynamics. The various parameters of ensembles and connections, such as synaptic delays or refractory time constants, heavily influence the behavior of the system.

3.4.1 Nengo*

Nengo is a neural compiler and simulator for models symbolically-specified according to the NEF principles. It uses ensembles to represent information via neural populations, and connections between ensembles to realize information transformations. The translation from high-level specification to low-level neural constructs happens partially automatically. The construction and simulation of models are decoupled, which allows for different simulators or backends to be used with the same model.

In more detail, Nengo's object model comprises six basic types of objects.

An ensemble represents a population of neurons and encodes information according to the representation principle of the NEF. The dimension of an ensemble is defined as the dimension of the encoded

* Xuan Choo's post clarified some important details, https://forum.nengo.ai/t/nengo-nef-algorithm-finding-gain-and-bias/1706/2.

vector space. At creation, the encoding weights can be specified. Alternatively, Nengo determines them as outlined previously.

A key part of an ensemble is the neuron model that is used for each of its neurons. Such models are parameterized by various values, which all affect the shape of the tuning curve.

We look at the equations for two common neuron types: the rate-based Rectified Linear Unit (ReLU) neuron type as well as the spiking Leaky Integrate and Fire (LIF) neuron type.

A ReLU neuron is modeled as a rectified line. That is, the neuron's activity scales linearly with current, unless it passes below zero, at which point the neural activity will stay at zero. The activation function is

$$a(x) = \beta \cdot \max(0, I(x)) \quad ,$$

where β defines a scaling of the neuron output.

The LIF model's behavior on the other hand is nonlinear. It is parameterized by four values. The membrane time constant tau_rc defines how many seconds it takes for the membrane potential to decay to 0 in the absence of input. The absolute refractory period tau_ref defines how many seconds the neuron is non-excitable after a spike, i.e. how long the membrane potential stays at 0. There is also a scaling factor β on the neuron output. Finally, I_{th} denotes the threshold current for the neuron to generate a spike. The firing rate a is computed as (adapted from Eliasmith and Anderson[18])

$$a(x) = \begin{cases} \frac{\beta}{\tau_{ref} - \tau_{RC} \ln(1 - \frac{I_{th}}{I(x)})}, & \text{if } I(x) > I_{th} \\ 0, & \text{otherwise} \end{cases}$$

Note that the described version of LIF is not perfectly biologically accurate since it does not include any form of adaption, e.g. the reduction in firing rate in response to constant input current. If such behavior is required, adaptive LIF models can be considered.

The second object are nodes. They are not part of the actual neural model, but are used for interfacing with it, i.e. handling input and output. Nodes can e.g. model sensory stimuli or entire environments that respond to the behavior of the neural model. At each timestep, the output of a node is computed directly from its input.

Then there are connections. These can be formed between ensembles and nodes and compute arbitrary nonlinear functions. When ensembles are involved, NEF's transformation principle is applied. Connections can either be conceptual, i.e. on the vector-space level, or direct, i.e. on the neuron level. For direct connections the user manually specifies the synaptic weights and the transformation principle is not applied. Furthermore, synaptic delay can be configured and is enabled by default. Learning rules provide functionality to modify the connection weights during simulation.

In order to analyze Nengo simulations, usually probe objects are used. They can monitor, record and store all kinds of data during a simulation. Examples are membrane potentials and spike trains. Since there are typically a large number of time steps, monitoring multiple variables or vectors is resource intensive and can slow down the simulation considerably.

The last two objects, networks and models, both encapsulate related functionality in form of interconnected ensembles and nodes. This grouping can be hierarchical. It potentially makes working complex neural models more approachable, since certain parts can be abstracted away. The only real difference between network and model is that a simulator takes a model as input. Thus, a model object needs to contain the entire neural model.

Nengo provides a platform-independent reference simulator simulators work with signals, that represent values, and operators, that represent computations performed on signals

There exists also a backend and simulator specific for Intel Loihi, which is already mentioned in Sec. 3.2.1 and is referred to regularly throughout Chap. 5.

3.5 Fourier Transform

Simply put, the Fourier transform converts between a representation, e.g. vector or function, in the time/spatial domain and a corresponding representation in the frequency domain. Essentially, a time/spatial representation is approximated by a weighted sum of complex exponentials, i.e. harmonics, of different frequency. Thereby, constructive and destructive interference work together. By going from sums to integrals, i.e. using infinitely many harmonics, the approximation becomes exact. The Fourier transform computes the weights in the weighted sum, which consist of amplitudes and phase shifts, i.e. complex numbers. These weights are referred to as Fourier coefficients. Working in frequency space has certain advantages. For

example, some mathematical operations may be more efficient or analyzing certain aspects may be easier. We employ the Fourier transform as a means for dimensionality reduction and, additionally, are able to manipulate the transformed data intuitively in frequency space.

More formally, as adapted from Osgood[52], let

$$\mathbf{x} = (x_1, x_2, ..., x_n)$$

be a point in $\mathbb{R}^n$, which we refer to as direct or physical space, and

$$\boldsymbol{\xi} = (\xi_1, \xi_2, ..., \xi_n)$$

be a (corresponding) point in the n-dimensional frequency space, also $\mathbb{R}^n$, which is often referred to as dual or reciprocal space. Let further $f(\mathbf{x})\colon \mathbb{R}^n \to \mathbb{C}^n$. The Fourier transform of f is then defined as

$$(\mathcal{F}f)(\boldsymbol{\xi}) = \hat{f}(\boldsymbol{\xi}) \coloneqq \int_{\mathbb{R}^n} e^{-2\pi i \mathbf{x}\cdot\boldsymbol{\xi}} f(\mathbf{x})\, d\mathbf{x} \quad , \quad (3.1)$$

where $\mathcal{F}$ is the Fourier transform operator and $i \in \mathbb{C}$ is the imaginary number.

Now, let $g(\boldsymbol{\xi})\colon \mathbb{R}^n \to \mathbb{C}^n$. The inverse Fourier transform of g is defined as

$$(\mathcal{F}^{-1}g)(\mathbf{x}) \coloneqq \int_{\mathbb{R}^n} e^{2\pi i \mathbf{x}\cdot\boldsymbol{\xi}} g(\boldsymbol{\xi})\, d\boldsymbol{\xi} \quad . \quad (3.2)$$

Note the missing minus sign in the exponent within the integral.

Since the points in direct and reciprocal space are defined as vectors, the above definitions hold for any number of dimensions.

When f maps to $\mathbb{R}^n$, then there is a certain symmetry in the set of Fourier coefficients of f. Specifically, the coefficients pertaining to $-\boldsymbol{\xi}$ are precisely the complex conjugate of the coefficients pertaining to $\boldsymbol{\xi}$,

$$(\mathcal{F}f)(-\boldsymbol{\xi}) = \overline{(\mathcal{F}f)(\boldsymbol{\xi})} \quad .$$

We work with the aforementioned approximation as sum of finitely many complex exponentials and, hence, are in the realm of the discrete Fourier transform (DFT). The equations are essentially the same as Eqs. 3.1 and 3.2, where the integrals are replaced by sums and normalization factors are included. The standard algorithm used to compute the DFT is the Fast Fourier Transform (FFT).

For an input array of length n, the DFT outputs an array of n coefficients, corresponding to n distinct frequencies. Stemming from the fact that we are working with complex exponentials and potentially get complex-valued inputs, the output components contain both positive and negative frequencies. When working in a time domain, the input points typically correspond to evenly spaced samples of a time-varying signal, e.g. one sample per second starting at 0. The DFT's output coefficients then correspond to frequencies in cycles per second. When working in a spatial domain, the input points are obtained similarly by sampling along some spatial dimension, e.g. one sample per meter starting at 0. This means the Fourier coefficients pertain to frequencies in cycles per meter. The spacing of the input points Δt determines the highest frequency component that can be computed. For example, if we want to obtain a frequency of at least $4\,\mathrm{Hz}$ then the sample spacing may be at most $1/(4 \cdot 2)\,\mathrm{s}$. Another important aspect is that the input is assumed to be periodic. For example, if we have a signal that repeats itself every 2 seconds, then we need to evenly sample the input for the Fourier transform within a 2-second window of the signal. This ties in nicely with the fact that the complex exponentials are periodic.

We discuss the output of the DFT at the example of SciPy's FFT implementation. As per the official documentation,* the coefficients in the output array pertain to the frequencies

$$\frac{1}{n\,\Delta t} \cdot \left[0, 1, \ldots, \frac{n}{2} - 1, -\frac{n}{2}, \ldots, -1\right] \quad (3.3)$$

for even n, and to

$$\frac{1}{n\,\Delta t} \cdot \left[0, 1, \ldots, \frac{n-1}{2}, -\frac{n-1}{2}, \ldots, -1\right] \quad (3.4)$$

for odd n, where the final frequency values are in the unit 'cycles per unit of the sample spacing'. We observe that the constant component comes first. It contains the average intensity of the signal. Then there are the positive frequency components in increasing order, followed by their symmetric negative counterparts in decreasing order. For even n the largest frequency is called the Nyquist frequency. Notably, it does not have a negative counterpart in the output array. SciPy implicitly assumes a sample spacing of 1 over a window of length n of the input signal.

* https://docs.scipy.org/doc/scipy/reference/generated/scipy.fft.fft.html#scipy.fft.fft, https://docs.scipy.org/doc/scipy/reference/generated/scipy.fft.fftfreq.html#scipy.fft.fftfreq

The discrete Fourier transform of multidimensional signals can be computed by applying a one-dimensional DFT over the individual dimensions one after the other, where the individual DFT operations work on the output of the respective immediately preceding operation. The set of Fourier coefficients as computed by SciPy's implementation of the multidimensional FFT,* which is an array of the same dimension as it's input array, is organized as follows: The 0-frequency component resides at index $[0, \ldots, 0]$. The positive frequency components occupy the first half of the axes. The Nyquist term is in the middle of any even-length dimension. The negative frequencies make up the rest, i.e. the second half of all axes.

Shift Theorem. This theorem essentially states that a shift in the input domain, i.e. time or space, corresponds to a phase shift in the frequency domain.

As adapted from Osgood[52], assume we shift $\mathbf{x}$ by the vector

$$\mathbf{b} = (b_1, b_2, ..., b_n) \quad .$$

Then, if $(\mathcal{F}f)(\boldsymbol{\xi})$ is the Fourier transform of $f(\mathbf{x})$, the shift theorem states that the Fourier transform of $f(\mathbf{x} \pm \mathbf{b})$ is

$$(\mathcal{F}f(\mathbf{x} \pm \mathbf{b}))(\boldsymbol{\xi}) = e^{\pm 2\pi i \mathbf{b} \cdot \boldsymbol{\xi}} \, (\mathcal{F}f)(\boldsymbol{\xi}) \quad . \tag{3.5}$$

For the discrete case and when representing $\boldsymbol{\xi}$ as a vector of interleaved real and imaginary parts Eq. 3.5 can be written as a rotation in vector space, where the complex exponential factor is transformed into a rotation matrix. The quick explanation is the following: In one dimension $\boldsymbol{\xi}$ is a complex number and can be represented as a vector with two elements, one for the real part and the other for the imaginary part. Further, we note that the multiplication with the complex exponential corresponds to a rotation in the complex plane about the origin since the factor's amplitude is 1. This complex exponential can be converted into a complex number via Euler's formula, $e^{ix} = \cos x + i \sin x$. Then we can construct a rotation matrix $\mathbf{R}$ that performs the desired rotation in vector space. This extends to higher dimensions and is discussed further in Sec. 5.2.

* https://docs.scipy.org/doc/scipy/reference/generated/scipy.fft.fftn.html#scipy.fft.fftn

4 Related Work

There are multiple engineering approaches that take inspiration from or try to emulate parts of the spatial processing system of mammals, both conceptually and architecturally. For example, simultaneous localization and mapping (SLAM) where the spatial map is constructed while at the same time being used as reference for localization, was implemented in a biologically-inspired fashion by Milford, Wyeth, and Prasser[43], who focus on the functional aspect; and by Kreiser et al.[35–37], who additionally introduce anatomical realism by using neuromorphic concepts and hardware. We discuss various works that were of relevance to our development process, inspired solutions to issues that arose along the way, or constitute an interesting contrast to our approach.

4.1 Different Neuron Arrangements for Modeling Grid Cells

Wang, Yan, and Tang[66] achieve multiscale hexagonal firing patterns of grid cells with recurrent neural networks. The agent's self motion is mapped to neuronal activity via a Jacobian matrix. The authors discuss path integration in 1D, 2D and 3D. To that extent, they evaluate the connection pattern between grid cells and place cells and demonstrate the functioning of grid cells as a metric for coding space.

In addition to analyzing the coding performance of different grid scales in terms of accumulated coding error, the authors compare the performance of different phase distributions in 2D and 3D space. The evaluated grid patterns are uniform random, square tiling, and hexagonal tiling. They report a significant performance benefit of the hexagonal over the square tiling, particularly with smaller network size. As the network size increases, the different phase distributions approach uniform coverage and the coding error performances approach a common limit value. Due to limited computing resources and, later on, hardware restrictions, the performance with lower numbers of neurons is most relevant for us. The authors settle for an FCC lattice structure to model the 3D grid cells. The conclusion for us is to arrange the neurons of each layer of the 3D pose network in a hexagonal grid.

Further, grid cells' metric power is quantified in the paper, i.e. the required numbers of place cells to achieve specific coding precision depending on the size of the environment.

Wang et al. also mention a representational model for grid cells, which represents the agent's position as a high-dimensional vector and realizes displacement by applying matrix transforms to that vector.[26] This seems to be conceptually similar our approach in Fourier space.

4.2 Gray Code Model for the Encoding of Grid Cells

We can also argue for a hexagonal neuron grid from the perspective of Gray codes and spatial indexing. Monteiro, Pedro, and Silva[46] propose new multilayer Gray encodings for representing geospatial data in one and two dimensions. For the former case, they consider periodic and non-periodic signal representations. For the latter case, an equilateral triangular coordinate system is employed. They test their 2D Gray encoding of grid cells in the navigation system of a robot and successfully demonstrate the formation of patterns that resemble those of real grid cells' firing fields.

The authors explain the advantages of hexagonal coordinate systems over traditional rectangular coordinate systems for spatial indexing. When considering neighboring coordinates in the rectangular case, they can either be connected by an edge or lie diagonal from each other, the distance between the neighbors being different for the two cases. In contrast, neighboring coordinates in the hexagonal coordinate system all have the same distance between them. Another way to look at it is to compare distributions of absolute distances between points within the squares or hexagons, and their centers. The standard deviation for a hexagonal shape is lower.

Monteiro et al. conclude that using the centers of identical and space-tiling hexagons for indexing points inside an area results in higher preci-

sion compared to square tiles. Consequentially, we should prefer a hexagonal lattice over a rectangular one when partitioning 2D space.

4.3 Toroidal Topology of Population Activity in Grid Cells

Gardner et al.[27] attempt to shed some light onto the logical structure underlying a single grid cell module's population activity. They simultaneously record hundreds of grid cells to generate a solid data set for topological analysis. Up to then, only small sample sizes had been available in the literature.

A number of plausible explanations for the hexagonal pattern of firing locations of grid cells have been proposed, recurrently connected continuous attractor networks (CAN) being the one under consideration by Gardner et al. . While CANs fit the population code's invariance across environments, independent of sensory inputs, they cannot be singled out as definitive underlying mechanism. Other feedforward-based explanations are still equally viable.

To advance the understanding of this matter, the authors employ persistent cohomology analysis. Topological holes, such as spheres, rings or higher-dimensional structures, are tracked over time in multidimensional spaces of point clouds. The set of holes with distinctly long lifetimes can be used to reliably classify the structure underlying the point cloud. Barcode plots can aid the visual identification of interesting topological holes. For the current use case, the point cloud pertains to firing rates of grid cells. Persistent cohomology analysis yields that a grid cell module's population activity is structurally similar to a (twisted) torus. Specifically, this requires four distinctly long-lived holes: one 0D hole, two 1D holes, and one 2D hole. The authors further apply nonlinear dimensionality reduction to the data set in order to create a 3D visualization of the population activity's toroidal structure.

Gardner et al. also identify a mapping between the population activity and the torus' surface. To this extent, they calculate coordinates from the two 1D holes, which are associated with circular structural features. When separately plotting the cosine of the two coordinates in 2D space, single-frequency plane waves with different orientations become visible. The orientations of the two waves are about 60 degrees apart. Hence, these angular coordinates directly correspond to the parameterization of a hexagonal torus.

Using the cohomology approach, the authors show that the toroidal coordinates of individual grid cells are invariant to external sensory inputs or environmental changes. They speculate, that the grid code itself is unaffected but that distortions in the grid pattern manifest in the mapping between space and toroidal coordinates.[10] They further argue, that since grid-cell toroidal coordinates remain invariant in scenarios where the 'correlation structure of place-cell activity'[27] changes, continuous attractor networks are likely at play here. These are the only models that predict such behavior, as opposed to other feedforward models. Hence, it is viable for us to engineer the toroidal dynamics of our pose-cells using intrinsic network connectivity.

Note, however, that alternative mechanisms to CANs may coexist, e.g. feedforward mechanisms. These may be particularly relevant before and during emergence of proper recurrent connectivity, potentially even enabling it.

In their extended material, Gardner et al. compare two different CAN models for implementing grid cell behavior. The first one is based on the approach of Burak and Fiete[10] (see Section 4.12) and uses local center-surround connectivity to generate population activity that resembles a hexagonal grid. Velocity input is incorporated via patches of neurons with different orientation tuning, i.e. having their center-surround outgoing weights offset into a specific direction. As proposed by Couey et al.[15] (see Section 4.11), Gardner et al. opt for purely inhibitory recurrent connectivity combined with excitatory drive from outside, potentially originating from the hippocampus. It is not explicitly stated in the paper, whether the borders of the neural sheet are connected in a twisted fashion, but just that opposite edges are connected. Nonetheless, the single-neuron responses exhibit firing fields arranged on a hexagonal lattice, as reported by Burak et al. and Couey et al. Importantly, as hinted above, the resulting population response features multiple bumps of activity. The second CAN model is based on the twisted torus model from Guanella, Kiper, and Verschure[32] (see Section 4.13). Its recurrent connectivity consists of local excitation and, in contrast to the first model, global inhibition. The outgoing excitation strength decreases with increasing distance from a neuron's center, while the inhibi-

tion strength increases with distance. The edges of the neural sheet are connected so that it forms a twisted torus. Another difference to the first model is that a neuron's outgoing weights are modulated by velocity input, i.e. the center of excitation is dynamically shifted away from the neuron's center on the neural sheet into the direction of the instantaneous velocity vector. Due to the global recurrent inhibition, the resulting population response exhibits a single bump of activity, which is precisely what we desire from our model. As outlined in the respective sections, we combine elements from Burak and Fiete[10], Couey et al.[15], and Guanella, Kiper, and Verschure[32].

Gardner et al. also compare two different versions of wraparound connectivity: square (i.e. simply connecting opposite edges) and twisted (e.g. connecting one opposing edge-pair regularly, and the other pair with an offset. Note, that there might be an offset for each pair of edges if the torus' flat representation as a parallelogram and the neural sheet are not aligned to one axis.). A cell's firing field on the torus is defined as a 2D Gaussian. When depicting the firing field on the unrolled torus and tiling 2D space with the corresponding parallelogram, a square wraparound results in a square grid, while the twisted wraparound gives rise to a hexagonal firing pattern. When plotting the cosine of the two toroidal coordinates separately, the two resulting plane waves have a relative angle of about 60 degrees for the square torus, and about 90 degrees for the twisted torus. This suggests that we ought to implement our neural sheet, or rather each horizontal slice of the 3D attractor network of neurons, with twisted wraparound connectivity in order to obtain a hexagonal single-neuron response pattern (see Section 5.4).

4.4 Biomimetic FPGA-Based Spatial Navigation with Grid Place Cells

Krishna et al.[38] present a biomimetic spatial navigation system comprising three modules, that they eventually test on a field-programmable gate array (FPGA) chip.

The grid-cell module consists of multiple independent continuous attractor networks (CAN) of neurons, as proposed by Burak et al. (see Section 4.12), each featuring a hexagonal grid-field of different spatial scale. This multi-layer architecture resembles the modules of varying grid-scale along the dorsoventral MEC. Each grid-cell layer consists of 30 by 30 neurons arranged on a torus. Actually, this is the smallest network size that supported the emergence of grid-like firing fields. Due to hardware constraints, the authors modify the conventional Mexican hat recurrent weight shape to a step-like ring of inhibition, resulting in a binary weight matrix. They pair this inhibitory connectivity with excitatory drive from outside. This stark simplification is not necessary in our case, since Intel Loihi supports more elaborate synaptic weights without much overhead (see Section 3.2.1). Nonetheless, since the binary ring of inhibition is reported to yield quantitatively and qualitatively similar grid cell firing as the Mexican hat excitatory-inhibitory connectivity (see Section 4.11), we opt for a trade-off: we crop small values in our weight matrix to zero in order to reduce the number of recurrent connections. Krishna et al. achieve the different grid-scales by adjusting the parameters of inhibition, i.e. the inner diameter and the width of the ring. For example, if the inner diameter is kept constant and ring width is increased, the spacing between firing fields increases; and if the width of inhibition is kept constant but the diameter is increased, the firing fields' size increases. In contrast, we envision to realize different spatial grid scales by keeping the meticulously tuned recurrent connection weights the same and instead only scale the velocity input to the individual pose networks. Apart from the recurrent connectivity, the neurons selectively receive velocity input. Each neuron has one of four preferred directions: 0, 90, 180, or 270 degrees. Correspondingly, the center of the inhibitory ring of outgoing weights is offset from the neuron's location on the torus. Velocity input to a neuron is modulated depending on how well the movement direction aligns with the neurons preferred direction. Each 2D grid-cell layer is uniformly tiled by patches of neurons of each orientation tuning, i.e. by repeating sets of four neurons. The authors do not explicitly state whether all four neurons are assigned to the same position on the torus, or to different adjacent positions. In any case, when there is no velocity input, the translational force of each neuron is canceled out by a neuron of opposite tuning direction, hence, the activity packets remain stationary. Their way of incorporating velocity input is considerably different from our approach. One similarity, however, is the pres-

ence of neurons with directional tuning and selective routing of velocity input depending on the heading direction (see Section 5.7).

Secondly, the place-cell module also comprises multiple layers of different spatial scales, but the neurons here typically fire at just one location in space. Place cells are not recurrently connected, but receive their only inputs from the grid-cell module. Specifically, each place cell gets one input from each grid-cell layer, one of which having a larger connection weight than the others. The selection of cells that connect to place cells is random. The convergence of inputs from grid cells of different spatial scales and phases results in the desired place fields. Krishna et al. report, that the grid of firing fields exhibits significant drift, supposedly caused by the attractor dynamics themselves. Decoding positions through the place cell module, instead of directly from the grid-cell activity, apparently mediates this issue. This means that we may eventually need to implement place cells to obtain good performance over longer time periods without visual reset input. However, our current model shows only mild drift behavior. Hence, decoding the instantaneous pose directly from the grid-cell-like pose network seems to be viable.

Lastly, the authors employ a two-layer neural network as a black-box decoding module, arguing that the exact mechanism of decoding grid-cell and place-cell activity in the brain is not known yet. The neural network receives inputs from all neurons in the place-cell module and outputs the estimated current position on the planar surface.

Krishna et al. are one of few who implement a comprehensive spatial navigation system, comprising both grid and place cells, on hardware, specifically on an FPGA chip. They use time-multiplexing to save resources and manage to fit the entire system on the same chip.

They demonstrate precise and stable estimation of the robot's trajectory in real time, notably any without vision-driven reset. Decoding efficiency is reported as an evaluation metric. To this end, the authors show the mean squared error (MSE) between the actual trajectory and the one estimated by the system for different setups. Good parameters are 250 place cells per module and 5 layers of grid modules. Drawbacks of their approach are increased power consumption over an analog system, as well as memory consumption.

4.5 Multi-Scale Grid and Place Cells for Cognitive Map Building

Wang, Yan, and Tang[67] design yet another model of grid cells and place cells for path integration based on continuous attractor networks. They put emphasis on optimizing the multiscale aspect, i.e. combining the activities of grid-cell modules of different spatial scales (and phases) to obtain position estimates in the form of place cell activations. To this end, they parameterize the recurrent weight function differently than e.g. Burak et al. (see Section 4.12). That allows them to easily compute the weight matrix for the different grid-cell layers. Note, that the general shape still follows the traditional Mexican hat function, i.e. center-surround excitatory-inhibitory connectivity. Place cell firing is improved, i.e. filtered and stabilized, by applying a sigmoid function to threshold weighted sums of grid cell activations. This simple but effective addition might prove useful for a future extension of our system to multiple grid scales. When considering an enclosing Gaussian function of the actual Mexican hat shape and using the three-sigma rule of probability theory, the area of effect of the outgoing weights can be approximately enclosed by a circle with a radius of three times the standard deviation. Wang et al. continue to visualize the roughly one-to-one correspondence of this radius and the period of the associated grid cell's firing fields.

The authors use a square neural sheet with wraparound connections, where each neuron is tuned to one of four directions 90 degrees apart. The neurons are organized into two-by-two patches containing one neuron of each orientation preference. These patches are repeated to tile the neural sheet. Velocity input is selectively fed to neurons depending on their preferred direction. If the velocity is zero, all neurons within the patches receive the same input and, thus, their directional drive cancels out to yield stationary grid-like firing. Wang et al. 's approach to integration of velocity information provided a valuable direction of research during our development process.

4.6 Implementation of Head Direction and Place Cells on Intel Loihi

Fieweger[22] propose a biologically-inspired localization and mapping system, comprising head direction cells, grid cells, and some functionality of place cells. They start out from a previous implementation, which is ultimately based on work from Kreiser et al. (see Sections 4.7 and 4.9), but that is limited in its ability to shift around activity bumps on the neural sheet arbitrarily. Fieweger not only address the theoretical and software aspects, but also consider robotic sensors and run their system on an Intel Loihi neuromorphic chip - the same one our implementation is designed to work with.

The head direction and grid modules are realized as continuous attractor networks of spiking neurons, based on previous work by Conklin et al. (see Section 4.15). They work in frequency space, meaning that orientation and position estimates correspond to the set of Fourier coefficients of appropriately centered one- or two-dimensional Gaussian functions, respectively. Manipulation of encoded estimates, i.e. translation in space, is done via rotation in frequency space. The neurons of the head direction module are connected in a ring. The neurons of the grid module are arranged in a rectangular grid on the square neural sheet, and opposite edges are connected to form a regular torus. Conveniently, the Fourier basis is circular and, thus, directly supports the periodic representation. Importantly, their grid cells, while not explicitly reported, have firing fields arranged on a square lattice, instead of a hexagonal one. The reason is the non-shifted wraparound connectivity, which only yields two axes of periodicity, whereas three would be required for a hexagonal grid. Also note that they only work within the unit cell of the grid-cell module and, therefore, effectively treat the neurons as place cells. While we adopt the approach of working in frequency space and using continuous attractor networks, our neural sheets for representing planar space have their neurons arranged on a hexagonal lattice and, crucially, have the edges connected with an offset so that they form a twisted torus. Moreover, we support the representation and propagation of multiple simultaneous activity bumps.

Fieweger employ the Neural Engineering Framework (NEF) and the corresponding Python library Nengo. They use one ensemble for the head direction ring attractor, and one ensemble with 63x63 neurons for the position attractor. Connections between ensembles are realized by simply specifying the desired transformation or function that the connection is to compute, and Nengo would then solve for the required weight matrices. In contrast, we specify most connections between ensembles in the form of explicit neuron-to-neuron weights. While we rely on Nengo to obtain a baseline weight matrix for some of these connections, all weights are meticulously fine-tuned by hand in order to achieve the desired behavior. Another difference between their work and ours, is that we manually split the ensembles to fit onto Loihi cores and to fulfill the associated connectivity restrictions, whereas they purely rely on Nengo to split the ensembles according to some generic rules.

Velocity information is incorporated as follows. The orientation estimate and the tangential velocity is combined to obtain the velocity components in x and y direction. This computation happens with a Nengo connection. A dedicated ensemble is required for this, because all variables, upon which a connection is dependent, need to be represented in the same ensemble. The two velocity components are then fed into the position ensemble, which, for the same reason, actually works in the representational space of position (in the form Fourier coefficients) plus the two velocity components. The recurrent connection of the position attractor network, is specified as the identity transform followed by a rotation along the x-axis proportional to the corresponding velocity component, and by a rotation along the y-axis scaled by the remaining velocity component. The separate application of rotations simplifies implementation, but is actually an approximation (see Section 4.15). Angular velocity is incorporated into the head direction ensemble in a similar manner, just with one fewer dimension. We started out by generalizing Fieweger's and Conklin et al. 's approach to three dimensions, which proved to be quite involved owing to the intricacies associated with the multidimensional frequency space. Despite successfully addressing those, Nengo was not able to properly represent and approximate the translation of pose estimates with the limited number of neurons per horizontal slice of our 3D pose network. Admittedly, the connection function with three rotations in high-dimensional representational space is quite complex. Our considered representational spaces have upwards of 1000 dimensions.

For reference, Fieweger's representational space for the head direction ensemble has 5 dimensions (3 Fourier coefficients, 1 angular velocity), and position representation comprised 12 dimensions (6 Fourier coefficients, 2 velocity components). We eventually came up with a different architecture to realize the shifting of activity bumps. Specifically, we implemented separate pure grid-cell-like cells to realize a stable attractor network, and a form of conjunctive grid, head direction and speed cells to orchestrate translation and rotation of pose estimates.

The previously mentioned aspects of place cells pertain to reset by visual inputs. Fieweger investigates various strategies to achieve this. All considered strategies compare the position inferred from current visual input with the position associated to a similar visual input in the past. They use QR code markers as abstract landmarks to simplify their detection. Hence, visual inputs correspond to marker IDs. They are stored in a dictionary. Visual input is only evaluated every 0.1 seconds. Due to the inert synaptic dynamics, any potential reset of the attractor network needs to be sustained for some time after its onset. The rest strategies differ in how aggressively they pursue a reset, whether the list of reset positions is already pre-initialized, whether they take into account remapping between different environments, and whether they consider the reliability of visual inputs.

For testing and evaluation, the authors intended to use NeRmo, a mouse robot with a biologically-inspired skeleton, but they eventually ran their system on a small wheeled robot. The test environment was an empty area with scattered QR code markers. Fieweger demonstrated the system's ability to capture changes in orientation and position, albeit for a limited range of tangential and angular velocities. Such limits are to be expected when using attractor models, and originate from stable points on the attractor manifold where a certain attracting force needs to be overcome in order to transition the attractor network to a different state. However, it might be that the attractor under consideration behaved more inertly than necessary. This may be explained by the strength of the ring of inhibition in the recurrent weight shape obtained from Nengo for the attractor network. By designing the recurrent connection weights ourselves, we are able to influence the inertia of the attractor network and, hence, make the system usable with a wider range of parameters. While our work pertains to three dimensions, it is highly likely that the two-dimensional setting results in a similar weight function when projected to 1D. The authors also report that equal velocity inputs have different effects depending on the current state of the attractor network, which might be due to the involved LIF neurons or some Nengo-related intricacies.

Fieweger shows the capability of topologically tracing several path shapes: straight line, circles of different radii, and modified figure of eight. The system suffers from noise and drift, and lacks repeatability and precision. Visual reset proves to be rather effective in countering drift. Their implementation is not real-time capable, for which they do not pinpoint a specific reason.

Our system is a significant improvement in terms of stability, while its real-time performance pends evaluation on Loihi.

4.7 Separate Path Integration of Yaw and Pitch with SNNs on Intel Loihi

Kreiser et al.[36] demonstrate a neuromorphic implementation for estimation of the head pose, as given by yaw and pitch angle, by integrating relative movement information. To mitigate the issue of error accumulation, reset via visual input and memorization of landmarks' locations are implemented. When some object is encountered, is visual appearance is associated with the current pose estimate via synaptic plasticity. Upon visiting the object again, the previously learned location is used to reset the two SNNs.

They propose two separate and identical spiking neural networks, one to represent the yaw angle and the other to represent the pitch angle. Similar networks have been developed in an earlier paper by Kreiser et al. (see Section 4.8). Each SNN comprises five different layers: Current Head Direction (CHD; essentially a special attractor network that keeps an activity packet corresponding to the current estimate; one neuron is most active at a time), Left Shift and Right Shift Layers (SL, SR; neurons herein receive inhibitory connections from all but the corresponding neuron in CHD; are activated by velocity input and project to IHD such that each neuron in SL or SR has one weak excitatory connection to the corresponding left-offset or right-offset neuron in IHD, respectively), Reset (RHD; is connected

to a vision module via plastic synapses, that adapt their weights depending on the objects in view and the current yaw and pitch angles; has strong one-to-one excitatory and inhibitory connections to IHD), Integration Layer (IHD; projects strongly in a one-to-one excitatory and inhibitory manner to CHD), and Goal Layer (GHD; used for goal-directed movement). In summary, if no object is detected, the shifting behavior via the shift layers is dominant and rotates the estimate in CHD clockwise (by activating SR) or counterclockwise (by activating SL); and if an object is detected, the associated estimate is enforced upon the CHD layer via the reset layer. Our approach differs in the input layer and corresponding incoming connections. We compute the neuron inputs corresponding to the desired input bumps directly and feed these values to the attractor ensemble. Similar to Kreiser et al. , we realize the shifting of the current attractor bumps via distinct shifting layers that feature the same neuron arrangement as the attractor network. However, we work with a conjunctive representation of multiple variables (x, y, theta), whereas they have separate systems for yaw and pitch, 1D each. Another difference pertaining to velocity input, is that our shift ensembles are activated by the attractor network, but inhibited by the absence of velocity input. Furthermore, they have an additional layer for combining shift and reset, which then projects to the attractor network. In contrast, we do not and cannot have that because of the large number of required neurons, but instead feed the shift and reset inputs directly to the attractor network. Also, we do not include an equivalent to their goal layer.

They test their system on an actual robot with an Intel Loihi chip. The entire system runs on the neuromorphic portion of the chip, including the learning part. The results show precise trajectory estimation by integrating motor commands, and working correction of drift via visual reset. It is real-time capable. An obvious limitation of Kreiser et al. 's approach is that shifts of the yaw or pitch angle can only happen in integer increments of neurons. Our continuous attractor network allows for a smoother representation of the desired values.

4.8 Separate Path Integration of Head Direction and Position on Neuromorphic Chips

Kreiser et al.[37] develop a neuromorphic system for path integration and mapping on a mobile robot. They are inspired by RatSLAM (see Section 4.16). Three modules are involved: one responsible for head direction integration, one for position integration, and another for learning the location of obstacles. A big distinction between their architecture and ours is that we represent orientation and position within a single neural ensemble, whereas they use two separate ensembles. This allows us to sustain multiple simultaneous estimates.

The head direction module, initially proposed in another paper by the same authors (see Section 4.9), performs 1D path integration of the heading angle and has four components. The HD layer's neurons maintain the current orientation estimate in the form of an activity bump, conceptually similar to an attractor network. There are also left and right shifting layers, an integration layer, as well as a reset population, which all work similarly to the ones described in Section 4.7. Different here is that there are multiple distinct shifting layers for various turning speeds, allowing for a more fine-grained integration of movement. The neuron index-offset in the connection between some shift population and the integration population is proportional to the speed associated with the shift population. Depending on the current angular velocity, the appropriate shifting layer is activated and shifts the orientation estimate via the integration layer. Our approach to changing the orientation of a pose estimate also makes use of distinct shifting populations, but there are only two of them - one corresponding to clockwise rotation and the other one for counterclockwise rotation. These populations feature the same neural architecture as our main attractor ensemble and essentially shift the entire set of its neural activities up or down, respectively. We do not require an additional integration layer. Another difference to Kreiser et al. 's system is that we control these shift ensembles via inhibitory modulation and, thus, are able to track smoothly varying angular velocities, while additionally keeping the required number of neurons reasonably small.

The position module performs 2D path integration of the robot's position. It comprises a two-dimensional network of neurons arranged in a rect-

angular grid (in comparison, we implement a hexagonal arrangement of neurons), where the neuron position corresponds topologically to the location in planar space it represents; as well as multiple shifting layers with the same neuron arrangement, one layer for each separately considered movement direction. The shifting layers receive input from the head direction module and translate the main network's activity packet accordingly (via an integration layer, as in the head direction module). It is not stated how different tangential velocities would be incorporated, but presumable this would happen via distinct sets of shifting layers, akin to the head direction module. Reset behavior is implemented as well. Our positional shift mechanism is similar in that it involves several distinct neural populations for shifting in different directions. It is more advanced, however, since these populations are part of a three-dimensional ensemble, where each 2D horizontal slice shifts activity in the corresponding slice of the main attractor ensemble into the preferred direction associated with that slice. Typically, more than one layer is involved in shifting one activity packet in the main attractor network. This enables choosing the movement direction from a more continuous spectrum. In contrast, Kreiser et al. 's system appears to be restricted to a fixed number of movement directions. Furthermore, our inhibitory modulation of the shifting ensemble's activities allows the system to smoothly track varying tangential speeds, akin to our head direction components.

A big difference between their head direction and position networks and our main ensemble is that they work with discrete state spaces, meaning that there is one neuron per network that has by far the highest activity and effectively selects the current angle or position estimate from a discrete set, whereas we implement continuous attractor dynamics to approximate a continuous 3D state space. Our readout algorithm is also more involved since it efficiently combines the activities of neurons contributing to a bump into a single pose estimate (per bump).

Thirdly, the collision detection module is responsible for associating obstacles, e.g. walls, with their position in space. This module consists of a winner-take-all neural population representing the position of a collision relative to the robot, as well as a neuron that is activated whenever some collision is detected. This neuron has incoming plastic connections from each position neuron. Whenever a collision occurs, the connection weight from the currently active position neurons to the collision neuron is increased using a spike-timing dependent plasticity (STDP) rule, effectively forming a collision map. Importantly, the STDP mechanism not only strengthens connections of co-occurring pre- and postsynaptic spikes, but also weakens connections where the presynaptic neuron regularly fires when the postsynaptic neuron does not. This counteracts incorrectly strengthened connections and allows to adapt the collision map upon entering a different environment. One consequence of this behavior, however, is that the position network cannot be circularly connected to deal with large environments, because after a wraparound any previously learned obstacle locations are overwritten.

The authors deploy and distribute their system across two different neuromorphic chips, DYNAP (DYnamic Neuromorphic Asynchronous Processor) for the head direction and position modules, and ROLLS (Reconfigurable OnLine Learning System) for the collision map. Velocity input is obtained from motor encoders, and collision input is provided by bumper sensors. Due to the limited number of available hardware neurons, the position shifting layers are shrunk so that one neuron therein connects to multiple neurons in the integration layer. Ultimately, they use six layers of 72 neurons for the head direction module, resulting in a discrete angular resolution of 5 degrees. The simulated position module consists of 32x32 neurons plus 8x32x32 neurons for the shifting populations, while the hardware tests are carried out with only 16x16 positional neurons. Regarding path integration precision, the authors demonstrate that the mean square error between target and actual neuron index at certain points of a trajectory is significantly smaller when using certain reset strategies, e.g. resetting the head direction module every 25 seconds and the position module every 37 seconds. The error increases with time, but seems to remain within some interval with proper reset of both modules. They report that errors in the heading angle estimate have a strong negative effect on the accuracy of the position network, as expected. The discrete nature of their networks also contributes to the accumulated path integration error, an issue our system does not suffer from. Kreiser et al. show the successful creation of collision maps, at least qualitatively, and further demonstrate unlearning.

In order to improve the resolution of the path integration and mapping system, more advanced neuromorphic chips with a larger number of neurons are required. The computation time would not be greatly affected by this upscaling due to parallelism and event-based communication on such architectures.

4.9 Neuromorphic Head Direction Estimation

Kreiser et al.[35] implement path integration of the heading angle of a mobile robot with a spiking neural network. The system is biologically inspired and deployed on ROLLS, an analog/digital neuromorphic chip, which makes it very power efficient.

They use several neural populations. To represent the current orientation estimate, one population of neurons (HD) sustains a size-constrained bump of activity. These neurons are recurrently connected in a ring and topologically tuned to a specific angle according to their position in the ring. Neurons excite their neighbors, i.e. neurons with similar orientation preference, and inhibit all other neurons that are further away, i.e. neurons with less or no similarity in orientation preference. This connectivity pattern is characteristic for an attractor network.

The activity packets are shifted clockwise and counterclockwise by velocity input from motor encoders, i.e. self-motion information. To this end, there are two more populations of neurons arranged in rings, one for each shifting direction (SL and SR). Each HD neuron has inhibitory connections to all but the neuron at the same index and the neurons to the immediate left and right thereof in the SL and SR populations. Hence, when SL or SR is excited by the positive or negative angular velocity populations (DL and DR), respectively, a localized bump of activity emerges, which is aligned with the activity in HD. DL or DR are active when the motor encoders signal rotational movement. Kreiser et al. note, that such combined inhibitory and excitatory connectivity of HD, SL/SR and DL/DR is more robust than a pure excitatory connectivity. The shifting populations project to yet another ring-like neural population (IHD) with offset excitatory connections, i.e. each neuron in SL or SR excites the neuron one position to it's left or right in IHD, respectively. The IHD neurons connect to the HD population such that they are able to enforce their state to the counterparts in HD, i.e. shifting the activity packets in HD left or right depending on velocity input. To prevent unwanted stochastic spiking, DL inhibits SR, and DR inhibits SL.

Visual input from an event-based camera can reset the orientation estimate to the correct value, thereby counteracting error accumulation due to drift and neuronal noise. This visual reset is incorporated via the IHD neural population. Furthermore, input from an IMU sensor can be used for reset in a similar way.

The evaluation consists of simple experiments where the robot turns left or right with constant angular velocity. The authors demonstrate that visual reset effectively corrects accumulated errors and, thus, increases precision of the orientation estimate. Nonetheless, they also argue that the system's accumulated error does not diverge but fluctuates around zero, with and without visual reset.

One limitation of the proposed architecture is that the left or right shift happens in discrete steps of one neuron per shifting operation. In contrast, our approach of variably inhibiting shifting ensembles that project to a main attractor ensemble with Mexican hat-shape weights allows for smooth integration of angular velocity. Furthermore, Kreiser et al. only consider the head direction component of a mobile robot's pose, while we additionally work with the 2D position. For a comparison of our head direction part with their system, see Section 4.8, where the same authors propose a more advanced approach but conceptually reuse the head direction integration discussed here.

4.10 Analog Implementation of Alternative Model for Grid Cells

Aggarwal[1] propose an analog hardware implementation of a spatial navigation system comprising grid cells and place cells. Grid cells are realized according to the GRIDSmap model, an alternative to CAN-based models for explaining hexagonal grid-cell firing fields. The idea here is to represent the entire space using three rings of so-called stripe cells, employing simple trigonometry to combine their activities. The authors compare two methods of obtaining place cell firing: computing weighted sums of multiscale and multiphase grid-cell activities fol-

lowed by thresholding; and using Bayesian integration of those activities. The latter option more closely resembles real place-cell firing fields and, therefore, is selected by Aggarwal.

Hardware-wise, three analog chips are designed and manufactured: one for the stripe rings, one for grid cells, and another one for place cells. Issues of the analog approach are limited scalability and difficult testing.[38]

Their system is vastly different from ours in that we engineer a CAN-based model, and further use digital hardware.

4.11 Purely Inhibitory Recurrent Connectivity for Grid Formation

Couey et al.[15] build upon the work of Burak et al. (see Sec. 4.12) and focus on how grid firing patterns can emerge from exclusively inhibitory recurrent connectivity between the neurons. Such configuration would fit the observed network architecture of cells in MEC layer 2. This is opposed to the classical Mexican hat-type connectivity often used in attractor network models of grid cells.

Together with constant excitatory input from outside, which might e.g. be coming from the hippocampus,[8] the purely inhibitory network connectivity yields multiple firing fields arranged on a hexagonal lattice. Specifically, the authors consider a simple bimodal connectivity: neurons in an immediate circular neighborhood are uniformly inhibited and all other neurons are not influenced. They call this an all-or-none connectivity. The spacing and field size of the resulting grid depend on the inhibition strength and the connection radius. Consider the hexagonal grid of neural firing activity. Neurons within an activity packet are inhibited by all other neurons within the same packet and potentially also by some neurons of neighboring packets. Apparently, the outside excitation is stronger than the total inhibition received by that neuron because it is firing. Compare that to neurons that are not firing, i.e. those that are located between packets. They receive inhibition from all neurons of the six surrounding activity packets, which outweighs the excitatory input. That reasoning directly explains why a hexagonal grid instead of e.g. a square grid is formed: the inhibitory effect between neurons, i.e. activity packets, is purely local and so the dynamical system approaches the state where the distance between neighboring activity packets is maximal. We also design our recurrent connection weights to have purely local effect. However, we use a Mexican hat shape, i.e. both inhibitory and excitatory interconnections. Qualitatively, this difference does not affect the grid cell activity. The reason for our choice is that we only want a single bump of activity per pose estimate in the population response at any time. Note that this does not contradict the fact that one of our core contributions is the representation of multiple simultaneous pose estimates, since there we introduce one activity bump per pose estimate.

Once the population activity is established, Couey et al. use directional velocity input to translate the activity pattern about the neural sheet. The incorporation of such input is done according to Burak and Fiete[10].

4.12 Intrinsic Velocity Tuning in Attractor Models of Grid Cells

Burak and Fiete[10] propose an implementation of a continuous attractor model for generating grid cell activity, which can perform accurate path integration. They note that previous models for dead reckoning with grid cells suffer rapid error accumulation and, therefore, would need periodic resets from outside stimuli. Their model takes velocity and direction inputs to generate and update the characteristic hexagonal firing grid. We use a similar input configuration of tangential and angular velocity.

Their neurons are topographically arranged on a square grid and recurrently connected via local center-surround weights. The resulting population activity resembles a hexagonal grid, where the spacing directly depends on the reach of the inhibitory ring and the size of each grid point depends on the size and strength of the excitatory center. The final weight configuration is such that the excitatory center is of zero magnitude and the neurons receive non-zero inhibitory input from their local neighbors. Constant excitation is provided from outside. Such configuration is discussed in detail by Couey et al.[15] (see Sec. 4.11). The inclusion of non-zero recurrent excitation would not qualitatively affect the result. Our final recurrent weights are also local and follow a center-surround pattern, but have a non-zero excitatory part. We do not em-

ploy constant excitatory input and instead only rely on self-sustained activity via the recurrent connections.

The single-neuron responses depend on the mapping of velocity to the neural activity. Burak et al. realize that via patches of neurons with different orientation tuning. There are four orthogonal preferred directions: up, down, left, and right. Each neuron is assigned one of these four and has it's outgoing recurrent weights shifted in that direction by some fixed amount. Additionally, each neuron selectively receives velocity input according to it's preferred direction, meaning that if the animal moves to the left neurons with that tuning are stimulated proportionally to the velocity. The neural sheet is uniformly tiled by 2×2 blocks of neurons representing each of the four directions. The authors note that the firing pattern is only stable for small directional offsets. They choose the offset as two grid positions. Another parameter to tune is the gain applied to velocity input. The velocity of activity blobs in the network in response to some velocity input is then determined by both the directional offset of the weights and the input gain. Intuitively, translation of the network activity works as follows. Each neuron has a tendency to drive the local activity into it's preferred direction. Due to the way neurons of different directional preference tile the neural sheet, there is another neuron nearby which tends to drive the local activity into the opposite direction. As long as all neurons receive the same (constant excitatory) input, as is the case in the absence of velocity input, all directional tendencies cancel each other out and the network activity remains stationary. When there is velocity input for a specific direction, e.g. to the left, the neurons with that directional preference receive stronger input than the others and, hence, their tendency to shift the network activity to the left prevails over the opposing rightward force. Upward and downward tendencies still cancel each other out, resulting in an overall rightward motion of the network activity. The population response's grid spacing together with the velocity mapping determines the spacing of the single-neuron response's grid. Burak et al. 's translation mechanism formed the basis of many interesting conceptual ideas for incorporating velocity input during our development phase. However, for the final system we opted for a pure attractor network to maintain the pose estimates and external shifting networks to move them around.

Another important aspect are boundary conditions of the neural sheet. They evaluate periodic and aperiodic boundary conditions and conclude that both can generate hexagonal firing grids and perform accurate path integration; some adaptations are required for the aperiodic case. The authors argue that for the periodic case, continuous rotations are not included in the attractor manifold. This means, for example, that the system is robust against noise that would otherwise introduce rotational errors. While Burak et al. simply connect the opposite sides of the square to realize the periodicity, we introduce an offset in the wraparound of one of the square's two dimensions. The reason is that we not only want a hexagonal single-neuron response but we also want the three-fold symmetry of such grid to apply within our neural sheet. Further, note that their population response features multiple bumps. In contrast, we introduce constant global inhibition and appropriately tune the recurrent weights in order to achieve a singular bump. This is because our ultimate goal is the representation of multiple simultaneous 'independent' pose estimates by our population. Also, our grid cell spatial scale matches the domain length, meaning that a single-neuron response's hexagonal grid has the domain length as distance between neighboring points. Overall, Burak et al. focus more on the emergence of the hexagonal grid from local connectivity and translating it with patches of different-directional neurons, while we go more top-down with the toroidal model to specifically account for the hexagonal firing pattern.

4.13 Twisted-Torus Model of Grid Cells

Guanella, Kiper, and Verschure[32] propose an attractor network with twisted wraparound connectivity and velocity-modulated recurrent weights to model grid cells.

They represent the network connectivity as a twisted torus. To this end, they define a rectangle in which neurons are arranged on a hexagonal grid with equal space between any two neighbors. The dimensions of that rectangle is chosen such that the number of neurons per row is larger than the number per column by a factor of about $2/\sqrt{(3)}$. While the neighbor relationships between neurons within the rectangle is clear, the boundary conditions need

to be defined explicitly. Guanella et al. conceptually connect the left and right sides, meaning e.g. that the neuron in the bottom left corner is a direct neighbor of the neuron in the bottom right corner. The authors further conceptually connect the left half of the top side to the right half of the bottom side and vice versa. This results in a twisted torus shape. It ensures that there are three axes of symmetry in the neural sheet connectivity and also yields a hexagonal firing grid when any single-neuron response is arranged to tile 2D space.

As to the recurrent connection weights per neuron, they are defined via a downward-shifted Gaussian that takes the absolute distance between source and target neuron as input. Thereby, the distance is computed on the toroidal manifold. The outgoing excitation strength decreases with increasing distance, while the inhibition strength increases with distance. Thus, nearby neurons excite each other, while distant neurons inhibit each other. This global recurrent inhibition results in the population response containing only one packet of activity. The described weight configuration yields attractor dynamics. To realize translation of the represented position, the recurrent connection weights are dynamically modulated and shifted by velocity input. For example, when there is leftward velocity input the Gaussian used for setting the recurrent weights is no longer centered at zero but is shifted towards the left. The shift is proportional to the magnitude of the velocity input. Since all neurons are affected by this, including the ones currently contributing to the activity packet, the entire neural activity moves towards the left; the speed of that movement naturally depends on the center-shift of the Gaussian. By varying the gain that is applied to the velocity input before it is passed to the network it is possible to change the size and spacing of the single-neuron response's grid. Specifically, the authors find a log-linear relationship. To keep the global neural activity in check, they add a neuron which receives input from all other neurons, sums it up, and provides proportional normalizing feedback to these other neurons.

In summary, Guanella et al. essentially design their model such that the population response, and, for that matter, also any single-neuron response, corresponds to one unit cell of a hexagonal grid. This necessitates the twisted wraparound. As a consequence, their system can represent large spaces with relatively few neurons. There are a number of similarities to our project and, in fact, some of them have their origin in Guanella et al. 's work. Due to the ultimate goal of representing multiple pose estimates simultaneously, we need our population response to feature exactly one activity packet per represented pose estimate. Instead of recurrent global inhibition, however, we use constant global inhibition to achieve that. The reason are hardware constraints on Loihi. The final recurrent weights follow the shape of a localized Mexican hat function, which is somewhat similar to a downward-shifted Gaussian. When the population response effectively corresponds to a unit cell of a hexagonal grid, this implies a direct relationship to any single-neuron response and further requires a twisted-torus topology. Consequently, our single-neuron responses tessellate space such that a hexagonal grid emerges. Regarding translation, our approach differs significantly from Guanella et al. 's. We implement external shifting ensembles with fixed connectivity, while they dynamically modify the recurrent weights of the sole neural sheet that is also responsible for providing attractor dynamics. Lastly, we do not require explicit normalization of the total network activity. This is already taken care of by the constant global inhibition.

Guanella et al. also implement a set of place cells that are wired via Hebbian-learnable synapses to the grid cell population. Since these place cells are predefined and can be assumed to be working properly, they essentially provide permanent reset input to the grid cells, thereby correcting path integration errors. In fact, the authors report the successful correction of large deviations due to high noise values. We leave such visual reset functionality for future work.

4.14 Path Integration with Distinct Shifting Layers

McNaughton et al.[42] present an early review of path integration based on a continuous attractor network and using intermediate neural layers with different directional tuning to translate the position estimate according to instantaneous velocity input.

There are separate systems for head direction and position. The former uses a 1D ring attractor and the latter a 2D plane attractor. We only consider the position system since the orientation system is just a simpler version thereof. Neurons

are arranged in a square grid with periodic boundary conditions, resulting in a torus. To get hexagonal single-neuron responses with a toroidal model it would need to be twisted. Excitatory recurrent connections are of 2D Gaussian shape, i.e. close neighbors are excited more and distant ones are not excited. There are two options for inhibition. Global feedback inhibition keeps activity from spreading and ensures that there is just a single activity bump, since this one bump inhibits the rest of the neural sheet so that no other bump can form. Then there is local recurrent inhibition, which also keeps activity from spreading but allows for multiple bumps to coexist. Regarding translation, there is a set of 2D 'hidden' neural layers, each responsible for a different direction of movement. Their neurons are tuned to position and orientation. The layers are selectively modulated by velocity input, proportional to alignment of the movement direction to a layer's preferred direction. They receive direct input from the position attractor network, along with velocity input, and project back with an offset according to the specific hidden layer's directional tuning. Hence, these layers effectively combine position with head direction and velocity. Thereby, a stronger activation by velocity input yields faster movement of the activity bump.

Our approach also uses continuous attractor dynamics to maintain activity packets that represent position and direction estimates, however, we implement their representation conjunctively in a 3D attractor network. The reason is that we ultimately want to be able to sustain more than one position-orientation tuple at a time. For most efficient coverage of space we opt for a hexagonal arrangement of neurons within horizontal layers. We employ Mexican hat type local connectivity to allow multiple bumps to coexist, and constant global inhibition to prevent random activity packets from forming. Similar to McNaughton et al. , we use distinct neural populations to incorporate movement. Given our conjunctive x-y-θ space, however, there are certain differences. They have one layer per direction and provide the velocity input to each layer according to its directional preference. We split velocity input into translational and rotational components. There is one shifting ensemble that conceptually shifts the activity in the main attractor network according to the directional tuning per x-y layer. It is disinhibited by non-zero translational velocity. Two rotation ensembles shift the main attractor's activity up for counterclockwise rotation or down for clockwise rotation. These are disinhibited by rotational velocity. While Milford et al. activate their shifting layers by velocity input, our shifting layer are active by default and need to be inhibited inversely to the corresponding velocity input. As another similarity, our shifting and rotation ensembles are also initialized directly from the main attractor.

Milford et al. go on to present an explanation of how their discussed model with (conjunctive) grid cell-like neurons, toroidal attractor connectivity, and multiple neural layers might form in the MEC through synaptic plasticity.

4.15 Attractor-Based Path Integration with Frequency Space Representation

Conklin and Eliasmith[13] propose a biologically realistic implementation of the attractor network model for path integration based on the Neural Engineering Framework (NEF). At the core, they incorporate the representation of the current position estimate as well as the translation of that estimate based on external velocity input into the same layer of neurons. In other words, they use the NEF's capabilities to piggyback the translation (via rotation modulated by velocity input) onto the regular attractor dynamics, i.e. with one total set of recurrent connection weights. This is an advantage because it requires fewer neurons. Specifically, they do not require any additional networks or layers of neurons to control the shifting of the current position estimate. In contrast to Conklin et al. , we opt for this second option of having logically-separate neural populations; one regular attractor network to represent the pose estimates, and three so-called shifting networks to incorporate the change of position and orientation per time step.

A position is represented as a two-dimensional Gaussian 'bump' on a torus, i.e. they use a square with cyclical axes as their domain. Hence, when the position estimate reaches the border of the square it simply wraps around to the opposite side. For mathematical convenience and because it works nicely with NEF, the representation, originally parameterized by mean and variance, is transformed into frequency space. The final representation of a position is of the form of a flattened vector of 2D Fourier coefficients. To compute it, the domain is sampled in

an evenly-spaced square grid. The Fourier transform assuming periodic input and outputting coefficients for a set of periodic complex exponential functions fits well with the periodic toroidal domain. Thereby, a smaller variance of the original Gaussian requires a higher number of Fourier components, or basis functions as Conklin et al. refer to them, for an accurate approximation. Similarly, we also represent pose estimates as Gaussians and work with their Fourier coefficients, but we extend it to three dimensions; the third one being the heading orientation. So, while Conklin et al. only consider the heading direction via the velocity input, we represent it conjunctively with the planar position. Another important difference is that we work with a hexagonal domain and implement the periodicity such that a twisted torus emerges. This is necessary to achieve truly hexagonal firing patterns akin to biological grid cells.

To convert the theoretical description into a neural substrate without having to perform meticulous tuning of low-level neural parameters, the NEF is employed. Neurons are conceptually arranged in a square grid so as to evenly cover the square domain, i.e. the unrolled torus. Each neuron's preferred represented position then corresponds to the neuron's conceptual location on the grid, and is defined as the vector of Fourier coefficients of a Gaussian centered at that location with the same variance as the Gaussians used for representing positions. As per NEF, a neuron's response is conceptually computed based on the dot product of the input, i.e. a Gaussian in frequency space, and the neuron's encoder, i.e. a (different) Gaussian in frequency space. Since both these vectors correspond to Gaussians, the dot product when plotted for all possible inputs in the square domain looks roughly like a Gaussian centered at the neuron's position. Therefore, we can speak of Gaussian tuning curves per neuron, as is also observed for neurons in rodents' entorhinal cortex. We also define a neuron's tuning via the Fourier coefficients of a Gaussian centered at the respective neuron's position in space/in the grid, but, as mentioned above, extend the entire concept to three dimensions. Working with the three-dimensional Fourier transform and manipulating the resulting coefficients is significantly more involved than in 1D and 2D. Like with the domain differing in shape and connectivity, we arrange the neurons per layer of our 3D network in a hexagonal grid, as opposed to a square grid, and realize the wraparound connectivity such that a twisted torus emerges, as opposed to a regular torus. Our (sliced) network topology yields three axes of periodicity as opposed to just two for Conklin et al. 's.

The authors instruct NEF to randomize each neuron's gain and bias parameters so as to achieve a heterogeneous neural population. NEF then computes the decoders per connection based on the encoders, gains, biases, and the desired function. In contrast, we choose the same parameters for each neuron to reduce the number of to-be-tuned variables. Of course, there are still the distinct encoders that cause the neurons to behave differently from each other.

By simply specifying a recurrent connection with the identity function, attractor dynamics can be obtained. A position is represented as packet of neural activity shaped like a Gaussian. In order to translate this activity packet about the neural sheet according to the actual movement in space, the authors consider the effect of real-space translation of a 2D Gaussian on the associated vector of Fourier coefficients. Starting from the Fourier transform of a two-dimensional Gaussian and substituting parameters to reflect translation they derive an equation that relates pre-translation Fourier coefficients to post-translation Fourier coefficients. The latter are obtained by left-multiplying the former with a rotation matrix. This is initially defined per coefficient represented as a length-2 vector containing the real and imaginary parts, but directly extends to multiple coefficients by constructing a large matrix with the rotation matrices arranged on the main diagonal. Thus, the translation in space amounts to a rotation in frequency space. They essentially rederive a version of the shift theorem (see Sec. 3.5). Such rotation can be split into two successive one-dimensional rotations, one per dimension of the frequency space; corresponding to translation first along e.g. x and then along y in planar space. The respective amounts of translation/rotation can be interpreted as velocity inputs. The rotation matrices have to be recalculated for each time step given the specific translation distances per dimension. Due to trigonometric functions being involved this is rather difficult for a neural network. Conklin et al. solve this by fixing rotation matrices for the maximum translation speed per dimension and then simply scaling them by the time-varying velocity input. The velocities per dimension are normalized such that

the total velocity vector's length lies in the range $[-1, 1]$. This works out just fine for one dimension, but requires an approximation for two dimensions since there the system dynamics equation contains a product of the two rotation matrices. The product prohibits independent translation per dimension via scaled rotation matrices. The authors propose an approximation of the dynamics equation that has the rotations per dimension separated in a sum (see Eq. 14 in their paper). For zero translation and translation in only one dimension the approximation is exact, while for simultaneous translation in both dimensions the approximation introduces a negligible error compared to the exact equation. Specifically, they argue that their approximation effectively corresponds to a summation of three Gaussians in the plane: one translated in the first dimension, one translated in the second dimension, and one at the original position. The third Gaussian is subtracted. This is in contrast to only one Gaussian in the case of exact translation. However, if the maximum translation speed, as reflected in the rotation matrices, is small compared to the variance of the Gaussians, then the distortion of the sum of Gaussians compared to the target Gaussian is very small. They further argue that no accumulation of this error happens because the attractor dynamics correct it continuously. To obtain the final connection weights, Conklin et al. specify the function to-be-computed by the recurrent connection using the rotation matrices and the velocity components available in the network. Note that the plain attractor dynamics are included via the special case of both velocity components being zero as the function then reduces to an identity function. The NEF then automatically determines the weights, which follow a center surround pattern. We extend the translation by rotation to our 3D frequency space, which turns out to be quite tricky when it comes to interpreting the specific meaning of individual coefficients or trying to optimize the representation by omitting redundant coefficients (see Sec 5.2). Different from Conklin et al. , we cannot directly use the weights obtained by specifying the recurrent connection's function via rotation matrices, but need to further tune these weights to achieve our goals of stable path integration and of representing multiple pose estimates simultaneously. Of course, there is also the significant difference that we use external neural populations to realize translation of the pose estimates. Conklin et al. 's considerations regarding the sum of Gaussians and a resulting slightly-distorted Gaussian can be applied to our system for the case of simultaneous translation and rotation; the translation ensemble tries to establish an activity bubble slightly offset in the forward direction, while one rotation ensemble tries to establish a bubble slightly offset above or below the current bubble, and since this happens at the same time we essentially end up with a slightly distorted bubble that is a combination of the two. The attractor dynamics continuously drive the neural activity towards the currently closest stable state consisting of only Gaussian-like bubbles. This is crucial to keep path integration errors as small as possible. Even after our manual tuning of the connection weights they exhibit a center-surround pattern. Furthermore, we make use of the rotation matrices to manipulate the Fourier representations during various stages of development and deployment. Basically, we compute the Fourier coefficients of a Gaussian centered at the origin and then obtain the representation corresponding to another location, e.g. to construct the encoder matrix, by rotating in frequency space.

To explicitly avoid multiplicative synapses, Conklin et al. include the two velocities as additional representation dimensions in the attractor ensemble. Therefore, the encoders per neuron are extended by two elements that are randomly chosen as -1 or 1 each. While we started out by including the velocity as additional dimensions, we have since switched to a modular architecture with external control populations.

For evaluation, a network of about 4000 spiking LIF neurons is used, which are arranged on a 63-by-63 grid. The involved Gaussians are defined with a variance of 1/3 and represented by 25 Fourier coefficients, i.e. a frequency resolution of 5 per dimension. Initialization of the network happens by brief stimulation with white noise, after which a Gaussian activity packet at a random position emerges. The authors demonstrate that their system follows a controlled stable attractor, exhibiting only minor drift when velocity input is 0. They present several experiments of time-varying velocity input, both in terms of direction and magnitude. Overall, the estimated trajectories are somewhat noisy due to the involved spiking neural dynamics. Nonetheless, the system displays the ability to follow changes in velocity with only a small amount of drift. Depending on the duration of the trajectory visual reset may be necessary to periodically cor-

rect accumulated errors. Alternatively, the number of neuron can be increased to reduce drift. To make the stability of the system more explicit, Conklin et al. provide a plot of the representational error surface of in dependence of the mean and variance of the encoded Gaussian, which essentially depicts the tendency to drift towards certain locations in the mean-variance space, i.e. the attractor points. The flatness of that surface indicates similar (stable) behavior of the system independent of the current location in representational space. Also, the behavior of the system with randomly perturbed connection weights is considered; while the drift increases proportionally with the amount of noise, the system is still able to qualitatively follow simple movement patterns. In contrast, we are forced to use fewer neurons per horizontal slice of our network to cover the actual planar space. This is because we are limited by computational resources and by hardware constraints of Loihi, while requiring more neurons in the first place since we work in 3D and utilize dedicated shifting networks. We use a $12 \times 12 \times 12$ grid on which we arrange our neurons, i.e. there are 1728 neurons just for the attractor network. The discrete sampling of our domain is performed on the same grid, resulting in a frequency resolution of 12 per dimension. This is significantly higher than Conklin et al. 's resolution and allows us to have lower variance of the Gaussians, which, in turn, is necessary if we want to pack more simultaneous bumps in the same space. Eventually, our variance is tuned such that the resulting response curves of the neurons have a reasonable amount of overlap for population coding. Since we allow multiple simultaneous activity packets and cannot use global dynamic inhibition, i.e. depending on the neurons' current activity, we specifically design the system such that no bumps emerge in areas of silence or random activity. This is achieved by constant global inhibition. As for their experimental results, it would be interesting to see their system's performance on longer and more complicated trajectories in order to compare it to ours. From the reported behavior it seems that Conklin et al. 's system behaves less regular than ours since we control our neural tuning more tightly and also have a seemingly more precise control over the translation and rotation. Our system appears to be more stable in the absence of velocity input, while it favors movement along or between grid axes. The magnitude of drift during movement is difficult to compare. Finally, evaluating our path integration system with noisy connection weights is an interesting aspect for future work.

4.16 3D Attractor-Like Network for Path Integration (RatSLAM)

Milford, Wyeth, and Prasser[43] developed a computational model for SLAM based on the functional understanding of the navigation system in the mammalian hippocampus. Later works made some slight adaptations but the basic ideas and components remain.[3,44,45,63] RatSLAM comprises three modules.

The pose cell network is a continuous attractor network (CAN) for representing the current pose estimate. The associated domain is three-dimensional with the orthogonal dimensions of x, y, and θ. Opposite sides effectively wrap around to each other, yielding periodic axes. The cells are topographically distributed within that domain and organized into horizontal x-y planes. The recurrent connectivity is locally excitatory and in some versions also locally inhibitory; both realized as 3D Gaussians with different variance. Additionally, cells are globally inhibited inversely proportional to their individual activations. The described elements lead to a stable state of one active cluster of cells; the centroid of which corresponds to the current pose estimate. Another implementation of RatSLAM uses slight constant global inhibition instead of the activity-dependent one to aid the network dynamics.[45] Then there is also global normalization, i.e. the global activity is forcibly normalized to a fixed total value. This allows to treat the population activity as a 'probabilistic volume'. The activity packet can be shifted around by idiothetic velocity input. This mechanism is implemented directly, meaning that the current global activity state is read out, mathematically translated according to the magnitude and direction of the current velocity input, and then assigned back to the cell population. Hence, representation and translation is handled by the same cell population. The pose cell network is rather coarse by implementation, noisy due to quantization, and only somewhat Cartesian. Over time, it accumulates error due to drift.

Therefore, the second module of local view cells is essential for the proper functioning of the first module. These cells are used to anchor poses to visual stimuli from the environment and to facilitate

loop closure. Specifically, there is a subset of local view cells that are active when a certain visual input is present. At the same time, a subset of pose cells is active representing the current pose estimate. Via Hebbian learning the connections from the active local view cells to the active pose cells are strengthened. Thus, over time poses become associated with visual stimuli. When the same visual input is observed again, the associated subset of pose cells receive excitatory input via the now-strengthened connections. This effectively provides a visual reset, meaning that if the associated pose of the visual input is close to the currently represented pose, then the activity packet in the pose cell network moves towards the associated pose; and if the currently represented pose is further away, then there will be two simultaneous activity packets competing with each other (since they exist in an attractor network with normalized global activity). Over time, one of the two packets will prevail due to being strengthened more via further 'matching' visual input. Importantly, the input from the local view cell network only really affects the pose cell network if it is somewhat persistent and coherent in time rather than spurious, because the attractor dynamics effectively filter the latter type.

Third, there is an experience map which links information from the other two modules into a coherent representation of space. For example, it resolves the ambiguity introduced by the periodic nature of the pose cell network's domain. Experiences, which constitute pairs of pose and view, are linked via relative translation and rotation information, i.e. how to get from one pose to the other. Loop closure events trigger a map correction procedure in the experience map. The map can be used to perform path planning.

Overall, RatSLAM can deal with ambiguous visual input by representing and updating multiple simultaneous pose estimates. The representation of the environment is consistent. Milford, Wiles, and Wyeth[44] specifically test RatSLAM on a wheeled robot in the presence of visual ambiguity. An advantage of such computational approach is that short-lived competition between multiple pose hypotheses can be easily analyzed, whereas capturing this sort of neural dynamics in live rodents is difficult. In brief, they use a square arena with a perceptually ambiguous corridor along the perimeter. Roughly midway in each of the 4 segments is a cue and right after it are rewards, which are only visible, however, when the cue has been passed. The cues in the bottom and left segments are identical, and the cues in the top and right segments are identical. Consequentially, moving along the bottom section looks identical to moving along the left section, while moving along the top section looks identical to moving along the right section. To successfully navigate the corridor arena, two simultaneous location hypotheses need to be maintained and propagated until enough evidence accumulates via integration and strengthens one of the hypotheses. Since probabilities sum to one, the other hypotheses are naturally downweighted. Here, such evidence corresponds to the coming-into-view and being-in-view of telling cues. From their experiments, the authors report successful representation of multiple pose hypotheses and the eventual dominance of the correct one. They postulate that conjunctive grid-by-head-direction cells are well suited for encoding space in the presence of uncertainty, which is precisely one of our objectives. Ball et al.[3] provide an open-source implementation of RatSLAM based on the Robot Operating System (ROS) framework. It consists of ROS nodes corresponding to the RatSLAM components pose cell network, experience map, and local view cell network. The nodes communicate via ROS messages. Yu et al.[69] implement a SLAM system for three-dimensional navigation. They separate 3D position a 1D planar head direction. Specifically, an attractor network of 3D grid cells is used to achieve a metric representation of 3D position. The neuron arrangement, recurrent connectivity, and iterative application of the 'dynamics' are very much like RatSLAM's pose cell network. For representing the orientation per z-layer, they use a multilayered head direction cell network, which resembles a stack of ring attractors. Similar to RatSLAM, the networks for position and head direction are updated by odometry input (i.e. path integration) as well as by local view cells, the connections to which are learned over time. Since the two networks interact, the authors refer to them collectively as conjunctive pose cell model, which might be slightly misleading. Also from RatSLAM comes the concept of an experience map, extended to a multilayer version. It combines the information from grid cells, head direction cells, and the vision module to build up a topological and locally metric of its 3D surroundings. Importantly, it appears that by separately storing position and orientation they lose the capability of representing multiple pose hy-

potheses. However, that might be compensated by the convergence of information in the experience map. The authors demonstrate successful operation in real-world three-dimensional settings that required the correction of large path integration errors.

While RatSLAM is inspired by the rodents' navigation system, this shows only on a high conceptual level. The pose cells are inspired by place cells and the wraparound is introduced to reuse computing resources, which results in characteristics similar grid cells (which were discovered after the publication of RatSLAM). The implementation purely functional and not biologically realistic. It uses rate-based cells and all described dynamics are implemented as an iterative model where the population activity is directly modified. For example, the described recurrent connectivity is 'applied' one component after the other (first local excitation, then local inhibition, and so on); and the shifting of activity is not neuromorphic but algorithmic. In contrast, one of our goals is to adhere functionally and implementation-wise as close as possible to the spatial navigation system of mammals. We specifically use spiking neuron models and connect them via realistic synapses. We also design the system such that it can later be run on neuromorphic hardware. Their pose cell module is similar in function to our main attractor network in that both are 3D continuous attractor networks with an associated periodic domain of conjunctive x-y position and orientation angle. RatSLAM is one of the few approaches to employ such conjunctive representation, which is crucial to both their and our approach's capability of maintaining and updating multiple simultaneous pose estimates. While RatSLAM implements pose cells as a wrapped-around 3D array of values in C++, we specifically model our main attractor neurons as conjunctive grid-by-orientation neurons. Per horizontal slice, our neurons are arranged on a hexagonal grid and periodically connected such that a twisted torus forms. Similar to RatSLAM, the connectivity in our main attractor network comprises local center-surround excitation-inhibition, but it is actually simulated like synaptic connectivity. We also implement a slight constant global inhibition to keep the population activity in check and prevent random bumps from spawning. In contrast to RatSLAM, we do not use dynamic global inhibition nor global normalization due to implementation and hardware constraints. Furthermore, regarding velocity input, while Milford et al. algorithmically implement the translation and rotation of the pose cell activity in C++, we incorporate such behavior in our neural substrate via additional populations of neurons that influence the activity in the main attractor network.

5 Methodology

Here we describe the workings of our final neuromorphic multi-pose representation for path integration under uncertainty. Our neural network must be capable to maintain stable pose estimates, update them according to outside velocity input, and achieve all this with as low as possible drift in order to successfully perform path integration.

An overview of the final system architecture is given in Fig. 5.1.

We note that many parameter values or design decisions are interdependent. Therefore, it is difficult to give exact and singular reasons for specific choices. Nonetheless, we try to give such reasons where possible and refer to interdependent parameters where appropriate.

5.1 Domain

To facilitate path integration on a planar surface in the presence of uncertainty, we want to be able to represent and update multiple simultaneous pose estimates. A pose comprises position (x, y) and head orientation θ, relative to some reference point or reference orientation, respectively. To avoid combinatorial ambiguity that might arise when dealing with multiple distinct poses at the same time, we work in the conjunctive three-dimensional space $\mathbb{R}^2 \times [0, 2\pi)$, henceforth referred to as *domain*, and define a pose as a 3-tuple (x, y, θ).

Position. First, we consider only position. Real 2D space is infinite but the number of neurons we can use to represent it is bounded. Therefore, given some desired positional accuracy, we can only represent a constrained 2D area, which we call unit area. By introducing periodic boundary conditions we can extend the representable area infinitely, thereby achieving the same local accuracy at the cost of global ambiguity. We choose the shape of the unit area such that 2D space $\mathbb{R}^2$ can be tessellated with it, i.e. covered without gaps and without overlap. Intuitively, we can then uniquely identify a position relative to the unit area, but cannot say in which copy we are. We go on to discretize the unit area, associate some neuron to each discrete point, and use population coding to smoothly and continuously represent the area (see Sec. 5.3 for more details). Motivated by the hexagonal arrangement of firing locations of biological grid cells (see Sec. 3.3) and the fact that a hexagonal grid features the lowest variance in distance between arbitrary points and the nearest grid point, we want to achieve single-neuron responses that form a hexagonal lattice when arranged to tile space. Such hexagonal grid has three axes of symmetry or periodicity, spaced $60°$ apart, along each of which the period is the same. We specifically desire this periodicity property not only for the single-neuron response relative to physical space, but also relative to the unit area. This implies that the unit area together with its periodic boundary conditions needs to form a twisted torus. We opt for a rhombus shape, i.e. an equilateral parallelogram, with opposite edges 'glued' together (see Fig. 5.2). The equal side lengths and their $60°$-spacing yield a toroidal manifold with the three equal-period symmetry axes shown in Fig. 5.3. Note that a rectangle of appropriate width-to-height ratio or a hexagon shape would also work for the unit area, but these require a more complicated wraparound connectivity to get a twisted torus, and further make the implementation of e.g. the pairwise distance matrix more complicated. Since we actually started out with a rectangular shape, some functions operate on that assumption. We provide conversion routines to switch between the rhomboidal and rectangular representations.

Orientation. Now, we additionally consider the θ-dimension of our conjunctive space. To this end, we add a third orthogonal dimension to the unit area. Making this dimension orthogonal comes from the considerations for pure rotations in Sec. 5.3. Since the angular quantity is intrinsically periodic, introducing periodic boundary conditions for this dimension is only natural.

Unit Domain. We denote the resulting three-dimensional volume as the *unit domain* Dom within

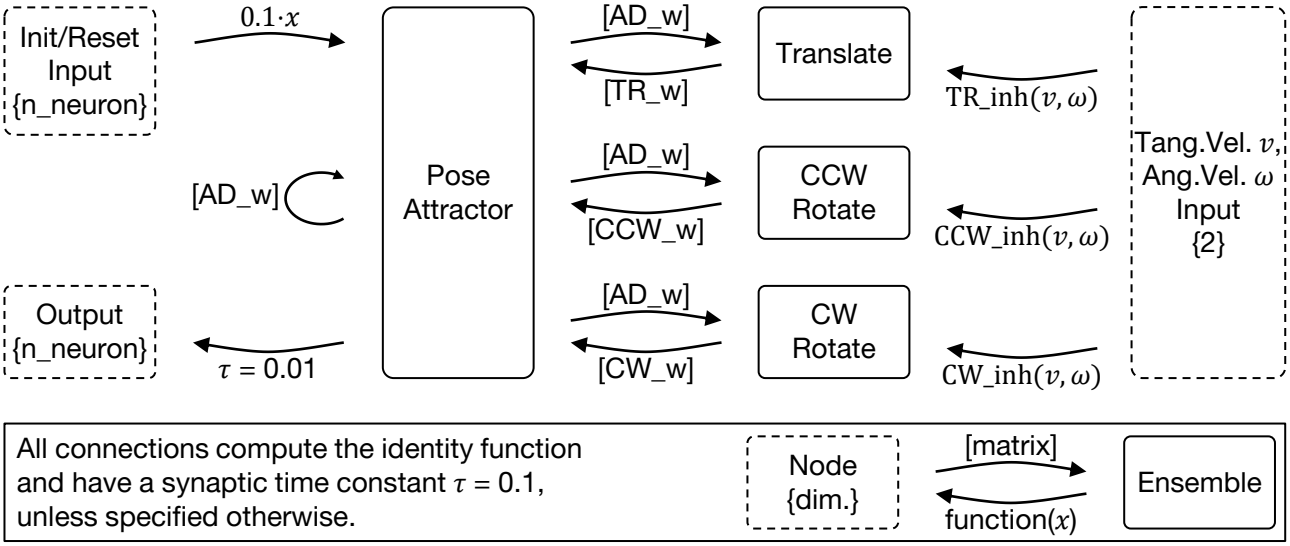

Figure 5.1 System architecture. We represent the current pose estimates in a neural network with continuous attractor dynamics (Pose Attractor). Translation and rotation of the pose estimates are realized by separate ensembles of neurons (Translate, CCW Rotate, CW Rotate), which are modulated by velocity input. The collection of all involved ensembles and connections is called the pose network.

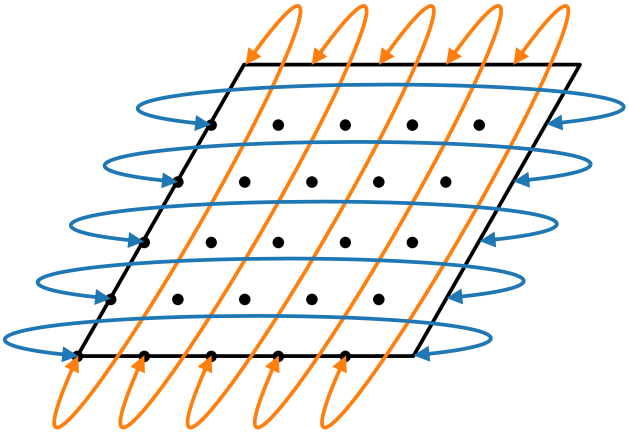

Figure 5.2 Rhomboidal unit area with periodic boundary conditions. It corresponds to the first two dimensions, x and y position, of our domain Dom. The shown wraparound effectively glues the opposite edges together, which results in a twisted torus. The arrows' endpoints indicate identical locations on the toroidal manifold. Grid points are included to indicate the discrete sampling and neuron grid that we work with (see Secs. 5.2 and 5.3). For example, the rightmost point in the bottom row is a direct neighbor of the point in the bottom left corner.

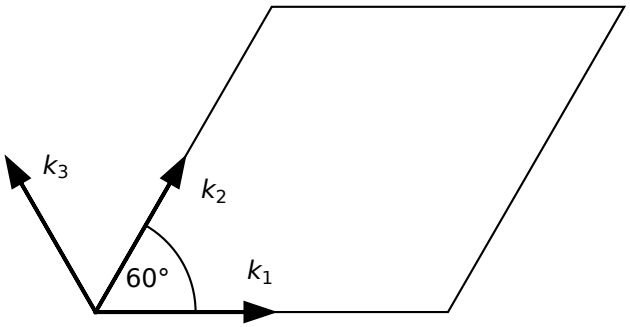

Figure 5.3 The unit area and its boundary conditions yield three symmetry axes of equal period.

our *domain* (see Fig. 5.4). The limits are set as $[0, 1) \times [0, 1) \times [0, 2\pi)$. Formally, the unit domain is defined as a right rhombic prism with an angle of $\pi/3$ between the two primitive vectors $\mathbf{a}_0$ and $\mathbf{a}_1$ of length 1 spanning the base. The third primitive vector $\mathbf{a}_2$ has length 2π. Note that for the implementation we use length 1 also for $\mathbf{a}_2$, but nonetheless interpret it as 2π.

Similar to how the unit area tessellates the plane spanned by the first two dimensions of our *domain*, the unit domain with its periodic boundary conditions can be interpreted to tessellate the actual three-dimensional *domain* $\mathbb{R}^2 \times [0, 2\pi)$. Dom is essentially unrolled according to the boundary conditions, conceptually replicating it along integer multiples of the two primitive vectors spanning the base. Figure 5.11 provides a helpful illustration of the tessellation, clearly showing the rhomboidal unit tile. By rearranging parts of this tile along the wraparound we obtain an equivalent hexagonal tile, thereby making the hexagonal nature of the unit domain and of the tessellation more clear. Figures 5.5 and 5.6 illustrate this relationship. In conclusion, the first two dimensions of the unit domain constitute a

periodic representation of position in planar space. Since the third dimension directly represents an angle, there is no conceptual replication happening along it.

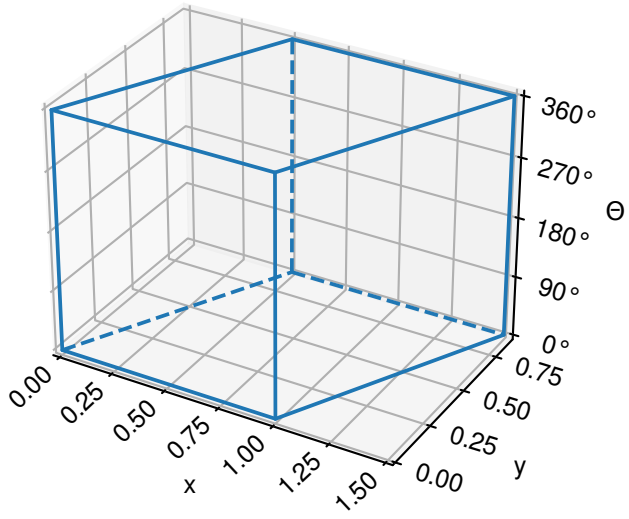

Figure 5.4 Our unit domain is a right rhombic prism with sidelengths 1, 1, and 2π. All opposite faces are conceptually 'glued' together to form a periodic volume. The shape of and connectivity within one horizontal slice are illustrated in Fig. 5.2. The unit domain conceptually tessellates our three-dimensional *domain* in a two-dimensional way. This facilitates a periodic representation of 2D position, along with the intrinsically periodic orientation angle.

From now on, we work primarily in the unit domain Dom and specify positional quantities in the unit u, i.e. relative to and in fractions of the unit domain. $1\,\mathrm{u}$ equals one sidelength of the unit domain's base.

Physical Mapping. At some point we need to map the first two dimensions of our *domain* to physical planar space, i.e. a real physical unit needs to be associated with u. For example, $1\,\mathrm{u}$ might be set equal to $0.5\,\mathrm{m}$. This would imply that our path integration uniquely represents a rhombic patch of sidelength $0.5\,\mathrm{m}$ in physical planar space. Formally, we map the base of Dom to some equilateral parallelogram in physical planar space. This space is then conceptually tessellated like the first two dimensions of our *domain* are by the base of Dom, just with a differently sized tile.

The mapping determines the scale of the population response's firing grid relative to physical space (see Sec. 5.3 for more details).

While we talk of the physical mapping as if it was simply a matter of interpretation, it is actually implicitly defined by how velocity input is incorporated into the system (see Sec. 5.7). For example, if a tangential velocity of $0.5\,\mathrm{m/s}$ is input to our system for $1\,\mathrm{s}$, and any pose estimates are translated by one unit domain length u as a result, then we effectively get the mapping of $1\,\mathrm{u}$ to $0.5\,\mathrm{m}$.

While we do not propose a default physical mapping for our system just yet, an implicit mapping occurs when defining time scaling factors for evaluating trajectory tracking performance in Sec. 6.3.

5.2 Representational Space

In this section we derive the final representation of poses, optimized for our needs.

5.2.1 Pose as Gaussian

We represent a pose estimate, denoted as P, not as a point in the domain or rather the unit domain, but as a three-dimensional Gaussian bubble of certain variance therein.

This comes from the fact that, inspired by the attractor network model of grid cells, we employ continuous attractor dynamics and the concept of population coding to sustain pose estimates (see Sec. 5.3). Individual neurons are tuned to Gaussian-shaped areas in the unit domain. For any pose a subset of neurons in its vicinity will be active, where a neuron will respond stronger the closer it is to the to-be-represented pose. Given the Gaussian neural tuning, the activity packet corresponding to a specific pose will resemble a Gaussian bubble centered at that pose. Therefore, treating pose estimates as 3D Gaussians centered at the desired pose in the unit domain is only natural.

Further, Gaussians superimpose nicely, i.e. they allow us to have continuous multidimensional functions representing multiple believes simultaneously. Also, interactions between nearby pose estimates can be modeled smoothly. Such situation occurs e.g. when reset input tries to spawn a new 3D Gaussian close to an existing pose estimate. Depending on the distance between their centers, the two Gaussians may be approximated by a single Gaussian of larger variance.

Importantly, the variance of a pose estimate's Gaussian does not directly correspond to the probabilistic confidence level in that estimate. Rather,

it stems from population coding as a tuning parameter for stability. An estimate's variance depends on the variance of the neural Gaussian tuning curves and on the synaptic connectivity between the neurons. Since the recurrent connectivity is static, all stable attractor states' pose estimates have the same fixed variance. Nonetheless, attractor dynamics allow the variance to vary around the stable value over time, e.g. due to varying or noisy reset or velocity inputs, or, as mentioned above, due to merging of nearby pose estimates. An increased variance in such cases could be interpreted as higher uncertainty of the pose estimate.

We use 3D Gaussians of variance 0.0025 for representing poses and defining neural tuning. One reason for that value is that the corresponding Gaussians fit nicely within the unit domain, i.e. the parts lost due to cropping and periodizing are mostly zero anyway. Another one is because then we can fit multiple simultaneous pose estimates into the unit domain without too much interference. Also, a discrete Fourier transform with resolution $12 \times 12 \times 12$ can capture most of the information content of such Gaussian. Regarding neuron tuning, this variance value yields appropriate response curves. More details are discussed in Secs. 5.2.2 and 5.3.

5.2.2 Computing Gaussians in the Periodic Unit Domain

We need to compute three-dimensional Gaussians respecting the periodic boundary conditions of our unit domain Dom. To do this, we need the true distances between any two points in Dom, such that when computing a Gaussian we know the distance between its center and all other points. Let $\mathbf{b}$ and $\mathbf{c}$ be two arbitrary points in Dom. The absolute distance between them can be determined by separately considering the absolute distance in x-y and the absolute distance in θ, since these are independent in that regard. For the x-y distance, we compare 9 values: the distance without wraparound, with wraparound along the first primitive vector $\mathbf{a}_0$ in the positive direction, with wraparound along the first primitive vector in the negative direction ($-\mathbf{a}_0$), with wraparound along the second primitive $\mathbf{a}_1$ vector in the positive direction, with wraparound along $-\mathbf{a}_1$, with wraparound along $\mathbf{a}_0$ and $\mathbf{a}_1$, with wraparound along $\mathbf{a}_0$ and $-\mathbf{a}_1$, and for all other combinations of $\pm\mathbf{a}_0$ and $\pm\mathbf{a}_1$ wraparound. The smallest of the 9 distance values is the true x-y distance between $\mathbf{b}$ and $\mathbf{c}$ in the periodic unit domain. For the distance along the third dimension, we simply compare the distances without wraparound, with positive wraparound along the third primitive vector $\mathbf{a}_2$, and with wraparound along $-\mathbf{a}_2$. As for x-y, the smallest of the 3 values is the true θ distance between $\mathbf{b}$ and $\mathbf{c}$ in Dom. The total true distance between the two points, i.e. considering all three dimensions, is then obtained by applying the Pythagorean theorem.

Next, we discretize the unit domain with a sampling grid. When constructing the grid, there are two goals. First, we want even coverage within the first two dimensions, such that the variance in distance between points and their closest grid point is minimal. This ensures that Gaussians are sampled as dense as possible in the position-related dimensions. Second, the third grid axis should align with the θ axis, such that there is less distortion related to the discretization when moving a Gaussian purely along the θ dimension; as is the case for pure rotations. This requirement specifically excludes the body-centered cubic lattice (BCC) and the face-centered cubic lattice (FCC), both optimal sampling lattices in three dimensions.[61] Eventually, we opt for a grid that is hexagonal in the first two dimensions and orthogonal in the third dimension. It is scaled so that the period fits into the unit domain in all three dimensions; the first two grid axes align with the first two primitive vectors $\mathbf{a}_0$ and $\mathbf{a}_1$ of the unit domain and the third grid axis aligns with the third primitive vector $\mathbf{a}_2$. We use a final sampling grid resolution of $12 \times 12 \times 12$. There are multiple reasons for that. Primarily, it ensures that when we sample a three-dimensional Gaussians of variance 0.0025 at this grid, the discrete Fourier transform thereof captures most of the information content. Also, while the sampling grid is not to be confused with the grid on which the neurons will be arranged, they may still be the same - structurally and/or resolution-wise. Due to overlapping goals, we choose the same structure. It further turns out that the same resolution of $12 \times 12 \times 12$ works best for either grid. For more information and a formal definition of such grid see Sec. 5.3 (included there since the neuron grid seems to have a greater effect on the overall performance). For example, Fig. 5.9 depicts a $5 \times 5 \times 5$ version of the final grid.

Combining both the wrapped distance computation and the sampling grid, we illustrate how an origin-centered Gaussian of variance 0.0025 can be computed in the unit domain Dom at a sampling resolution of $12 \times 12 \times 12$. First, we obtain the vertices of the hexagonal-orthogonal 3D sampling grid in our unit domain, with the vertex at index $(0,0,0)_\mathrm{g}$ (subscript g for *grid* index/position) being located at the origin and the vertex at index $(11,11,11)_\mathrm{g}$ being located at $[11/12 + 0.5 \cdot 11/12, \sqrt{3}/2 \cdot 11/12, 11/12 \coloneqq 11/12 \cdot 2\pi]$. Then we compute the absolute distances between the origin and all grid points considering the wraparound of the unit domain; as described above for arbitrary two points. Finally, we can plug these distances along with the arguments for a $[0,0,0]$-mean and $0.0025 \cdot \mathrm{diag}(1,1,1)$ covariance matrix into `scipy.stats.multivariate_normal` to get the desired sampling values of the 3D Gaussian in our unit domain with correct wraparound.

We show an interesting visualization of how this representation can be obtained alternatively. There, the relationship between the rhomboidal unit tile of the conceptual tessellation of 2D space by our unit domain and the equivalent hexagonal tile is exploited. First, a 3D Gaussian centered at the origin is computed 'regularly', i.e. without wraparound, with sampling distance $1/12$ in all three dimensions and a hexagonal arrangement of sampling points in the first two dimensions. Figure 5.5 shows a 2D slice of the sampled Gaussian at $\theta = 0$, along with our rhomboidal unit tile and the equivalent hexagonal tile. Evidently, the Gaussian fits nicely within the hexagonal tile and, therefore, also in our unit domain. The values outside the hexagon are essentially all zero. The annotated parts a, c, and d are simply rearranged as permitted by the periodic boundary conditions and as shown in Fig. 5.6, in order to obtain the final sampled Gaussian in the unit domain. As for the third dimension, we rearrange the samples with $\theta < 0$ and $\theta > -\pi$ into the upper half of the unit domain.

The final sampled Gaussian, sliced at $\theta = 0$, is depicted in Fig. 5.6. It represents the pose $\mathrm{P} = [0,0,0°] = (0,0,0)_\mathrm{g}$. The former are vector coordinates in the domain or unit domain. The latter are grid coordinates in the sampling grid.

At some point during the project the computation of 3D Gaussian bubbles within the periodic unit domain Dom, i.e. with proper twisted wraparound on all sides, was only directly implemented for a

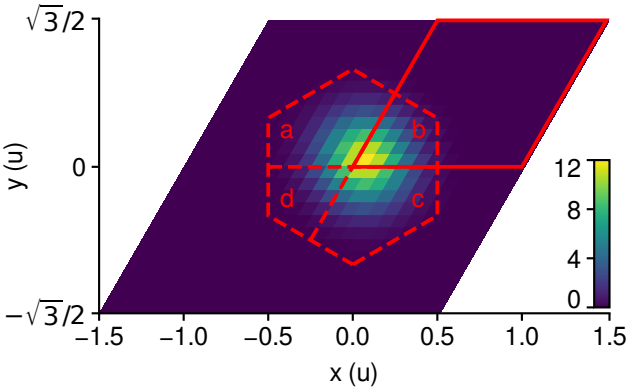

Figure 5.5 Two-dimensional slice at $\theta = 0$ of an origin-centered 3D Gaussian with variance 0.0025. The sample spacing is $1/12$ in all three dimensions and the sampling points are arranged hexagonally in the first two dimensions. Also shown are the rhomboidal unit tile corresponding to the conceptual tessellation of 2D space by our unit domain, as well as the equivalent hexagonal tile. By design, the Gaussian fits nicely within the hexagonal tile, as sample values outside it are mostly zero, and, therefore, fits within our unit domain. To obtain the representation of the Gaussian within Dom, we conceptually rearrange the annotated parts into the rhomboidal tile, as shown in Fig. 5.6. This further illustrates the correspondence between the rhomboidal and hexagonal tiles, both of which generate the same tessellation/lattice. The annotations are inspired by Fig. 6.4 in Odland[51]. For the third dimension we simply wrap around at $\theta = 0$ (not shown).

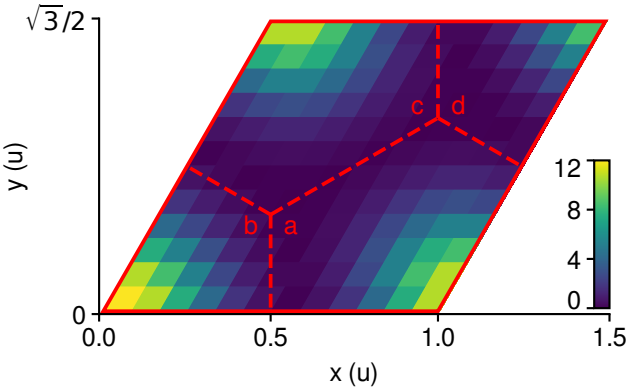

Figure 5.6 Two-dimensional slice at $\theta = 0$ of an origin-centered 3D Gaussian with variance 0.0025 in our unit domain Dom and sampled at a hexagonal-orthogonal grid of resolution $12 \times 12 \times 12$. This corresponds to pose $\mathrm{P} = [0,0,0°]$. The annotated parts illustrate how the representation is obtained from the non-wrapped Gaussian. They also indicate how the rhomboidal tile relates to the hexagonal tile. Both aspects are shown in Fig. 5.5. The wraparound along the third dimension happens straightforwardly at $\theta = 0$ (not shown). The annotations are inspired by Fig. 6.4 in Odland[51].

center at the origin. This limitation specifically stemmed from the complicated wraparound configuration. We later also implemented the computation of correctly-wrapped 3D Gaussians purely in the time domain using NumPy's `roll` function. However, this method is essentially limited to integer values, i.e. by the resolution of the sampling grid. To address this issue, we compute a Gaussian centered close to the origin such that the original wraparound logic still holds and such that the 'rest' of the offset from the origin can be achieved by integer `roll` operations.

5.2.3 Moving to Frequency Space

From the previous section it becomes clear that obtaining the sampled representation of a continuous function, specifically a 3D Gaussian, in our unit domain Dom with its periodic boundary conditions is quite cumbersome. Manipulating such representation, e.g. shifting it around in the unit domain, is only possible without error in discrete steps according to the sampling grid, whereas any other shift introduces significant errors from the target representation (as obtained by shifting the continuous function *before* sampling it). This limitation does not just apply to our specific case, but is inherent to sampled representations of continuous functions.

Also, since we want to use Nengo/NEF for implementing the neural networks, it is not feasible to simply work with the parameterized continuous functions directly. Designing and evaluating the system and its dynamics would be difficult.

Therefore, rather than working with continuous Gaussian functions parameterized by mean vector and covariance matrix, or with arrays of the sampled domain volume, we opt to work in the more convenient frequency space. A sampled and wrapped 3D Gaussian corresponding to a pose estimate in the unit domain is expressed as a weighted sum of multidimensional complex exponentials, which are continuous and periodic functions. The weights are obtained from the discrete Fourier transform and called the Fourier coefficients (see Sec. 3.5 for a formal definition). We can then work with and manipulate these discrete coefficient vectors, while under the hood the continuous harmonics are smoothly affected.

The Fourier transform of a Gaussian function is again a Gaussian (in frequency space), which also holds in multiple dimensions.[61] This means that we can still talk and think about pose estimates as 3D Gaussians in the unit domain, centered at the pose and with a particular variance, just in the corresponding frequency space. Even the wraparound 'looks' the same in the coefficient vectors.

A major reason the representation in frequency space is well-suited for our use case is that the basis functions are orthonormal and periodic. Our unit domain is also periodic in all three dimensions. Since the sampling grid is constructed to respect that periodicity, the frequencies of the complex exponentials perfectly align with, capture, and support the periodic space of Dom.

Further, translations of the real space representation on the multidimensional manifold of the unit domain correspond to rotations in the frequency domain, as stated by the shift theorem (see Sec. 3.5). By applying a discrete rotation matrix to the discrete vector of Fourier coefficients we can accurately realize smooth translations of the pose estimates. Smooth translations cause smooth variations of the Fourier coefficients.

Since the Fourier transform has the property of linearity, representing two or more Gaussians works just like in real space: their Fourier coefficient vectors are simply added together. Smoothly translating multiple bumps separately results in smooth variations of the combined Fourier coefficients. Similarly, when bumps get close they essentially merge into a 'larger' bump with greater variance. The entire merging process is representable in frequency space.

In fact, as mentioned in Sec. 3.5, any function can be approximately represented in frequency space, with the approximation being more accurate with more frequency components. We make sure to use enough components so that our arbitrarily-centered pose Gaussians and superpositions thereof, including overlaps and complete merges, are sufficiently well captured by their Fourier coefficient vectors. Specifically, the number of required coefficients or basis functions (per dimension) is determined mainly by the variance (in that dimension). See Sec. 5.2.4 for more details.

Another reason for working with Fourier coefficient vectors is that it goes well together with NEF/Nengo: a neuron's tuning is defined by such vector and the neuron's response is directly proportional to the dot product of its tuning vector with some input, also in the form of a vector in the same space. If both vectors correspond to the frequency space representation of Gaussians in our unit do-

main, then, due to the underlying periodic basis functions, the dot product constitutes a measure of similarity or closeness of the Gaussian centers in the periodic unit domain. Conveniently, the neural response curves then are also of Gaussian-like shape. This way of computing the neural responses would not work in real space. Furthermore, specifying connection functions between Nengo ensembles and analyzing the dynamics is easier in frequency space. See Secs. 3.4.1 and 5.3 for more details.

In summary, we choose to transform the unit domain Dom to frequency space because it simplifies or rather enables various aspects of our approach. This transformation is just an abstraction that does not void the biological plausibility and workings at the neural level. A pose estimate is represented by the set of Fourier coefficients of appropriately centered three-dimensional Gaussians in the unit domain. We also use the terms reciprocal space or Fourier space to refer to the frequency space.

5.2.4 Fourier Transform with Hexagonal Sampling Grid

In order to obtain the representation in frequency space, we use the multidimensional discrete Fourier transform. It assumes and requires its input to be sampled in a space with orthogonal dimensions. This is partly to ensure the correct relation between the frequencies in different dimensions. As introduced in Sec. 5.2.2 and more formally in Sec. 5.3, our sampling grid is hexagonal in the first two dimensions, and the third dimension is orthogonal to the plane spanned by the first two dimensions. Hence, the first two dimensions are not orthogonal but at a $60°$ angle.

There exist methods to compute the discrete Fourier transform on general lattices, e.g. on a 2D hexagonal lattice[65] or on 3D FCC/BCC lattices.[70] However, for analysis and interpretation we strive to be able to apply the regular multidimensional Fourier transform 'as directly as possible'. This is an additional reason against using an optimal 3D sampling lattice.

Eventually, we approach the problem by rectifying the second dimension. This amounts to shearing the rhombus base into a rectangle. Then all three axes are pairwise orthogonal. In practice, we simply treat the dimensions of the array of sample points as being orthogonal, without any modification. Naturally, the conceptual rectification retains the neighbor relations, semantics, and, most importantly, the periodicity of the data. Remember that the Fourier transform assumes periodic input. In Fig. 5.6 we showed a horizontal slice through an origin-centered 3D Gaussian in our unit domain. The rectified version of this, as seen by the multidimensional Fourier transform, is plotted in Fig. 5.7. Here, we identify the sampled points by their index in the data array.

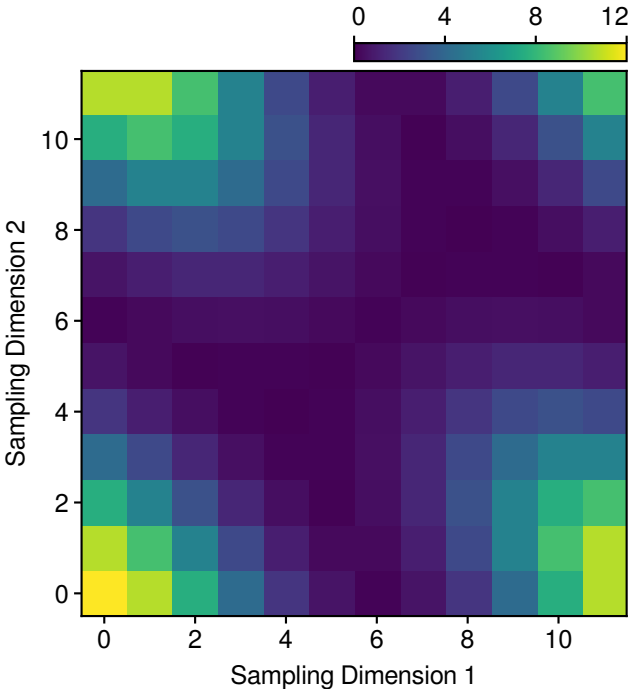

Figure 5.7 Horizontal slice through the rectified sample values of a 3D origin-centered Gaussian. This can be directly input to the multidimensional Fourier transform, which assumes orthogonal dimensions. Importantly, the periodicity of our unit domain, specifically in the positional dimensions as illustrated in Fig. 5.2, still holds for the rectified data. The original data, i.e. non-rectified, is shown in Fig. 5.6.

Now, we can apply the regular three-dimensional Fourier transform. Given how we obtain its input data, intuitively, this amounts to applying one-dimensional Fourier transforms along the axes of our unit domain.

The Fourier transform assumes a sampling domain length of 1 for each dimension. This is true for our first and third dimension, but the second dimension has sampling domain length $\sqrt{3}/2$ after the rectification (remember that we conceptually shear the rhombus to the left). This not only affects the interpretation of the frequencies of the Fourier transform's output for the second axis, but also intro-

duces a scaling factor on the entire coefficient vector. To restore the correct scaling of the Fourier coefficients with respect to our hexagonal-orthogonal sampling grid (in the unit domain), we multiply the result of `np.fft.fftn` by $\sqrt{3}/2$. We verified this with the `abelian` package* by Odland[51], who inspired the rectification approach in the first place.

When transforming from frequency space back to real space, i.e. computing the multidimensional inverse FFT on the reciprocal hexagonal lattice, we also have to account for the specific domain extents; this time, by dividing the vector of Fourier coefficients by $1 \cdot \sqrt{3}/2 \cdot 1$ before passing it to `np.fft.ifftn`. Equivalently, since the IFFT is linear, we can also just divide its output by that value.

Furthermore, we apply the normalization factor $1/N$, with N being the number of sampled values in the input array, on the forward transform. This ensures that the 0th Fourier coefficient represents the true average value of the input signal. For general information regarding the Fourier transform see Sec. 3.5.

We depict a slice of the final custom-scaled and normalized Fourier coefficients of an origin-centered 3D Gaussian in our unit domain in Fig. 5.8. The slice is taken where the frequency in the third dimension is zero. Note that the output of `np.fft.fftn` has the following structure: per dimension, the zero-frequency component is at index 0, followed by the positive-frequency components in increasing order of frequency, and then the negative-frequency components in decreasing order of absolute frequency. For visualization, we use `np.fft.fftshift` to rearrange each axis as follows: positive-frequency components in decreasing order of frequency, zero-frequency component, negative-frequency components in increasing order of absolute frequency.

We deliberately do not include a Figure of hexagonally arranged Fourier coefficients, as one would expect given the hexagonal sampling in the first two dimensions. The correspondence between reciprocal space and direct space is complicated. For our rhombus sampling shape in the first two dimension the corresponding shape in frequency space is also a rhombus, but 'somehow flipped'. Odland[51] use a Voronoi mapping to get one correct visualization of the coefficients (Fig. 7.3f therein). We found this flipping extend to the reciprocal unit domain in three dimensions. Still, the shape and periodicity

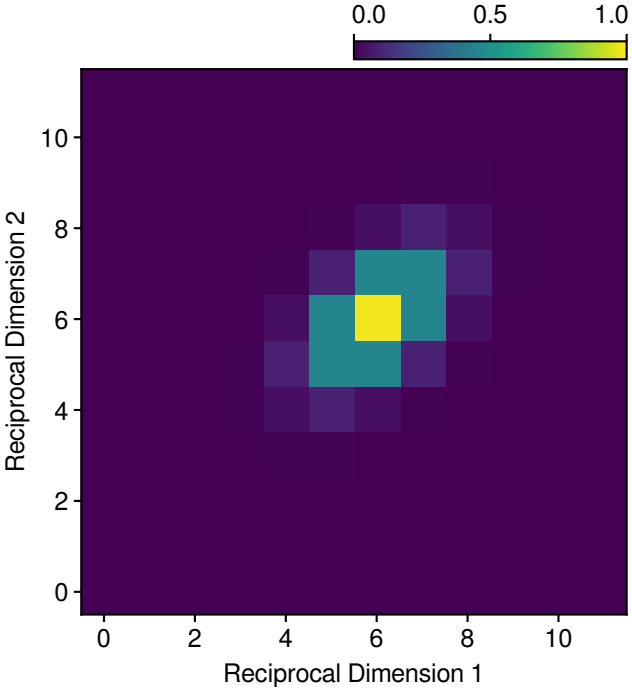

Figure 5.8 Horizontal slice in the middle of the third axes, i.e. where third component corresponds to a frequency of zero, of the post-processed Fourier coefficients of an origin-centered 3D Gaussian in our unit domain. Specifically, the rectified data partially depicted in Fig. 5.7 is input to the regular multidimensional Fourier transform. The returned output is scaled by $\sqrt{3}/2$ to account for a reduced domain length in the second dimension due to rectification from a rhombus. We apply the normalization factor $1/N$ on the forward transform. Each axis first contains the positive-frequency terms in decreasing order of frequency, then the zero-frequency term, and finally the negative-frequency terms in increasing order of absolute frequency. Stemming from the hexagonal-orthogonal sampling grid, the 'pattern' appears elongated along the main diagonal.

of our unit domain or of the hexagonal-orthogonal sampling grid is somehow transferred to the reciprocal space. Furthermore, it is not entirely clear how much interdependency there is between the dimensions in frequency space and their associated Fourier coefficients. These circumstances make the three-dimensional Fourier transform and reciprocal space very difficult to work with and to interpret individual parts of the coefficient vector.

After a lot of experimenting and interpretation efforts, both of individual and combined dimensions in reciprocal space, e.g. by omitting some dimensions and analyzing the effect on the information content, by considering and comparing projections along various axes, by visualizing the content in various coefficients, by checking for any symmetries, or by

* https://github.com/tommyod/abelian

explicitly relating the flipped rhombic prism in reciprocal space to the one in direct space; we found that for our purposes, as long as we keep nearly all reciprocal dimensions, we can treat the coefficients as if they were arranged on our original sampling grid, i.e. non-flipped, and as if there was a direct correspondence between frequencies/frequency indices and the sampling points/grid indices.

In other words, from now on we assume that the frequency dimensions directly correspond to the respective dimensions in direct/physical space. This prompts us to use the term *reciprocal grid* synonymously for our hexagonal-orthogonal sampling grid.

Now, we are in a position to further discuss the sampling resolution. Multiple factors are at play here. The sampling resolution directly determines the frequency resolution. For example, when we sample our domain on a $12\times12\times12$ grid, the resulting frequency resolution is also 12 per dimension. The number of Fourier coefficients roughly corresponds to half the dimensionality of our final representational space (see Sec. 5.2.5). For computational efficiency we want as low a frequency resolution as possible. The minimal expected variance (per dimension) of the 3D Gaussians used to represent our pose estimates directly influences the required minimum number of frequency components (per dimension). A smaller variance means higher-frequency exponentials are necessary, which, in turn, means more Fourier coefficients are needed. To fit multiple simultaneous and preferably non-interfering 3D Gaussians into the unit domain we have to keep their variance low. An upper bound on the variance (per dimension) is given implicitly via the extent of the sampling domain (per dimension). The first Fourier coefficient should have a magnitude of 1. This indicates whether all or most of the Gaussian is captured by the Fourier transform. For example, if the first coefficient is only 0.8, then this means that some non-zero part of the Gaussian lies outside the sampling domain. To get an idea of the relationship between the frequency resolution and the variance we considered the following metrics: histograms of the Fourier coefficient magnitudes and phases, to see how many of the values are zero; the Fourier coefficient spectrum, to compare the relative importance of all coefficients; and how accurately the frequency representation of a 1D Gaussian can be rotated from the origin center to some target position.

From various sampling grid resolutions, including $4\times4\times4$, $8\times8\times8$, $12\times12\times12$, and $16\times16\times16$; and various Gaussian variances, we arrived at the combination $12\times12\times12$ and 0.0025 variance as a good balance of the discussed factors and considerations. The variance influences the neural response curves and the attractor dynamics. Hence, the value of 0.0025 is largely determined based on neural dynamics (see Sec. 5.3).

We also tried sampling on a large grid, applying the Fourier transform, and then cropping the coefficients to a smaller grid. Eventually, reconstruction could be performed onto the larger grid, by reintroducing the cropped dimensions with zero values. Due to the complicated multidimensional frequency space this effort of manipulating, i.e. cropping and reconstructing, the coefficient vectors was quite involved. Our finding is that, compared to the true large-grid values, the additional points of the large-grid reconstruction are far off from the actual value, especially for smaller variance. There are also artifacts in the form of negative components that should be zero or slightly positive. Staying on the smaller grid, i.e. not reintroducing the 'lost' dimensions as zeros, gives a better reconstruction. We further note that, for the Gaussian variances tested, applying the Fourier transform on the smaller grid yields almost the same coefficients as applying it on the larger grid but then cropping the coefficient vectors. We conclude that switching between grids is not beneficial.

Another note on the sampling grid. For the discrete Fourier transform the alignment of the sampling grid relative to the signal influences the result. The fewer sampling points, i.e. the coarser the sampling grid, the greater the effect. We try to convey this issue by an example in 1D on the interval 0 to 1. Assume we have a Gaussian centered at 0. Then there are at least two options of how to distribute 5 equally spaced sampling points: we can distribute them asymmetrically in the domain (e.g. at [0.1, 0.3, 0.5, 0.7, 0.9]) or symmetrically (e.g. at [0, 0.2, 0.4, 0.6, 0.8]), where symmetry is evaluated about the center of the domain. Considering the periodicity assumption of the Fourier transform, i.e. if there is even spacing between points and if we go that same distance from the rightmost point further to the right and wrap around at the domain border then we land precisely at the leftmost point, the results, after having been adjusted to use the same phase shifts, may be different from each other. Further,

one of the results may be closer to its respective analytical counterpart, which is obtained by computing the analytical Fourier transform and then sampling at the desired frequencies. The bottom line is that, in general, the position of the sampling points relative to the domain needs to be considered when working with Fourier coefficients. However, since we deal with Gaussians that may be centered anywhere within the domain, we have to choose one option and apply that consistently. We opt to simply place the first sampling point in the lower-hand corner of our three-dimensional domain, and distribute the rest of them such that they are evenly spaced and that the periodicity assumption holds. Considering that we typically start out from a Gaussian centered at 0, transform it to frequency space, and then perform any necessary operations to get the desired data, with the outlined sampling grid convention we capture the value at the center of the Gaussian, and also nicely capture its periodic nature, i.e. no duplicate values where the ends of the Gaussian meet at 0.5. These considerations are especially important when debugging rotation matrices, which are introduced later in this section.

5.2.5 Flattened and Reduced Representational Space

Most of the time we work in the flattened complex/reciprocal space, where the three dimensions are collapsed into one dimension in row-major order and the real and imaginary parts of the complex coefficients are interleaved.

To further optimize the representation, i.e. reduce the number of dimensions, we exploit two properties of the coefficients obtained from the Fourier transform of real-valued signals (see also Sec. 3.5). First, the coefficient at index 0 is a real number. Second, for an even length input array the largest frequency is called Nyquist frequency. Its corresponding coefficient is located at the center index and also a real number. Since the imaginary part of a real number is zero, we can omit it without losing any information.

The two properties apply in a slightly more complex manner to the multidimensional Fourier transform. See Sec. 5.2.4 for more information on the inherent difficulty of working in multidimensional frequency space. We apply the simplified interpretation proposed therein. We found that Fourier coefficients at a combined index $\mathbf{i}$ are real numbers if every element $\mathbf{i}_k$ of $\mathbf{i}$ satisfies: ($\mathbf{i}_k = 0$) or ($\mathbf{i}_k$ corresponds to the center-index of an even-length dimension). These coefficients are often collectively referred to as Nyquist terms. We compute an index mask that selects all positions in the flattened coefficient vector except the ones corresponding to the imaginary parts of coefficients fulfilling this extended rule. Hence, when applying the index mask to a flattened coefficient vector we are left with a smaller vector that represents the same information.

Removing those dimensions is also useful from a Nengo-perspective, since representing constant values, i.e. zero for the imaginary parts under consideration, unnecessarily reduces an ensemble's capacity for the other non-constant dimensions.

Recall that in Nengo the neuron responses are based on the dot product between their encoders and some to-be-represented (input) vector in representational space. Since we only omit dimensions from our flattened coefficient vector which are always zero, the dot product and, consequentially, the response curves are not affected.

We define the *representational space* Rep as the flattened and reduced reciprocal space.

Limits of Reducing Dimensions. A lot of effort was spent on trying to further optimize the representation, i.e. reduce its dimensionality by identifying and omitting redundant or constant/zero parts of the Fourier coefficient set.

First, there is the fact that the Fourier transform of real-valued signals yields conjugate symmetric coefficients, meaning that the coefficients for the negative frequency components are precisely the complex conjugate of their corresponding positive frequency components. Thus, the entire information content is contained in just half the coefficient set, since the other half can be computed form it. In multiple dimensions the conjugate symmetry applies to the last transformed dimension, in our case the θ-axis. We conducted extensive tests where we omit the second half of the θ-axis for the representational space, e.g. using `np.fft.rfftn` which does not compute the redundant coefficients in the first place, thereby reducing the dimension of this space by 50 percent. In theory, this should significantly improve an ensemble's representation performance, since the same number of neurons is now available for just half the representational demand. However, we find that omitting the conjugate

symmetric part of the coefficient vector does not improve but rather worsen the performance of our system. With half of the θ-dependent values missing there are distortions in the dot product in representational space and, therefore, of the response curves along the axis affected by the omission, i.e. the θ-axis. Intuitively, a cropped θ-dimension has less influence than any of the other two dimensions on the dot product, which distorts the dot product 'volume' and, ultimately, the population response.

We also tried omitting first Fourier coefficient from the representational space. It corresponds to the average intensity of the signal. Under perfect conditions, i.e. when there is always exactly one Gaussian, it remains constant for the entire simulation. The first coefficient is essentially independent of the center of that Gaussian. However, when there are multiple simultaneous pose estimates and corresponding Gaussians, then the average increases with their number, since we explicitly do not normalize the representational space. Also, we deal with dot products of Fourier coefficient vectors later on and the first coefficient is required to get properly scaled values, with the minimum being zero. Therefore, to correctly represent and decode multiple bumps we need to include the real part of the first Fourier coefficient in our representational space.

Then there are the so-called Nyquist terms, which we introduced before. Apart from successfully omitting their zero imaginary component, we additionally experimented with omitting some or all of the Nyquist terms completely. The reason behind this is that some of them are zero for a Gaussian centered at the origin. When we compute encoders and input vectors based on the Fourier coefficients of that Gaussian, i.e. by exclusively manipulating the values in frequency space, then all encountered vectors in representational space have zeros at the corresponding dimensions. Thus, these dimensions could be omitted without affecting the simulation. If, however, we were to compute encoders and inputs directly from Gaussians in real space, then the same Nyquist terms might not be zero. This discrepancy comes from the discrete sampling in real space when computing the Gaussians and the subsequent application of the discrete Fourier transform. The error becomes smaller with more sampling points. For overall consistency and since we want to introduce as little computational error as possible we keep the real parts of all Nyquist terms in the representational space.

In summary, all except the imaginary parts of the first coefficient and the Nyquist terms are deemed necessary and, thus, included in the representational space.

5.2.6 Manipulating Representations in Frequency Space

We need to be able to manipulate representations in frequency space and representational space Rep. First, for handling arbitrary poses in the unit domain we require the wrapped representation of an arbitrarily-centered Gaussians, which, as indicated in Sec. 5.2.2, is rather cumbersome to compute directly in time space. It is more intuitive and efficient to start out from an origin-centered Gaussian in time space, transform it to frequency space, shift it to the desired location, and then back-transform it to time space. Second, pose estimates need to be moved around within the unit domain according to velocity input. Since we represent poses as 3D Gaussians in frequency space it would be convenient to directly manipulate the Fourier coefficients to capture the changes of the pose estimates. Further, we use Nengo to specify the neural substrate. As encoded space we choose the representational space Rep. Therefore, especially during development, it is useful to be able to directly specify operations in that space when defining NEF-style connections.

Currently, we only need to handle real-space translations of the pose Gaussians. Such translation corresponds to a rotation in reciprocal space. This is stated by the shift theorem of the discrete Fourier transform (see Section 3.5). Given a flattened vector of Fourier coefficients (of a pose Gaussian) and a desired real-space translation, we can construct a rotation matrix, which, when applied from the left, realizes this translation in frequency space.

Formally, let $\mathbf{x}_{i_x,i_y,i_z}$ be a point in reciprocal space, i.e. a Fourier coefficient, with the coefficient index in subscript; let $\mathbf{f} = \begin{bmatrix} f_x & f_y & f_z \end{bmatrix}^\mathsf{T}$ the vector of actual frequencies associated with $\mathbf{x}$; and let $\mathbf{s} = \begin{bmatrix} s_x & s_y & s_z \end{bmatrix}^\mathsf{T}$ be the vector of signed shift values in each dimension of the reciprocal space.

The value of the Fourier coefficient $\mathbf{x}_{i_x,i_y,i_z}$ after the shift $\mathbf{s}$ is applied can be computed as follows:

$$\begin{bmatrix} \mathrm{Re}\left(\mathbf{x}_{i_x,i_y,i_z}(t+\Delta t)\right) \\ \mathrm{Im}\left(\mathbf{x}_{i_x,i_y,i_z}(t+\Delta t)\right) \end{bmatrix} = \begin{bmatrix} \cos\left(2\pi\left(\mathbf{f}^\intercal\mathbf{s}\right)\right) & \sin\left(2\pi\left(\mathbf{f}^\intercal\mathbf{s}\right)\right) \\ -\sin\left(2\pi\left(\mathbf{f}^\intercal\mathbf{s}\right)\right) & \cos\left(2\pi\left(\mathbf{f}^\intercal\mathbf{s}\right)\right) \end{bmatrix} \times \begin{bmatrix} \mathrm{Re}\left(\mathbf{x}_{f_x,f_y,f_z}(t)\right) \\ \mathrm{Im}\left(\mathbf{x}_{f_x,f_y,f_z}(t)\right) \end{bmatrix} \quad , \tag{5.1}$$

where we define the involved matrix as the rotation matrix $\mathbf{R}$ for the specific coefficient index and shift values. This matrix satisfies $\mathbf{R}^\intercal = \mathbf{R}^{-1}$. Note that the coefficient's new value only depends on its previous value, but not on any other coefficients. This allows us to construct the full 'rotation matrix' $\tilde{\mathbf{R}}$ by arranging the different individual 2×2 rotation matrices $\mathbf{R}$ along the main diagonal of a large zero-matrix. When this large matrix is applied (from the left) to the flattened array of Fourier coefficients, each individual Fourier coefficient interacts only with its corresponding individual rotation matrix $\mathbf{R}$. Note that the full 'rotation matrix' is not actually a rotation matrix in the mathematical sense, but we refer to it as such for simplicity and because it intuitively translates Fourier coefficients around in frequency space by rotation.

We validated the rotation matrix by comparing its effect/results to those of directly applying the appropriate complex exponential factors (as used for the definition of the shift theorem in Sec. 3.5); the two methods behave identical. Since the rotation operation is crucial for our approach, we also verified correctness by comparing coefficients obtained from frequency-space rotations with the coefficients obtained from time-space translation followed by a Fourier transform. As required, the resulting coefficient sets are equal.

While we can work in the reduced flattened reciprocal space for representing poses and getting correct dot products (see Sec. 5.2.5), all dimensions are required when doing rotations; specifically, to get perfectly invertible rotation matrices, i.e. to get the identity matrix back when applying the proper series of rotation matrices. For example, when we first rotate by $30°$ and then rotate by $-30°$ we desire this to amount to no rotation at all. The entries of the flattened coefficient vector always interact in pairs. Thus, we cannot omit just one value per any pair, as is done in the reduced space Rep, because then we could never obtain an identity matrix from a series of rotations. Another reason for keeping all dimensions is to accurately capture non-grid-constrained shifts. Before arriving at this conclusion, however, we designed and implemented complicated reductions of the flattened reciprocal space with the goal of retaining the ability of correct rotations. Particularly, we went back and forth with omitting the Nyquist terms, in part or completely.

5.3 Neuron Parameters and Arrangement

Poses are to be represented by an ensemble of spiking neurons with continuous attractor dynamics. Individual neurons are tuned so a specific pose in the unit domain, specified in the form of a 3D Gaussian. By arranging the neurons topographically in the unit domain, i.e. at their preferred pose, and ensuring that the tunings are such that the neurons are spread out across the entire unit domain, we can represent any pose therein via population coding. Since our unit domain is three-dimensional, we end up with a 3D attractor network, where a pose corresponds to a 3D bubble of neural activity. In this section we discuss the arrangement and corresponding tuning of neurons, the resulting response curves, as well as some aspects of the dynamics. The next section deals with the interconnection required to achieve attractor dynamics and enable population coding.

At this point we note that, while a topographic arrangement of neurons is not what happens in networks of biological grid cells, it does not annul the biological realism of our system: randomly rearranging the cells would not qualitatively change the system's behavior. We opt for the topographic arrangement as it simplifies implementation, visualization, and evaluation.

5.3.1 Neuron Grid

Since we arrange our neurons topographically, one can say that the neuron grid discretizes the unit domain Dom. From that perspective we motivate the choice of grid. The goals are the same as for the sampling grid in Sec. 5.2.2: best possible coverage of the positional space, i.e. the space spanned by the first two dimensions of Dom; and minimal discretization-related distortion when traversing Dom along the θ-dimension.

For multiple reasons, the first goal is achieved by a hexagonal grid structure in the first two di-

mensions. Such structure gives the densest possible packing ratio in two dimensions (circle packing problem).[41] In other words, when considering the Voronoi tessellation of two-dimensional space, the hexagonal arrangement of points yields the smallest possible worst-case distance between any location in space and the nearest grid point.[32] Also, from the perspective of information encoding, a hexagonal arrangement of discrete 'encoding points' is most efficient.[53] As the number of neurons gets larger, their exact arrangement matters less (e.g. hexagonal vs. square vs. uniformly random), whereas for smaller neuron numbers, as is the case for our system due to hardware restrictions, there is a difference in accuracy of position representation - hexagonal arrangement performs best.[44,66]

The second goal prompts vertical stacking of hexagonally-arranged layers of grid points, such that the third grid axis runs in parallel to the third dimension of the unit domain. Then, for pure rotations, neural activity packets can move upward smoothly through the aligned neural sheets. The activity's cross-section of supporting grid points looks the same for all consecutive layers, which prevents the introduction of distortions related to the discrete nature of the neuron grid. In other words, a pose estimate should be able to move along the θ dimension while keeping the x-y position nearly constant; this would be jeopardized when shifting every second x-y plane since then the location of the activity packet relative to a traversed plane's neurons would differ depending on whether we are on an even-index or an odd-index plane. For completeness, we mention that if we wanted optimal coverage in all *three* dimensions of Dom, then e.g. the face-centered cubic grid (FCC) would be a possible choice as it has the densest packing ratio (sphere packing problem).[28,41] Since not all of our three dimensions are qualitatively the same, i.e. two correspond to position coordinates and one to the orientation angle, we do not have such optimal volumetric coverage goal. Furthermore, note that not even biological 3D grid cells, which, notably, operate in a space where all dimensions have the same units, e.g. x, y, z in meters, are arranged in such theoretically optimal grid. Rather, they likely only feature a local ordering (see Sec. 3.3.5).

To summarize, we opt for vertically stacked hexagonal neural sheets, because we want optimal packing in x-y space together with 'linear', consistent behavior along the θ dimension.

More formally, our neuron grid corresponds to a conventional unit cell of a three-dimensional hexagonal Bravais lattice.[68] The infinite lattice is generated by three primitive translation vectors as

$$\mathbf{R} = \frac{k_1}{n_1}\mathbf{a}_1 + \frac{k_2}{n_2}\mathbf{a}_2 + \frac{k_3}{n_3}\mathbf{a}_3 \quad , \tag{5.2}$$

where the $k_i \in \mathbb{N}$ are integer weights,

$$\mathbf{a}_1 = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}, \quad \mathbf{a}_2 = \begin{bmatrix} 1/2 \\ \sqrt{3}/2 \\ 0 \end{bmatrix}, \quad \mathbf{a}_3 = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} \quad , \tag{5.3}$$

are the primitive vectors, or generators, which coincide with the primitive vectors of the unit domain Dom, and $n_i \in \mathbb{N}$ is the number of grid points along dimension i contained in the conventional unit cell of the Bravais lattice, or, equivalently, in the unit domain. Thus, the n_i define the resolution of our neuron grid per dimension. Eventually, we choose $n_1 = n_2 = n_3 = 12$. We usually specify the grid resolution in the form $12 \times 12 \times 12$.

The choice of neuron grid is actually interdependent on the shape of the unit domain, so that the latter is optimally covered with respect to the first two dimensions. Note that we use the same letters for the primitive vectors for specifying either. While the length of the third vector differs, 2π vs 1, the implementation uses length 1 and we simply interpret it as 2π, as mentioned earlier.

Furthermore, the overlapping goals for sampling grid (Sec. 5.2.2) and neuron grid lead to both having the same hexagonal-orthogonal structure. Regarding the resolutions, we found that $12 \times 12 \times 12$ works best for either grid. Some of the reasons are related. For sampling, we argued that $12 \times 12 \times 12$ is able to represent multiple, e.g. three, essentially non-overlapping Gaussians (of variance 0.0025) simultaneously with sufficient accuracy, both in real (in terms of discretization error) and in reciprocal space (in terms of available frequencies). Of course, the specific variance is tuned together with the sampling grid resolution. For the neuron grid, we are dealing with 3D activity bubbles rather than 3D Gaussians. Since these bubbles directly correspond to pose estimates, we want to be able to sustain multiple of them simultaneously without much intersection. A neuron grid of $12 \times 12 \times 12$ achieves sufficiently stable and accurate population coding behavior under that requirement (see next section). Also, using $12 \times 12 \times 12$ neurons for the main attractor network and considering the entailed resource

requirements of the final system (including all ensembles and connections) gives a just-managable computational complexity, both in time and space, after all our optimizations. It is also the maximum that fits onto the Loihi chip, again, after optimizations and tweaks. More information on the Loihi-related design decisions is spread out across the remaining sections of this chapter. Specifically for the third dimension, a resolution which is a multiple of 6 should intuitively be preferred since then the shifting layers' directional tunings include the grid axes, both in positive and negative direction. Thereby, a higher θ-resolution is better for path integration. (see Sec. 5.7 and Ch. 6). During development, we experimented with several neuron grid resolutions, including $6\times6\times6$, $8\times8\times8$, $12\times12\times12$, and $16 \times 16 \times 16$, combined with various variances of the involved 3D Gaussians.

For visual clarity, we only depict a lower-resolution version of our final neuron grid in Fig. 5.9. Shown is a grid with resolution 5 per dimension.

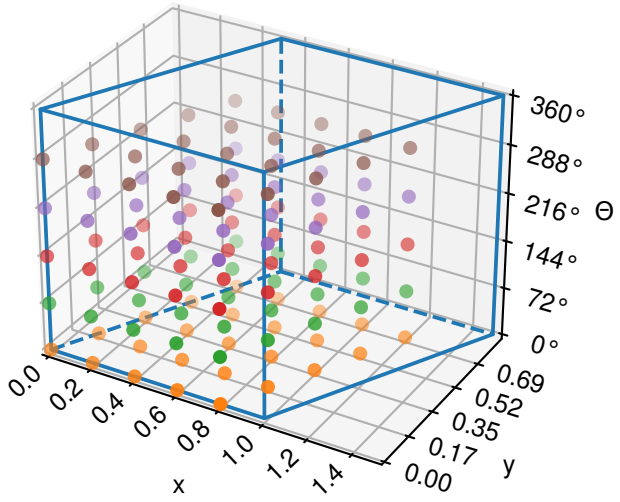

Figure 5.9 Three-dimensional hexagonal-orthogonal neuron grid, where the neurons are topographically arranged in the unit domain. Based on their tuning, the neurons resemble grid-by-direction cells. The resolution per dimension is 5, whereas our final grid has a resolution 12 per dimension. Compared to Fig. 5.4, here the axis ticks have been adjusted to align with the grid points. We use separate colors per horizontal layer.

Our neuron grid regarding position, i.e. only considering the first two dimensions, is significantly smaller than the ones employed in other literature. For example, Burak and Fiete[10] use 128×128 and 256×256; Edvardsen[17] use various resolutions from 40×40 to 64×64. We only use 12×12 neurons (per θ-layer). On the one hand, we need more neurons than other works since we implement a three-dimensional attractor network together with dedicated shifting ensembles, and, on the other hand we are limited by computational and hardware resources. For reference, rats potentially use 5000 or more grid cells in total, i.e. across the different-scale modules.[58] Of course, pose representation gets more accurate with more neurons.[13,66]

5.3.2 Encoders and Response Curves

We construct the main attractor network for representing and maintaining pose estimates as a Nengo ensemble and set the representational space Rep as its encoding space (see Sec. 3.4 for more on Nengo). This attractor network is called pose ensemble. To actually convert a pose estimate, i.e. a 3D Gaussian in the unit domain Dom that has been transformed to reciprocal space and cropped, to an activity state of the pose ensemble, we need to define the neural tuning. To this end, we associate each neuron with a vector in Rep corresponding to the Fourier coefficients of a 3D Gaussian centered at the neuron's conceptual grid location in the unit domain Dom and with a variance equal to the one of our Gaussian-represented pose estimates (0.0025).

The encoder Gaussians' centroids are chosen as the vertices of the hexagonal-orthogonal grid discussed in the previous section. Hence, the neuron grid is implicitly defined via the neurons' encoders and topographic arrangement.

For the computation of the encoders it is useful to implement a generic function `rotate_to_grid` that efficiently propagates a base Fourier coefficient vector to all grid points via rotation in reciprocal space. It exploits the grid structure and NumPy's vectorization. Specifically, it takes as input a vector of Fourier coefficients, which have been computed in the 'regular' unit domain. We compute three rotation matrices, one to rotate one grid position along the first dimension, one to rotate one grid position along the second dimension, and the third one to rotate one grid position along the third dimension. Then, we apply the third rotation matrix repeatedly to the input vector to rotate it, or rather its underlying domain, once through the third grid axis, thereby storing all intermediate results. We end up with 12 vectors of Fourier coefficients conceptually pertaining to the 12 horizontal layers of our sampling/hexagonal grid in the unit domain. Next, the

second rotation matrix is repeatedly applied to the stacked 12 vectors from the previous step in a vectorized fashion, effectively propagating the coefficient vectors along the second grid axis. We end up with a matrix of Fourier coefficient vectors conceptually pertaining to a plane of the grid spanned by its second and third axes. Finally, we similarly propagate this matrix of coefficient vectors along the first grid/domain dimension using the first rotation matrix and vectorization. The result is a four-dimensional array where the first three dimensions index the grid points, and the last dimension corresponds to the respective Fourier coefficients 'at that grid point'.

To obtain the encoders, we compute the Fourier coefficients of a 3D Gaussian bubble in the unit domain centered at the origin and of variance 0.0025 in each dimension (using our `get_0_coefs` function). We propagate these coefficients to all grid vertices, i.e. such that the coefficients pertain to a Gaussian centered at the respective vertices, using our `rotate_to_grid` function. Lastly, we normalize the coefficient vectors to unit length to obtain the encoders.

With the encoders in place, how are the neural activations determined? Let $\mathbf{P} \in Rep$ be some pose in the form of a 3D Gaussian with variance 0.0025, which we want to encode with the neural ensemble. To determine some neuron's firing activity in response to $\mathbf{P}$ we start by taking the dot product between the neuron's encoder e and the pose, $e \cdot \mathbf{P}$. The result, a measure of alignment or similarity between the two vectors, is the so-called raw input current, which gets scaled by the gain, offset by the bias, and then passed to the neuron's activation function. By default, Nengo is configured so that a raw input current of 1 (or -1) causes a neuron to be maximally active. We adopt that default. A pose neuron should respond maximally for a pose estimate located/centered exactly at the neuron's preferred location in the unit domain. Therefore, we normalize the vectors in Rep corresponding to a single 3D Gaussian (of variance 0.0025) to length 1. This affects both any pose estimates and the neuron encoders, and causes their dot products to fall into the interval $[0, 1]$ (the dot product is directly proportional to the magnitude of either argument). The neuron responses being determined based on dot products between vectors in some encoding space is one of the motivations for working in reciprocal space, for using Gaussians to represent poses, and for using the same Gaussian parameters for pose estimates and neuron tunings (see also Secs. 5.2.1 and 5.2.3). We want to emphasize that the dot product behaves as desired in the reciprocal of the periodic unit domain, i.e. the dot product of two Gaussians in Rep only depends on the absolute distance between their centers in Dom and, in particular, does not depend on the centers' location relative to the boundary of Dom. The dot product between two vectors from Rep effectively computes the dot product in complex space. A dot product of 1 is attained for poses and encoders with an identical underlying Gaussian, i.e. perfect alignment. 0 is attained if pose and encoder are maximally misaligned (with respect to the unit domain), i.e. if the underlying Gaussians are phase-offset in all three dimensions by $180°$. Such offset causes the involved Fourier coefficients to be orthogonal and, thus, the dot product to be 0. In between, we get roughly Gaussian-shaped values, which highlights that Gaussians in reciprocal space are well-suited for our use case. Ultimately, we can speak of Gaussian neural tuning. This is also the kind of tuning found e.g. in biological grid cells. From that perspective the neurons of our main attractor network correspond to *conjunctive grid-by-direction cells*.[30] Since our unit domain is periodic, the Gaussian tuning curves are periodic, and so we effectively end up with von Mises/von Mises-Fisher functions.[23]

We depict a 2D slice of the response curves of some of our neurons in Fig. 5.10. A response curve pertains to a neuron's raw input current for different input stimuli, i.e. Gaussian pose estimates in Rep. The periodic Gaussian-like shape is evident. Our response curves are not to be confused with Nengo's response curves. While both concern the preferred direction of neurons in the encoding space, the former project to raw input current, whereas the latter project to firing rate.

From the response curves it becomes clear how a 3D-Gaussian pose estimate corresponds to a Gaussian-like activity bubble in our attractor network. A single pose is represented by multiple neurons 'close to it'. The closer a neuron is to the to-be-represented pose the more active it is. For example, when only considering the 12 neurons shown in Fig. 5.10, for pose $\mathbf{P} = (0.5, 0, 0°)$ neuron $(6, 0, 0)$ is maximally active, neurons $(5, 0, 0)$ and $(7, 0, 0)$ are about 50 percent active, neurons $(4, 0, 0)$ and $(8, 0, 0)$ are barely active, while all other neurons are not active. This works similarly when representing poses that do not coincide with a neuron's pre-

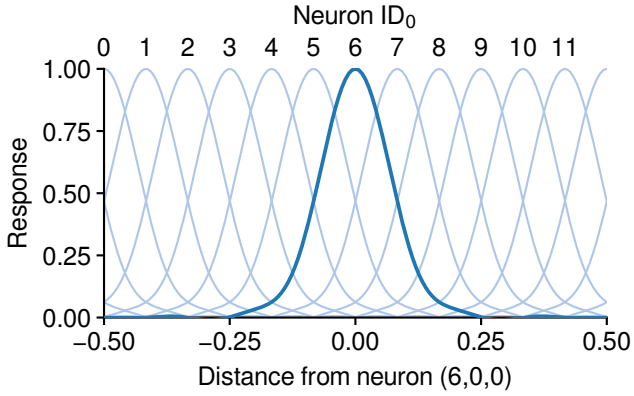

Figure 5.10 Two-dimensional slice of neural response curves of neurons along the first grid axis. A neuron's response curve comprises the dot products of its encoder with Gaussian pose estimates centered at different locations in the unit domain. Shown are responses to bumps centered anywhere on the first neuron grid axis with the other coordinates being zero. The curve for neuron $(6, 0, 0)$ is highlighted. It is shaped like a 2D Gaussian. The bottom scale shows the signed distance from neuron $(6, 0, 0)$ to points on this axis. The extent of the unit domain in the corresponding dimension 1, so the figure wraps around at ± 0.5. Top labels indicate the neuron index along the first axis of the neuron grid. From the response curves we can clearly see how our neurons perform population coding, i.e. represent continuous values with a discrete number of encoders/neurons.

ferred pose. In fact, by combining the activations of multiple neurons to represent a single pose we can smoothly represent any pose in Dom. This is an instance of population coding and also its main advantage: being able to encode continuous values with a discrete set of encoding points. Furthermore, this explains why we effectively work with a *continuous* 3D attractor network. The required connectivity for attractor dynamics is discussed in the next section.

Enabling population coding and, ultimately, attractor dynamics is another strong reason for using 3D Gaussians to represent poses; it is a natural choice. Actually, population coding is optimal for neurons with Gaussian/von Mises tuning.[34] The variance of the Gaussians determines the width and overlap of the individual neurons' response curves. These need to overlap 'just the right amount' for most efficient and effective population coding. We tune the Gaussian variance such that for a pose estimate coinciding with a neuron's preferred pose, the immediately neighboring neurons in the same horizontal slice (6 neurons) are about 50 percent active and the next-closest neighbors are barely active. This gives us the benefits of population coding but also keeps activity bubbles as small as possible in order to allow for multiple simultaneous bubbles, corresponding to distinct pose estimates, without much intersection/interference. A natural trade-off here is that more neurons per pose estimate yield a more stable and accurate representation, but require more neurons in total and/or reduce the number of simultaneously representable pose estimates. Naturally, the response curves directly influence the (feasible range of) recurrent connection weights and attractor dynamics. We arrive at a variance of 0.0025 (per dimension). Of course, this value strongly depends on the neuron grid resolution and, therefore, was tuned conjunctively. Note that the dot product appears to be rather independent of the sampling grid and neuron grid resolutions, unless the resolution is very small.

The effect of the variance (and grid resolution) on the response curves (and recurrent connection weights) is the final consideration, in addition to the ones discussed in various previous sections, that led us to the combination of a 0.0025 Gaussian variance (per dimension) with a $12 \times 12 \times 12$ neuron grid.

Finally, in Fig. 5.11 we show a two-dimensional slice of the response pattern (raw input currents) of our 3D neuron grid for the pose $(0, 0, 0°)$. Note that here we do not consider any recurrent connectivity or attractor dynamics yet. Ideally, however, the final population response looks very similar. In the figure we not only depict the values within the unit domain, but also in the rest of the domain by unrolling the unit domain according to its periodic boundary conditions. The effective hexagonal pattern of neural activity with respect to the domain becomes visible. We speak of a hexagonal population response (in the first two dimensions). Furthermore, Fig. 5.11 equivalently shows the response (raw input currents) of neuron $(0, 0, 0)$ to poses centered at vertices of the neuron grid, i.e. a single-neuron response. At this point we highlight that a single-neuron response depicts the response of a neuron to multiple separate inputs (e.g. over time), while a population response depicts the instantaneous response of all neurons of the population to a specific input.

The encoder setup is only the first step toward a hexagonal firing pattern (in the first two dimensions). Specifically, we want to have a single bump of activity per unit domain for both the population

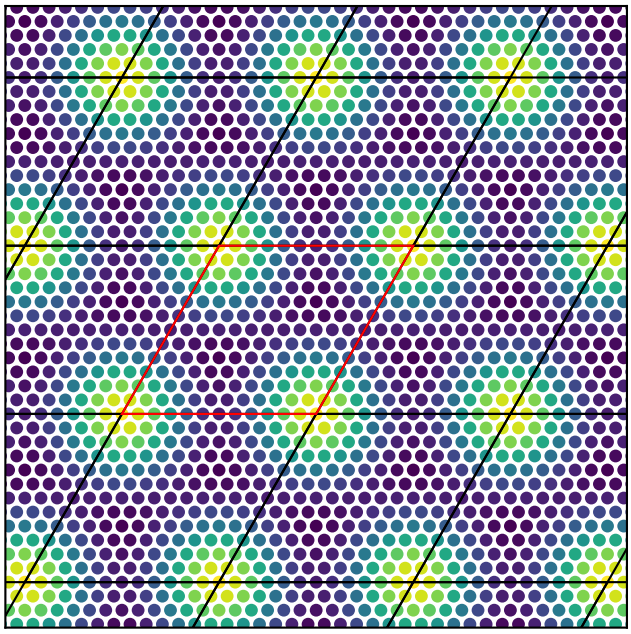

Figure 5.11 First two dimensions of the raw population response relative to the unit domain for pose $(0, 0, 0°)$; raw in the sense that only the dot product values of the neurons' encoders with the input vector is shown. Additionally considering the recurrent connectivity (gains, biases, activation functions) and the resulting attractor dynamics would not qualitatively change the firing pattern. The horizontal slice is taken at $\theta = 0$. Unrolling the first two dimensions of the unit domain according to its periodic boundary conditions yields a hexagonal grid. Equivalently, the figure shows the raw response of neuron $(0, 0, 0)$ to varying poses in the '$\theta = 0$'-plane, i.e. a horizontal slice of a single-neuron response. Evidently, it can be arranged to hexagonally tessellate positional space.

response and the single-neuron response. Relating to the formal specification of the neuron grid as the conventional unit cell of a hexagonal Bravais lattice, here we are dealing with a *primitive* unit cell of a hexagonal Bravais lattice, i.e. the smallest part of the lattice able to generate the lattice. The primitive unit cell contains only one grid vertex, which corresponds to the single activity bump. The other element required to get a hexagonal firing pattern is the recurrent connectivity (see Sec. 5.4).

5.3.3 Neuron Parameters and Implementation Specifics

We encapsulate the pose ensemble in a Nengo network with the parameters listed in Tab. 5.1. Regarding the frequency resolution, remember that we omit the imaginary parts of the Nyquist terms for the encoding space (see Sec. 5.2.5).

Table 5.1 Settings and parameter values for our Nengo ensembles.

Frequency resolution	$12 \times 12 \times 12$
Neuron grid	$12 \times 12 \times 12$
Neuron model	`LIF` / `LoihiLIF`
Input bias current	0.968
Input gain	2.68423963

Based on the simulator backend, default or NengoLoihi, one of the two neuron models `LIF` or `LoihiLIF` , respectively, is used. Both are simulated with spiking dynamics. The latter type is consistent with Loihi's LIF implementation. During development, we experimented with other neuron types such as `nengo_extras.neurons.FastLIF`, but found the default types to work best. For designing and analyzing the dynamics we can use a rate-based mean-field abstraction and talk about firing rates instead of spike trains, even though the actual spiking dynamics are simulated.[38]

As explained in Sec. 5.4.2, we initially configure the pose ensemble with intercepts 0 and nominal firing rates 120, and the recurrent connection as a decoded connection with the identity transform. Nengo then computes connection weights, gains (2.68423963), and biases (1). We go on to tune the weights and biases. For the final system, we use a direct neuron-to-neuron connection with manually specified weights, gains of 2.68423963, and biases of 0.968 (see Sec. 5.4.3).

We choose same parameters for each neuron to reduce the number of to-be-tuned variables. Of course, there are still the distinct encoders that cause the neurons to behave differently from each other. Realistic would be to randomize each neuron's gain and bias parameters to achieve a heterogeneous neural population, as observed in the brain.[13]

When using the NengoLoihi backend, ensembles are automatically split to fit onto Loihi cores. By default, consecutive groups of neurons are spread evenly across the cores. If successful, this obviously respects the 1024 neuron-per-core limit. However, other constraints such as number of input axons, number of output axons, and synaptic memory. For our system, including the pose ensemble with its recurrent connections (see Sec. 5.4) as well as the shifting ensembles and associated connections (see Sec. 5.7), the default splitting does not work. It would e.g. require too many input ax-

ons. Therefore, we specify a desired splitting unit, or `BlockShape`, of $(4, 4, 4)$ at ensemble creation. It no only affects the number of neurons per core, but, more importantly, exploits the sparse and localized connectivity structure of our pose ensemble. When thinking of the neurons in a three-dimensional grid, we cut the grid into 27 blocks of equal size (64) and shape $(4 \times 4 \times 4)$. Each block is put onto a distinct (simulated) Loihi core and the resulting intra- and inter-block connectivity fulfills the (simulated) hardware constraints. Finally, when using the NengoLoihi backend, model parameters are discretized at build time and, occasionally, not all relevant bits can be kept. Together with the difference between `LIF` and `LoihiLIF`, and the different initial voltages per neuron (random vs. 0), this explains the slight differences between simulations with default backend vs. NengoLoihi backend.

We refer the reader to `run_experiment` and `get_pose_network` for all details on the network construction.

5.4 Connectivity Profile

To stably sustain the Gaussian-shaped activity bubbles corresponding to pose estimates after they have been initialized at an arbitrary location (as described in Sec. 5.3.2), we implement attractor dynamics by introducing recurrent connections between the neurons. Eventually, we employ a Mexican hat-type local connectivity to allow multiple bumps to coexist, and use constant global inhibition to prevent random activity packets from forming.

5.4.1 Wraparound for Hexagonal Firing

As mentioned in Sec. 5.3.2, we want the first two dimensions of our population response and single-neuron responses to resemble a hexagonal grid of firing locations, akin to biological grid cells. This grid should not only hold relative to physical space, but also relative to our unit domain. To achieve this the recurrent connectivity needs to be able to keep the activity packets alive and prevent them from spreading/dissolving. Provided we have correctly implemented velocity input (see Sec. 5.7) and as long as there are no distortions at the boundary of the unit domain, that could already be sufficient to obtain the hexagonal pattern relative to physical space. To get the hexagonal pattern also relative to the unit domain or neuron grid, particularly, to get the three axes of periodicity characteristic for a hexagonal pattern to hold within the unit domain/neuron grid, the recurrent connectivity needs to wrap around correctly. Our unit domain is set up such that the 'correct' neural wraparound connectivity can be achieved by simply following the domain's shape and periodic boundary. This amounts to computing the recurrent weights based on the manifold of the stacked, twisted tori underlying the unit domain and, with the described wraparound connectivity, also the neuron grid. An interesting comparison between square and hexagonal toroidal models for grid cell activity is presented in extended Fig. 7c and d in Gardner et al.[27]. The square torus is an example of 'incorrect' wraparound connectivity for our case.

Together with the unit domain setup (see Sec. 5.1), the encoders introduced in Sec. 5.3.2, and some form of global inhibition (see Sec. 5.7) a properly-wrapped recurrent connectivity completes the requirements for a population response that follows a hexagonal grid in the first two dimensions such that there is only a singular activity packet within the unit domain per pose estimate. Further, it completes the direct relationship between population response and single-neuron response. Thus, they effectively 'look' the same. The hexagonal grid has a spatial period of $1\,\mathrm{u}$ relative to the unit domain; its period relative to physical space depends on the mapping of velocity input (see Secs. 5.1 and 5.7). For a visualization of how the single-neuron response (in the first two dimensions) can be arranged to tessellate 2D space and the resulting hexagonal grid, consult Fig. 5.11.

Interestingly, due to the periodic boundary conditions in the neural grid the attractor manifold does not include rotations of the firing grid. What this means is that we can persistently translate the firing grid along all three dimensions, i.e. to capture translator and rotational movement, but we cannot rotate the firing grid about any axis. This is because any rotation would cause the attractor state to leave the energy minimum and sort of create a tension or mismatch at the border of the unit domain. Consequently, the system is robust against noise that would otherwise introduce rotational errors.[10]

5.4.2 Recurrent Weights

Here we discuss the actual connection weights between the neurons in the pose ensemble. The goal when designing the weights is to realize continuous

attractor dynamics for pose estimates. As defined by the neural tuning in Sec. 5.3.2, a pose is represented in the form of a 3D Gaussian-shaped bubble of neural activity centered at the pose in Dom. Such activity bubble is initialized from some outside input stimulus. Afterwards, it needs to be sustained independent of the input stimulus. We want stable behavior with as low as possible drift. The width of an activity packet should preferably remain similar to the initial width. Furthermore, it should be possible to represent multiple separate pose estimates simultaneously, which amounts to sustaining multiple *independent* activity packets at the same time. If these packets are far enough apart, they should not interfere. Once they get closer, they may interfere by merge or pushing each other apart. If one of the bumps receives more support from outside it may take dominance and inhibit all other bumps. Control of activity packets, i.e. translating them around the neuron grid, is discussed in Sec. 5.7.

The attractor dynamics (for grid cell-like firing behavior) could be achieved purely by appropriate recurrent connectivity, e.g. local and/or global, constant and/or dynamic, excitation and/or inhibition. Alternatively, the recurrent connectivity could be just inhibitory and combined with constant excitation from outside.[8,36] We choose the former option (with local excitation and inhibition), because we only want a single bump of activity per pose estimate.

Since we implement the neural substrate with Nengo, we first try a decoded connection from the pose ensemble to itself (cf. Conklin and Eliasmith[13]). We want integrator-like behavior, i.e. the longer some pose is input the stronger the respective neurons should respond (until saturated) and the more intense the pose estimate should show up in the decoded (reciprocal) pose volume. Looking at integration from a control theoretic perspective as a dynamical system, $\dot{\mathbf{x}}(t) = \mathbf{A}\mathbf{x}(x) + \mathbf{B}\mathbf{u}(t)$, with state $\mathbf{x}$ and input $\mathbf{u}$, we find that the system matrix $\mathbf{A}$ needs to be 0 and the control matrix $\mathbf{B}$ needs to be 1. Accounting for the neural dynamics and exponential synapses with time constant τ of the recurrent connection (see later in this section), we obtain 'neural forms' of the two matrices,* $\tilde{\mathbf{A}} = \tau\mathbf{A} + \mathbf{Id}$ and $\tilde{\mathbf{B}} = \tau\mathbf{B}$, which become

$$\tilde{\mathbf{A}} = \mathbf{Id} \tag{5.4}$$

and

$$\tilde{\mathbf{B}} = \tau \quad , \tag{5.5}$$

after inserting the values for $\mathbf{A}$ and $\mathbf{B}$. Therefore, according to Eq. 5.4, we specify the identity transform on the recurrent connection. Also, we configure the ensemble with parameters that are compatible with/recommended for Loihi†: intercepts 0 (makes sense for us because our dot products go from 0 to 1; recommended range for Loihi is $[-0.5, 0.5]$), nominal firing rate 120 (recommended for Loihi is $[100, 120]$). Note that we tested various intercept distributions and nominal firing rates. Based on the encoders and a set of evaluation points, Nengo then conveniently solves for the connection weights. We set the evaluation points equal to the encoders in order to evenly represent the unit domain volume. During development, we also experimented with piking evaluation points from the unit domain at random and optionally concatenating the encoders, but found no performance gains. The resulting outgoing recurrent connection weights per neuron feature local excitation centered at the neuron's grid position surrounded by local inhibition. Beyond, there is further excitation targeting more distant neurons. We found the system with raw Nengo weights to be non-functional for our purpose. The attractor network was not consistently able to properly sustain one or more bumps. Observed behavior includes divergence into large activity bubbles, divergence into random activity patterns, fading of activity bubbles, and emergence of new activity bubbles at random locations.

Since the desired behavior should theoretically be attainable with even fewer neurons, e.g. with a $9 \times 9 \times 9$ neuron grid, we looked for solutions at other parts of the system: neural parameters, connection parameters (e.g. synaptic time constant τ, scaling factors), neuron grid structure, simulation parameters, and the weights themselves. After a lot of trial and error we settled on the following major changes: domain and neuron grid from square to hexagonal (in the first two dimensions; see Secs. 5.1 and 5.3.1), periodic boundary conditions from square to rhombus (in the first two dimensions; see Secs. 5.1 and 5.4.1), and modified connection weights. We experimented with various modifications of the raw weights computed by

* Adapted from Conklin and Eliasmith[13] and https://www.nengo.ai/nengo-fpga/v0.2.2/examples/notebooks/03-integrator.html.

† https://www.nengo.ai/nengo-loihi/api.html#nengo_loihi.set_defaults

Nengo. For example, we added global inhibition. This did not yield a robust attractor. However, the speed of the previously mentioned divergence is influenced by this global inhibition component. Getting just the right amount of inhibition is difficult or impossible. Using too little results in divergence, albeit slower than without any, while too much results in any bumps' intensity converging to zero quickly. By manual search, we were unable to find a value that was 'just right' and would keep any bumps stable, even tough we determined the range of potential values, if any, down to 1e-4. Furthermore, adding such global inhibition directly violates the locality requirement as discussed below. Hence, we proceeded to test different excitation and inhibition strengths and 'ranges'.

We define the *weight function* as the multivariate function from which the connection weights are sampled, or which is fit to approximate existing connection weights. Thereby, distances between neurons are computed on the stacked toroidal manifold. Importantly, the neural connectivity needs to be localized, meaning that any neuron projects only to neurons in its immediate local neighborhood. Such requirement stems from Loihi's limited synaptic resources. These must be considered not only in total, but also per core. As mentioned in Sec. 3.2.1, the largest fully connected network that could be run on Loihi comprises about 300 neurons, which would be put onto a single core. For reference, our $12 \times 12 \times 12$ attractor network would require about three million ($\approx 12^6$) synapses for an all-to-all configuration, and that does not even include the synapses required for the shifting ensembles. Taking into account the splitting of the attractor ensemble onto cores, full connectivity is not compatible with Loihi. Localized connectivity patterns also seem to be more biologically realistic. Another benefit is that we can make use of sparse matrices to specify the weights.

We opt for a Mexican hat function. It is popular for computing connection weights for attractor networks and also biologically realistic (see Ch. 4). When plotted in two dimensions, it resembles a sombrero. It features center-surround excitation-inhibition. A neuron thus excites its immediate neighbors, the excitation strength being indirectly proportional to the distance between source and target neuron; inhibits neurons that are immediately outside the excitatory center; and does not influence neurons that are located further away. Hence, such weight function is particularly well suited for our use case since any neuron has purely local effect.

A Mexican hat function can be obtained as the difference of Gaussians.[38] Formally, let $G(\mathbf{x}, \mu, \Sigma)$ be the probability density function of a multivariate normal distribution, with the mean vector μ and the covariance matrix Σ. Our Mexican hat *weight function* can then be defined as

$$M(\mathbf{x}) = k_{exc} \cdot G(\mathbf{x}, \mu, \Sigma_{exc}) - k_{inh} \cdot G(\mathbf{x}, \mu, \Sigma_{inh}) \quad , \tag{5.6}$$

with its center μ, the scaling factor of the excitatory part k_{exc}, the covariance of the excitatory part Σ_{exc}, the scaling factor of the inhibitory part k_{inh}, and the covariance of the inhibitory part Σ_{inh}. Per dimension, Σ_{inh} needs to be larger than Σ_{exc}.

Intuitively, the excitatory center sustains activity packets and the surrounding inhibitory region prevents activity packets from spreading.[42] We refer to maximum the excitatory value at the center of the function as center amplitude; and to the lowest value in the inhibitory region as inhibition depth. The extent of various features of the weight function influences the final attractor behavior, especially after initialization/reset input is turned off, i.e. when it is 'free running'. By modifying the scaling factors and variances in Eq. 5.6 we can change the relative strength of excitation vs. inhibition as well as the width of excitation and inhibition.

In more detail, the width of the excitatory center determines how many neurons are involved in representing some pose estimate, where a greater width corresponds to more involved neurons. Note that it is difficult to predict the exact number of involved neurons from the weight curve, because it only tells the influence of some neuron to its neighboring neurons. These neighboring neurons, in turn, excite their neighbors. The effects of the inhibitory ring also need to be considered. Furthermore, the response curves influence the behavior during or immediately after reset input. Essentially, the full dynamics of the attractor network need to be computed, i.e. simulated, to obtain the final number of neurons involved in representing any individual bump. Still, we can argue that the excitatory width (together with the inhibitory width) of the weight curve determine a trade-off between general representational stability of individual pose estimates and how close two individual bumps can coexist without interfering, i.e. the maximal num-

ber of representable simultaneous pose estimates. When more neurons are involved, a pose estimate can be kept at locations that do not coincide with any neurons' preferred location for longer. In contrast, fewer neurons lead to a tendency of bumps to be drawn to and center at locations that correspond to a nearby neuron's tuned location. From the perspective of multiple simultaneous pose estimates, more involved neurons per bump mean that the corresponding activity packets are larger and interfere sooner, i.e. when the bump's centers are farther apart; compared to when fewer neurons are involved and the activity packets are smaller. When the inhibitory range/width is increased, the required spacing between simultaneous pose estimates increases.

The strengths of excitation and inhibition influence the inertia of the attractor network. What that means is a stronger excitatory center or a stronger surrounding inhibition cause an activity bump to be more 'locked in place', i.e. make it more difficult and slower to move it around. This makes the system appear more inert. Consequently, stronger shifting forces may be required to achieve desired translation/rotation velocities, or the highest possible velocity may be lower altogether (see Sec. 5.7). Far-reaching inhibition may have a similar effect of 'locking' bumps in place and making them harder to move. From another perspective, by controlling the parameters of the connection weight function we can make the system usable with a wider/different range of shifting parameters. We noticed a tendency that the bump moves faster for smaller center excitation and shallower surround inhibition, but that the trajectory gets more wiggly. Furthermore, using a stronger center excitation leads to longer required init/reset periods. Our goal is to get short init/reset times. Regarding the inhibition depth, apart from the 'locked in place' aspect, we need to consider the trade-off between crisp bumps and bumps slowing down stronger when approaching each other. Stronger inhibition makes bumps more defined but increases the slowing down-effect, and vice versa.

The widths and strengths of the excitatory and inhibitory parts of the weight function cannot be set arbitrarily, but need to lie within certain feasible ranges. All lower and upper bounds primarily depend on the shape (i.e. width and overlap) of the neural response curves (see Fig. 5.10). Specifically, there is potentially only a single parameter combination for Eq. 5.6 such that the resulting weights sustain activity packets exactly as initialized by some input corresponding to a pose estimate. This directly implies that deviating from this 'ideal' weight function changes the width of activity packets representing pose estimates, and, consequently, the variance of the decoded Gaussians. (Note that this variance is no perfect indication of uncertainty in pose estimates, see Sec. 5.2.) Apart from the specific shape of (stable) activity packets, the parameter ranges and their effect on the excitation and inhibition strengths and extents determine whether activity packets decay, split, are stationary, or diverge. A balance between excitation and inhibition is important.[64]

Now, we discuss the computation of the final weights. As it turns out, the connection weights computed by Nengo for the recurrent connection of the attractor ensemble roughly follow a Mexican hat function. However, as mentioned before, these weights do not yield usable attractor dynamics, whether we add a global inhibitory part or no. It is difficult or rather impossible to directly calculate the optimal weight shape. Therefore, we manually adjust the weights by modifying parameters of the Mexican hat function in an iterative fashion. Starting out from Nengo's raw weights provides the huge benefit of already being in the right ballpark scale-wise. Note that small changes of features of the weight curve could have large effects on the network dynamics. The relationship/influence is highly nonlinear, e.g. shifting the weight curve downwards by a constant amount could slightly decrease the activations of neurons in the center of a bump but increase activations of neurons that are slightly off-center in a bump. We tried many distinct weight parameterizations over countless experiments, going back and forth on individual parameter values. While the obtained parameters for the 'final' weights are most likely not optimal, they certainly yield good results. Specifically, we tuned until we got the following behavioral traits: keep an activity packet stable at any neuron's center indefinitely (without input); move activity packets along different directions that are not aligned to the grid (in addition, of course, to grid axis-aligned movement; see Sec. 5.7); sustain three bumps simultaneously. Note that any further tuning might result in overfitting to the particular random seeds used.

Primarily, we modify two qualitative aspects to achieve the desired attractor dynamics. Firstly, the amplitude at the center of the function is reduced.

This reduces the overall intensity of bumps and also reduces the force with which the activity tries to spread out from the bump center. Secondly, the depth of the inhibition ring is increased. This increases the resistance against the spread of activity. Taken together, these two modifications allow us to obtain connection weights that are usable with a wider range of values for the constant global inhibition. Also, we lower the center width/'variance' in order to be able to maintain more simultaneous bumps. The final parameters for Eq. 5.6 are listed in Tab. 5.2.

Table 5.2 Final parameters for computing the recurrent Mexican hat-type connection weights of the main attractor network. These values were obtained by manual tuning.

Variance of exc. Gaussian	k_{exc}	0.0108485147
Variance of inh. Gaussian	k_{inh}	0.01084861
Scaling of exc. Gaussian	Σ_{exc}	1.00115141
Scaling of inh. Gaussian	Σ_{inh}	1.001155

We depict the corresponding final Mexican hat-type weight function in Fig. 5.12 (orange curve). The original weight function obtained by fitting a Mexican hat function to slightly modified (see below) raw weights computed by Nengo is shown in blue. Conceptually, the actual three-dimensional connection weights are then sampled from our weight function with distances computed on the stacked toroidal manifold, i.e. respecting the periodic boundary conditions of Dom. In Fig. 5.13a, we show two-dimensional slices of the recurrent connection weights. The center-surround structure as well as the periodic boundary conditions are clearly visible.

Next, we discuss some implementation specifics. The connection weights computed by Nengo, when plotted in one dimension with the absolute distance between neurons on the x-axis and the respective weight on the y-axis, does not only feature an excitatory center and an adjacent inhibitory region, but additionally has excitation beyond that, i.e. possibly globally. We only need/want the non-global part as basis for our weights. Thus, we set any values beyond the inhibitory region to zero. (For completeness, note that we also tried shifting the weights downward before fitting the parameters to have a weaker excitation, a stronger inhibition, and zero effect beyond that; but this did not yield appropriate attractor dynamics.) Then we fit our generalized

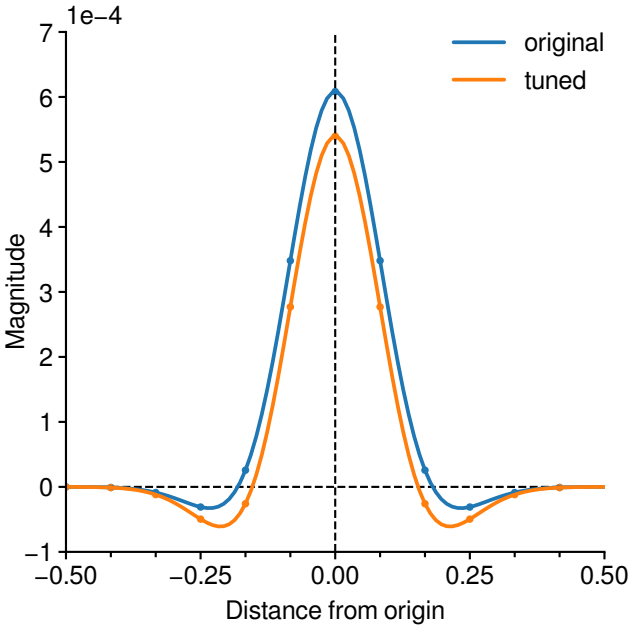

Figure 5.12 The three-dimensional recurrent connection weights are sampled from a univariate Mexican hat function, using the Euclidean distance between the two neuron endpoints as input argument. As a basis, a Mexican hat function is fitted to the recurrent connection weights computed by Nengo given our encoders (blue), where any values outside the inhibitory region are set to zero. The final system uses a hand-tuned version thereof (orange). The center magnitude was lowered, the width of the excitatory center at height 0 was reduced, and the depth of the inhibitory ring was increased. Both functions approach 0 for large absolute distances from the origin. The origin denotes the start point of the connection, and the x-axis represents the signed length of the distance vector in x-y-θ-space to the end point of the connection. This highlights the weight function's symmetry - we could just work with absolute distances and obtain the same result. For reference, the weights of connections to neurons along a grid axis are marked by dots; neuron 0 is taken as start of the connections and we wrap around at neuron 6.

Mexican hat function `mex_hat` to the data using SciPy's `curve_fit` * utility. The result is the blue curve in Fig. 5.12. We tune and modify the obtained parameters as described above, ending up with another set of parameters (see Tab. 5.2). Importantly, our final weight function, as shown in Fig. 5.12 (orange), is also zero beyond the inhibitory region. To get the actual weight matrix we need to evaluate the weight function for each neuron-to-neuron connection. This amounts to roughly three million connections, many of which will have zero synaptic weight.

* https://docs.scipy.org/doc/scipy/reference/generated/scipy.optimize.curve_fit.html

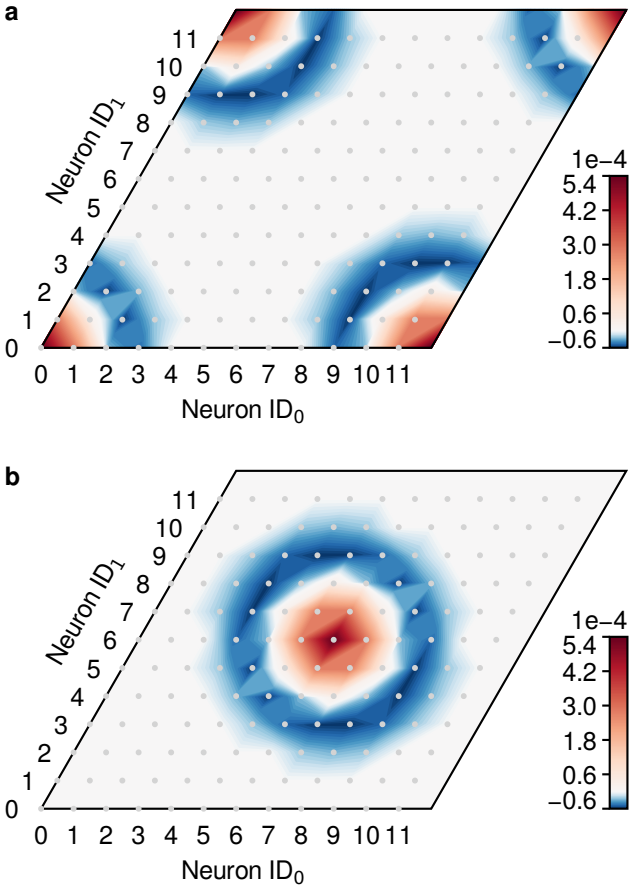

Figure 5.13 Recurrent connection weights plotted in the first two dimensions (position) of our unit domain. The axes labels denote the index in the neuron grid along the dimension indicated in subscript, e.g. ID_0 pertains to dimension 0. **a**, The recurrent weights from the neuron at the origin to all points, including the other neurons, within the $0°$-layer. These are obtained by sampling from the univariate Mexican hat function depicted in Fig. 5.12 (orange). A center surround excitatory-inhibitory structure centered at the source neuron is clearly visible in this 2D view. Remember that the extent of the domain along the first axis (x) is 1. Note that we actually show an interpolation of the discrete synaptic weights (2D slice) to better visualize the center-surround structure. Unfortunately, such interpolation introduces some visual artifacts. **b**, Recurrent weights from neuron $(6, 6, 0)$ to all other neurons in the same horizontal slice. These are essentially the same weights/values as in a, just shifted (with wraparound) to a different center.

Instead of individual evaluation for each connection, we devise a more efficient way of computing the weight matrix for the pose ensemble's recurrent connection. First, obtain a three-dimensional array of distances from neuron $(0, 0, 0)$ to all other neurons, respecting the wrapped domain. Then, apply our `mex_hat` function to this array to get the outgoing connection weights for neuron $(0, 0, 0)$. We transform these weights, which are essentially just a sampled function in the unit domain, to frequency space (as described in Sec. 5.2.4). Now, we can apply our `rotate_to_grid` function (see Sec. 5.3.2 for a description of that function) to efficiently compute the frequency-space representation of the outgoing weights for every neuron. Transforming the stack of three-dimensional arrays back to real space (in a vectorized fashion), we arrive at the final and complete weight matrix. To exploit the locality property of our weights and as a first step to fulfill Loihi's synaptic constraints, we convert the matrix to a sparse storage format (`scipy.sparse.csr_array`) before passing it to Nengo. At this point we realize that we need to further reduce the number of non-zero connections in order to fit our system (pose ensemble plus shifting networks) onto Loihi. To this end, we crop very small values in our weight matrix to zero *before* converting to the sparse representation. The tuning of the threshold for zeroing happened in conjunction to multiple other system parameters, such as the sampling/neuron grid size or the weight shape. The final threshold is $2.3 \cdot 10^{-5}$. Then, we construct a sparse matrix transform from the sparse weights with `nengo.Sparse`, and set it as the transform argument for the neuron-to-neuron recurrent connection on the pose ensemble. We found the synaptic time constant $\tau = 0.1$ to be a viable default. Larger values, such as 0.2 lead to smoother trajectory tracking but more inertia and smaller range of feasible input velocities; whereas smaller values make the system more volatile.

All details on the recurrent connectivity can be found in `run_experiment`, `get_pose_network`, and any functions called therein. See Fig. 5.1 for an overview of the final system, i.e. pose network, including details such as the functions computed per connection and synaptic time constants.

5.4.3 Constant Global Inhibition

To prevent the packets of activity from spreading to distant neighboring neurons, keep the population response from diverging, and contain random noise-induced bumps from their onset, a certain amount of global inhibition is required. Furthermore, we only want a single bump of activity per pose estimate in the population response at any time. This does not contradict the fact that one of our core contributions is the representation of mul-

tiple simultaneous pose estimates, since there we introduce one activity bump *per pose estimate*.

The 'right amount' of global inhibition is strongly dependent on the shape of the weight function. While the relationship is nonlinear, as a rule-of-thumb the center amplitude is directly proportional to the global inhibition strength, whereas the depth of the inhibition ring is indirectly proportional to it.

We distinguish between dynamic/feedback and constant global inhibition. The former is accomplished by a constant negative offset on the recurrent connection weights and, hence, is dependent on the individual neuron activations. As mentioned previously, this type of inhibition is not suited for our use case since it would require non-local recurrent weights. Another reason for implementing constant global inhibition instead of global feedback inhibition is necessary to ensure that the conditions to initialize a new bump at some location do not change depending on the number of currently active bumps. Also, if there was global feedback inhibition then the magnitude of stable bumps would decrease with an increasing number of bumps. Such normalization-like behavior is not required nor desired.

Constant global inhibition is independent of neural activity and can be *equivalently* achieved by either adding an inhibiting input connection, or by reducing the bias of the involved neurons. Since the former option involves a Nengo node outputting a constant negative value over connections to *all* neurons in the pose ensemble (and potentially also the shifting networks), it adds a lot of incoming connections to Loihi cores - too many, in fact, for our network size. We instead choose the second option of a negative offset on the neuron biases. Specifically, we reduce the bias from 1 to 0.968. This value was obtained by manual tuning. No additional resources like nodes or connections are needed. That seemingly simple way of implementing global inhibition actually required some sophisticated insight into Nengo's inner workings.

5.5 Initialization And Stationary Dynamics

How do we actually get a pose estimate 'into' the attractor network? And how does the resulting activity packet behave over time without velocity input? These are two important aspects we cover in this section. Before correct/intended path integration may be expected, we need properly initialized pose estimates (in stable state).

5.5.1 Single Pose Estimate

To initialize neural activity that represents some pose estimate P, we compute the (flattened) frequency space representation of pose $(0, 0, 0°)$ (i.e. a Gaussian centered at the origin and transformed to frequency space), shift it to P's location via multiplication with the appropriate rotation matrix, crop the shifted/rotated representation to Rep, normalize the vector to unit length, take the dot product with the encoders to get the target neuron activations, and finally input the target activation values via a node-to-neurons connection to the pose ensemble. This connection has synaptic time constant τ and also applies a scalar transform of τ. Importantly, the latter factor needs to be the same as the time constant on the *pose ensemble's recurrent connection* (see Sec. 5.4.2). This is because we want integrator-like behavior, i.e. the longer some pose is input the stronger the respective neurons should respond (until saturated) and the more intense the pose estimate should show up in the decoded (reciprocal) pose volume. In Sec. 5.4.2 we look at integration from a control theoretic perspective and treat our recurrent neural network as a dynamical system. From the resulting neural input matrix in Eq. 5.5 it is evident that we need to use a scalar transform of τ on the input connection to correctly approximate integrator dynamics. We choose a node-to-neuron connection from the input node to the pose ensemble because we want to have full and fine-grained control over the entire initialization/input process. Also, this is necessary to reduce the required synapse memory on Loihi. A regular decoded input connection would need 9300 input axons per core, with only 4096 available. Compare that to 2527 input axons needed when directly inputting to neurons. That is about 105 inputs per neuron saved. Further, the simulation seems to be significantly faster with that direct connection.

In biology, such initialization input might come from place cells, anchoring the grid cell activity to the environment.

The desired behavior is as follows. Prolonged input of some pose estimate should establish the corresponding activity bump as self-sustained, i.e. it should persist after the input is turned off and in the presence of constant global inhibition and potential small inhibition from other nearby bumps. There

is some intensity threshold that a bump needs to reach to be able to sustain itself. The integration dynamics with scaling factor τ, the inertia due to neural state dynamics and synaptic delays, and the mentioned inhibitory forces all imply that initialization input must be applied for more than one time step in order for a bump to reach this intensity threshold. We can vary the input frequency, e.g. every second time step, and the total number of time steps at which input is applied. These two variables determine how long it takes to reach the threshold. There is an inverse relationship between the two when it comes to minimum time-to-threshold, meaning that for a higher input frequency the number of input time steps can be smaller, whereas for a lower input frequency we need more total input time steps. The neural state and dynamics influence the minimum time-to-threshold and make its dependence on the two parameters nonlinear. For example, half the input frequency does not double the minimum time-to-threshold but might only increase it by 80 percent. One of our goals was to reduce the minimum time-to-threshold to a practical value. It is influenced, for example, by the excitatory strength of the recurrent weights. Eventually, initialization at maximum frequency (i.e. input at every consecutive time step) for $0.2\mathrm{s}$ worked reliably (for spawning a bump in near-silence state).

Note that the state of a bump at threshold is most likely one of lower intensity compared to the stable state. It will then gain in intensity after the input phase until it is stable. On the other hand, if a bump is initialized longer than needed, it might gain a bit of intensity over the stable state. We designed the system so that this gap cannot become too large before saturation of the neurons sets in. Overall, there is a continuous range of bump states which all converge to the stable state over time (without input). This range is called the *basin of attraction* of the stable state. We cover that in more detail later in the section. With all the nonlinear neural dynamics going on it is difficult to initialize a bump for exactly the right amount of time so that it is in a stable state at the time the input is turned off. We usually apply the input a bit longer than needed just to be sure. Also, it makes sense to include a short period of 'free running', i.e. no input, after initialization, so that the system can settle to a stable state.

In Fig. 5.14, we show the neural activations after initializing pose $(0.5, \sqrt{3}/6, 0°) = (4, 4, 0)_{\mathrm{g}}$ and letting the system settle to a stable state. Given the small number of neurons involved in representing a pose estimate and the discrete nature of simulating the dynamics, it is normal that the activity wiggles or 'pulsates' a bit around some theoretical stable state. We depict a snapshot of this pulsation. It resembles a sphere. Being centered at a neuron's position in the neuron grid, the actual shape in the first two dimensions is that of a hexagon. In the layers above and below the bubble's center it also has a hexagonal shape, but with the respective centers being more prominent. Of course, since the grid is hexagonal in the first two dimensions but has an orthogonal third dimension, the shape and behavior of the activity slightly differ from a sphere. An ellipsoid would be a better description. This transfers to the decoding.

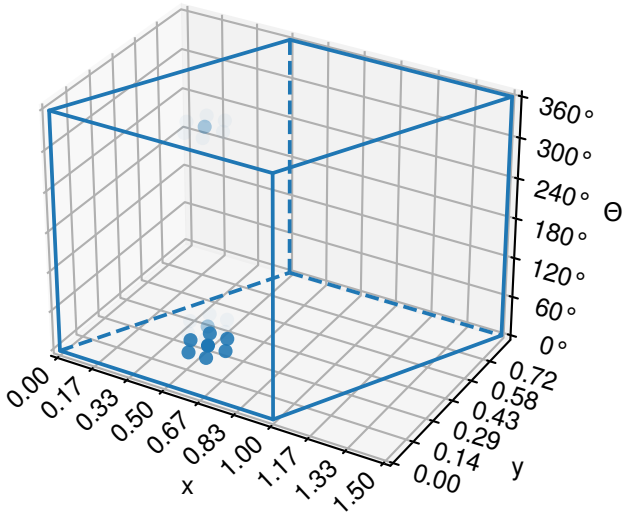

Figure 5.14 Stable neural activity when representing pose $(0.5, \sqrt{3}/6, 0°)$. Depicted is a snapshot of the slightly 'pulsating' population response activity around the theoretical stable state. The data points lie on the *neuron grid*. There is a resemblance to a sphere/ellipsoid, which manifests itself as a hexagonal slice through the bubbles center $(4, 4, 0)_{\mathrm{g}}$ and two hexagonal patterns above and below, both with a prominent center point. This highlights the hexagonal structure of the grid. Note that only the immediate neighbors of the bubble's center neuron are active in the hexagonal plane(s); and only the immediate and diagonal neighbors in the rectangular planes are active.

In the absence of any input, the activity packet should persist indefinitely. The single-neuron responses as well as the population response relative to the unit domain Dom form a hexagonal-orthogonal grid when unrolled according to the unit domain's boundary conditions. A horizontal slice thereof is shown in Fig. 5.11. Technically, the figure

depicts the raw current response, but it is qualitatively equivalent to the final responses in terms of firing locations (see also Sec. 5.4.1).

The corresponding three-dimensional Gaussian, after decoding the neural activity and applying the inverse Fourier transform, is depicted in Fig. 5.15. Again, the structure looks like a (larger) sphere/ellipsoid, but now in the sampling grid. Together with the faint halo around it, we recover a three-dimensional Gaussian. The decoded pose estimate matches the initialized one.

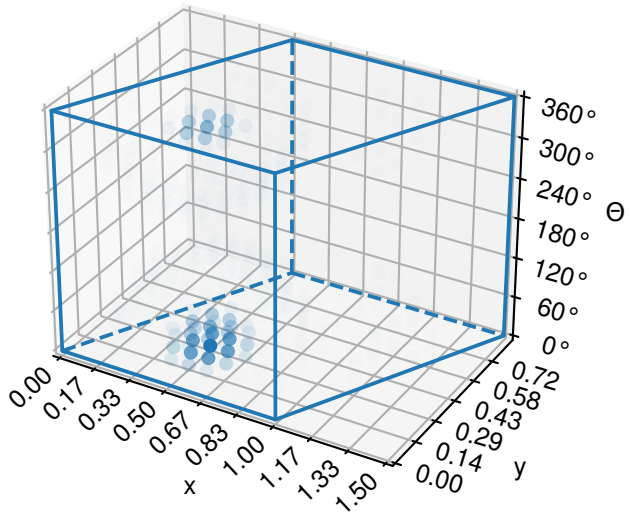

Figure 5.15 Three-dimensional Gaussian representing the pose estimate $(0.5, \sqrt{3}/6, 0°)$. Obtained by decoding the neural activity in Fig. 5.14 and applying the inverse Fourier transform. The data points lie on the *sampling grid*. There is a resemblance to a sphere/ellipsoid, i.e. circular shape in the first two dimensions and elliptical shape of vertical cross-sections. The hexagonal nature of the sampling grid is evident. Looking closely, there is a faint halo around the strong center bubble, which fits with a 3D Gaussian.

5.5.2 Multiple Pose Estimates

Now we consider multiple simultaneous pose estimates and corresponding activity packets. Such situation may e.g. arise in biology/operation following ambiguous visual input. We can establish multiple bumps either by initializing them simultaneously, i.e. superimposing multiple Gaussians when constructing the input data, or by initializing them one after the other. As long as the bumps are far enough apart, the dynamics do not depend on the number of bumps. This is because there is no global feedback inhibition.

The target behavior is that any activity bumps should persist indefinitely if they do not interfere with each other or are otherwise inhibited. If, however, two bumps happen to be close by, then they move towards each other (of course, if one bump is stronger it 'pulls' with greater force) and merge. Similarly, if a new pose/bump is input close to an existing bump, then the latter moves towards the former (and merges with it).

Then there is also the requirement that persistent input of some pose should not only initialize the corresponding activity packet, but also inhibit any existing bumps that are different from the 'new' bump. If the input and corresponding inhibition are applied long enough, all other bumps should drop below their self-sustain-threshold so that they vanish. Only one pose estimate/bump remains. This can be likened to visual reset in the mammalian spatial navigation system, where an existing estimate is corrected to some other location based on visual input. Path integration errors due to drift or temporary visual ambiguity can be resolved in this way.

We accomplish such behavior by adding an inhibitory component to the input procedure. Specifically, after the target neuron activations have been computed from the desired pose(s)/bump(s), we reduce all values below 0.01 by 0.08, i.e. setting them to approximately -0.08. All other values remain unchanged. The resulting target activations are then directly fed to the pose ensemble, resulting in inhibition outside the desired (new) bump location(s). Applying the input for more time steps causes stronger inhibition. This method of inhibition/reset is rather clever as it requires no additional resources. The recurrent connection weight matrix can be kept sparse.

Just as with initialization itself, the inhibition is influenced by the frequency and number of time steps at which input is applied. For example, lower frequency means other bumps vanish slower, and vice versa. The relationship is nonlinear due to the neuron dynamics and synaptic delay. When we inhibit every second time step, then the neurons might still have some state so that the vanishing does not occur at exactly 0.5 the speed. So, the time it takes for the other bumps to fall below their self-sustain-threshold depends on the frequency and strength of the reset input; once the threshold is reached the vanishing may progress quickly. Also, considering this new inhibitory aspect, to get additional

bumps without destroying existing ones, we need to apply new input just long/often enough to initialize a self-sustained bump but without inhibiting the existing bumps below their threshold activity. When movement is involved (see Sec. 5.7), for an initialization/reset to be effective it needs to be consistent with the movement. This means that some bump can only prevail over others if it is strengthened over time via persistent input matching its time-varying location. Spurious input or input that does not account for the changing locations of pose estimates under movement do not influence the attractor lastingly; they are essentially filtered by the attractor dynamics and neural dynamics. Relating that to the visual reset in biology/path integration, when moving through an environment the visual inputs vary and so do the associated reset stimuli - they are synchronized to the movement.

5.5.3 Attractor Dynamics and Continuity

Any involved parameters influence the attractor dynamics. Examples are the balance between excitatory strength and width and inhibitory strength and width in the recurrent connectivity; the constant global inhibition; the neural model and associated parameters (we use Leaky Integrate-and-Fire (LIF); see Sec. 5.3.3); or the synaptic time constant. For example, to get stable behavior with a smaller time constant on the recurrent connection, we need proportionally more neurons.* From the previous section it is evident that the system's behavior, e.g. its response to some input, also depends on the current state.

The only dynamically stable activity states of the network are: no activity (near-silence), a Gaussian-like bubble, or multiple sufficiently spaced-out bubbles. Due to the constant global inhibition, a random activity state is not guaranteed to approach any stable state other than near-silence. Ideally, there are infinitely many such states, to allow stable representation of pose estimates in the continuous three-dimensional domain. In other words, the pose corresponding to the stable state is the same as the poses corresponding to the states in the stable state's basin of attraction. In a true continuous attractor this would be the case. However, with our discrete set of spiking neurons and fixed connection weights it is impossible to achieve. Realistically, the stable states correspond to bubbles centered at neurons' preferred positions, i.e. the set of stable states is finite. Over time pose estimates are pulled towards the pose associated with the nearest neuron. To get good/smooth trajectory tracking this process may not be too fast. We could slow it by adapting various parameters, most importantly by increasing the number of neurons involved in representing any single pose estimate.

We inspect the internals of Nengo, such as voltage potentials, spike trains, inputs, outputs, and decoders/weights, to verify correct (stationary) behavior of the pose network. An important detail here is that NengoLoihi (and Loihi, for that matter) only supports an initial voltage of zero for the LIF neurons.† Consequently, all neurons start at the same state and the behavior at the beginning of a simulation might look deterministic. This quirk is one reason the simulations with Nengo's default backend might differ slightly from simulations with the NengoLoihi backend. By default, Nengo initializes neuron voltages randomly between 0 and 1.

Lastly, a note on reproducibility. For development and debugging it is crucial to get consistent behavior across simulation runs. Many parameters of Nengo objects will be randomly assigned during build time, e.g. encoders and evaluation points (decoders). We not only set a top-level network seed, but also set a seed per ensemble and per connection to 'control the randomness'. Otherwise, the 'random' parameters might change when adding or removing network elements.‡

5.6 Readout of Pose Estimates

We need to be able to read out the pose estimates represented by the population of grid cells in the main attractor network at any time. In biology, this potentially happens via place cells.[67] Some engineering works propose readout via a black-box neural network[38], and others address the problem from a theoretical and mathematical perspective.[58]

We set out to devise an algorithmic readout of the neural activity in the pose ensemble. This pertains to obtaining all pose estimates represented at any time. Specifically, since we represent a pose estimate as a Gaussian bubble in the domain Dom,

* https://forum.nengo.ai/t/encoders-and-stable-dynamics-for-a-head-direction-network/1253/2

† https://www.nengo.ai/nengo-loihi/api.html#nengo_loihi.set_defaults

‡ https://forum.nengo.ai/t/how-to-directly-access-and-modify-the-connection-weight-matrix-for-a-nengo-connection/256/2

the goal of the readout is to find the most likely centers of the bubbles corresponding to the pose estimates at any time. Note that we exclusively work with a single spatial scale, as defined by the mapping Dom to physical space. The problem of combining multiple scales akin to the modular organization of grid cells with increasing spatial scale along the dorsoventral axis of the medial entorhinal cortex is left for future work (see Ch. 8).

Various approaches are considered and implemented. To get started, we can use the maximally activated neuron's associated pose as readout of the current estimate. This results in jumpy behavior of the pose readout over time due to the possible values being constrained to the discrete neuron grid. Also, it does not leverage the power of population coding, i.e. the power to smoothly represent values between the neurons' preferred poses. Hence, this approach is obviously not as accurate as possible. Further, only a single pose estimate is obtained for any time point, specifically the one with the highest single-neuron firing rate in its associated activity packet. However, there are potentially multiple simultaneous poses represented by our system.

Another class of approaches first decodes the population activity using the decoder weights computed by Nengo and then works with the pose volume in the reciprocal space Rep. One option is to use SciPy's `curve_fit` function to fit a 3D Gaussian bubble to the pose volume. This fitting can either be done directly on that volume, or after applying the inverse Fourier transform, i.e. in time space. To accommodate multiple simultaneous bumps, an auxiliary function representing the sum of Gaussians is defined. It takes as parameters the mean, variance and scaling factor of each Gaussian it represents. Hence, the maximum number of pose estimates that can be captured is hard coded. The auxiliary function is then passed as argument to `curve_fit`. By allowing the scaling factor to be 0 we effectively support a variable number of bumps (up to the hard limit). As indicated in Sec. 5.2.2, computing the wrapped representation of an arbitrarily-centered Gaussians in time space is rather cumbersome; it is less intuitive and seems to be slower than computing an origin-centered Gaussian in time space and doing everything else in frequency space. So, if Gaussians centered at a different location were required, the origin-Gaussian had to be transformed to the frequency space, rotated to the corresponding location therein, and potentially back-transformed into time space. Therefore, it becomes clear that talking of fitting a sum of multiple Gaussians with different centers within Dom or Rep introduces a certain level of complexity. Back to the readout: The approach via fitting a function to the decoded volume has the advantage of significantly smoother and less noisy pose estimates. Unfortunately, it is very computationally intensive and slow, which stems mainly from `curve_fit` and the large number of parameters it is supposed to determine. As a simplification we can fix the variance to the value that is also being used to compute the input Gaussians in the first place. We found it most efficient to fit the Gaussians' rotation parameters and scale factors in frequency space. Regarding `curve_fit`, there are some options pertaining to the fitting resolution that sometimes influence the computation speed. For example, `ftol` specifies the 'tolerance for termination by the change of the cost function'.* Lowering it from the default 10^{-8} to $5 \cdot 10^{-3}$ significantly speeds up the entire process with almost the same results.

An idea of further improving the readout via fitting sums of 3D Gaussians to the pose volume is to provide better initial guesses for the centers, as opposed to just using the origin for each bump. Such approximate center estimates are computed via clustering. First, localized clusters of sufficiently active neurons are identified within the pose volume. Then, the most active neuron of each cluster can be used as center estimates for `curve_fit`. While this approach yet again improved on the computation speed of the previous one, it was still deemed not efficient enough - even when `ftol` is increased.

The final approach is to combine clustering with an approximation of the multi-Gaussian fitting. It works on the decoded and inverse Fourier-transformed pose volume. We use the DBSCAN algorithm, which stands for Density-Based Spatial Clustering of Applications with Noise and is very fast to perform. Essentially, it finds seed samples of high density in the to-be-clustered point volume and expands clusters from them. The big advantage of this clustering algorithm is that we can provide a precomputed distance matrix to use as metric, and we can construct this matrix such that the periodic boundaries of our 3D sampling grid are respected.

* https://docs.scipy.org/doc/scipy-1.9.3/reference/generated/scipy.optimize.least_squares.html

We further provide a matrix of sample weights to the algorithm, which, in our case, correspond to their respective value in the pose volume. After running DBSCAN, we compute the weighted positional average per cluster in time space. This again requires proper handling of the periodic boundaries of our domain Dom (see Sec. 5.7.3). There are two parameter values that influence the quality and behavior of the clustering. First, we threshold the sample points in the pose volume before clustering, essentially setting a minimum value that a point needs to have in order to be considered for clustering in the first place. We choose this value as 0.4. Second, DBSCAN has an ϵ parameter, which defines the maximum distance between two samples such that they are considered to be in each other's neighborhood. We found $\epsilon = 1/12 \cdot 1.1$ to work well, where 1/12 represents the distance between neighboring points in the sampling grid (assuming it to be uniform along all axes). The threshold and ϵ determine how close two bumps can get before the corresponding clusters merge and are returned as one cluster by DBSCAN. Clustering plus approximation of fitting multiple simultaneous Gaussians turned out to work really well and is also computationally efficient.

Regarding the system architecture some modifications on the output-related parts were necessary to make it work with Loihi. Initially, we used a decoded connection from the main attractor network to the output node. However, this required too many output axons per block/core, 7924 to be specific, whereas Loihi only has 4096. We changed that connection to a non-decoded one, i.e. feeding the raw neuron activities to the output node. This reduces the number of required output axons per block to 1331 and, thus, fits Loihi's constrained resources per core. In general, one would want to minimize the communication between off-chip components and on-chip components, because this increases power efficiency.* An additional benefit of directly outputting neuron activations is that the simulation is significantly faster. Performing the decoding after the simulation is straightforward.

5.7 Translation and Rotation

The pose estimates need to be shifted around in the unit domain Dom according to the actual movement in real space. When poses are translated through Dom, their representing 3D Gaussians are also translated through Dom. Since the Fourier transform is linear, such translation similarly occurs in reciprocal space or Rep. Ideally, the corresponding activity packets/bubbles in the neuron space, initialized via the encoders and sustained by the attractor dynamics, behave equivalently. Therefore, we need to be able to stably and smoothly shift neural activity around.

We eventually incorporate or *integrate* tangential and angular velocity inputs via distinct ensembles of *conjunctive grid-by-direction-by-speed cells*, which are modulated by velocity and project back to the pose ensemble. It is plausible that the mammalian spatial navigation system performs path integration in a similar way[42,54] (see also Ch. 4).

5.7.1 Include Translation in Attractor Network

Initially, we incorporate the representation of the current pose estimate, i.e. attractor dynamics, and the velocity-modulated translation of that estimate into the same network of neurons, inspired by Fieweger[22] and Conklin and Eliasmith[13]. The recurrent connection is realized by specifying the desired function that the connection is to compute, and Nengo then solves for the decoders, i.e. the weight matrix. We define this function using the rotation matrices from Sec. 5.2.6 and the current tangential and angular velocities. We determine three orthogonal velocity components along the dimensions x, y, and θ in reciprocal space. Then, we can use three precomputed rotation matrices which rotate by a fixed angle/shift by a fixed amount along either dimension, scale these matrices by the respective velocity component, and apply them successively to the current encoding space. Formally, let $\mathbf{x}$ be the current state, v_a the current velocity component along dimension $a \in \{x, y, \theta\}$, and $\tilde{\mathbf{R}}_b$ a fixed rotation along dimension $b \in \{x, y, \theta\}$, then the next state is computed as

$$\hat{\mathbf{x}} = (v_\theta \tilde{\mathbf{R}}_\theta) \cdot (v_y \tilde{\mathbf{R}}_y) \cdot (v_x \tilde{\mathbf{R}}_x) \cdot \mathbf{x} \quad . \tag{5.7}$$

This gives a good approximation to the true effect of the tangential and angular velocities in/after the current time step. Further, this approach is significantly more efficient than to recompute two or three time step-specific rotation matrices for every function evaluation. The standard attractor dynamics

* https://forum.nengo.ai/t/nengo-loihi-computable-functions-and-spiking-behaviour/1646

are included via the special case of all three velocity components being zero; the function then reduces to an identity transform.

Importantly, when a Nengo connection function should nonlinearly combine multiple input dimension, all required information needs to be stored in the same ensemble. To this end, we include the velocity components as additional representation dimensions in the attractor ensemble. The encoders per neuron are extended by three elements that are randomly chosen as -1 or 1 each. Thus, any neuron is sensitive to negative/positive velocity along each dimension of the reciprocal space, which makes the velocity information available for the recurrent connection. While such alteration of the representational space does interfere with the pure attractor dynamics and associated properties, like perfect local center-surround excitatory-inhibitory connectivity, Nengo should ideally still be able to determine suitable weights. These potentially resemble some center-surround-like pattern. The new weights are responsible for attractor dynamics as well as shifting the neural activity/representation based on outside velocity input. One could say that, using NEF, we piggyback translation/rotation onto the standard attractor dynamics. An advantage of such approach is that it requires fewer neurons and fewer synapses.

Unfortunately, the resulting system was not able to properly represent pose estimates nor capture their translation/rotation. A possible explanation is that the decoded connection function with three rotations in high-dimensional representational space (about 1700 dimensions) is too complex to be approximated by our limited number of neurons (per horizontal slice) in the 3D pose ensemble. Since we cannot increase the number of neurons, other architectures need to be devised.

One alternative that we briefly considered involves patches of neurons with different directional preferences and correspondingly offset outgoing recurrent weights per horizontal layer of the neural sheet. A neuron receives excitatory input proportional to how much the current velocity vector aligns with the neuron's tuning direction. The skewed weights then shift activity into that direction. Individual velocity modulation per neuron is difficult to achieve neuromorphically; we might need Nengo nodes for that. Each patch contains neurons of every possible directional preference out of a predefined set, e.g. $[0°, 90°, 180°, 270°]$. Neurons of different tuning can be 'assigned' to the same location or to different locations. The horizontal layers are evenly tiled with these patches. When activated equally, neurons with opposing tuning cancel each other's shifting force, resulting in standard attractor-like dynamics.[25] If the current velocity vector points in a specific direction, neurons with aligned directional preferences are activated more that the opposing neurons, resulting in a shifting of the neural activity into the velocity direction. Neurons with tunings that are orthogonal to the velocity direction are still equally activated and oppose each other. Works that deal with that kind of translation mechanism include Burak and Fiete[10] (see also Sec. 4.12) and Krishna et al.[38] (see also Sec. 4.4). Eventually, we decide against this approach due to the difficult selective velocity modulation (at least, neuromorphically), and the potentially large neuron numbers required to get stable and smooth behavior.

5.7.2 Distinct Shifting Ensembles

Our final approach to translation is via distinct neural populations. We separate the representation of poses, via attractor dynamics, from their velocity input-based translation and rotation.

To this end, we leave the attractor network unchanged and introduce three new neural ensembles, so-called shifting ensembles/networks. One is responsible for translation, one for counterclockwise rotation, and the third for clockwise rotation. Each has the same neuron grid and neural tuning (encoders) as the attractor network. They receive input directly from the attractor network using the same connection matrix as is used for the attractor's recurrent connection. Intuitively, the attractor network's neural activity is replicated in each of the shifting ensembles. Instead of connecting these ensembles recurrently, however, we connect them back to the attractor network. In fact, the shifting networks only differ from each other in their outgoing connectivity.

Here is where the actual shifting happens. It is best explained by an example. Assume only one generic shifting ensemble for now. So, on the one hand we have the pose ensemble where the attractor dynamics, brought about by appropriately tuned recurrent connections with a local center-surround excitatory-inhibitory structure, sustain activity bubbles corresponding to (Gaussian) pose estimates. And, on the other hand, there is the shifting ensemble. Neglecting synaptic delay and neural inertia,

the connections from the attractor cause the activity in the shifting ensemble to match the one of the attractor network. The shifting ensemble's outgoing connection weights might e.g. resemble an offset version of the attractor's recurrent weights. They would then try to shift neural activity into the direction of the offset, without destroying the activity bubbles. Since the connection goes from shifting ensemble to attractor ensemble and their activity states are ideally equal, the attractor's activity is effectively shifted into the offset direction. In the next time step, the shifting ensemble's activity is updated to match the shifted activity, and so on.

Translation. We first discuss the shifting ensemble responsible for translation, called the translation module. It is supposed to shift the attractor's activity according to the directional tuning per x-y layer. Remember that each horizontal layer of neurons is associated with a particular heading orientation. For example, the pose estimate $(0, 0, 45°)$ needs to be shifted differently than $(0, 0, 70°)$. Consequently, the outgoing weights of each 2D horizontal slice of the translation module shifts activity in the corresponding slice of the attractor ensemble into the preferred direction associated with that slice.

We compute the required connection weights by shifting the tuned Mexican hat-type weights (of the attractor's recurrent connection) in frequency space and subtracting (i.e. canceling out) the original recurrent attractor weights. This is implemented in the `get_shift_con_weights` function. Specifically, we start out from the frequency space representation of the outgoing weights for every neuron in the pose ensemble (as derived in Sec. 5.4.2), which effectively is a three-dimensional array (source neuron) of three-dimensional arrays (target neuron) of scalars. For each horizontal neuron layer, we compute a rotation matrix that rotates $0.25 \cdot 1/12$ into the layer's preferred direction. Per layer, we apply this matrix to the corresponding weights, using vectorization and some array reshaping tricks. Then, we transform the values back to direct space. At this point, the outgoing weights of each neuron resemble a center-shifted Mexican hat function, where the direction of shift corresponds to the neuron's θ-coordinate in the grid. Finally, we subtract the recurrent weights of the attractor from the shifted weights. This is an ingenious design choice and crucial to maintaining approximate attractor dynamics in the pose ensemble despite the presence of the shifting input. Since the incoming connections roughly add up in the attractor network, and the respective weights (recurrent and shifting) operate roughly on the same neural activity (not exactly due to synaptic and neural dynamics in between), the effective weights operating on the activity in the main attractor network approximately correspond to the center-shifted Mexican hat-type weights. No 'extra' activity is introduced.

In Fig. 5.16a, we depict a horizontal slice at height $\theta = 0°$ of the final outgoing weights of neuron $(6, 6, 0)$ of the translation module. There is an excitatory region in the shifting direction $(0°)$, and an inhibitory region in the opposite direction. When we re-add the recurrent attractor weights, we get the center-shifted Mexican-hat type weights shown in Fig. 5.16b. Thus, neuron $(6, 6, 0)$ effectively results in skewed attractor dynamics. Of course, this is a simplification. There are neural dynamics in both ensembles, synaptic delays of the connections to and from the translation ensemble, and potentially additional influence from one rotation ensemble (as discussed later in this section). Still, the conceptual summation of the incoming weights of the attractor works.

In general, a single pose estimate/activity bubble comprises neurons in multiple θ-layers. According to the connectivity described above, each slice of an activity packet is propagated in the associated preferred direction. Viewed in isolation, this seems to dissect packets of activity instead of propagating them as a whole. However, the attractor dynamics are operating in parallel and, when balanced properly, keep the activity packets together. Ultimately, it enables choosing the movement direction from a more continuous spectrum.

Importantly, the specific setup and connectivity of the translation module allows for different pose estimates to be translated into distinct directions, independent of each other (see Sec. 5.8).

Rotation. Next, we introduce the shifting ensembles responsible for rotation, called rotation modules. They are simpler than the translation module. The counterclockwise rotation module is supposed to shift the attractor's activity upwards along the θ-dimension; whereas the clockwise rotation module shifts it downwards. So, while the translation module operates intra-neuron layer, the rotation modules operate inter-neuron layers.

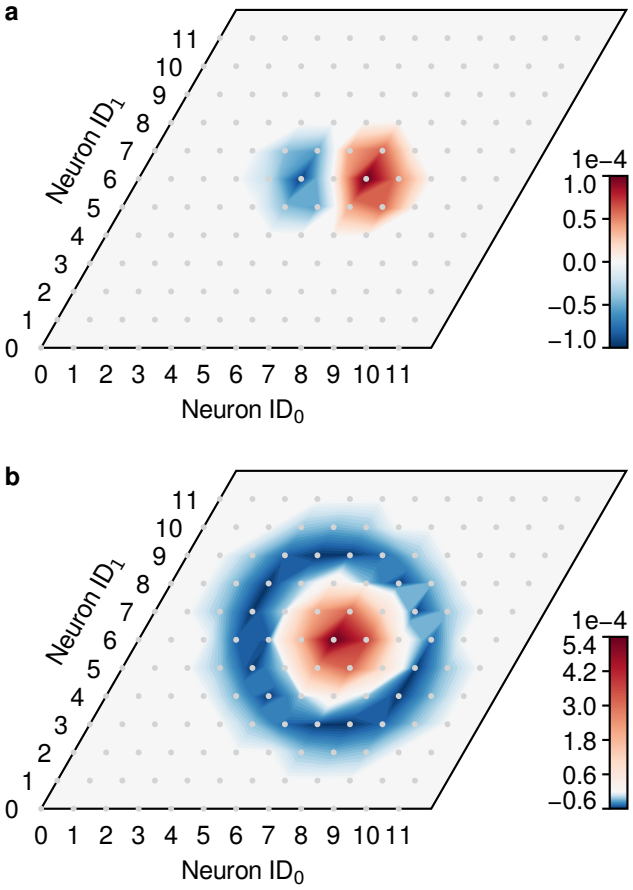

Figure 5.16 Outgoing connectivity of the translation module and the effect on the attractor dynamics in the pose ensemble. The axes labels denote the index in the neuron grid along the dimension indicated in subscript, e.g. ID_0 pertains to dimension 0. Both colorbars are mapped symmetrically around 0, but their scales differ. The line artifacts stem from interpolation in combination with the sample points being specified on a hexagonal grid. **a**, Outgoing connection weights from neuron $(6, 6, 0)$ in the translation ensemble to the $\theta = 0°$ neuron layer ($\mathrm{ID}_2 = 0$) in the attractor ensemble. There is excitation to the right of neuron $(6, 6, 0)$'s position, i.e. in the shifting direction, and inhibition on the other side. The two regions appear flattened where they meet in the middle. **b**, Effective weights operating on the neural activity of the attractor ensemble. Shown is a horizontal slice of the neuron grid at height $\theta = 0°$ ($(\mathrm{ID}_2 = 0)$). These effective weights are obtained by adding the corresponding weights from the translation module to the attractor (shown in a) and the corresponding recurrent weights of the attractor (shown in Fig. 5.13b) together. The weights follow a center-surround excitation-inhibition pattern, which is offset to the right compared to Fig. 5.13b. For clarity, we neglect the rotation ensembles' effect.

The required weights are computed similarly to the translation weights, i.e. by shifting the frequency space representation of the attractor network's recurrent Mexican hat-type weights around, and subtracting the original values (see `get_rot_con_weights`). For the counterclockwise rotation weights, we compute a single rotation matrix that rotates $0.25 \cdot 1/12$ along the positive θ-direction. Using vectorization and reshaping, we apply this rotation efficiently to the frequency space representation of the entire outgoing weights for every neuron in the pose ensemble. After back-transforming to direct space, we subtract the recurrent attractor weights from the shifted weights. Equivalently, we obtain the clockwise rotation weights, just with rotation along the negative θ-direction.

A vertical slice at $y = 0$ of the final outgoing weights of neuron $(6, 0, 6)$ of the counterclockwise rotation module is plotted in Fig. 5.17. There is upward excitation and downward inhibition. The corresponding weights of the clockwise rotation module are essentially flipped upside down. If we were to re-add the recurrent attractor weights, the result would again be center-shifted Mexican hat-type weights.

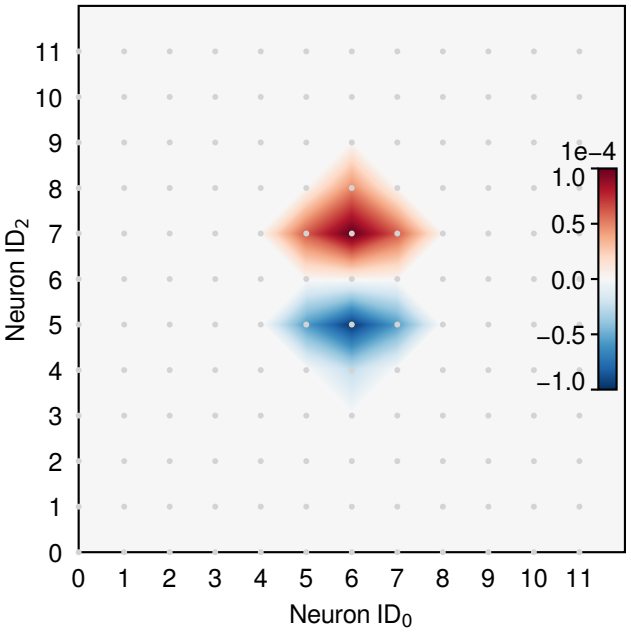

Figure 5.17 Outgoing connection weights from neuron $(6, 0, 6)$ in the counterclockwise rotation module to the vertical neuron layer with $y = 0$ ($\mathrm{ID}_1 = 0$) in attractor ensemble. There is excitation above neuron $(6, 0, 6)$'s position, i.e. in the direction of rotation, and inhibition below. The axes labels denote the index in the neuron grid along the dimension indicated in subscript, e.g. ID_0 pertains to dimension 0. The colorbar is mapped symmetrically around 0. The line artifacts stem from interpolation in combination with the sample points being specified on a hexagonal grid.

Interestingly, we found that using decoded connections from the shifting ensembles to the pose ensemble, specifying the connection functions with rotation matrices (or, equivalently, a set of evaluation points), and letting Nengo determine the weights, results in exactly the same weights that we present above (the re-added versions). This implicitly verifies our shifting weights.

We also tried a scaled sparse identity connection from the main attractor to the shifting ensembles, with the intention to reduce the number of connections, specifically, the number of output axons. Occasionally, this would perform similar. At other times, it exhibits drift behavior. Overall, it seems to be worse (more noise, less stable) than the 'full' connection. Moreover, it might be interesting to try shifting weights that have been constructed from Mexican hat-type base weights with different parameters than the attractor's recurrent weights.

5.7.3 Velocity-Modulated Inhibition

Based on the magnitude of the tangential v $(\mathrm{u/s})$ and angular velocity inputs ω $(\mathrm{rad/s})$, we want the pose estimates to be translated/rotated at different speeds. To achieve this, we inhibit the shifting ensembles indirectly proportional to the relevant velocity input, which is how the term 'velocity modulation' comes about. Take the translation module, for example. No inhibition corresponds to maximum translation speed; full inhibition corresponds to no translation, i.e. no movement in the first two dimensions. It works equivalently for the rotation modules. We also speak of disinhibition of the shifting ensembles: higher shifting speed is achieved with greater disinhibition.

Our inhibitory modulation of the shifting ensembles' neural activities allows the system to smoothly track varying tangential and angular velocities, while additionally keeping the required number of neurons reasonably low. Essentially, adding the velocity aspect makes the neurons of the shifting ensembles behave like *conjunctive grid-by-direction-by-speed cells*, compared to the conjunctive grid-by-direction cells in the main attractor network.

The incorporation of velocity input completes our neural path integration system. A schematic of the final architecture, i.e. pose network, is shown in Fig. 5.1.

Implementation-wise, a Nengo node receives the tangential and angular velocities, and is connected to the neurons of each shifting ensemble. So, there is an additional input connection per shifting ensemble apart from the initialization connection from the attractor ensemble. The inhibition connection functions differ per ensemble; each neuron per ensemble receives the same input. The two rotation modules must operate mutually exclusive, i.e. at any time, at least one of them is fully inhibited. These inputs amount to some direct current input less than or equal to zero. Intuitively, we can treat them as a negative offset of the neurons' bias current, but additionally scaled by the neurons' gains.

As to the feasible value range of the inhibition input it can be roughly bounded based on the operating currents of the shifting ensembles' neurons. Clearly, one bound is 0, at which we attain the highest shifting speed. Considering that (a) 1 is the largest nominal value to expect as a regular raw current input, i.e. the value after the dot product with the neuron's encoder (note that this value is multiplied by the gain before being fed to the neuron); and (b) a change of -0.032 on the neuron bias current already has a noticeable and stabilizing effect on the attractor dynamics (see Sec. 5.4.3), a direct current input of -1 would definitely result in full inhibition of the shifting ensembles. Therefore, we start out with -1 for the other bound, corresponding no movement.

Velocity Calibration. Now, we need to find the *true* bounds for the inhibition input, which essentially means that varying it outside does have the same effect as setting it to the closest true bound. The true value range depends on the attractor dynamics, the spiking neural dynamics, synaptic delays, and the self-sustain-threshold. Further, these factors make the relationships between velocity input, inhibitory strength, and actual movement speed of activity packets nonlinear. Overall, it is very difficult to compute an exact relation that maps velocity inputs to the 'correct' inhibition values. Therefore, we perform a so-called velocity calibration to empirically determine such mapping (and inhibition bounds).

First, we determine a tighter bound on the inhibition values. Since we exclude 'negative inhibition', i.e. excitation, one bound remains at 0. When considering any shifting ensemble in isolation, any inhibition input equal to or smaller than -0.07 results in sufficient inhibition of the shifting ensemble for it not to move any pose estimates in the pose ensemble. It does not necessarily mean that there is no

effect on the attractor's neural activity at all. Also, when additionally considering the respective other two shifting ensembles, the combined effect may be large enough to influence the pose estimates (even though one shifting ensemble is inhibited with -0.07). Still, defining a feasible range of $[-0.07, 0]$ for the inhibitory input turns out to work well for velocity calibration.

Next, we generate data about our system's behavior with a single pose estimate for various combinations of translation inhibition i_{T} and counterclockwise (CCW) rotation inhibition values i_{CCW}. Thus, we directly specify constant inhibition values as input for the Nengo simulation. Note that $i_{\mathrm{T}} = a$ means an inhibitory input of $-a$ is applied. Specifically, we run a $5\mathrm{s}$ simulation for each combination of $i_{\mathrm{T}} \in \{0, 0.01, 0.02, 0.03, 0.04, 0.05, 0.06, 0.07\}$ and $i_{\mathrm{CCW}} \in \{0, 0.01, 0.02, 0.03, 0.04, 0.05, 0.06, 0.07\}$. i_{CW} is set to 1. This results in left turns of different radii. The system behaves perfectly symmetric for right turns. The system's trajectories, i.e. the trajectories of the represented pose estimate, for some of the input combinations can be seen in Fig. 6.4. We then manually fit circumcircular arcs to the trajectories and determine their radii and lengths relative to the unit domain length u. These values allow to compute the tangential and angular velocities with respect to the unit domain. Additionally, we plot the orientation angle over time and determine the angular change over the simulation duration, leaving us with a second estimate for the velocities. Averaging over the two estimates for tangential and angular velocities (per input combination) results in the values listed in Tab. A.1.

We need to turn these values into functional relationships. Specifically, since we eventually want to input time-varying velocities to the path integration system, we need to determine mappings from the velocity tuples to the required inhibition values. To this end, we define parameterized bivariate functions, fit them to the raw data, and evaluate how well the data is captured. Numerous nonlinear functions were considered and tested, including quadratic functions, third order functions, and ones where there is nonlinear interaction of the two velocity inputs (e.g. concatenations of different exponential functions with quadratic and fourth order functions). We found that the data does not exhibit nonlinear interaction between the two input arguments, and that more complex/higher order functions tend to overfit the data. Still, the inhibition values each depend on *both* the tangential and angular velocity inputs. This is because if one of the two inhibition values operates near its to-be-determined lower limit, i.e. the respective shifting ensemble exerts maximal force on the attractor ensemble, then the effect of the other shifting ensemble is diminished. Similarly, if one of the shifting ensembles is barely active, the other shifting ensemble's effect on the attractor ensemble takes dominance. For example, a maximally active translation module together with a barely active CCW rotation module results in pure translation of pose estimates in the pose ensemble; whereas an inactive translation module together with a barely active CCW rotation module results in pure counterclockwise rotation of pose estimates. Overall, the third order bivariate polynomial

$$f_{vi}(v, \omega) = av^3 + bv^2 + cv + d + e\omega^3 + f\omega^2 + g\omega, \quad (5.8)$$

with the fitted coefficient values from Tab. 5.3 worked best. Note the separate coefficient sets for computing the tangential inhibition i_{T} and the rotational inhibitions i_{CCW} and i_{CW}. Henceforth, we indicate the used coefficient set in the second part of the parameter list: $f_{vi}(v, \omega; \mathrm{T})$, $f_{vi}(v, \omega; \mathrm{CCW})$, and $f_{vi}(v, \omega; \mathrm{CW})$.

Table 5.3 Coefficients for computing inhibition from velocity with f_{vi} (Eq. 5.8), as obtained by fitting hand-labeled simulation data, where all edge cases are included and outliers removed.

Param. of f_{vi}	Values for Trans. Inhib.	Values for Rot. Inhib.
a	$-5.127\,898\,81$	$2.244\,294\,58$
b	$1.632\,213\,62$	$-0.834\,833\,08$
c	$-0.364\,301$	$0.099\,670\,6$
d	$0.063\,760\,09$	$0.065\,622\,74$
e	$-0.005\,292\,29$	$-0.022\,847\,53$
f	$0.013\,479\,63$	$0.047\,004\,35$
g	$-0.005\,943\,36$	$-0.068\,187\,54$

We depict the function from Eq. 5.8 for computing the translation inhibition in Fig. 5.18. Clearly, the translation inhibition primarily depends on the tangential velocity; angular velocity having a smaller influence.

Similarly, we depict the function for computing the CCW rotation inhibition in Fig. 5.19. In this case, the value depends mostly on the angular velocity and less on the tangential velocity.

The approximation quality of our fitted function of the calibration data in Tab. A.1 is demonstrated in

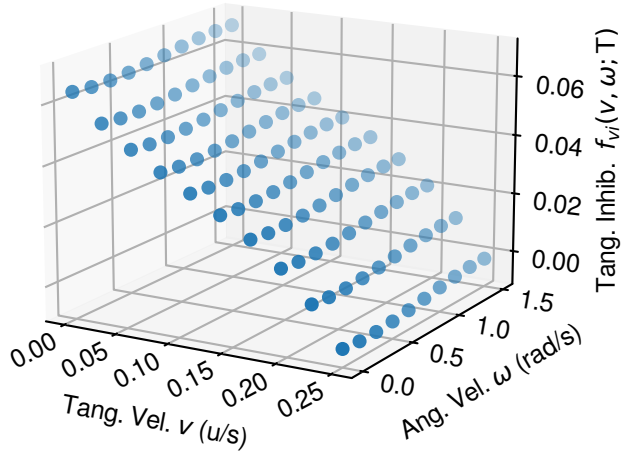

Figure 5.18 The translation inhibition can be computed from the tangential and angular velocity via the third order polynomial in Eq. 5.8 and the correct parameter set from Tab. 5.3. The tangential velocity has greater influence on the translation inhibition. The plot is generated by evenly sampling the tangential velocity in $[0, 0.25]$, and the angular velocity in $[0, 1.5]$.

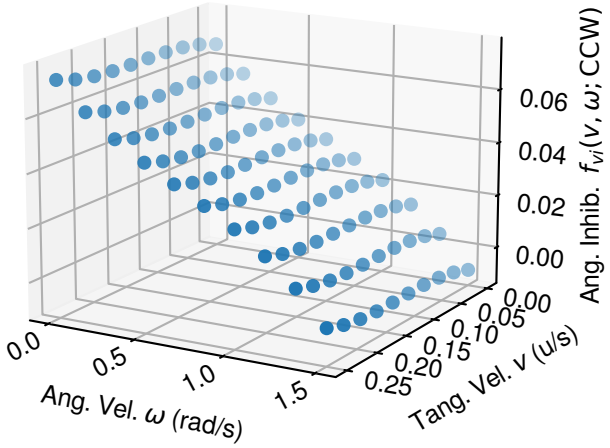

Figure 5.19 The counterclockwise rotation inhibition can be computed from the tangential and angular velocity via the third order polynomial in Eq. 5.8 and the correct parameter set from Tab. 5.3. The angular velocity has greater influence on the CCW rotation inhibition. The plot is generated by evenly sampling the tangential velocity in $[0, 0.25]$, and the angular velocity in $[0, 1.5]$. Note the different view angle compared to Fig. 5.18.

Figs. 5.20 and 5.21. In the first figure, we plot translation inhibition over tangential velocity; in the second one we show CCW rotation over angular velocity. Essentially, the gray bands correspond to the value range of Eq. 5.8's polynomial with the respective coefficients and projected along one of the two input dimensions. The blue points represent the raw data from Tab. A.1. The orange points show our fitted functions' approximation to the raw data. Note that the orange points lie strictly within the gray bands. Also, note the distinct vertical layers in which the scatter points are located; these reflect the 'reverse' way in which the raw data was obtained, i.e. input inhibition values and observe velocities. The blue points here are different from the blue points in Figs. 5.18 and 5.19.

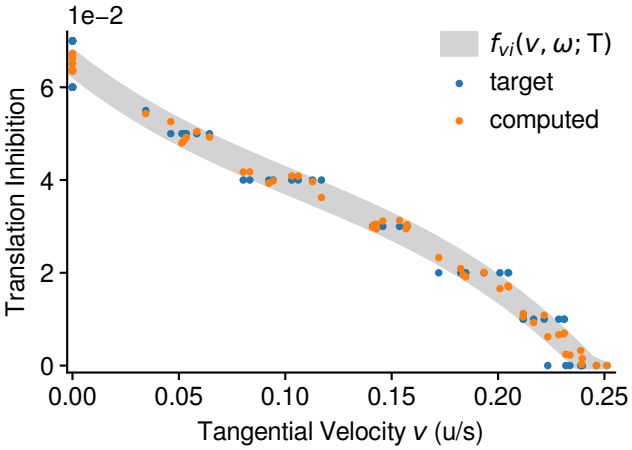

Figure 5.20 Approximating the translation inhibition from the tangential and angular velocities; projected along the latter input dimension. The gray band indicates the value range of the polynomial in Eq. 5.8 with the appropriate coefficients from Tab. 5.3. The raw calibration data from Tab. A.1 is shown in blue, and the approximations of our fitted function are overlaid in orange. The scatter points are clustered in distinct vertical layers, which stems from the data collection process.

Preprocessing. Next, we define feasible ranges for the velocity inputs. There are four goals here: (a) stay within the well-defined region of our polynomial; (b) each input may vary freely within its range, independent of the respective other input; (c) the entire ranges are usable in the sense that the bounds reflect the smallest/largest velocities our system can represent; and (d) the ranges are as large as possible. This yields the ranges $[0, v_{max}] = [0, 0.223]$ for the tangential velocity, and $[\omega_{min}, \omega_{max}] = [-1.256, 1.256]$ for the angular velocity. Any velocity input to our system is preprocessed by cropping it into these ranges. Note that the angular velocity range is symmetric around zero. This is because the system behaves symmetrically for positive and negative values thereof.

Final Mapping. After cropping, the velocity values are passed to the polynomial in Eq. 5.8 with

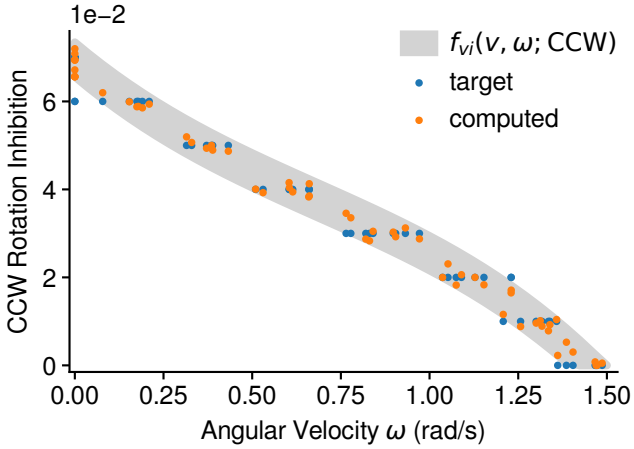

Figure 5.21 Approximating the counterclockwise rotation inhibition from the tangential and angular velocities; projected along the former input dimension. The gray band indicates the value range of the polynomial in Eq. 5.8 with the appropriate coefficients from Tab. 5.3 (indicated by 'CCW' in the second part of the parameter list). The raw calibration data from Tab. A.1 is shown in blue, and the approximations of our fitted function are overlaid in orange. The scatter points are clustered in distinct vertical layers, which stems from the data collection process.

coefficients from Tab. 5.3. We restrict the final inhibition values to be within certain ranges, aligned with the feasible velocity ranges. The minimum inhibition values required to prevent the corresponding shifting ensemble from causing any apparent movement in the attractor network are 0.0637 for translation, and 0.0656 for rotation. On the other end we ensure non-negativity, i.e. a lower bound of 0.

In summary, the computation of the translation inhibition from the preprocessed tangential and angular velocities amounts to

$$\begin{aligned} &\text{inh}_{\text{T}}(v, \omega) = \\ &= \begin{cases} \max\left\{0, f_{vi}(v, |\omega|)\right\}, & \text{if } > 0.0637 \\ 1, & \text{otherwise} \end{cases} \end{aligned} \tag{5.9}$$

where f_{vi} is parameterized by the appropriate coefficients from Tab. 5.3. To ensure complete inhibition in the case of zero tangential velocity, we map the inhibition value to 1 if it is larger than 0.0637. The counterclockwise inhibition and clockwise inhibition can be determined from the preprocessed tangential and angular velocities as

$$\begin{aligned} &\text{inh}_{\text{CCW}}(v, \omega) = \\ &= \begin{cases} \max\left\{0, f_{vi}(v, |\omega|)\right\}, & \text{if } > 0.0656 \text{ and } \omega > 0 \\ 1, & \text{otherwise} \end{cases} \end{aligned} \tag{5.10}$$

and

$$\begin{aligned} &\text{inh}_{\text{CW}}(v, \omega) = \\ &= \begin{cases} \max\left\{0, f_{vi}(v, |\omega|)\right\}, & \text{if } > 0.0656 \text{ and } \omega < 0 \\ 1, & \text{otherwise} \end{cases} \end{aligned} \tag{5.11}$$

respectively, where f_{vi} is parameterized by the appropriate coefficients from Tab. 5.3. Note the slight difference in the 'if' conditions of Eqs. 5.10 and 5.11. In the former, we map to 1 if larger than 0.0656 or if the angular velocity is *greater than* or equal to 0; in the latter, we map to 1 if larger than 0.0656 or if the angular velocity is *less than* or equal to 0.

Physical Velocity. At some point we could define a mapping from the tangential velocities relative to our unit domain, i.e. u/s, to tangential velocities in physical space, e.g. m/s. Such mapping essentially corresponds to applying some attenuation/gain factor to physical velocity input. Also, the mapping directly determines the spatial scale of the single-neuron response grid and, in turn, the scale of the population response and the unit domain itself. In other words, the grid scale is related to how sensitive the pose network is to velocity input.[66]

As for the angular velocity, any scaling here can only be done if the tangential velocity is scaled equally - so as not to distort the trajectory.

Inhibition and Attractor Dynamics. The maximum velocity can only be achieved when the corresponding input is applied long enough. Put differently, for short spikes of maximum velocity input the behavior of the bump is more likely to deviate from the desired behavior, i.e. move slower or not at all.

At very low velocities, the input from the shifting ensembles is going to be weak relative to the attractor dynamics and, therefore, cannot push the activity packets over to the respective next stable positions, i.e. the neurons' grid locations on the grid. This issue is called pinning. It is something to consider when defining the physical velocity mapping. Relating to biology, the behavior can be likened to the vestibular system not being able to pick up very subtle movement.

Consequentially, if there are long periods of low velocity, the deviation from the ideal trajectory is large.

Potential fixes for the pinning issue include adding more neurons, which is out of question for

our case; widening tuning/weight curves to have more neurons active per pose estimate, which would reduce the number of simultaneously representable poses, i.e. not an option; or changing the way we inhibit shifting ensembles.Edvardsen[17] We test the third idea.

Modulated Inhibition. As an alternative to directly and continuously inhibiting the shifting ensembles by input from a Nengo node, we implement so-called modulated inhibition. Per time step, a shifting ensemble can either be fully inhibited ($i_{\mathrm{T/CCW/CW}} = 1$) or not inhibited ($i_{\mathrm{T/CCW/CW}} = 0$). To get completely disable a shifting ensemble, we inhibit it fully at every time step; for maximum speed, we do not inhibit it at any time step; and for e.g. 66 percent of the maximum speed we might inhibit it at *every third time step*. Essentially, the proportion of time (steps) that we inhibit a shifting ensemble inversely corresponds to the proportion of the maximum bump speed. It is not quite as simple as that, because there are (inert) nonlinear neural dynamics and synaptic delays involved. A time step of full inhibition followed by a time step of no inhibition do not necessarily result in 50 percent of the maximum speed. The true effect of such modulated inhibition is more like a nonlinear smoothing or averaging (actually, below average) of the inhibition over a certain time period.

To implement this form of inhibition, we add a single-neuron ensemble per shifting ensemble. We use a one-dimensional encoding space, set the encoder to 1, intercept to 0, and nominal firing rate to 1000 (i.e. equal to the number of time steps per second, with $dt = 0.001$). As neuron type we choose `nengo.SpikingRectifiedLinear`. Each of these new ensembles receives input from the input node via an encoded connection, which rudimentarily converts from velocity to inhibition strength; we did not tune a complicated mapping. The specific ensemble configuration results in the single neuron spiking proportionally for input values in the range $[0, 1]$. Hence, the desired inhibition strength is converted to a spike train. Each new ensemble projects to the respective shifting ensemble via a direct neuron-to-neuron connection with a transform of $-dt$. The scaling is necessary because Nengo sets the spike amplitude to $1/dt$ by default, but we want it to be 1.

Regarding the case of two shifting ensembles being active simultaneously (see more in Sec. 5.8), as might happen for a left turn movement, there is the idea of interleaving translation and rotation and to modulate them separately within their timeframes. For example, every odd time step could be designated for translation, whereas every even time step is for rotation. This would change the available time steps per second per shifting ensemble from 1000 to 500.

We conducted various experiments with modulated inhibition and compared the results to the continuous inhibition. For the same inhibition input values, the system's behaviors in terms of tangential velocity, angular velocity, and trajectory smoothness are very similar. The representable ranges of the two velocity dimensions are comparable. Such one-to-one correspondence is interesting since the effects of the two inhibition variants on the neural dynamics of the main attractor are conceptually quite different. Regarding the initial motivation for modulated inhibition, i.e. potentially fixing the continuous inhibition's performance at very low velocities, we find that the lowest possible velocity that our system can follow when using either inhibition method is basically the same. It does not seem that one variant of inhibition works where the other does not. One reason is that we get into the limit region where neural dynamics and the associated inert behavior are relatively strong relative to the shifting strength (for each variant). On the other end, similar to continuous inhibition, the maximum velocity can only be achieved when there are enough consecutive 'no inhibition'-time steps. See Sec. 6.4 and Fig. 6.22 for more on the relevant simulations. We decided to stick with continuous inhibition for now, since it is conceptually simpler and the resulting trajectories appear to be slightly smoother at times. For completeness, we note that we also tried a combination of continuous and modulated inhibition. While it worked, there did not seem to be any immediate benefit.

Input Specification. Ultimately, we want to be able to input time-varying velocity values (both tangential and angular) to our pose network. For simulation, we use a Nengo process object of type `nengo.processes.Piecewise`. It takes a dictionary mapping time points to 'values'. The time points can be arbitrary, but should be in order. The 'values' should be of the same type and shape, e.g. NumPy arrays of shape $(2,)$ for tangential and angular velocity. The process object is then passed

to the input node, which will output the desired values between two time points in a piecewise constant way.

For visualization, we implement helper functions to convert from velocity arrays to sequences of 2D waypoints (by integrating the change caused by the instantaneous velocities over time), and back (by taking the pairwise difference of successive waypoints, some trigonometry, and the elapsed time between the waypoints).

When talking about trajectories, at some point the need for unrolling our unit domain according to its periodic boundary conditions arises. As explained in earlier sections of this chapter, the first two dimensions of the unit domain tessellate 2D physical space in a rhombic/hexagonal fashion. This means that following velocity input corresponding to some trajectory results in a wrapped version of that trajectory. To unwrap it, we implement `wrap_closer`. This function takes as input a reference point and an array of points, that are to be brought as close as possible to the reference point in the *unrolled* domain. The function operates in reciprocal space. Essentially, per point in the array, it tries out many different domain-length offsets in the first two dimensions and checks which one brings the point closest to the reference point. An array of the closest points is returned. As an inverse operation, `wrap_into_parallelogram` is implemented. It also works in reciprocal space. Essentially, the first two dimensions of a point' coordinates are brought into the range $[0, 1]^2$ by a modulo-like logic. For efficiency, the input is modified in-place. For now, the assumption here is that all values are in the range $[-1, 2]$, but this could easily be extended.

5.8 Path Integration Dynamics

Having the mechanisms for translation and rotation of pose estimates in place, we can now discuss the response of the attractor network when these mechanisms are enabled. There are many factors that influence the path integration dynamics.

We build upon the initialization and stationary dynamics outlined in Sec. 5.5, where velocity input was assumed to be zero. In short, the attractor dynamics are determined by the balance of recurrent local excitation and inhibition, the constant global inhibition, as well as the nonlinear neural and synaptic dynamics. While the set of dynamically stable states ideally contains all possible combinations of one or more pose estimates in the unit domain Dom, it realistically is discrete - as is the case for our attractor network. Specifically, the set contains combinations of sufficiently spaced pose estimates at neurons' grid locations, including the state of zero pose estimates. When there is no input, the system remains in a stable state indefinitely. This is equivalent to saying pose estimates or bumps that are sufficiently far apart in Dom persist indefinitely. In the section on stationary dynamics we also covered the behavior of existing activity packets (in the form of a 3D Gaussian) when there is reset input trying to establish a new activity packet (also a 3D Gaussian) at some location. If the new pose estimate is far from all existing packets, the existing ones are affected only by the inhibitory input component. Otherwise, the new activity packet interacts smoothly with at least one existing packet. Depending on the actual distance between the interacting bumps and on the strength and duration of the input, the affected existing bumps may be slightly pulled towards the input bump's center, or pulled essentially all the way to the input's center.

The considerations in this section are complemented by the experiments and insights in Ch. 6.

5.8.1 General Nonstationary Dynamics

Now, we assume scenarios with non-zero velocity input. In terms of the dynamically stable attractor states mentioned above, this means that we essentially transition between them. The maximum time between two states depends on many parameters, e.g. on the number of neurons involved in representing a pose estimate. How good our attractor dynamics plus translation and rotation mechanisms can interpolate in between the states determines the path integration performance.

For the shifting to work, the attractor dynamics not only need to be properly balanced themselves, but also need to be balanced with the shifting dynamics. For example, constructing the shifting weights with an offset of $0.25 \cdot 1/12$ in Sec. 5.7.2 is not arbitrary, but was tuned simultaneously with the recurrent attractor weights. Combining that with the inhibitory modulation of the shifting ensembles' neural activities allows the system to smoothly track varying tangential and angular velocities.

As with stationary dynamics, the system's behavior depends on its current state. Examples are the response to reset inputs that cause merging vs. ones that do not; or the response to velocity inputs

when an activity packet is not yet fully established vs. when it is in a stable state. Note that a bump needs to be fully established before correct movement can be expected. There is also a difference between translation in a direction which aligns with a grid axis or a layer of the translation module vs. translation in some direction between two grid axes or θ-layers. In general, movement of activity packets along a grid axis is favored by the pose network. This is due to the discrete neuron grid and the associated discrete set of stable states. If an activity packet is at a position where the forward movement direction lies between two grid axes, then a translatory input might not result in a straight trajectory but rather the activity packet is pulled towards the closer grid axis (on a curve-like path) and then travels along it. The behavior might be different if the activity packet were to start out exactly in between two grid axes. Note that a grid axis may not only pertain to a direction in the plane spanned by the first two dimensions, but can also be perpendicular to that. Another important factor are the θ-layers and how/if they coincide with grid axes. For example, we end up using 12 vertical layers for our sampling and neuron grids, corresponding to direction tunings of $\theta \in \{0°, 30°, 60°, 90°, 120°, \dots, 330°\}$. Thereof, only every second layer coincides with a grid axis, i.e. $\theta \in \{0°, 60°, 120°, 180°, 240°, 300°\}$. Put simply, most attractive are directions where a grid axis coincides with a layer's direction, then come directions that align either with a layer or with a grid axis. It is difficult to say whether layers or grid axes are more 'powerful'. The actual behavior depends strongly on the circumstances, e.g. which direction is closest to the pose estimate in question, or how active the translation and rotation modules are. We try to convey an intuition of the matter in Ch. 6.

Inert Behavior. We have mentioned the issue of inert dynamics at multiple occasions. It essentially stems from the neural state, i.e. the voltage potential and the Leaky Integrate-and-Fire (LIF) dynamics associated with it; from the synaptic delay of connections to and from neuron ensembles; and from the fact that the shifting ensembles operate on a delayed replication of the attractor state.

As a consequence, the system cannot follow sharp or short input spikes. Further, rapid or high-frequent input fluctuations between zero and some nonzero value are impossible to capture. For example, assume that the shifting ensembles are fully inhibited and that their neurons are at a voltage potential of zero. When the velocity input jumps to a nonzero value, the inhibition of at least one shifting ensemble drops and the respective shifting ensemble is supposed to exert some influence on the attractor network. However, it takes some time to accumulate voltage potential and generate spikes in the shifting ensemble. The onset of the desired shifting influence is delayed. This directly explains why the maximum velocity can only be achieved when the corresponding input is applied long enough.

While increasing the synaptic time constant τ generally makes the system's behavior smoother, it also makes it more inert. It also changes the feasible range of velocity inputs and potentially requires retuning of the recurrent and shifting connection weights to get (closer to) the best performance. Therefore, we leave a thorough evaluation of different τ's for future work.

Drift. The preference for activity packets to move along grid axes, the inertia of the attractor and shifting ensembles, or noise, among others, cause drift of the represented pose estimates relative to their respective target values/trajectories. This might even happen in the absence of velocity input. These errors add up over time. Hence, to perform successful path integration, the drift must be low.

Note that a larger deviation from the target trajectory does not result in an increased variance of the involved 3D Gaussians. Rather, by design, the variance in all dimensions stays constant, i.e. the attractor network does not represent uncertainty in the classic probabilistic sense (see Sec. 5.2).

One way to reduce drift is by increasing the number of neurons in the pose ensemble and, in turn, in the shifting ensembles.[10] As we are constrained in the number of neurons by Loihi, this is no viable option.

Depending on the duration of the trajectory, visual reset may be necessary to periodically correct accumulated errors (see Ch. 8).

5.8.2 Single Activity Packet

Consider a single pose estimate and its corresponding activity bubble. As the naming implies, there are usually neurons from multiple adjacent θ-layers involved. The connectivity described in

Sec. 5.7.2 indicates that each horizontal slice of an activity packet is propagated in the associated preferred direction. Attractor dynamics counteract a dissection of the activity packet, and instead enable a propagation into the mean direction of the involved θ-layers. This directly explains why we can represent movement directions from a continuous spectrum, as opposed to the discrete set of θ-layers. Also, including the heading orientation in the representational space along with the translation module setup and connectivity makes it possible that different pose estimates are independently translated into separate directions. Intuitively, ignoring any rotation, each pose estimate is translated into its forward direction, which is different for each one.

During pure rotation, the activity packet moves vertically through neuron layers, i.e. inter-plane. There is no shearing of the activity packet.

For simultaneous translation and rotation, i.e. when the translation module and one rotation module are disinhibited at the same or at a similar time, the bump moves in the average instantaneous direction. Intuitively, per time step, the result of just the translation and the result of just the rotation are combined/averaged to get the next state. When thinking of movement vectors, this corresponds to applying the Pythagorean theorem. The effective tangential and angular velocities of the bump are slightly different compared to pure translation or pure rotation, respectively.

In more detail, the translation module tries to establish an activity bubble slightly offset in the forward direction, while one rotation module tries to establish a bubble slightly offset above or below the current bubble, and since this happens at the same time we essentially end up with a slightly distorted bubble that is a combination of the two. The attractor dynamics continuously drive the neural activity towards the closest stable state consisting of only Gaussian-like bubbles. This is crucial to keep path integration errors as small as possible.[10,13]

Specific to the modulated inhibition, due to randomness in the neuron or synapse states the single-neuron ensembles may not always fire at the exact same time for the same inhibition value. Hence, the shifting ensembles may not always interfere when converging onto the attractor network. Along these lines, there is also the idea of interleaving translation and rotation and to modulate them separately within their timeframes (see Sec. 5.7.3).

5.8.3 Multiple Simultaneous Activity Packets

In a realistic setting, we might encounter two or potentially also three simultaneous pose estimates. These need to be represented and propagated properly until one single estimate emerges as the 'true' one.

The previous considerations generalize readily to multiple simultaneous bumps that are sufficiently far apart, so that they exert little to no influence on each other. As long as the bumps do not intersect, they will exist and move 'in parallel' indefinitely.

When two bumps get closer, they interfere and deviate from the single-bump dynamics. The magnitude of the deviation is indirectly proportional to the distance between the bump centers. The exact behavior can be very diverse and cannot be quantified. The involved overall dynamics are highly nonlinear: LIF dynamics of the attractor network, dynamics due to the recurrent connection, LIF dynamics of the shifting networks, dynamics due to the connection between shifting networks and attractor network, the functional shape of the connection weights, the intensity of the involved bumps, how close they get, the relative angle of approach, and other static and dynamic factors. Generally, bumps get slower upon approaching each other. When the closest distance is not too small, the bumps merely get deflected from their respective target trajectories. If they get too close, however, one is destroyed or they merge. There might also be some kind of momentum involved so that even with a close pass-by both involved bumps persist and continue on their altered trajectories. In Ch. 6, we provide a more qualitative understanding of the multi-bump interfere.

5.9 Experiment Framework

Here, we highlight some interesting details about our framework for running and evaluating Nengo simulations of our pose network.

Configuration Management and Simulation Data. There are a lot of parameters that define our system architecture, or influence the Nengo build and simulation process. In order to keep track of their values during development and fine-tuning, we use Sacred* for configuration management

* https://github.com/IDSIA/sacred

and organizing simulation results. The final configuration includes: seed values, the simulation time step length dt, the resolution of the sampling grid, the resolution of the neuron grid, the variance of the 3D Gaussians used to represent poses (same for all dimensions), the probing interval of any enabled Nengo probes, the simulation duration, the main synaptic time constant to be used, all parameters of the recurrent and shifting connection weight functions, the input data (see below for specification), whether to use the default backend or the NengoLoihi backend, the neuron bias for ensembles with constant global inhibition, whether to use continuous or modulated inhibition, whether to use 'full' connections from the attractor to the shifting ensembles or sparse scaled identity matrices, a textual identifier, and some comment for evaluation.

We set up a MongoDB database to organize all data related to experiments. There is one table containing metadata and configuration parameters, which is indexed and efficiently searchable. Per experiment, we also store snapshots of any involved and important source files, which makes it easier to evaluate the effect of any specific changes in the code that are not entirely controlled by the configuration parameters. In a second table we keep any data generated by experiments during simulation. Primarily, this concerns time-varying quantities in the neural network that are explicitly probed for. A prominent example for such quantity is the multidimensional output of the main attractor ensemble, which, depending on the function defined for the decoding node, amounts to some form of the represented pose estimates over time. Since performing the decoding from neural activities to representation space Rep as well as the subsequent conversion to our domain Dom after the simulation yields a speedup thereof, we simply configure the decoding node as a pass-through node, i.e. no function or transformation is applied.

We define a default configuration with values for each entry of the final configuration specification introduced above, which have been found to work well in general. For each (relevant) simulation run, a new Sacred experiment is instantiated. Conveniently, we only need to specify the configuration values that are to be modified from the default configuration.

During evaluation, we need to find and read out experiment data from the database. For this we use the Omniboard GUI* and the Incense toolbox.†

For portability, we deploy the database and Omniboard in separate docker containers.

Simulation Input. For specifying input to a simulation via the `input` config parameter, we created a hierarchical data scheme. At the base level, there is a list of *input blocks*. Each block has a duration and a list of *commands*. Starting at time $t = 0$, the first block is used for generating and providing input to the simulation until t equals its duration (note that per block the input will be constant). Then, we switch to the second block, and so on. All commands per block are used for generating the respective input. A command can be: a list of pose estimates, which are to be initialized (as 3D Gaussians); the inhibition inputs i_{T}, i_{CCW}, and i_{CW} for three shifting ensembles; the scalar tangential and angular velocity values v and ω, respectively; or three equal-length arrays containing durations, tangential velocities, and angular velocities, respectively. The latter command requires preprocessing to convert its arrays to a list of blocks with (a single command with) one pair of tangential and angular velocity and the corresponding per block. Via the 'array'-command we can effectively input arbitrary trajectories to the system.

The input for the pose network, specifically, for its input node, is expected to be a vector of the following shape: the lower dimensions pertain to the representational space of roughly $2 \cdot 12^3$ dimensions (containing any to-be-initialized poses/activity packets) and the last two dimensions contain the tangential and angular velocities. We call this the *input vector*. If there are multiple commands per block, their respective input vectors are simply added together.

Evaluation. For evaluating the simulation results, primarily snapshots or the evolution of the neural activity state of the pose ensemble, we created numerous Jupyter notebooks and devised many sophisticated plotting techniques and functions. We refer the reader directly to the project source to learn more.

* https://github.com/vivekratnavel/omniboard
† https://github.com/JarnoRFB/incense

Example of Running an Experiment. Here, we walk through a small example of how to run the pose network on the Nengo Loihi backend, with velocity input corresponding to straight movement. This example has been directly adapted from the documentation of our Python package.

```
from exp import ex

# Run the simulation. Specify duration, input commands, and enable the Nengo
# Loihi backend. All other parameters get default values as defined in
# 'exp.default_config'.
ex.run(config_updates={
  'simulation_duration': 5.7,
  'input': [{  # Initialize a pose estimate, specified in reciprocal grid
               # coordinates.
    'duration': 0.2,
    'cmds': [{'cmd': 'input_freq',
              'bump_centers': [(4, 4, 0)]}]
  }, {  # Short period of no movement to allow the attractor network reach
        # a stable state.
    'duration': 0.5,
    'cmds': [{'cmd': 'manual',
              'shift_inhib': 1,
              'pos_rot_shift_inhib': 1,
              'neg_rot_shift_inhib': 1}]
  }, {  # Constant tangential velocity input of 0.5, angular velocity input
        # of 0.
    'duration': 5.0,
    'cmds': [{'cmd': 'manual_vel',
              'tangential_vel': 0.5,
              'angular_vel': 0}]
  }],
  'use_loihi': True});
```

The data generated during simulation can be accessed and decoded as follows. Assuming defaults for everything, you only need to plug in `your_mongo_uri`.

The `decoders` can be obtained as the weights of the main recurrent connection as computed by Nengo. To this end, disable our custom direct neurons-to-neurons connection and enable the NEF-style connection in `nengo_utils`. Then get the decoders like `decoders = sim.data[pose_network.attractor_network.rec_con].weights.copy()`. This needs to be done only once. Do not forget to undo the changes in `nengo_utils` afterwards.

Eventually, we arrive at a set of pose estimates for each time point.

```
import incense
import nengo
import numpy as np

from exp import ex
from pose.experiments import get_cluster_centers_timeseries,
                                        reconstruct_timeseries
from pose.freq_space import get_0_coefs
from pose.hex import fspace_base, get_3d_coordinates_unwrapped_vectorized
from pose.nengo_utils import encoding_mask
from pose.plotting import plot_experiment_xy

# Load the experiment's metadata and data from the database.
loader = incense.ExperimentLoader(
  mongo_uri=your_mongo_uri,
  db_name='sacred'
)

def load_experiment(loader):
  exp = loader.find_latest()  # we want the most recent experiment
  _config = exp.to_dict()['config']
```

```
  data_raw = exp.artifacts['sim_data.pkl'].as_type(
    incense.artifact.PickleArtifact)
  data = data_raw.render()  # load pickle
  return exp.to_dict(), _config, data
_exp_info, _config, data = load_experiment(loader)

# Get some config values.
variance_pose = _config['variance_pose']
cov = np.eye(3) * variance_pose
fgrid_shape = _config['fgrid_shape']

# For each time point, decode the neuron activations into representational
# space.
decoded_sim_data = decoders.dot(data['pose_network.output'].transpose())
decoded_sim_data = decoded_sim_data.transpose()

# Reintroduce the ommitted imaginary parts of the Nyquist terms.
output_restored =np.zeros((decoded_sim_data.shape[0],np.prod(fgrid_shape)*2))
output_restored[:, encoding_mask(fgrid_shape).ravel()] = decoded_sim_data

# Undo normalization to obtain the final reciprocal space representation.
gauss0_f_cropped_flat = get_0_coefs(fgrid_shape, cov)
fact = gauss0_f_cropped_flat[0] / gauss0_f_cropped_flat[2]
output_restored[:, 0] = output_restored[:, 2] * fact

# Back-transform into real space.
pose_reconstructed = reconstruct_timeseries(output_restored, fgrid_shape)

# Compute pose estimates from their 3D-Gaussian representation.
bump_centers =get_cluster_centers_timeseries(pose_reconstructed, fgrid_shape)

# Plot the projection of the pose estimates along the third axis, color-coded
# by time.
plot_experiment_xy(_exp_info, _config, bump_centers, time_slice=np.s_[7:])
```

6 Evaluation

In this section, we demonstrate the capabilities of our approach. First, numerous proof-of-concept experiments and simulations are presented. These include the behavior of the path integration system in various basic situations such as linear and circular movement. This is followed by a subsection on sustaining and propagating multiple simultaneous activity packets. There, the phenomenon of interference between bumps is discussed as well. Finally, we deal with more complex trajectories specified by vectors of 2D waypoints or, equivalently, vectors of tangential and angular velocity tuples. The input data can be obtained from simulation, or be recorded from actual robots moving around in a suitable environment. We demonstrate the ability of our system of following realistic continuous velocity input series.

Unless specified otherwise, the configuration and parameters listed in Table 6.1 are used for the following experiments. Positional values are given in unit u. As a reminder, $1\,\mathrm{u}$ is defined as the total length of our unit domain along the x-axis (see Sec. 5.1). With a neuron grid of $12 \times 12 \times 12$, the distance between two neighboring neurons is equal to $1/12\,\mathrm{u}$. Furthermore, we opt for continuous inhibition for most experiments, since initial testing showed that it yields similar but seemingly smoother movement patterns than modulated inhibition. For comparison, we also include some experiments using modulated inhibition.

When referring to simulation time, we define the onset of velocity input or the start of the evaluated behavior as $t = 0$. Therefore, any simulated activity before that, such as initialization via reset input or periods of free-running to reach a stable state, is referred to with negative time information.

Simulation time is not to be confused with the actual time it takes to run the simulation. This quantity strongly depends on the available compute resources. Over the course of the project, we implemented various optimizations to bring the time required to run a couple of seconds of simulation down from hours to minutes. These optimizations are discussed in the respective sections. The fact that the individual neurons and their spiking dynamics need be simulated cannot be 'optimized away'. When using the actual Loihi chip, however, the neural substrate is run much more efficiently. Hence, we expect a significant performance boost, potentially even real-time capability.

Note that when we refer to two points in the same θ-plane being at the greatest possible x-y-distance apart, it means that the absolute distance between the points on the extended plane, i.e. considering wrap-around, cannot be increased by moving just one of the two points. Importantly, however, this does not mean that the distance along the movement direction of either bump to the other is maximal.

For the trajectory following experiments we use two common evaluation metrics, the absolute trajectory error (ATE) and the relative trajectory error (RTE), to quantify how closely our system's estimated trajectories matches the target trajectories. Both metrics are computed using the freely-available† python scripts by Sturm et al.[60], modified for our use case. Often, the target trajectory and the estimated trajectory have different sampling rates or even missing data points, necessitating a preprocessing step to associate sampled points between the two trajectories. Here, this is done by computing best matches, i.e. associating each sampled point from one trajectory with the closest sampled point in time of the other trajectory. Optionally, the association quality can be improved by interpolating one of the sampled trajectories.

Absolute Trajectory Error (ATE).[60] Concerns the absolute distances between the target trajectory and the estimated trajectory at each time point or sample point. The ATE thus gives information about the global consistency of the estimated trajectory.

Usually, one of the two trajectories is rigidly transformed such that both trajectories are aligned as closely as possible prior to computation of the ATE. This is particularly relevant for SLAM systems and can e.g. be achieved by solving a least-squares problem. Since we set up our trajectories such that

† https://vision.in.tum.de/data/datasets/rgbd-dataset/tools

Table 6.1 Default parameters used for evaluation experiments. These values were obtained by meticulous manual tuning while simultaneously developing the neural architecture. The parameters are highly dependent and influential on each other, i.e. changing one or more values would most likely necessitate tuning multiple others as well. Parts are reproduced from Tabs. 5.1 and 5.2.

General	Frequency resolution	$12 \times 12 \times 12$
	Neuron grid	$12 \times 12 \times 12$
	Variance of Gaussian for tuning curves and pose estimates	0.0025
Simulation	Readout time interval (s)	0.1
	Simulation time step (s)	0.001
Neurons & Synapses	Synaptic time constant τ	0.1
	Neuron input bias current	0.968
	Neuron input gain	2.68423963
Connection weights	Variance of excitatory Gaussian	0.0108485147
	Variance of inhibitory Gaussian	0.01084861
	Scaling of excitatory Gaussian	1.00115141
	Scaling of inhibitory Gaussian	1.001155
	Constant offset	0.0
	Center-offset for translation weights	$0.25 \cdot 1/12$
	Center-offset for rotation weights	$0.25 \cdot 1/12$
	Threshold for zeroing weight components	$2.3 \cdot 10^{-5}$
Nengo Loihi	Block shape for splitting ensembles onto cores	$4 \times 4 \times 4$

they start at the same position and orientation, the alignment via a rigid-body transformation is omitted.

The ATE only directly considers the translational errors, but indirectly also contains information about rotational errors since those lead to translational errors at consecutive time points.

In addition to the raw ATE, statistics such as the root-mean-square error (RMSE) or the mean value over the entire trajectory are interesting. The RMSE is computed by taking the square over the normalized sum of squared individual values. The mean is less influenced by outlier points since there is no square-operation involved.

Relative Pose Error (RPE).[60] Measures the difference in relative motion between two trajectories within a fixed-length time period. Hence, it provides information about local errors, i.e. drift. A time parameter determines the length of the time period. For example, choosing delta as one second makes the RPE equal to the drift per second. The smallest possible time delta is achieved by considering consecutive samples, while the largest delta directly compares first and last pose estimate. It is also possible to average over multiple deltas. We specifically choose the smallest delta. Since our estimated trajectories are sampled ten times per second, the RPE then corresponds to the drift per 0.1 seconds. We postulate that such choice for delta makes the RPE highly dependent on the quality of association between the sampled points of the two trajectories.

The relative pose error considers both the translational and the rotational movement components. As with the ATE, the RPE's translational component implicitly contains information on the rotational errors. It is therefore often sufficient to only report the translational part and still refer to it as 'RPE'. ATE and RPE are strongly correlated.

Just like for the ATE, statistics such as the RMSE or the mean value over the entire trajectory are interesting to consider for the RPE.

6.1 Basic Movement

Here, we directly specify the inhibition values applied to the shift, clockwise and counterclockwise rotation ensembles. We consider values between 0 and about 0.07, since this range roughly aligns with the range of operating currents in the involved neural ensembles, accounting for synaptic low-pass filtering and inertia of spiking neural dynamics. Note

that when some experiments use inhibition values of 1, this is similar to using a value of about 0.07, since there the inhibited ensemble also has little to no influence on the main attractor.

6.1.1 Pure Translation

Fully inhibiting the two rotation modules, by setting the corresponding inhibition values i_{CW} and i_{CCW} to 1, while varying the inhibition strength of the translation module yields purely translational movement at different speeds. Figure 6.1 depicts multiple experiments covering the entire range of possible tangential velocities, from standstill to about $0.25\,\mathrm{u/s}$. Apart from including the behavior at inhibition values in equidistant steps within this interval, we also show the situations of barely moving and moving at near-maximum speed. We conclude that our path integration is capable of capturing small changes of the tangential velocity input, i.e. boasts a high resolution for representing this quantity. Furthermore, the nonlinear relation between tangential velocity and tangential inhibition strength becomes apparent when comparing the plots. For example, the velocity difference between $i_{\mathrm{T}} = 0$ and $i_{\mathrm{T}} = 0.01$ is about $(1/12)\,\mathrm{u/s}$, between $i_{\mathrm{T}} = 0.01$ and $i_{\mathrm{T}} = 0.02$ it is about $(2/12)\,\mathrm{u/s}$, while it is about $(3/12)\,\mathrm{u/s}$ between $i_{\mathrm{T}} = 0.02$ and $i_{\mathrm{T}} = 0.03$ as well as between $i_{\mathrm{T}} = 0.03$ and $i_{\mathrm{T}} = 0.04$.

When we vary only the angular dimension of the starting location and keep all other configuration the same, specifically a constant translation inhibition of $i_{\mathrm{T}} = 0.02$ and fully inhibited rotation ensembles, the bump moves in a straight line into different directions. As introduced in Sections 5.2 and 5.3, a bump starting in layer 0 of the 3D pose ensemble corresponds to a heading direction of 0 degrees. Going upwards through the layers, the corresponding heading direction increases in 30 degree steps, wrapping around from 330 degrees to 0 degrees when going from layer 11 to layer 0. With this in mind, in Fig. 6.2a-d and Fig. A.1 we first show experiments that have bumps move only *within* layers. The pose estimates are propagated in perfectly straight lines into the intended direction, i.e. remain vertically centered in the neural layer that they were initialized to. This demonstrates a certain level of stability of the system. Furthermore, independent of layer, the bumps travel roughly at the same speed. This is despite the fact that in some instances the movement direction is aligned with one of the periodicity axes of the hexagonal grid and in other instances the movement direction is right in between two axes (angle-wise). In these two cases the arrangement of neurons representing the translating activity packet is different from the perspective of the activity packet - compare e.g. the different spacing of neurons encountered by the bumps in Fig. 6.2a and Fig. 6.2b.

Next we look at initial heading angles other than multiples of $30°$. For this part, a constant translation inhibition of $i_{\mathrm{T}} = 0$ is used. Further, the weight parameters 'center-offset for translation weights' and 'center-offset for rotation weights' are set to $0.45 \cdot 1/12$. This is done to obtain a greater effective shifting force, as explained below. Fig. 6.2e-h depicts experiments where the bumps are initialized at varying locations between horizontal neural layers 1 and 2 of the 3D pose ensemble. There are some important aspects that influence the behavior, i.e. how long bumps move into the intended direction before potentially drifting to the layer immediately below or above their starting layer. Firstly, if a bump starts between two layers but relatively closer to one of them, it has a tendency to drift towards the closer layer. The more skewed the starting location, the more likely and the faster the closer layer is approached. Secondly, the connection weight parameters and the inhibition strength can make the difference between some pose estimate remaining at the desired heading direction, or drifting to a neighboring layer's orientation. Specifically, if the shifting force, influenced by the center-offset of the shifting weights as well as by the tangential inhibition, is relatively strong compared to the attractor dynamics, which try to pull the bump into a stable state or keep it in one, then the bump tends to remain in between layers longer. Note that stable states are predominantly centered at neuron locations. The drift behavior can also be influenced by the width of the effective neural response curves and the shape of the recurrent connection weights. For example, by setting the parameters such that more neurons are active to represent a specific pose estimate, the system's behavior would be more stable and the drift would be slower and/or less severe. Additionally, if the preferred orientation of one of the bump's two neighboring layers is aligned with a grid axis while the other layer's preferred orientation is not, the bump will naturally be drawn more towards the former layer. We can see that when comparing Fig. 6.2g, where the initial orientation is $42°$ and slowly drifts counterclockwise, with Fig. 6.2h, where

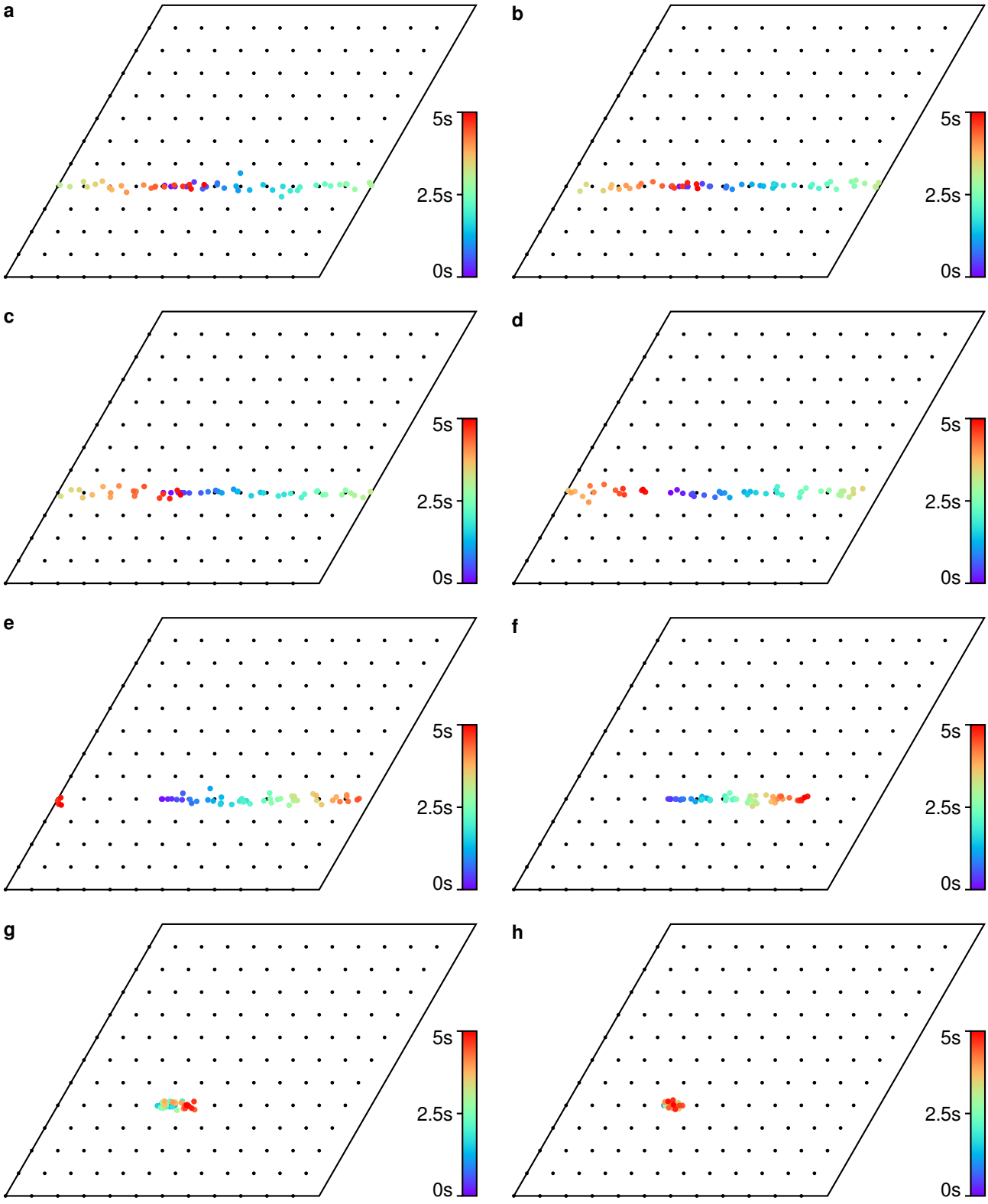

Figure 6.1 Pure translation with different values for the continuous shift inhibition i_{T}. Both rotation inhibition values, i_{CW} and i_{CCW}, are 1, so that the rotation modules do not have any influence on the main attractor. All plots show a time range of 5s, for which the inputs are kept constant. The bump is always initialized at position $(4, 4, 0)_{\mathrm{g}}$. **a**, The translation module is fully active ($i_{\mathrm{T}} = 0$). This shows the maximum tangential velocity. **b**, $i_{\mathrm{T}} = 0.005$. The speed is only slightly lower than at $i_{\mathrm{T}} = 0$, demonstrating that very small velocity changes can be captured by the system near it's maximum velocity. **c**, $i_{\mathrm{T}} = 0.01$. **d**, $i_{\mathrm{T}} = 0.02$. **e**, $i_{\mathrm{T}} = 0.03$. **f**, $i_{\mathrm{T}} = 0.04$. **g**, $i_{\mathrm{T}} = 0.05$. Rather low speed, showing that small velocity changes can also be followed near standstill. **h**, $i_{\mathrm{T}} = 0.055$. Effectively zero translational movement.

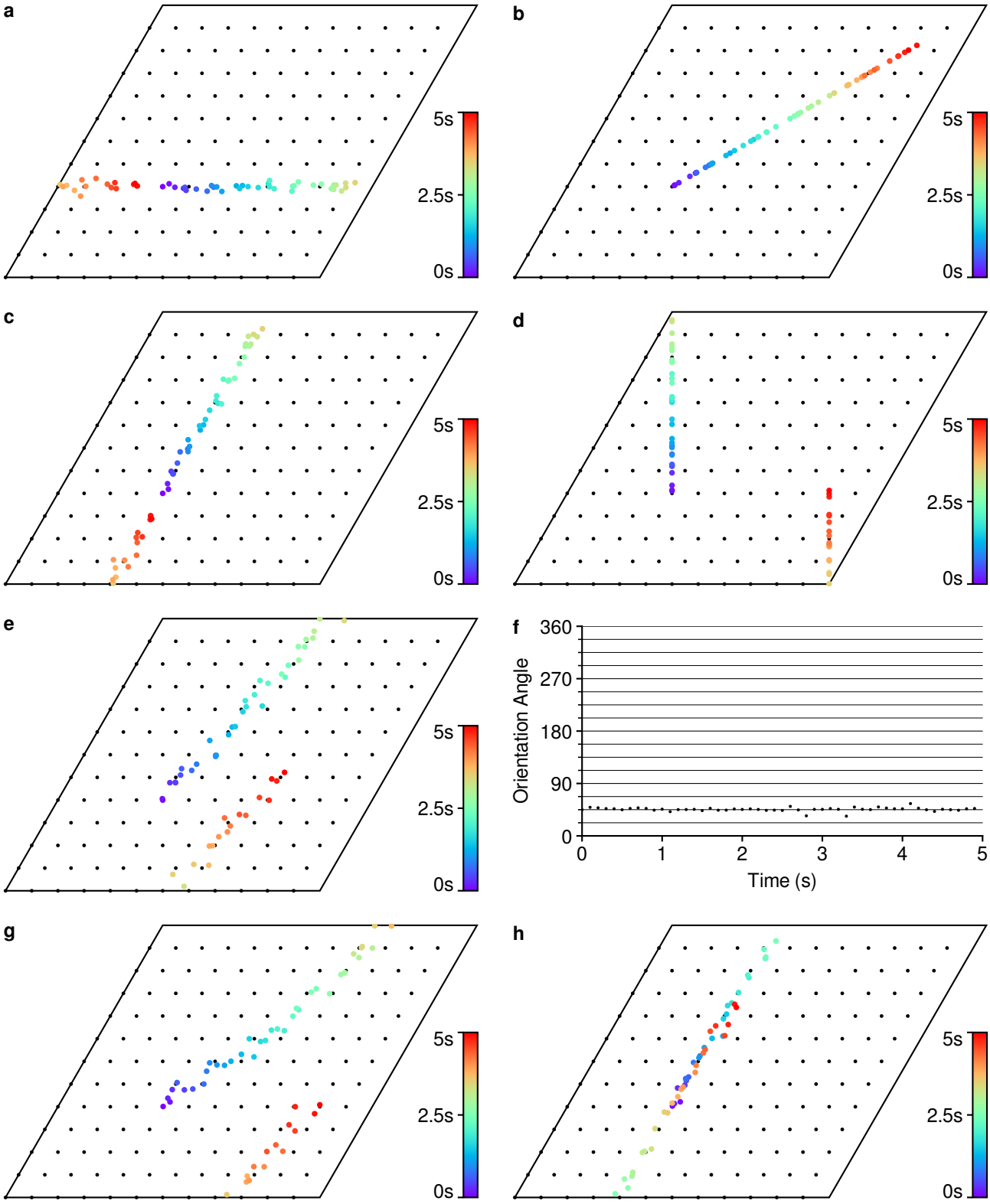

Figure 6.2 Pure translational movement into different directions. All plots show a time range of 5s, for which the inputs are kept constant. The bump is always initialized at position $(4, 4, *)_{\mathrm{g}}$, where the heading orientation is different for each experiment. **a**-**d**, Shown are the first four of 12 directions corresponding to the 12 layers in the 3D pose ensemble (see Fig. A.1 for the remaining ones). Independent of the starting layer, the bumps travel roughly at the same speed in $\mathrm{u/s}$. **a**, Initial pose at $t = 0$ is $\mathrm{P}(0) = (4, 4, 0)_{\mathrm{g}}$, i.e. oriented at $0°$. **b**, $\mathrm{P}(0) = (4, 4, 1)_{\mathrm{g}}$, $30°$. **c**, $\mathrm{P}(0) = (4, 4, 2)_{\mathrm{g}}$, $60°$. **d**, $\mathrm{P}(0) = (4, 4, 3)_{\mathrm{g}}$, $90°$. **e**-**h**, Pose estimates are initialized at varying locations between layers 1 and 2 of the 3D pose ensemble. Here, a constant translation inhibition of $i_{\mathrm{T}} = 0$ is used, and the center-offsets for translation and rotation weights are each set to $0.45 \cdot 1/12$. *(continued on next page)*

Figure 6.2 ***(continued)*** The bumps either remain at their designated heading direction, or drift to a neighboring layer sooner or later. **e**, **f**, The starting position is equidistant between layer 1 and 2, at $\mathrm{P}(0) = (4, 4, 1.5)_{\mathrm{g}}$, i.e. $45°$. The pose estimate remains nicely at its initial orientation. **g**, Initializing the bump slightly closer to layer 1, at $\mathrm{P}(0) = (4, 4, 1.4)_{\mathrm{g}}$, i.e. $42°$, results in a flatter absolute angle of the linear movement and has the pose estimate's orientation component vary between slightly below and up to around $45°$. **h**, Starting slightly closer to layer 2, at $\mathrm{P}(0) = (4, 4, 1.6)_{\mathrm{g}}$, i.e. $48°$, we observe a rapid drift toward the nearby axis of periodicity of the hexagonal grid.

the bump is initialized at $48°$ and very quickly approaches layer 2, i.e. $60°$, which is aligned to a grid axis. Both, the angular distance between the initial orientation and layer 1 in the former case, as well as the angular distance between the initial orientation and layer 2 in the latter case, are $12°$. Nonetheless, in the former case, the bump moves toward $45°$, i.e. right between layer 1 and 2, whereas in the latter case, the bump moves toward $60°$. Hence, the attracting force of layer 2 is stronger than layer 1's. Of course, there is also slight variability of the behavior given the same inhibition inputs but having the bump's starting location somewhere in between neurons (within a layer) rather than exactly at a neuron's preferred position.

We conclude that the pose ensemble's behavior regarding maintaining the initial orientation of a pose estimate slightly varies depending on the movement speed and particular heading direction. Still, the target heading direction is approximately captured in all cases.

6.1.2 Pure Rotation

When we completely inhibit the shifting ensemble, i.e. $i_{\mathrm{T}} = 0$, as well as one rotation ensemble, the pose estimates move solely in the vertical direction, i.e. only rotate. In this section we primarily consider counterclockwise rotation, i.e. $i_{\mathrm{CW}} = 0$, but the clockwise case is perfectly symmetric. Figure 6.3 conveys an impression of the range of representable angular velocities, from standstill at $i_{\mathrm{CCW}} = 0.065$ and above, to about $1/5$ rotations per second at $i_{\mathrm{CCW}} = 0$. As for the purely linear movement covered in Sec. 6.1.1, apart from showing the behavior at equidistant points within this interval, we also include the cases of barely rotating and rotation at almost maximal representable angular velocity. From the respective results it is clear that our angular path integration can capture small changes of the rotational velocity input, i.e. features a good resolution of this variable. It is also evident, that the system can track pure rotations without any positional drift. Again, similar to the translation case, we can highlight the nonlinear relationship between angular velocity and rotation inhibition strength. To this end, consider the following differences in effective rotational speed (rotations per second): from $i_{\mathrm{CCW}} = 0.02$ to $i_{\mathrm{CCW}} = 0.03$ it is about 0.028, from $i_{\mathrm{CCW}} = 0.03$ to $i_{\mathrm{CCW}} = 0.04$ it is about 0.044, and from $i_{\mathrm{CCW}} = 0.05$ to $i_{\mathrm{CCW}} = 0.055$ it is about 0.017.

Note that there is a slight difference in effective rotation speed when using $i_{\mathrm{T}} = 0.07$ compared to $i_{\mathrm{T}} = 1$ for varying values of i_{CCW}. The effective rotation speed is greater for $i_{\mathrm{T}} = 1$, with the difference being between about 0.015 and 0.04. The difference tends to be at the lower end of this range for lower i_{CCW}, and at the higher end for higher i_{CCW}. With $i_{\mathrm{T}} = 0.07$, there is no rotational movement for $i_{\mathrm{CCW}} = 0.05$ and above, while with $i_{\mathrm{T}} = 1$, minor movement is achieved up to $i_{\mathrm{CCW}} = 0.06$. This can be explained, especially for higher rotation inhibition i_{CCW}, as follows: there the desired bump for the next time step $t + 1$, i.e. the activity obtained by applying attractor dynamics, input from the shift module, and input from one rotation module to the current activity for one time step, does not resemble a clear-enough defined Gaussian bump and, hence, is not able to establish itself at the new location over the old bump. The old bump better resembles a 3D Gaussian with proper self-excitation and surround inhibition, while the desired new bump does not. Therefore, the new bump is not able to sustain itself and inhibit the old bump, but the opposite happens, i.e. the new one is inhibited by the existing one. The reason for the less-clear defined new activity state is that with an inhibition value of $i_{\mathrm{T}} = 0.07$ the input from the shift ensemble is just strong enough to mess with the already weak influence of the rotation ensemble at higher i_{CCW}. Also, the shift influence is too weak to cause translation of the bump. The described discrepancy between $i_{\mathrm{T}} = 0.07$ and $i_{\mathrm{T}} = 1$ in the case of pure rotation is not present for pure translation, i.e. when comparing $i_{\mathrm{CCW}} = 0.07$ and $i_{\mathrm{CCW}} = 1$ for varying i_{T}.

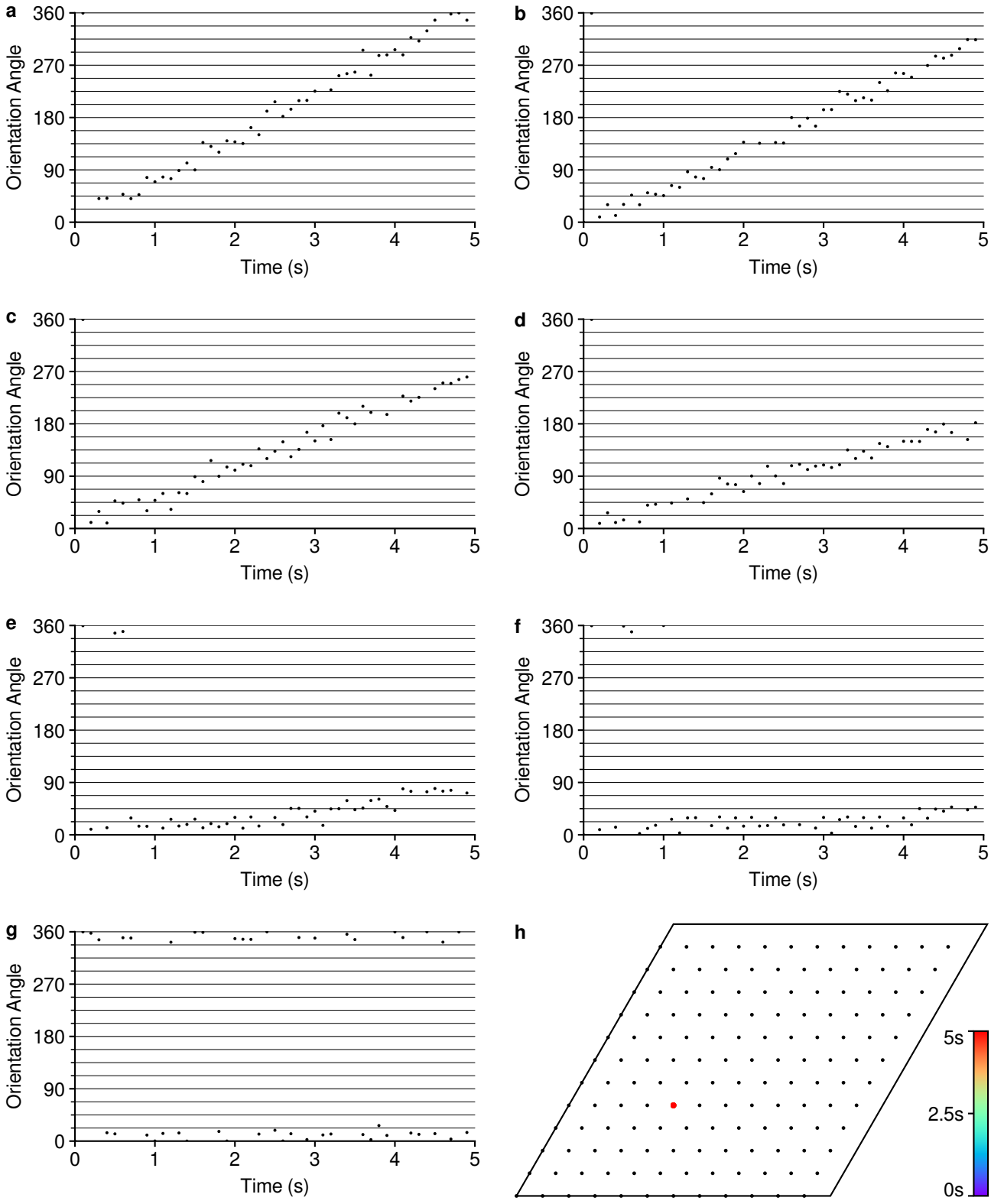

Figure 6.3 Pure counterclockwise rotation with different values for the continuous rotational inhibition i_{CCW}. The two parameters i_{T} and i_{CW} are set to 1, so that the translation and clockwise rotation modules do not have any influence on the main attractor. All plots show a time range of 5s, for which the inputs are kept constant. The bump is always initialized at position $\mathrm{P}(0) = (4, 4, 0)_{\mathrm{g}}$. **a**, Maximum angular velocity at about $1/5$ rotations per second, with $i_{\mathrm{CCW}} = 0$. **b**, $i_{\mathrm{CCW}} = 0.02$. Demonstrates the capability of capturing small changes of the angular velocity input near the system's upper representable limit. **c**, $i_{\mathrm{CCW}} = 0.03$. **d**, $i_{\mathrm{CCW}} = 0.04$. **e**, $i_{\mathrm{CCW}} = 0.05$. **f**, $i_{\mathrm{CCW}} = 0.055$. Shows that small input changes can also be followed close to rotational standstill. **g**, The system shows no rotation behavior at $i_{\mathrm{CCW}} = 0.065$ and above. **h**, As desired, there is zero translational movement, independent of the angular velocity.

6.1.3 Circular Movement With Constant Velocities

Still keeping inputs constant for the duration of the simulation, we now set the inhibition values so that both the shift module and the counterclockwise rotation module are active simultaneously. The result are circular paths with a constant radius depending on the particular combination of shift inhibition and rotation inhibition. As in the previous section, we fully disable the other rotation module with $i_{\mathrm{CW}} = 1$, since clockwise circular movements are perfectly analogous to counterclockwise ones. Figure 6.4 contains related experiments, all of which have a duration of 5 seconds. Overall, we can see the tendency that increasing i_{T} while keeping i_{CCW} constant yields a smaller turn radius and, naturally, a higher tangential velocity v, whereas increasing i_{CCW} while keeping i_{T} constant results in curves of larger radius and lower angular velocity ω. In the latter case, the arc lengths for the different i_{CCW} are similar, but tend to be slightly longer for smaller i_{CCW}. The achievable ranges for tangential and angular velocities depend on the desired turn radius. Also, for each value of i_{T} or i_{CCW} there is some threshold for the other, where we recover pure translation, pure rotation, or no movement at all. Specifically, as the rotation inhibition becomes larger, the circular movement transitions to linear movement. With $i_{\mathrm{T}} = 0.06$ and above we get pure rotation, whereas with $i_{\mathrm{CCW}} = 0.07$ and above we get pure translation or no movement. We demonstrate the near-smallest practical radius of about 0.08 in Fig. 6.4a. It is obtained with $i_{\mathrm{T}} = 0.04$ and $i_{\mathrm{CCW}} = 0.01$. While achieving an even smaller turn radius of about 0.04 is technically possible, using e.g. $i_{\mathrm{T}} = 0.05$ and $i_{\mathrm{CCW}} = 0.01$, it is very close to the system's limits and, therefore, potentially not as consistent in various situations as a slightly larger radius. For comparison, the largest radius of about 1.1 is shown in Fig. 6.4b. Here, we use $i_{\mathrm{T}} = 0$ and $i_{\mathrm{CCW}} = 0.06$. Further, we include examples of same radii but different tangential and angular velocities (Fig. 6.4c and d), same tangential speed but different radii and angular velocities (Fig. 6.4d and f), and same angular velocity but different radii and tangential velocities (Fig. 6.4e and f).

Overall, we witness good repeatability, meaning that with constant inputs the bump circles back to its starting position rather accurately. This holds for a large portion of the parameter space. Note that, for large i_{T} and comparatively small i_{CCW}, the likelihood for undesired behavior might be higher. For example, $i_{\mathrm{T}} = 0.04$ and $i_{\mathrm{CCW}} = 0$ gives a circular movement where the pose estimate overshoots upon return to the start position. This is the largest observed deviation from expected behavior of all circular movement experiments.

6.2 Multiple Simultaneous Pose Estimates

Next up are scenarios where more than one pose estimate need to be sustained and propagated at the same time. The behavior of the individual bumps is equivalent or similar to the previously discussed single-bump experiments, unless bumps get too close to each other - then they interfere and potentially deviate from the target behavior. Here, closeness is defined as the Euclidean distance between two bumps in the three-dimensional x-y-θ-space. An important factor determining the threshold distance at which signs of interference start to become visible is the width of the Mexican hat function for computing the recurrent connection weights (see Sec. 5.4). In short, with a wider excitatory center more neurons are involved in representing a single bump and, consequently, the neural activity associated with two individual bumps overlaps sooner, i.e. already at a greater distance apart. So, to fit more simultaneous bumps into the unit domain, the excitatory part of the Mexican hat function can be narrowed. This comes at a trade-off with general representational stability and the capability to represent pose estimates at locations between positions associated to neurons - to improve those aspects, more active neurons per bump are necessary. Note that the width of the excitatory center region can only be varied within some feasible interval, which is implicitly determined by the neural response curves (see Sec. 5.3).

For the final weights presented in Sec. 5.4 we separately consider the system's behavior with multiple simultaneous bumps that do not interfere, as well as with bumps that interfere.

6.2.1 No or Little Interference

When two bumps are far enough apart, they do not interfere and behave exactly as if they were on their own. This holds for all relative positions, but only as long as they remain at a certain distance from each other.

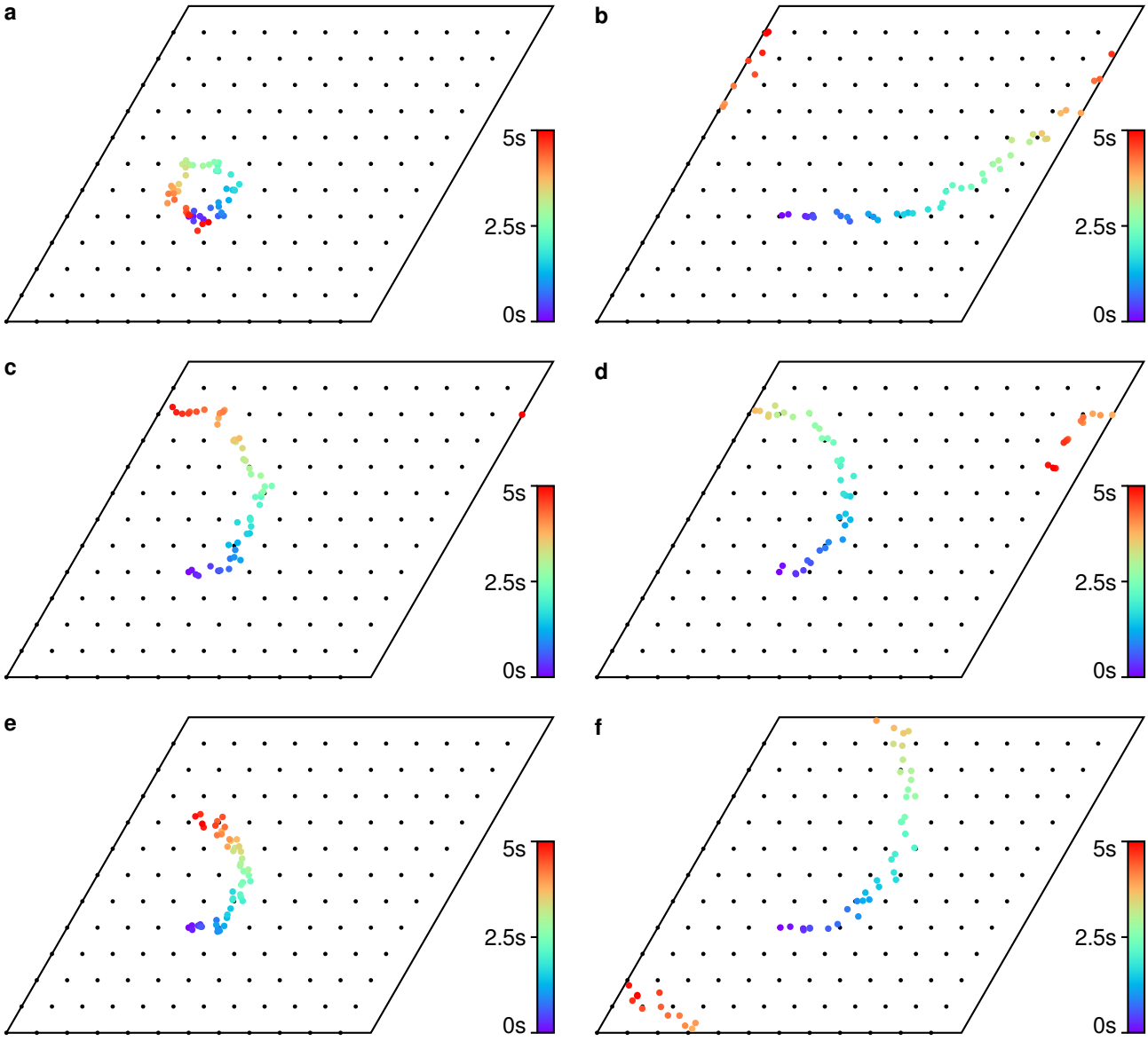

Figure 6.4 Circular movement with varying turn radii, tangential velocities, and/or angular velocities is obtained using different combinations of i_{T} and i_{CCW}. The inputs are kept constant over the entire simulation duration of 5 seconds. **a**, Near-smallest practical turn radius of about 0.08. $i_{\mathrm{T}} = 0.04$ and $i_{\mathrm{CCW}} = 0.01$. **b**, Maximal radius of about 1.1. $i_{\mathrm{T}} = 0$ and $i_{\mathrm{CCW}} = 0.06$. **c**, $i_{\mathrm{T}} = 0.03$ and $i_{\mathrm{CCW}} = 0.04$. **d**, $i_{\mathrm{T}} = 0.02$ and $i_{\mathrm{CCW}} = 0.03$. **c**, **d**, Same turn radii, with different tangential and angular velocities. **e**, $i_{\mathrm{T}} = 0.04$ and $i_{\mathrm{CCW}} = 0.04$. **f**, $i_{\mathrm{T}} = 0.01$ and $i_{\mathrm{CCW}} = 0.04$. **d**, **f**, Same tangential velocity, with different radii and angular velocities. The bumps travel the same absolute distance in planar space. **e**, **f**, Same angular velocity, with different radii and tangential velocities. The bumps cover the same angular distance.

For bumps at the greatest possible distance apart in the same θ-plane, this is demonstrated in Fig. 6.5a for linear movement, and in Fig. 6.5b for circular movement. Similarly, Fig. 6.5e shows three bumps being initialized at the pairwise-greatest distance apart in the same θ-plane and being translated interference-free. Figure 6.5f shows the analogous experiment with circular movement. When talking about greatest possible distance between two bumps, we need to consider the twisted nature of our toroidal representation space. Intuitively, when looking at the rhombus, as which we show the representation space in figures, one might think that the point farthest from $(0, 0, 0)_{\mathrm{g}}$ is $(6, 6, 6)_{\mathrm{g}}$. However, since $(0, 0, 0)_{\mathrm{g}}$, $(12, 0, 0)_{\mathrm{g}}$, $(0, 12, 0)_{\mathrm{g}}$, and $(12, 12, 0)_{\mathrm{g}}$ are actually the same point in our x-y-θ-space, the farthest point from $(0, 0, 0)_{\mathrm{g}}$ is $(4, 4, 6)_{\mathrm{g}}$.

Even for pose estimates existing a bit closer together, their behavior is nearly identical to the corresponding single-bump scenarios. Figure 6.5c

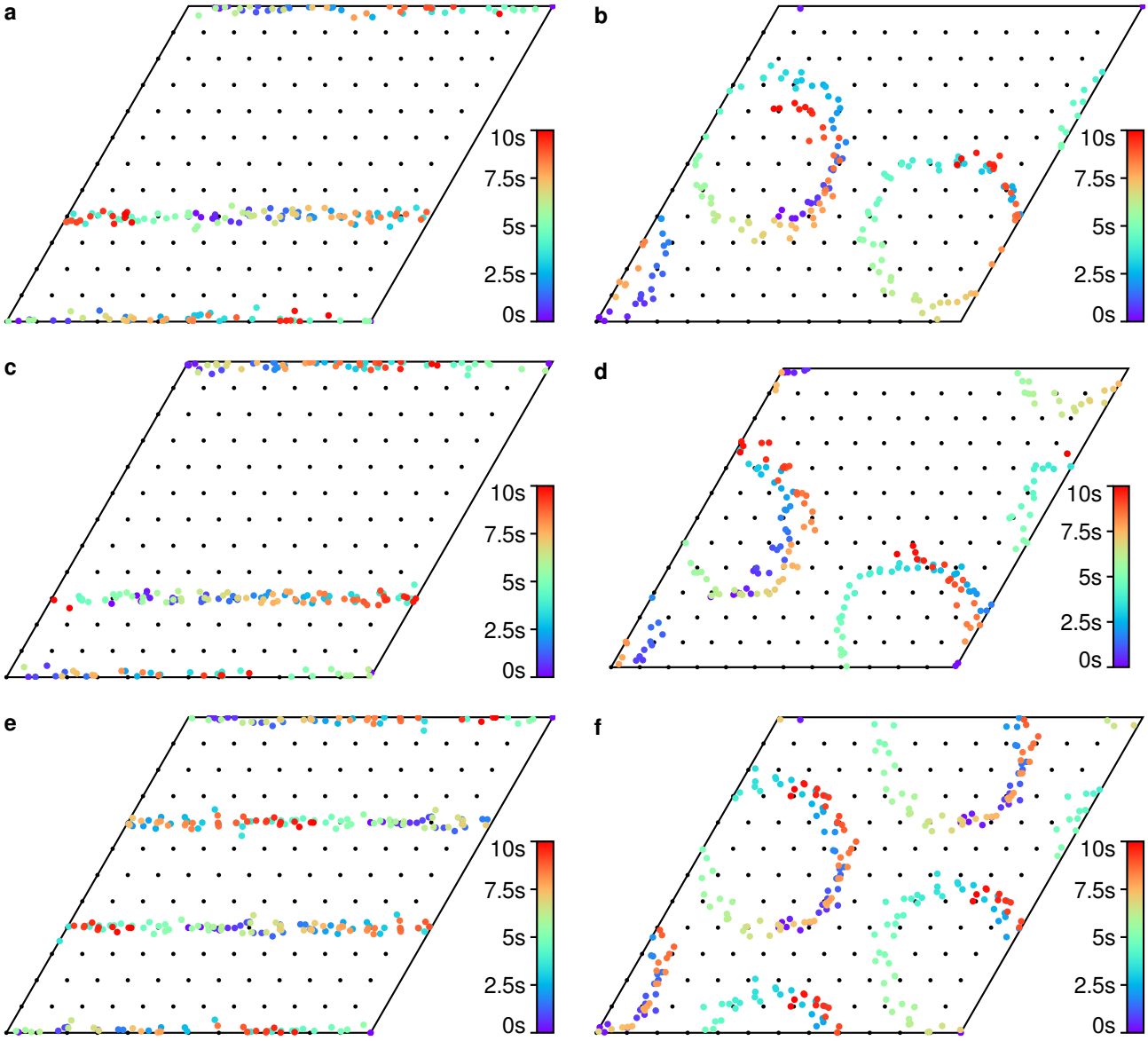

Figure 6.5 Multiple simultaneous pose estimates can coexist and be propagated correctly without any major interference between them. All simulations are run for 10 seconds. For straight movement, we use $i_{\mathrm{T}} = 0.02$, while for circular movement we use $i_{\mathrm{T}} = 0.02$ and $i_{\mathrm{CCW}} = 0.03$; all other inhibition values are set to 1. The bumps per experiment are indexed starting from 0, so that $\mathrm{P}_i(t)$ denotes the position of the i-th bump at time t. Here, we only consider cases where all bumps are initialized in the same θ-plane but at different distances apart. **a**, **b**, Two bumps are initialized at the largest distance apart, $\mathrm{P}_0(0) = (0,0,0)_{\mathrm{g}}$, $\mathrm{P}_1(0) = (4,4,0)_{\mathrm{g}}$. **a**, Both bumps are propagated properly towards the East. **b**, Circular movement is as desired. **c**, **d**, Two bumps are initialized closer together, $\mathrm{P}_0(0) = (0,0,0)_{\mathrm{g}}$, $\mathrm{P}_1(0) = (2,3,0)_{\mathrm{g}}$. **c**, Both bumps are still propagated towards the East. Specifically, bump 1 is not pushed North. Note that there appears to be some minor interference since the velocity is a bit lower compared to **a**. **d**, The circular movement behaves similarly to **b**, but bump 0's trajectory seems to be pushed slightly southward by the nearby bump 1. **e**, **f**, Three bumps are initialized so that each bump is equal distance apart from the other two, $\mathrm{P}_0(0) = (0,0,0)_{\mathrm{g}}$, $\mathrm{P}_1(0) = (4,4,0)_{\mathrm{g}}$, $\mathrm{P}_2(0) = (8,8,0)_{\mathrm{g}}$. **e**, We observe ideal straight movement towards East. The trajectories of bumps 0 and 1 are nearly identical to **a**. **f**, As in the straight movement case, there are no signs of interference and the trajectories of bumps 0 and 1 are very similar to the related two-bump experiment in **b**.

shows this for linear movement, and Fig. 6.5d addresses the case of circular movement.

Importantly, even if two (or more) bumps are initialized far enough apart, they may get close due to two reasons. One is that they are on different θ-planes and approach each other in x-y-space after certain movement patterns. The second one comes from wrap-around in the twisted-torus-like representational space.

6.2.2 Interference Between Bumps

Simultaneous pose estimates that are too close to each other may deviate from the expected behavior. How close two bumps can get before interference starts depends largely on the parameterization of the connection weights. Specifically, the wider the excitatory center portion of the Mexican hat function used for sampling the weights, the larger the threshold distance at which interference begins. Also, in theory, the deeper the inhibitory ring of said function, the more severe the interference behavior will be.

Generally, the form of interference behavior is situational and depends on the relative position of the involved bumps. Intuitively, the following holds. If the bumps are on the same θ-layer, then they will deviate from their target trajectory in the x and y dimensions but not in the θ-dimension. If the bumps are on different layers, then they will additionally be pushed up or down along the θ-dimension, respectively. If one bump is traveling on or along a grid axis and the other one is traveling off-axis, there is a chance that the axis-aligned bump will remain on its designated path while the other one is deflected. Similarly, but still different, if one bump is traveling in line with neurons and the other one is traveling between neurons, the in-line bump will assert more influence on the not-in-line bump than vice versa. The described two cases of one bump being in a stronger position than the other can be explained by the involved attractor dynamics. Points in the representation space that are aligned with neurons' tuned locations have a higher attractive force than other points. If there is a third bump involved, the deviation behavior is difficult to predict. For example, when the bumps get close but are on different θ-layers and at the same x-y-positions, the lowest bump might get pushed downwards, the uppermost bump might get deflected upwards, and the center bump might remain at it's desired location.

In Fig. 6.6 we convey an overview of the system's behavior in various interference situations, thereby covering some of the aforementioned forms of interference. Figures 6.6a and 6.6c contain straight line experiments with two and three bumps, respectively, moving in different directions where interference primarily shows as one bump being deflected downwards, i.e. making a right turn, and another bump being deflected upwards, i.e. making a left turn. The 10-second simulation of Fig. 6.6c is analyzed in more detail in Figs. 6.6e-6.6h. Analogously, Figures 6.6b and 6.6d show circular trajectories of two and three simultaneous bumps initialized at different orientations, respectively. Situations of turn radii of interfering pose estimates being decreased or increased, depending on the relative angle at which the involved bumps approach each other, are covered. Figures 6.6i-6.6l illustrate the experiment of Fig. 6.6d in more detail.

It is evident, that after all situations of interference the bumps return to regular and expected behavior. This demonstrates a certain stability of the path integration system.

To quantify the magnitude of interference, we isolate single variables in the experiment setup and only vary one of those at a time. Figures 6.7a-6.7d show two bumps being initialized in the same θ-layer at different but fixed y-positions and varying x-positions, i.e. varying relative x-distances from each other, and moving North on straight paths. As the starting point of bump 1 shifts closer to bump 0, interference in the form of slowing bump 0 down and speeding bump 1 up begins to show more and more. Furthermore, there is some slight repelling between the bumps in x-direction. However, even at the smallest distance apart, both bumps roughly follow their target trajectory. A similar series of experiments is depicted in Figs. 6.7e-6.7h. Both bumps are initialized at the same y-position, while the x-position of bump 1 is moved closer to bump 0 with each experiment. The smaller the distance between the bumps, the stronger their interference in form of repelling in x-direction. The movement in y-direction is unaffected. When being initialized only 0.25 units apart, the bumps merge rapidly into a single bump centered between the original starting positions, which then properly travels North on a straight path. The merging process is depicted in detail in Fig. 6.8. It can be seen that the readout via clustering identifies the two bumps as one bump right from the beginning, i.e. before they have merged. When the two bumps start in line with respect to their heading direction, i.e. at the same x-coordinate, but at varying y-distances apart, only their tangential velocities are affected by interference (see Figs. 6.7i-6.7l). Specifically, bump 0, which is technically behind bump 1, is slowed down slightly, whereas bump 1 is sped up slightly. The closer the starting positions, the more prominent this effect becomes. When starting only about 0.15 units apart, the bumps merge into a single bump

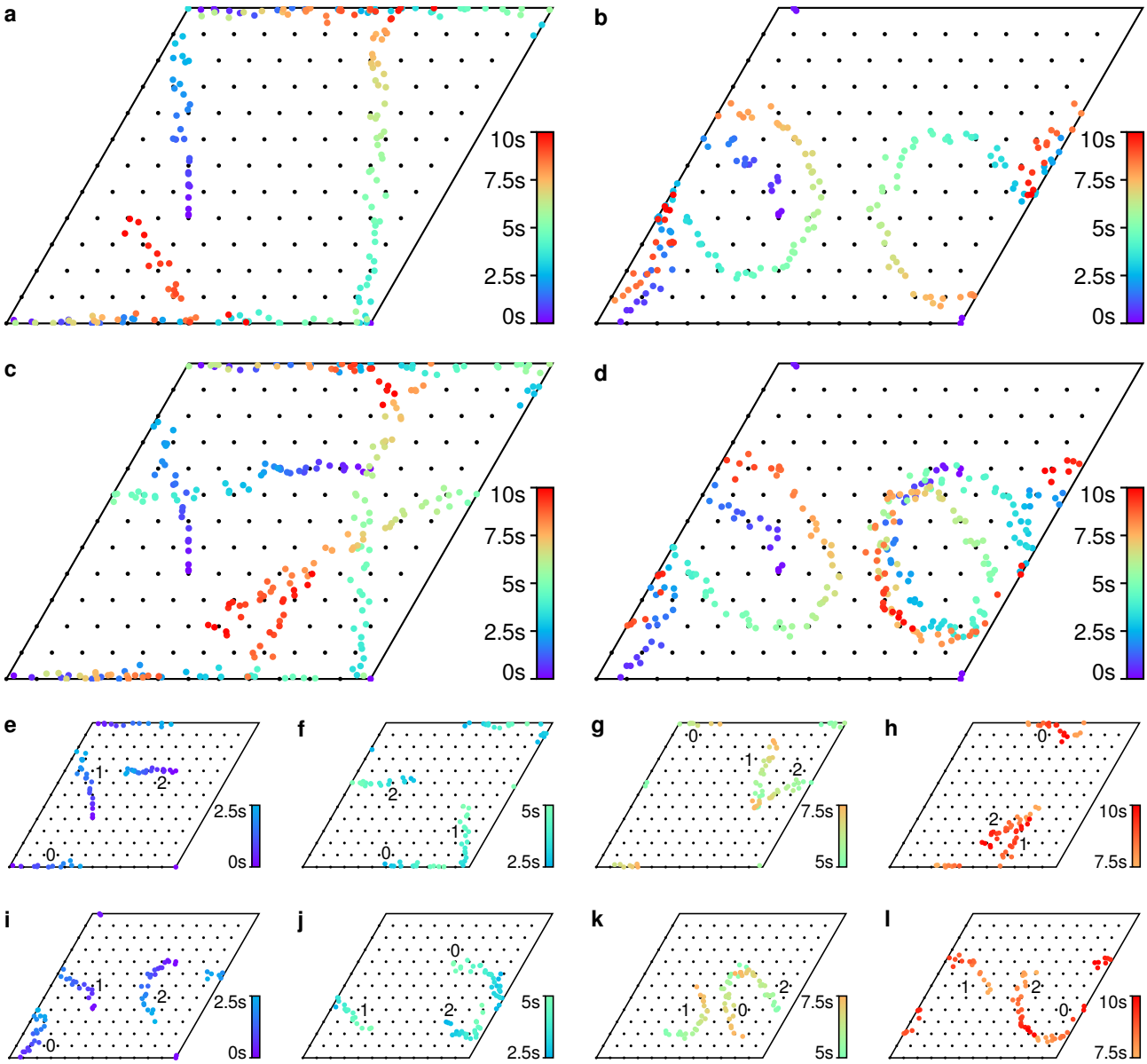

Figure 6.6 Multiple simultaneous pose estimates may interfere if they get within a certain proximity of each other. How close two bumps can get before interfering depends largely on the width of the excitatory center of the Mexican hat weight function used here. All simulations are run for 10 seconds. For straight movement, we use $i_{\mathrm{T}} = 0.02$, while for circular movement we use $i_{\mathrm{T}} = 0.02$ and $i_{\mathrm{CCW}} = 0.03$; all other inhibition values are set to 1. The bumps per experiment are indexed starting from 0, so that $\mathrm{P}_i(t)$ denotes the position of the i-th bump at time t. **a**, **b**, Two bumps are initialized at the largest x-y-distance apart, one facing East, $\mathrm{P}_0(0) = (0, 0, 0)_{\mathrm{g}}$, and the other facing North, $\mathrm{P}_1(0) = (4, 4, 3)_{\mathrm{g}}$. **a**, Both bumps are propagated properly until they get close at about 7.5s. Then, the northbound bump is deflected towards the West, while the eastbound one stays in it's lane. The asymmetrical effect is potentially because the eastbound bump is traveling along a grid axis, whereas the other one is not. After the encounter, both pose estimates are propagated along straight lines again. **b**, The bumps are translated correctly for about 2.5 seconds. Then, they get close and bump 0 is pushed downwards, i.e. the turn radius is momentarily larger, while bump 1 is pushed upwards, i.e. the turn radius is briefly smaller. From there on, the two bumps follow trajectories with the correct turn radii and do not interfere again. *(continued on next page)*

between them that is then propagated North as desired.

In Fig. 6.9, we consider another situation of two bumps merging. There, the relative offset between the involved bumps is not purely along the x-dimension, but is along both x- and y-dimension. The resulting bump is still perfectly centered between the two original bumps. Importantly, this centered merging also works for combined x, y, and θ offsets between bumps - they not only merge towards their x-y-center, but also along the third dimension.

Figure 6.6 ***(continued)*** **c**, **d**, Three bumps are initialized at the largest pairwise x-y-distances apart, one facing East, $\mathrm{P}_0(0) = (0, 0, 0)_{\mathrm{g}}$, the next facing North, $\mathrm{P}_1(0) = (4, 4, 3)_{\mathrm{g}}$, and the last facing West, $\mathrm{P}_2(0) = (8, 8, 6)_{\mathrm{g}}$. **c**, The three bumps are mostly propagated in straight lines. Occasionally, two of them get close and interfere. In most of these instances, one bump is deflected downwards while the other is deflected upwards, i.e. they make turns in opposite directions and afterwards continue on straight paths. The various interference situations are analyzed in more detail in **e**-**h**. **d**, The three bumps follow circular paths with occasional pairwise interference where one bump's turn radius is decreased, i.e. it takes a sharper turn, and another bump's turn radius is increased, i.e. it takes a wider turn. In **i**-**l** we look at the interactions between bumps in more detail. **e**-**h**, The trajectories of the three bumps in **c** are split into four equal-length parts and plotted individually. Interference situations can be explained more clearly in this manner. **e**, Bump 1 and 2 interfere slightly: 1 is slightly pushed upwards along the θ-dimension, i.e. is deflected towards the West; whereas 2 seems to get a bit slowed down but stays on it's designated path (potentially because 2 travels along a grid axis while 1 travels off-axis). **f**, All bumps are far enough apart to not influence each other noticeably. **g**, Bumps 1 and 2 get close again: 1 is pushed downwards, i.e. is deflected to the East; whereas 2 is pushed upwards, i.e. is deflected to the South. **h**, Bumps 0 and 1 interfere slightly: 0 deviates slightly towards South the end of its trajectory; while 1 seems to stay on its current path. Bump 2 might also be mildly involved here. **i**-**l**, The trajectories of the three bumps in **d** are split into four equal-length parts and plotted individually. **i**, Bumps 0 and 1 get close and start to interfere at about 2.5 seconds. **j**, Bump 0 is pushed slightly downwards, while bump 1 is pushed upwards and performs a rather sharp turn just after 2.5 seconds. It further seems that bumps 1 and 2 interfere a bit, since bump 2 performs a sharper-than-expected turn. **k**, When considering the three-dimensional representational space, the bumps stay relatively far apart at all times. Hence, they travel on their respective circular paths without any apparent interference. **l**, Bumps 1 and 2 get close at about 7.5 seconds: 1 is deflected slightly downwards along the theta-dimension, while 2 is pushed upwards. Other than that, the three bumps follow circular paths of desired radii.

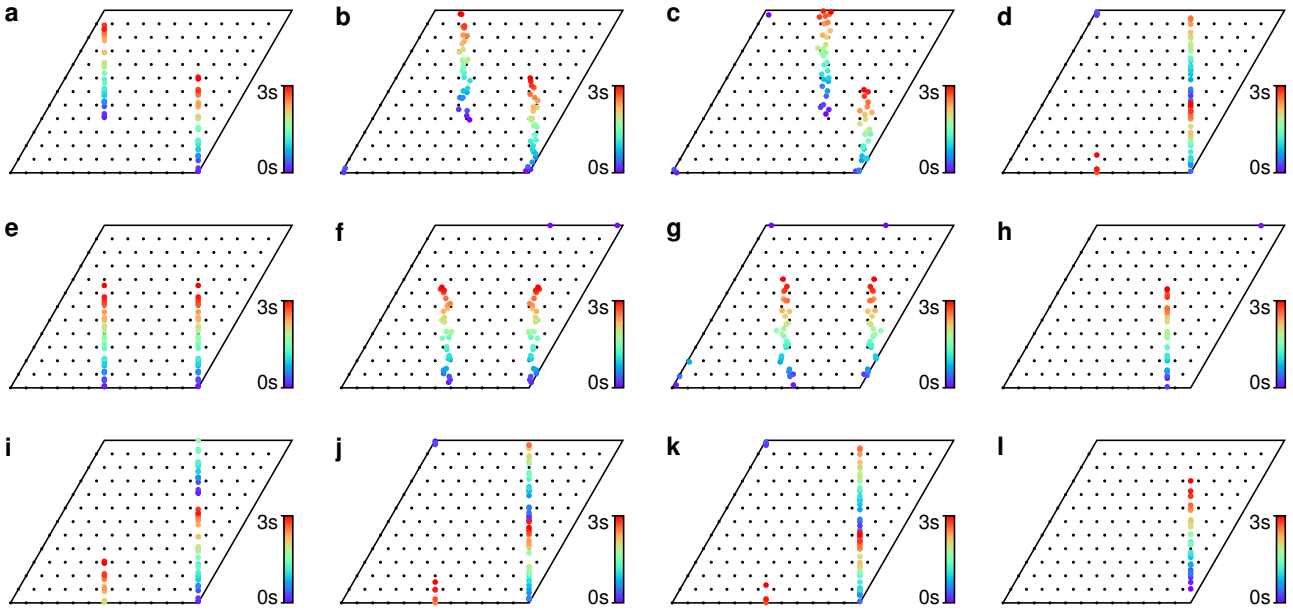

Figure 6.7 The magnitude of interference between two bumps depends on the distance between them as well as the relative positioning. To demonstrate this, three series of four experiments each are shown. All simulations consider pose estimates at $\theta = 90°$ and 3 seconds of straight movement with $i_{\mathrm{T}} = 0.02$; the other inhibition values are set to 1. The bumps per experiment are indexed starting from 0, so that $\mathrm{P}_i(t)$ denotes the position of the i-th bump at time t. **a**-**d**, The two bumps are initialized at $\mathrm{P}_0(0) = (0, 0, 3)_{\mathrm{g}}$ and $\mathrm{P}_1(0) = (k, 4, 3)_{\mathrm{g}}$, where k is increased in 2-steps from 4 to 10, i.e. the starting point of bump 1 is moved along the x-dimension closer to bump 0's starting point $\mathrm{P}_0(0)$. As the second bump starts closer to the first one, they interfere more. Specifically, bump 0 gets slowed down, whereas bump 1 gets sped up. This is because bump 1 is in front of bump 0 when looking in their movement direction and considering that the North edge wraps around to the South edge with an offset. Note that in all four cases the interference is minor and the bumps essentially behave as desired. **a**, Baseline with maximum x-y-distance between the bumps. **b**, Compared to the baseline, bump 0 is slowed down slightly, while bump 1 is sped up slightly. Both bumps still move North on straight paths. **c**, The slowing down and speeding up of the two bumps, respectively, is a bit more pronounced, but still minor. The bumps appear to repel each other a bit in the x-dimension, with this effect being more prominent on bump 0 since it is technically behind bump 1. *(continued on next page)*

Figure 6.7 ***(continued)*** **d**, Both bumps start in line. Compared to the baseline, the slowing down of bump 0 and the speeding up of bump 1 are more clearly visible. Even in this closest starting scenario of the experiment series both bumps are propagated near-target. **e**-**h**, The two bumps are initialized at $\mathrm{P}_0(0) = (0, 0, 3)_\mathrm{g}$ and $\mathrm{P}_1(0) = (k, 0, 3)_\mathrm{g}$, where k is increased in 1-steps from 6 to 9. Since both bumps always start at the same y-position, their tangential velocity is not affected by the interference. The closer they get, the more they repel each other in x-dimension, until they actually merge when initialized 0.25 units apart. **e**, Baseline with both bumps initialized at the greatest x-y-distance apart. The bumps remain precisely in their designated lane. **f**, Pretty much equivalent to the baseline. **g**, The bumps repel each other noticeably, i.e. bump 0 is pushed right and bump 1 is pushed left. **h**, Both bumps rapidly merge into a single one right in the middle of the original starting locations. The new bump is propagated correctly North in a straight line. Figure 6.8 provides a detailed illustration of the merging process. **i**-**l**, Bump 0 is initialized at $\mathrm{P}_0(0) = (0, 0, 3)_\mathrm{g}$, while bump 1 starts right in front of it, i.e. at the same x-coordinate, with the y-starting position getting closer to bump 0 for each successive experiment; until they merge when initialized about 0.15 units apart. As the bumps start closer together, bump 0 is slowed down more and bump 1 is sped up more. As expected, no deviation from the target trajectory is observed along the y-dimension. **i**, Baseline with both bumps at the greatest x-y-distance apart, $\mathrm{P}_1(0) = (8, 8, 3)_\mathrm{g}$. The bumps do not interfere and behave nearly identical to **a**. **j**, $\mathrm{P}_1(0) = (9, 6, 3)_\mathrm{g}$. Bump 0 is slowed down slightly, while bump 1 is sped up slightly. **k**, $\mathrm{P}_1(0) = (10, 4, 3)_\mathrm{g}$. The slowing-down and speeding-up effects, respectively, are more pronounced. Nonetheless, the bumps still follow their target trajectory rather closely. **l**, Bump 1 starts only about 0.15 units from bump 0, $\mathrm{P}_1(0) = (11, 2, 3)_\mathrm{g}$. Right after or even during the initialization phase both bumps merge into a single bump centered between them.

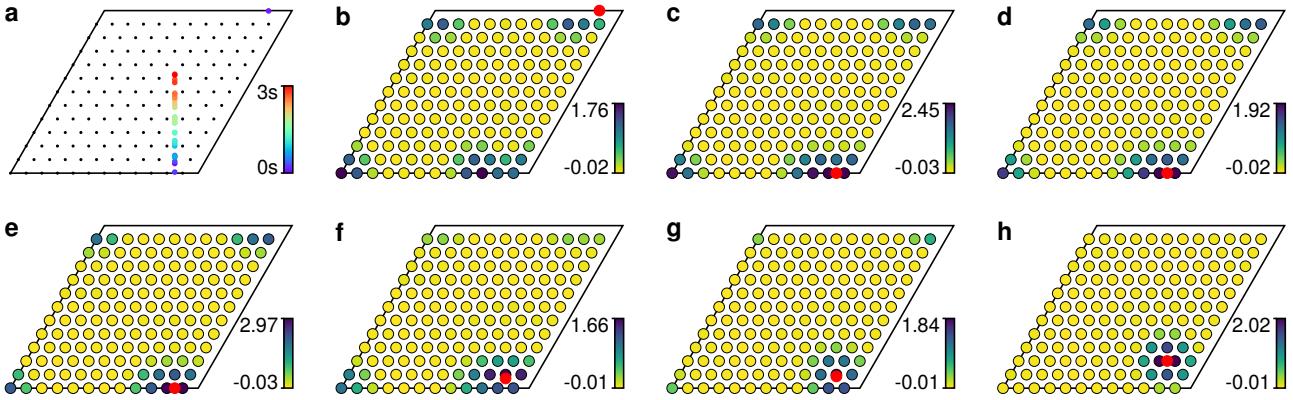

Figure 6.8 Merging of two bumps. One starts at $(0, 0, 3)_\mathrm{g}$, the other at $(9, 0, 3)_\mathrm{g}$, i.e. they are supposed to move into the same direction. Due to their close start positions and the recurrent connection weight shape, the two bumps merge quickly into a single bump centered right in between the two. The resulting bump moves straight north, as intended. The start of the actual trajectory, i.e. onset of the velocity input, is defined at $t = 0$. **a**, Trajectory of the single merged pose estimate. **b**-**h**, Snapshots of the reconstructed pose estimate volume, projected along the z-dimension with the max function. The bump centers obtained by clustering are shown in red. **b**, Two distinct bumps are visible at their starting locations. The clustering identifies only one bump, $t = -0.6$. **c**, $t = -0.5$. **d**, $t = -0.3$. **e**, The two pose estimates have fully merged, $t = -0.1$. **f**, Onset of the velocity input and northbound movement, $t = 0$. **g**, $t = 0.3$. **h**, $t = 0.8$.

6.3 Trajectory Following

In this section, we evaluate the path integration system on tangential and angular velocity data recorded from two different mobile robots. The goal is to demonstrate the ability of following realistic continuous velocity input series.

The recorded velocity data is noisy, contains potential outlier points, has sharp rising and falling edges at non-zero velocity sections, and features periods of rapid fluctuation between 0 and some non-zero value range. Therefore, a certain amount of error accumulation due to inert dynamics is inevitable. This is actually a general issue with path integration systems. Thus, we only consider a limited time slice of any available input data at a time, i.e. per simulation. For longer total durations, some form of reset mechanism needs to be deployed to achieve satisfactory performance.

Regarding representable input ranges, we take the conservative maxima of $0.223\,\mathrm{u/s}$ for the tangential velocity v, and $1.256\,\mathrm{rad/s}$ for the absolute angular velocity $|\omega|$. The lower bound for v is 0. These limits ensure that the path integration sys-

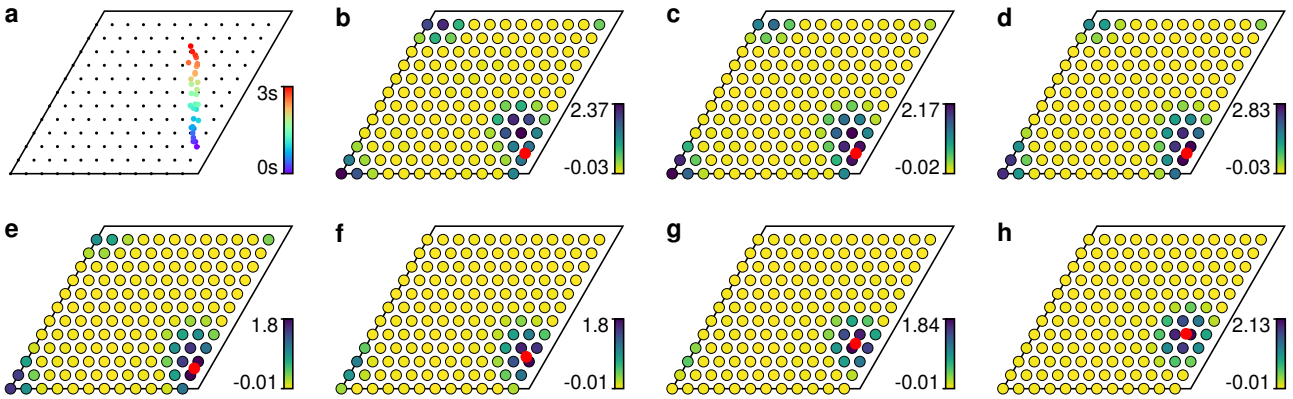

Figure 6.9 Merging of two bumps. One starts at $(0, 0, 3)_{\mathrm{g}}$, the other at $(10, 3, 3)_{\mathrm{g}}$, i.e. they are supposed to move into the same direction. Due to their close start positions and the recurrent connection weight shape, the two bumps merge quickly into a single bump centered precisely in between the two. The resulting bump moves straight north, as intended. The start of the actual trajectory, i.e. onset of the velocity input, is defined at $t = 0$. **a**, Trajectory of the single merged pose estimate. **b**-**h**, Snapshots of the reconstructed pose estimate volume, projected along the z-dimension with the max function. The bump centers obtained by clustering are shown in red. **b**, Two distinct bumps are visible at their starting locations. The clustering identifies only one bump, $t = -0.6$. **c**, $t = -0.5$. **d**, $t = -0.4$. **e**, The two pose estimates have fully merged, $t = -0.1$. **f**, Ongoing velocity input and northbound movement, $t = 0$. **g**, $t = 0.5$. **h**, $t = 0.8$.

tem can, in theory, represent any input combination of tangential and angular velocities.

The target trajectories are computed by Euler integration of the provided velocities. Given the current ideal pose $\mathrm{P}_I(t) = [x, y, \theta]$ at time t, the next pose at time $t+\Delta t$ is obtained as $\mathrm{P}_I(t+\Delta t)[0{:}2] := \mathrm{P}_I(t)[0{:}2] + (\cos(\theta(t)), \sin(\theta(t))) \cdot v(t) \cdot \Delta t$ and $\mathrm{P}_I(t + \Delta t)[2] := \theta(t) + \omega(t) \cdot \Delta t$, where $\theta(t)$ is the current orientation angle, $v(t)$ is the tangential velocity, and $\omega(t)$ is the angular velocity. The durations Δt between the velocity inputs are variable. Note that multiplication of a tuple and a scalar is assumed to be the same as multiplying each element of the tuple by that scalar, resulting in a new tuple. The indexing notation is adopted from Python. For example, $[0{:}2]$ includes all elements from index 0 (included) up to index 2 (excluded), so effectively the two elements at index 0 and 1.

Krishna et al.[38] mention the destabilization of grid cell activity for heavily fluctuating head direction inputs. Based on this, they argue for smooth rat-like trajectories instead of completely random ones. While our system would not suffer from destabilization when provided completely random velocity input, including heavily fluctuating head direction changes, it would simply not move at all due to the inert dynamics being unable to follow the high-frequent inputs. Compared to destabilization behavior this is likely preferable, but it does still not make sense to benchmark it against perfect integration. Therefore, we only consider smooth non-random trajectories in this section.

Specifically, the first subsection deals with acceleration and velocity data recorded from an inertial measurement unit (IMU) on a wheeled mobile robot driving around in a small maze. The data is preprocessed by some noise filtering, clipping, and time scaling. In the second subsection we use raw odometry data, i.e. tangential and angular velocities, obtained from a mouse-shaped robot moving in a miniature road system that has the shape of Australia. Overall, we report satisfactory performance with a truthful representation of relative distances and properly captured topological features such as curves and turns.

As mentioned in the beginning of this chapter, we use the absolute trajectory error (ATE) as well as the relative pose error (RPE) to quantify the performance of our path integration system. To recap in different words, for any time point t, the ATE is defined as the Euclidean norm of the vector from the current position estimate to the desired position on the ideal trajectory at t. Intuitively, whenever the current slope is 0, the path integration perfectly represents the changes induced by velocity input. This is primarily the case for linear segments where the actual and target head directions are aligned, or for time periods of standstill. The RPE for any time interval from t to $t + 1$ is defined as the difference in (translational) motion within that interval between

the ideal trajectory and actual trajectory. This value represents the drift per time step.

6.3.1 Wheeled Robot in Maze

Here we work with data obtained from a wheeled mobile robot. There are four motors - one per wheel. These can be controlled individually via a Raspberry Pi and a motor controller. Hence, very tight turn radii, even turning on the spot, are theoretically possible. Note that Fieweger [22] reports some issues with spinning wheels and small turns due to the quality of the physical components. This needs to be considered when working with any data obtained from experiments involving this robot.

An IMU sensor is mounted on top of the robot and run by a separate controller. It provides linear acceleration and angular velocity information. These are noisy by nature and somewhat erroneous due to imperfect mounting and operating conditions. Hence, they are preprocessed in an appropriate way. The linear acceleration data is integrated to obtain tangential velocities.

We work with the preprocessed linear and angular velocities, which are provided with corresponding time stamps. The duration Δt between subsequent data points varies. Since both the linear and the angular velocities at times leave the calibrated range of our path integration system, we apply time scaling: the time deltas are multiplied by a factor k, while the respective linear and angular velocities are divided by k. This kind of scaling not only fixes out-of-range issues, but potentially also mitigates short input spikes by reducing their amplitude and increasing their duration. Such spikes are difficult for the path integration system to follow due to inert dynamics of the attractor network and the neurons.

To maintain a trajectory's shape and topology while only changing it's overall scale in space, it is enough to use the same factor for the time deltas and for the angular velocities. The tangential velocities could be scaled arbitrarily to further modify the overall scale of the trajectory. The latter trick can be applied to mitigate situations that would otherwise see both the linear and the angular velocity close to their respective absolute maxima and, consequently, would have higher probability of undesired behavior. We tested various factors for scaling the tangential velocities with mixed results: for some factor values the tangential velocities are underrepresented, while for others the angular velocities are underrepresented. Therefore, we only consider uniform time scaling of time deltas, tangential velocities, and angular velocities in this report.

Due to the previously mentioned drift issues of path integration systems, we only consider the first 45 seconds of the available trajectory. With a time scaling factor of $k = 2.5$ this is stretched to about 110 seconds. At this scale the preprocessed velocities take representable values most of the time, i.e. the tangential velocity mostly remains between 0 and 0.223, and the angular velocity remains between -1.256 and 1.256. These are the bounds imposed by the path integration system and the velocity calibration. The final velocities, which are provided as input to the Nengo simulation, are shown over time in Fig. 6.10. Some clipping happens at around 90 seconds, but this is a necessary trade-off. If we wanted to avoid any clipping, a time scaling factor of at least 4 would be required; then the non-zero velocities would be unnecessarily small, and the non-zero angular velocities would be so small that it would be hard for our system to capture. There are multiple potential sources of errors. Most rising and falling edges of non-zero velocity periods are still quite sharp, and there are some spikes in value for each velocity input. These are difficult to follow for the inert dynamics of the path integration system. The fact that the tangential velocities are very large, relative to the maximum of 0.223, during both turns, is also important: it means that the system is operating near or at limit and, therefore, is potentially less accurate.

The corresponding inhibition values of the translation, counterclockwise rotation, and clockwise rotation modules are plotted over time in Fig. 6.11. They roughly look like a scaled version of the absolute velocity values that have been flipped vertically.

In Fig. 6.12, the trajectory estimated by the path integration system is depicted. For direct comparison, the ideal trajectory obtained by Euler integration is included. Evidently, when looking at the three straight segments, the traveled distances are represented quite truthfully. The most significant topological features such as turns are also captured properly in terms of radius and arc length, e.g. the first curve has a smaller radius than the second one. Even though the first turn is directly sandwiched by two points without movement, it is represented well.

In general, periods of 0 velocity input result in perfectly stationary behavior, i.e. non-moving bumps that remain at the same intensity. Specifically, the

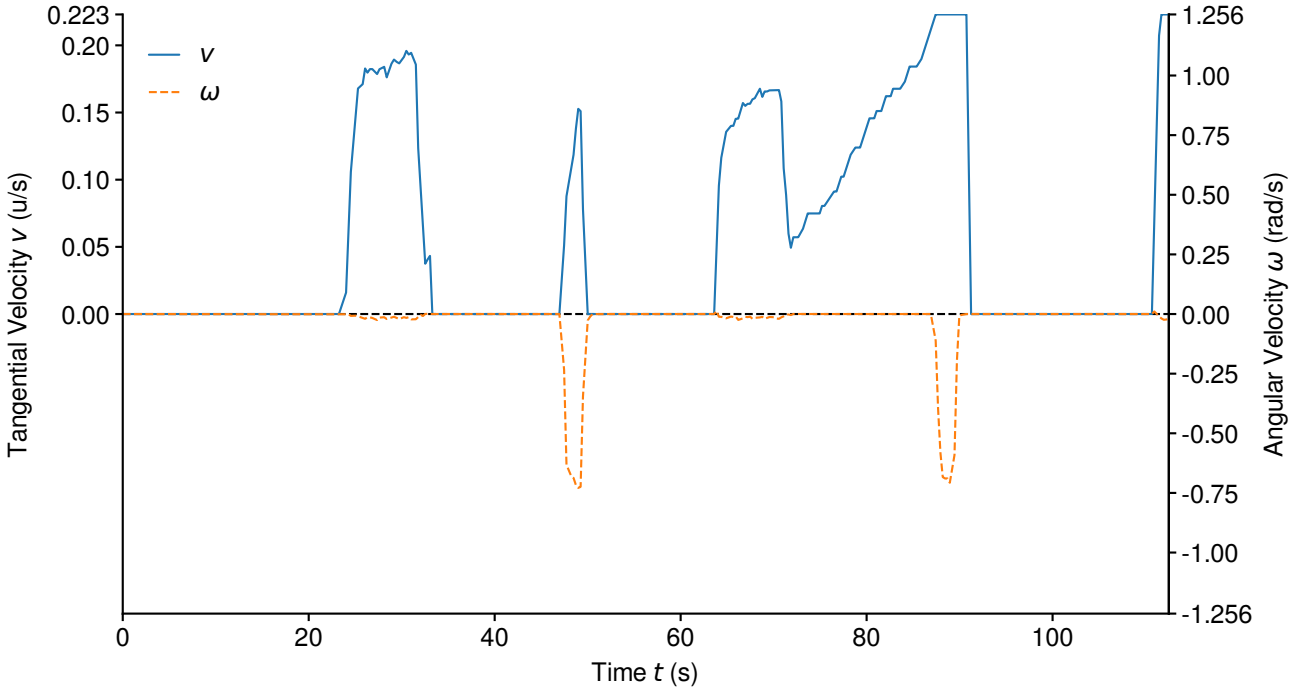

Figure 6.10 Tangential and angular velocities of the maze experiment over time. Both quantities have been preprocessed, time-scaled, and clipped. According to the limits of the path integration system, the tangential velocity v is bounded by 0 and 0.223, and the angular velocity is bounded by ± 0.256. Actual clipping only happens at around 90 seconds for a short time period. Overall, there are multiple long periods without any movement, i.e. zero tangential and angular velocities. The majority of rising and falling edges of non-zero velocity periods are rather sharp. This is a potential cause for deviation from the target trajectory. Furthermore, the tangential velocity is quite large or even maximal for each of the two turn periods.

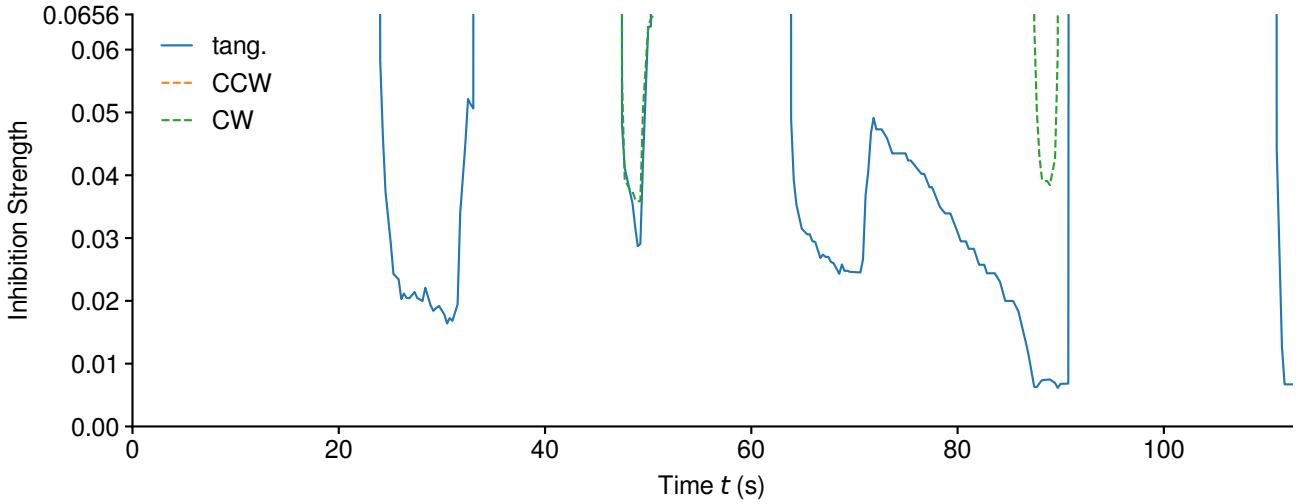

Figure 6.11 Inhibition values for the translation (blue), counterclockwise rotation (orange), and clockwise rotation (green) modules in the maze experiment over time. These quantities correspond to the inhibitory negative input current that is fed to the respective shifting module's neurons. The minimum inhibition values required to prevent the corresponding shifting ensemble from causing any apparent movement in the attractor network are 0.0637 for translation, and 0.0656 for rotation. Note that this does not necessarily mean that the shift modules are fully inhibited, but that they might still slightly influence the behavior of the attractor network. These values are obtained from the velocity calibration step (see Sec. 5.7.2). Any values out of view are equal to 1, i.e. definitively and fully inhibit the shift modules. For example, not a single green data point is visible. This means that, at least in the considered time frame, the robot does not perform any left turns. The upper bounds imposed on the velocities show up briefly at about 90 seconds, and the sharp flanks of the velocity data are also visible here.

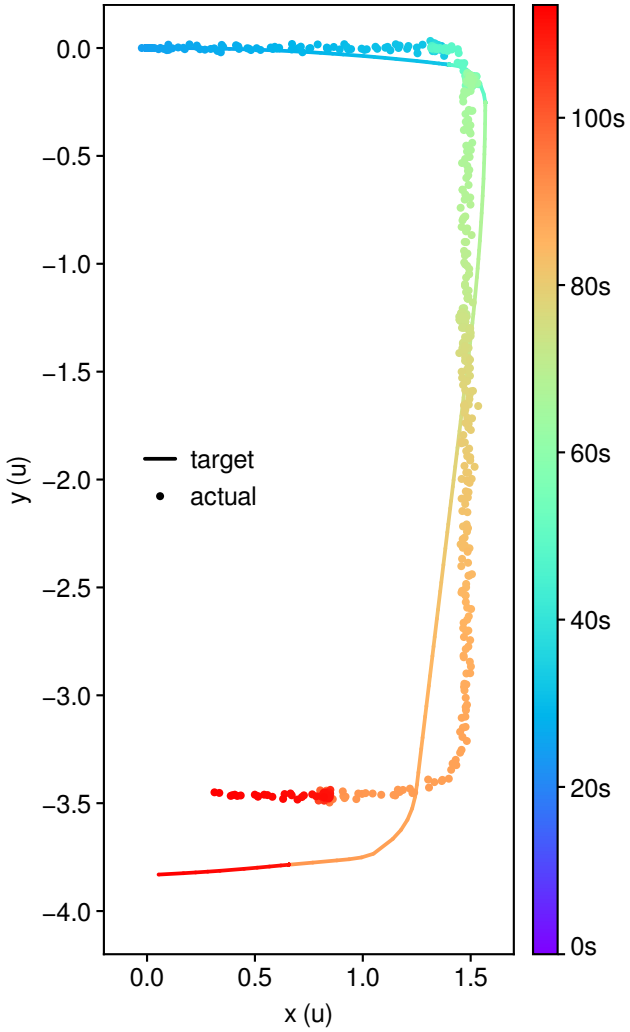

Figure 6.12 Target trajectory and path integration result of the maze experiment. Overall, traveled distances are represented very accurately. The important topological features such as curves and significant bends are also captured properly. For example, turn radii are correct: the first turn is sharper, while the second turn has a larger radius. As dictated by the velocity data, movement only starts after 20s of standstill. This is perfectly captured and demonstrates good stationary stability of the system. Periods of standstill are also very well represented before and after the first turn. Further, the varying tangential velocity between the two turns is reflected in the estimate. These visually described qualities are confirmed when looking at the ATE and RPE values (see later figures). We note, however, that short and/or small velocity inputs or changes thereof are difficult to capture. Instead, the respective velocity component tends to get ignored and any bumps stay on paths that are aligned with grid axes or prefer movement within a θ-layer. For example, while at the end of the second curve the bump is oriented slightly South, it eventually switches to purely westbound movement.

existing bumps are attracted to stable points nearby, i.e. at some neuron's location, and remain there. This shows a general stability of the system.

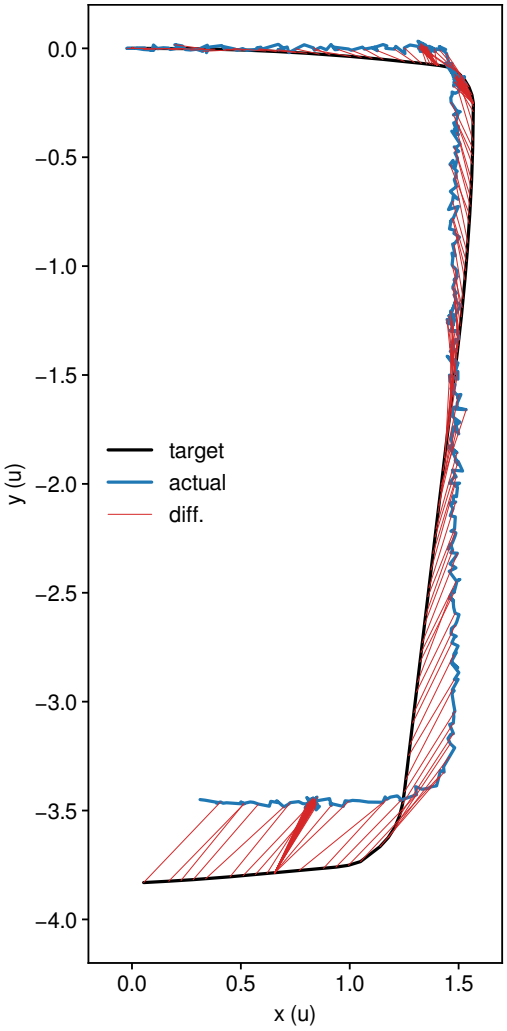

Figure 6.13 Absolute trajectory error (ATE) of the path integration estimate from the target trajectory for the maze experiment. Red lines show the point pairs of target and actual trajectory that have been associated via their time stamp. Periods of standstill are visible as densely clustered red lines.

As expected, short and/or small velocity inputs or changes thereof are not picked up by the path integration system. For example, the bend between start and the first turn is missed. Furthermore, the system tends to prefer movement within a θ-layer as well as along grid axes or at least right in the middle of two grid axes. Long off-axis movement that is supposed to happen in close proximity to a grid axis will likely be drawn to and move along the axis. For example, the long linear segment, while actually slightly curved, snaps to the vertical grid axis.

For a more quantitative view on the system's global performance, consider Fig. 6.13, where the absolute trajectory error (ATE) is highlighted for each point of the estimated trajectory; as well as

Fig. 6.14, where the evolution of the ATE is plotted over time. Phases of constant deviation mostly correspond to standstill. Overall, the absolute deviation remains below 0.5 for the entire duration of the simulation. This is despite sharp flanks and spikes in the velocity input. The largest increase can be attributed to the long straight segment with a slightly misaligned heading orientation. The handful of small jumps likely correspond to the actual trajectory and the target trajectory crossing over. Note that the ATE does not directly represent the heading orientation, but only includes it indirectly. While the ATE informs about the absolute deviation from the target trajectory at any point in time, the relative pose error (RPE) represents the local difference in movement per time step. Hence, it gives insights about the drift behavior of the system. Figure 6.15 shows the RPE over time.

6.3.2 Rodent Robot on Australia Map

The second velocity data set comes from a small mouse-shaped robot driving around in a miniature road system. There are visual landmarks scattered over the map and its outer shape is similar to that of Australia, hence the section name. The mechanical design with two separately controllable wheels allows for tight turns on the spot.

In contrast to Sec. 6.3.1, here we work with the raw and unprocessed odometry data, consisting of timestamps, linear velocities, and angular velocities. These are plotted over time in Fig. 6.16. Again, because of unavoidable drift, we limit the simulation to some time slice of the available data series; specifically, the first 110 seconds. No time scaling is required, as the velocities are essentially fully within the representable ranges. As a reminder, these are 0 to 0.223 for the tangential velocity, and -1.256 to 1.256 for the angular velocity. Not being preprocessed shows in the considerable amount of noise, especially for the angular velocity. The path integration system's inert dynamics are not able to follow such variance and, hence, can be thought of as smoothing its inputs. Thereby, the effective velocity should, in theory, lie closer to the lower bound of an enclosing tube of the noisy data series. In other words, when the input velocity fluctuates rapidly between two values, just like noisy behavior does, our system follows some effective velocity that lies closer to the lower of the two values. Note that this is merely a rule-of-thumb and does not always apply; it depends on the specifics, such as lower value, upper value, and frequency of fluctuation. There are multiple periods where the velocities jump between near-0 and some non-zero value at a high frequency. These correspond to curves in the trajectory. Similar to noise, it is difficult for the inert dynamics to precisely follow such inputs. However, since these jumpy periods are more regular and usually have a sufficient base value to which they fall, the path integration system is able to roughly capture the associated turns. A similar argument can be made for the sharp rising and falling edges at periods of non-zero velocities. While the input cannot be followed exactly, it is still represented well since the inertia of our system is small compared to the involved non-zero input durations. Furthermore, the angular velocity wiggles around 0 most of the time. This makes it harder to represent accurately. The fact that the tangential velocity drops whenever the angular velocity increases in absolute value contributes to the system being able to follow the noisy, spiky, and inherently small angular velocities at turns.

Figure 6.17 shows the inhibition values corresponding to the velocity input at hand. Compared to the previous subsection, they have a significantly higher variance. Still, the vertically mirrored image of the absolute velocities is apparent. The rotation inhibition values are close to their maximum of 0.0656 most of the time, if they are not 1. This is due to the angular velocity wiggling around 0.

The trajectory estimated by the path integration system is plotted in Fig. 6.18. The ideal trajectory obtained by Euler integration is shown for comparison. Relative traveled distances are captured truthfully. Because of the aforementioned issue of underrepresentation of noisy or otherwise rapidly fluctuating velocity input, the actual trajectory is of smaller scale than the target trajectory. Significant topological features such as curves are picked up, but with a tendency for too-small radii. For example, the turn at 80 seconds is accurate, whereas the curve at about 65 seconds is too shallow. Some slight bends are missed entirely. This is expected of short and/or small velocity inputs or changes. These angular velocity-related issues are the reason that the estimated trajectory is more 'curled up' compared to the ideal trajectory. Furthermore, the system tends to prefer movement within a θ-layer as well as along grid axes or at least right in the middle of two grid axes. Interesting is also the sharp turn at 20 seconds, which involves slowing down to

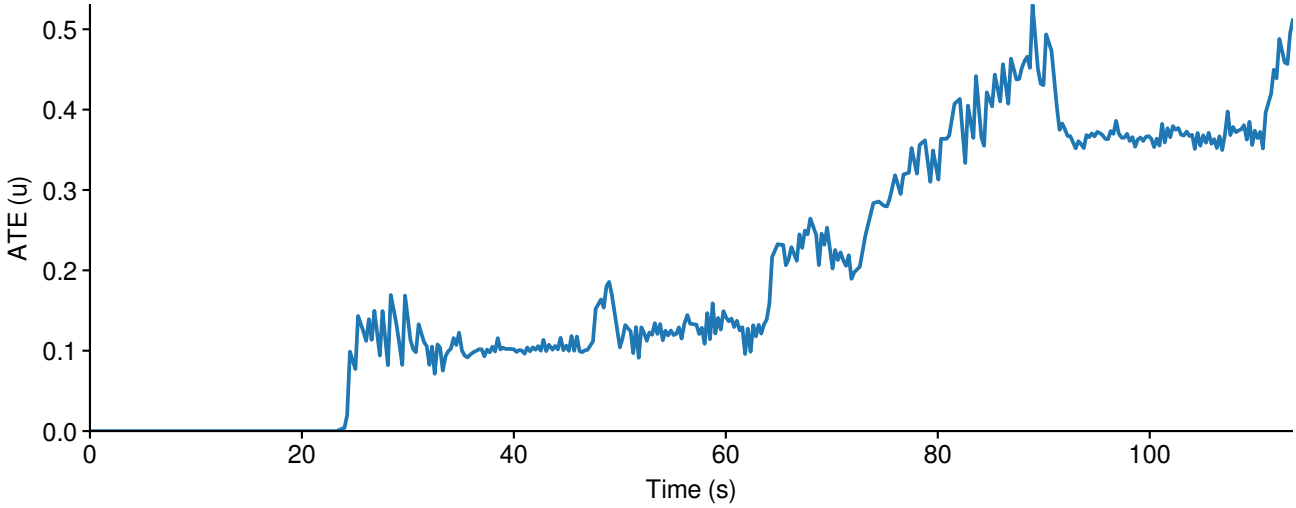

Figure 6.14 Absolute trajectory error (ATE) of the path integration estimate from the target trajectory for the maze experiment, plotted over time. Periods of constant deviation, i.e. horizontal linear segments, indicate precise path integration behavior (of the position), since the error does not increase there. For the trajectory at hand, these correspond to periods of standstill. Also, during phases of non-zero velocity input the deviation does not increase much. Specifically, when the heading angle currently represented by the path integration system differs from the actual heading angle and there is pure translation, then the deviation increases linearly with the slope being proportional to the angular difference. Some small jumps can be observed, presumably when target and actual trajectory cross.

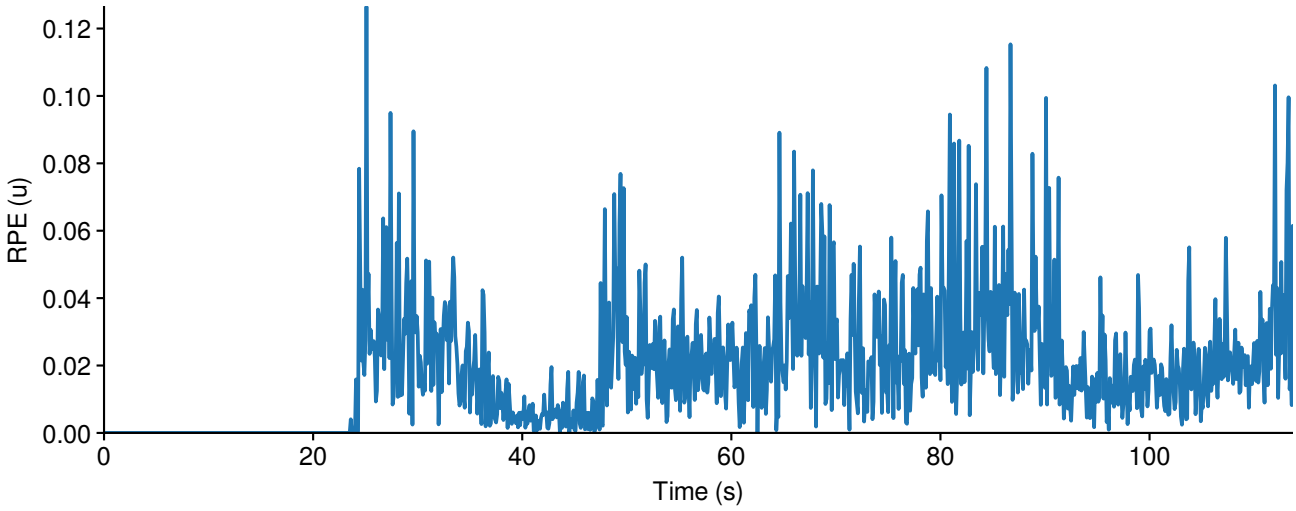

Figure 6.15 Relative pose error (RPE) of the path integration estimate from the target trajectory for the maze experiment, plotted over time. The value represents the difference in motion between the two trajectories for each 0.1-second period. Depicted is the translational part of the RPE. The value remains mostly below 0.04, with an RMSE of 0.026, indicating rather accurate path integration behavior. Perfect or near-perfect following of the target trajectory manifests itself as near-0 RPE.

Table 6.2 Evaluation metric statistics for the trajectory following experiments. Specifically, the root-mean-square error (RMSE), mean, and maximum for the absolute trajectory error (ATE) and relative pose error (RPE) are reported. All values are of unit u.

	ATE			RPE		
Trajectory	RMSE	mean	max	RMSE	mean	max
Maze	0.244	0.192	0.530	0.026	0.019	0.126
Australia	1.010	0.909	1.656	0.029	0.025	0.081

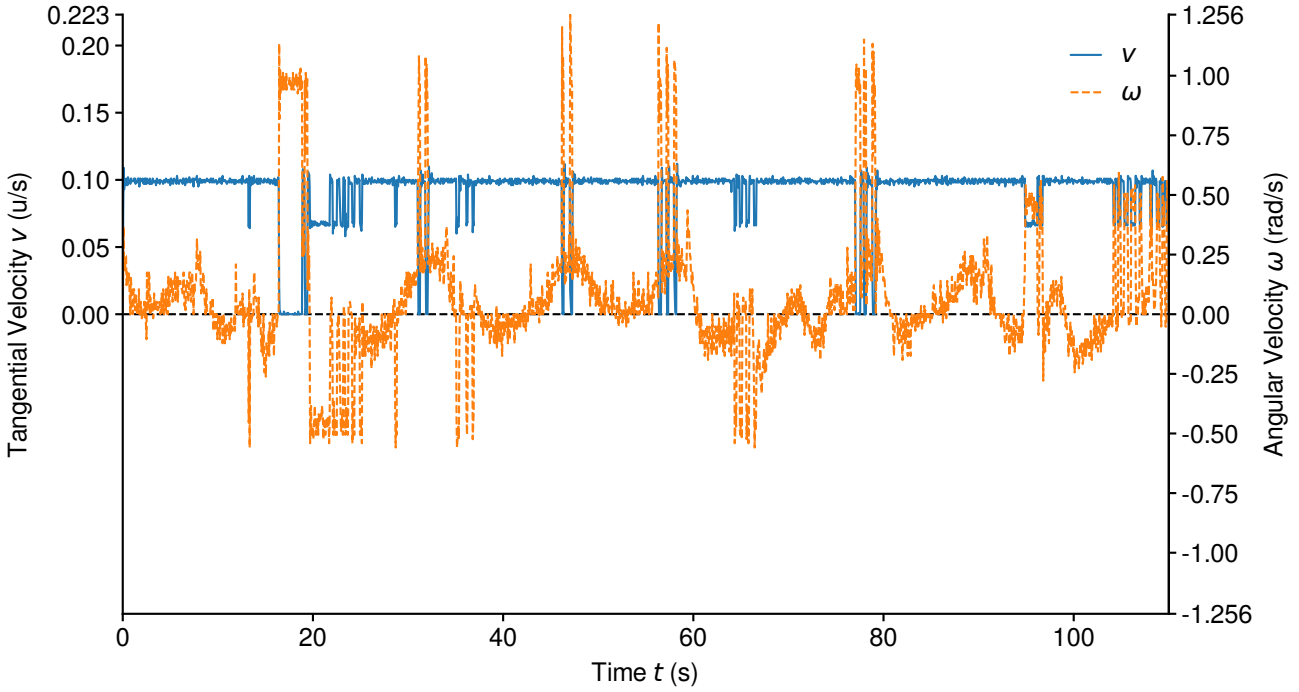

Figure 6.16 Tangential and angular velocities of the Australia experiment over time. No preprocessing has been applied. Therefore, the data is noisy and contains seemingly erroneous outlier points. Of the greatest concern are phases where the velocity jumps rapidly between near-zero and some non-zero value, for example just before 60 seconds. These inputs are impossible to follow precisely due to the inert dynamics of the attractor network and the neurons. Inputs of this nature typically lead to underrepresentation of velocities, e.g. too shallow turns or missed bends. There are also some sharp rising and falling edges of non-zero velocity periods, which potentially lead to some deviation from the target trajectory. Even without clipping, the data already lies within the limits of the path integration system: the tangential velocity v is bounded by 0 and 0.223, and the angular velocity is bounded by ± 0.256. Overall, the velocities are on the lower side, e.g. the angular velocity wiggles around 0 most of the time, which makes it more difficult for the path integration system to capture. Interesting here is that the tangential velocity drops in magnitude whenever the absolute angular velocity increases.

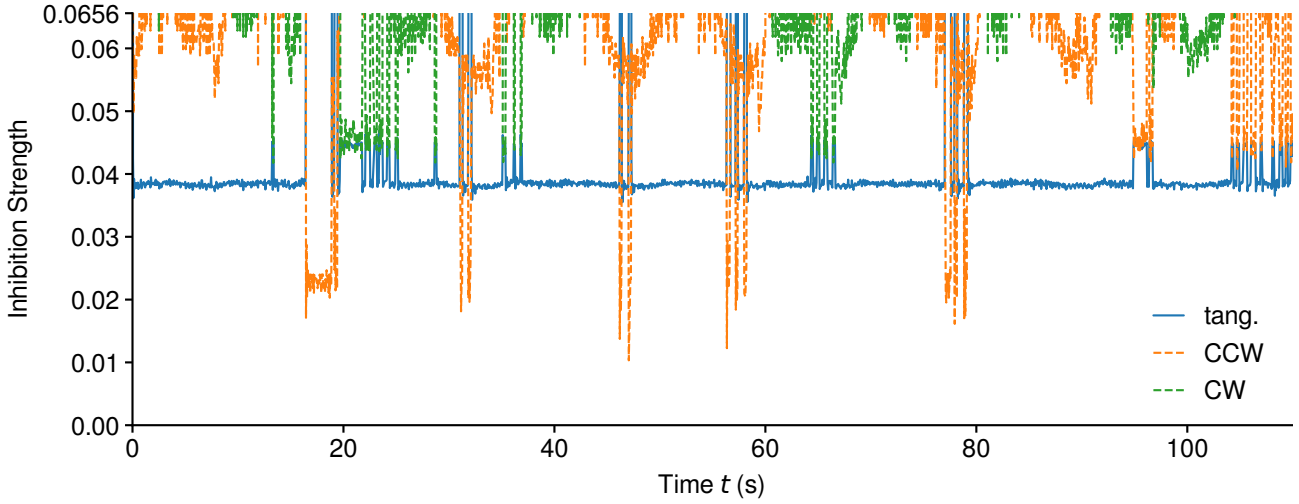

Figure 6.17 Inhibition values for the translation (blue), counterclockwise rotation (orange), and clockwise rotation (green) modules in the Australia experiment over time. These quantities correspond to the inhibitory negative input current that is fed to the respective shifting module's neurons. The minimum inhibition values required to prevent the corresponding shifting ensemble from causing any apparent movement in the attractor network are 0.0637 for translation, and 0.0656 for rotation. Note that this does not necessarily mean that the shift modules are fully inhibited, but that they might still slightly influence the behavior of the attractor network. These values are obtained from the velocity calibration step (see Sec. 5.7.2). Any values out of view are equal to 1, i.e. definitively and fully inhibit the shift modules. Actually, most of the time the rotation inhibition values are rather close to their upper limit, when not equal to 1. This fact matches the observation of rather small absolute angular velocities in Fig. 6.16. The jumpy nature of the velocity input can also be seen in the inhibition data.

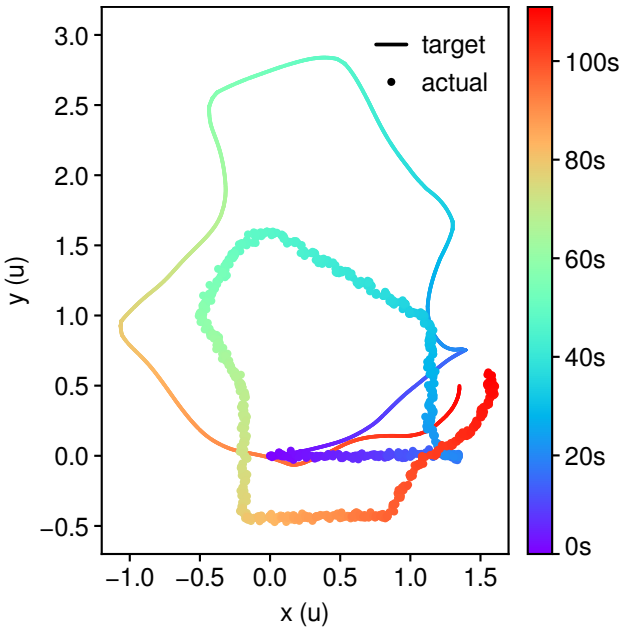

Figure 6.18 Target trajectory and path integration result of the Australia experiment. Overall, the relative traveled distances are represented quite accurately. Most important topological features such as curves and significant bends are captured. For example, the sharp turn of about $170°$ at 20 seconds is represented correctly. Shallow turns, however, are not followed by the path integration system, e.g. the left bend right at the beginning. Due to the noisy and jumpy velocity data, curves and turns tend to be underrepresented, i.e. be too shallow. In fact, for the same reasons also the tangential velocities are partly underrepresented, which explains the smaller-than-desired scale of the estimated trajectory. Considering the trajectory's total duration of almost two minutes, the visual similarity to the target trajectory seems appropriate. Consult the following figures for an illustration of the ATE and RPE.

standstill, rotating by about $170°$, and speeding up again. This feature is represented properly.

As a concrete measure for the system's global performance, Figure 6.19 highlights the absolute trajectory error (ATE) from the target trajectory. The fact that angular velocities cannot be captured precisely due to wiggling around 0 most of the time results in the estimated trajectory looking 'curled up' compared to the ideal trajectory. This becomes apparent in the red lines that associate positions with similar time stamps. Fig. 6.20 shows the ATE over time. We identify two phases of constant deviation, the first of which corresponds to the pure rotation input. Since the absolute deviation pertains to position differences, it makes sense for it to remain constant in the absence of translatory input. Nonetheless, this behavior demonstrates stationary stability of the system. The linear segments with non-zero slope primarily pertain to translation where the actual heading angle differs slightly from the target. The bends in the deviation curve correspond to turns in the trajectory. Considering these factors, the noise and fluctuations of the velocity data, as well as the total duration of 110 seconds, we deem the performance to be appropriate. Furthermore, we compute the relative pose error (RPE) as a measure of the local drift per time step (see Fig. 6.21). This value being non-zero most of the time indicates an ever-present drift. It could be due to the underrepresentation of the angular velocity (which shows up as translational errors at subsequent time steps), and/or due to the trajectory containing no periods of standstill (as opposed to the maze experiment).

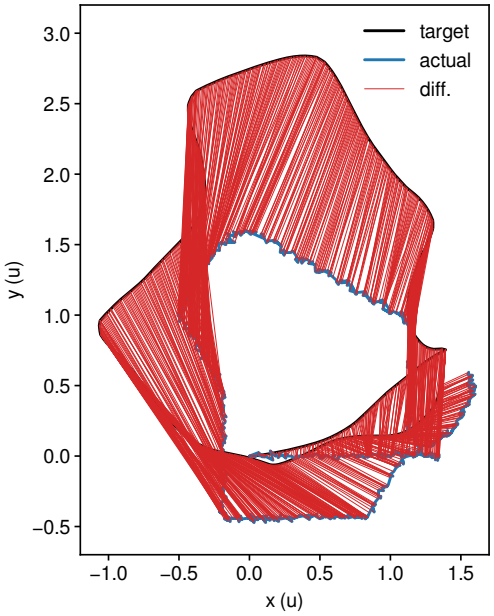

Figure 6.19 Absolute trajectory error (ATE) of the path integration estimate from the target trajectory for the Australia experiment. Red lines show the point pairs of target and actual trajectory that have been associated via their time stamp. Patches of these lines being parallel indicates that translational movement is captured accurately in terms of tangential velocity. It is also apparent that the estimated trajectory is 'curled up' compared to the target, which stems from the misrepresentation of angular velocities that are noisy and wiggle around 0. The absence of any clustered patches of red lines indicates that there are no periods of standstill as well as that the path integration does not get stuck in some attractor state.

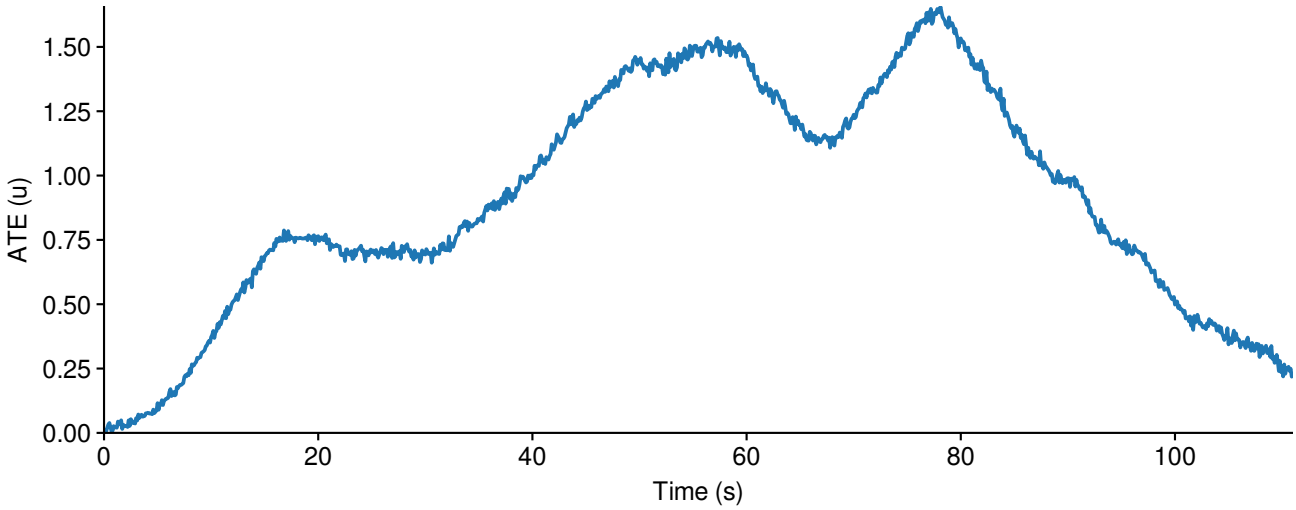

Figure 6.20 Absolute trajectory error (ATE) of the path integration estimate from the target trajectory for the Australia experiment, plotted over time. Periods of constant deviation, i.e. horizontal linear segments, indicate precise path integration behavior (of the position), since the error does not increase there. For the trajectory at hand, one such period is from around 15 to 30 seconds and pertains to a pure rotation. This can be explained with head orientation being essentially independent of distance and, therefore, errors thereof do not directly show up in the deviation plot, whereas translation is dependent on head direction. When the heading angle currently represented by the path integration system differs from the actual heading angle and there is pure translation, then the deviation increases linearly with the slope being proportional to the angular difference. Many of the linear segments in the deviation plot are caused by this. Turns in the target trajectory can be attributed to bends in the deviation plot. The deviation reaches its maximum of about 1.65 at 80 seconds. The final approach towards 0 deviation is a coincidence. Overall, considering the trajectory's duration of almost two minutes, the accumulated deviation seems appropriate.

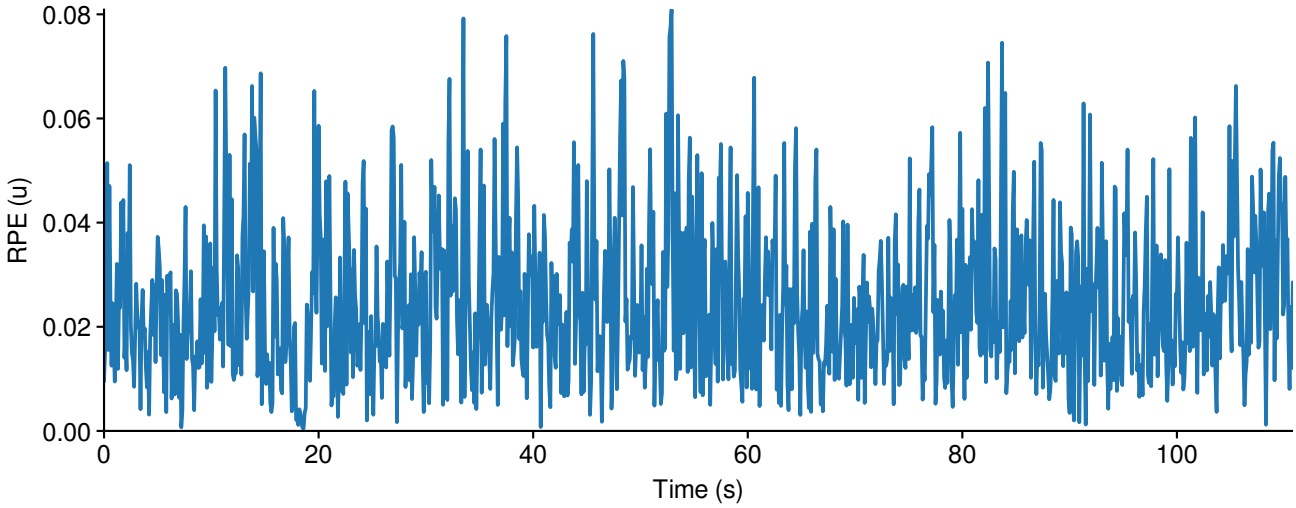

Figure 6.21 Relative pose error (RPE) of the path integration estimate from the target trajectory for the Australia experiment, plotted over time. The value represents the difference in motion between the two trajectories for each 0.1-second period. Depicted is the translational part of the RPE. The value remains mostly below 0.05, with an RMSE of 0.029. However, it is also mostly non-zero, indicating that some drift occurs at every time step. This matches the observation that the estimated trajectory is of slightly smaller scale than the ideal trajectory, as well as 'curled up' compared to it. Nonetheless, the RPE statistics demonstrate quite accurate overall path integration behavior.

6.4 Modulated Inhibition

In addition to the default shifting implementation, which uses a Nengo node to directly and continuously inhibit the shift ensembles, we briefly discuss simulations that have the so-called modulated inhibition enabled (see Sec. 5.7.3). The results of these simulations are depicted in Fig. 6.22. For the same values of i_{T}, i_{CCW}, and i_{CW} the bump's behavior in terms of tangential velocity, angular velocity, and smoothness is very similar to the behavior with continuous inhibition. Also the representable ranges of the two velocity dimensions are comparable. This one-to-one correspondence is interesting since the effects of the two inhibition processes on the neural dynamics of the main attractor are conceptually quite different.

In particular, for straight movement the trajectories are essentially the same. Standstill is achieved at $i_{\mathrm{T}} = 0.055$ (see Fig. 6.22a and cf. Fig. 6.1h). At $i_{\mathrm{T}} = 0.05$ we demonstrate the lower bound for the tangential velocity, which roughly matches the one for continuous inhibition (see Fig. 6.22b and cf. Fig. 6.1g). At a second glance, while the bump's final position after 5s is the same, it is reached a bit sooner here compared to continuous inhibition. This can be seen by comparing the positions of the orange trajectory segments (at about 3.5-4s). However, it is difficult to compare behavior at these low speeds since the attractive force of the stable points of the attractor, i.e. the neuron's associated locations in representational space, becomes large relative to the involved shifting forces exerted by the shifting modules. With $i_{\mathrm{T}} = 0.03$ and $i_{\mathrm{T}} = 0.005$ the results are pretty much identical with the ones for continuous inhibition (see Figs. 6.22c and 6.22d, respectively; and cf. Figs. 6.1e and 6.1b, respectively). This demonstrates that the ability to capture velocities near the upper limit remains unchanged. As expected, the behavior at $i_{\mathrm{T}} = 0$ is also identical (no figure shown).

For circular movement we consider examples at either end of the representable radius range. Overall, the velocities tend to be a bit lower compared to continuous inhibition with the same inhibition values. The smallest discernible circle is achieved at $i_{\mathrm{T}} = 0.05$ and $i_{\mathrm{CCW}} = 0.01$ (see Fig. 6.22e). While the corresponding experiment for continuous inhibition is not depicted, the difference is that with modulated inhibition we get an approximately half as fast movement at the same radius. With $i_{\mathrm{T}} = 0.04$ and $i_{\mathrm{CCW}} = 0.01$ we get the smallest reliable radius, just as with continuous inhibition and at similar speed (see Fig. 6.22f, cf. Fig. 6.4a). The combination $i_{\mathrm{T}} = 0.02$ and $i_{\mathrm{CCW}} = 0.03$ demonstrates the one-to-one correspondence at a larger radius (see Fig. 6.22g, cf. Fig. 6.4d). At $i_{\mathrm{T}} = 0$ and $i_{\mathrm{CCW}} = 0.06$, which yield the maximum-radius circle with continuous inhibition, we get an even wider curve with modulated inhibition. The traveled distance seems to be the equivalent. However, instead of following a perfect circular path, the bump moves on linear segments joined by shallow bends. This can be explained similarly to the near-standstill translatory movement case: the attraction force of the stable points at the neurons' positions is relatively strong compared to the desired movement along the θ-dimension. Hence, the bump sticks to the closest preferred direction per x-y-layer for large portions of the simulation duration. We conclude that the largest reliably attainable radius for circular movement is similar to, specifically not larger than, the corresponding upper bound with continuous inhibition.

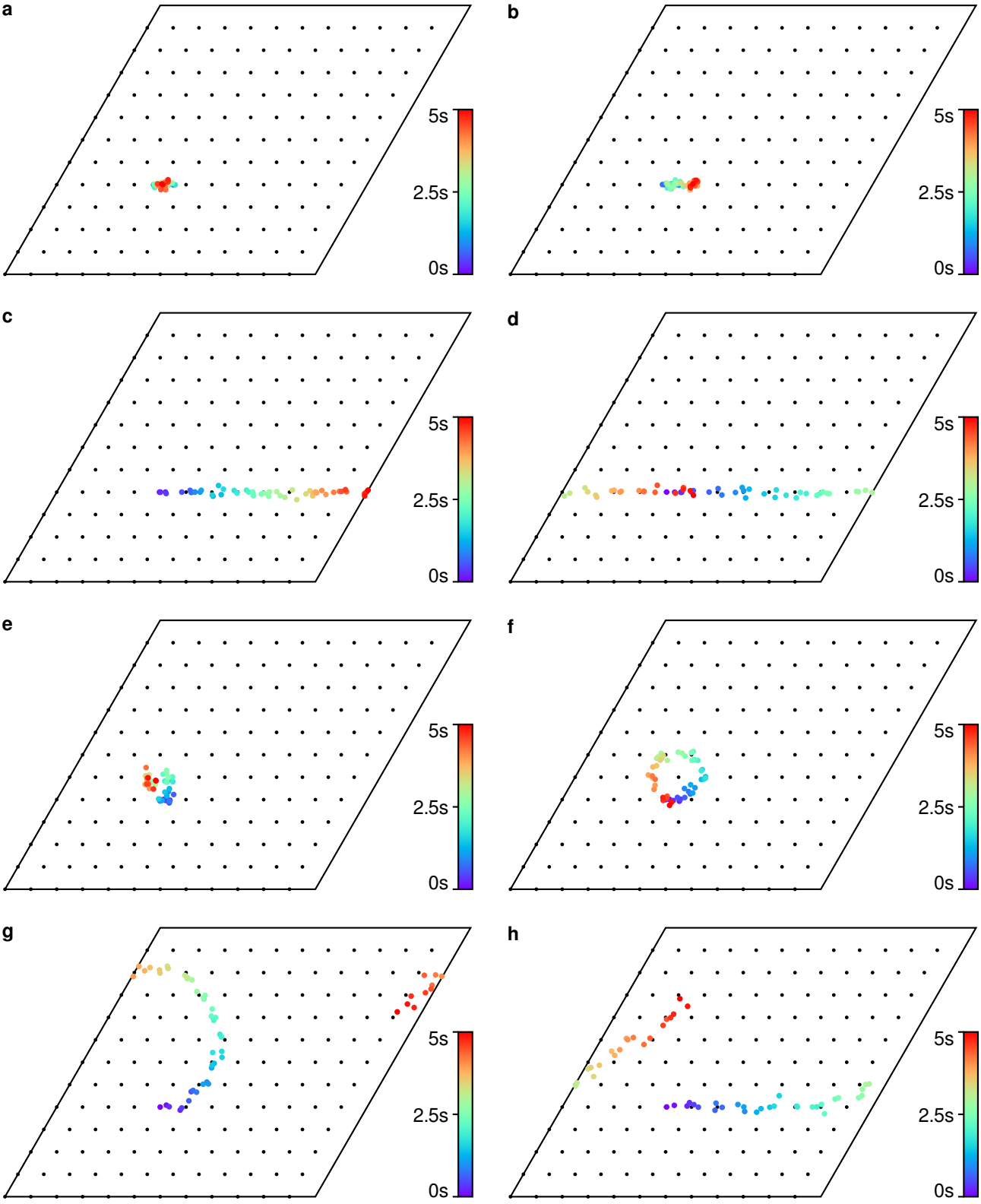

Figure 6.22 Use modulated inhibition instead of continuous inhibition to control the influence of the shift modules on the main attractor network. For each shifting ensemble there is an input neuron that spikes at a rate proportional to the specified inhibition value, i.e. inversely proportional to the desired velocity. The connection weight from these input neurons to the shift modules is negative, i.e. inhibitory. Comparing the results with the experiments in Figs. 6.1 and 6.4 it becomes clear that the achievable movement patterns, smoothness, and velocity ranges are very similar for the two types of inhibition. Interestingly, the correspondence seems to be 1:1, i.e. for the same values of inhibition the results are nearly the same. All plots show a time range of 5s, for which the inputs are kept constant. The bump is always initialized at position $(4, 4, 0)_{\mathrm{g}}$. *(continued on next page)*

Figure 6.22 ***(continued)*** **a**-**d**, Pure translation with different values for the continuous shift inhibition i_{T}. The results are essentially identical with the continuous inhibition experiments. **a**, Standstill at $i_{\mathrm{T}} = 0.055$. **b**, Slowest practical translation with $i_{\mathrm{T}} = 0.05$. **c**, $i_{\mathrm{T}} = 0.03$. **d**, Slightly lower than maximum speed, $i_{\mathrm{T}} = 0.005$. **e**-**h**, Circular movement with varying turn radii, tangential velocities, and angular velocities. While the results are similar to the continuous inhibition, the movement tends to be a bit slower here. **e**, Smallest radius, $i_{\mathrm{T}} = 0.05$ and $i_{\mathrm{CCW}} = 0.01$. Seems to be 50 percent slower than the corresponding continuous inhibition experiment (not depicted)). **f**, $i_{\mathrm{T}} = 0.04$ and $i_{\mathrm{CCW}} = 0.01$. Marginally slower than continuous inhibition. **g**, $i_{\mathrm{T}} = 0.02$ and $i_{\mathrm{CCW}} = 0.03$. Marginally slower than continuous inhibition. **h**, $i_{\mathrm{T}} = 0$ and $i_{\mathrm{CCW}} = 0.06$. While the radius is objectively larger here compared to continuous inhibition, the bump appears to travel not on a circle but rather on linear segments joined by wide-radii curves. The tangential velocity is similar.

7 Discussion & Conclusion

We propose and implement a neuromorphic model for representing and updating multiple simultaneous pose estimates, thereby closely following the mammalian spatial navigation system on both a functional and a neurophysiological level. We consider spiking neuron dynamics, continuous attractor network dynamics, efficient neuron arrangement and local connectivity, and hexagonal firing patterns. In short, position and orientation are conjunctively represented with a 3D continuous attractor network of spiking neurons. Translation and rotation of pose estimates are realized by three distinct shifting ensembles. These operate on a delayed replication of the attractor state and project back to the attractor via offset connections. One such ensemble shifts activity per horizontal neuron layer into the respective preferred directions; the other two ensembles shift activity up and down, respectively. This pertains to translation, counterclockwise rotation, and clockwise rotation, respectively. The shifting ensembles are selectively and proportionally inhibited based on tangential and angular velocity input. Importantly, the entire shifting mechanism is realized in the neural substrate, i.e. with spiking neurons and appropriately weighted synaptic connections.

During development and deployment, we are constrained by two main factors: computational resources and Loihi's hardware restrictions. The former concerns memory size and processing power. Larger neural networks with more neurons and more involved connectivity patterns make the Nengo build process more complex, require more memory to store all associated parameters and connection weights, increase simulation time, and result in more data being generated during simulation (which also needs to be stored). The Loihi-related restrictions pertain to the maximum number of neurons we can use for our attractor and shifting ensembles; and to the possible connectivity patterns, which includes the recurrent connections of the attractor, the connections from the attractor to the shifting ensembles (for replicating the attractor state), and the feedback connections from the shifting ensembles to the attractor. Eventually, Loihi is more constraining than the computational resources for simulation. To get to this point, however, various design changes and optimizations were necessary, some of which we mention below. The final attractor ensemble comprises 12^3 neurons; the recurrent connections of which follow a meticulously-tuned Mexican hat function. This yields localized center-surround excitatory-inhibitory connectivity where just enough neurons are active per activity packet so that stable stationary dynamics and reasonably smooth nonstationary dynamics are exhibited, and that as many simultaneous activity packets as possible can be sustained (three, in our case; provided they are far enough apart). Widening the excitatory region of the Mexican hat function would increase the number neurons active per pose estimate and therefore improve representational stability and smoothness of trajectories, but it would significantly impact the system's ability to sustain more than one pose estimate at a time. In addition to tuning the Mexican hat function, we set all weights below some small threshold to zero to further reduce the number of recurrent connections. Then we convert the weight matrix to a sparse data format. This optimization does not significantly change the attractor dynamics but is crucial for the system to fit onto Loihi. We reuse the recurrent weights for the connections from the attractor ensemble to the shifting ensembles, adding to the importance for these weights to be as sparse as possible. For the shifting connections we start out from the same Mexican hat function and, put simply, center-offset it into different directions depending on the shifting ensemble (and horizontal neuron layer). The same thresholding as for the recurrent attractor weights is applied, followed by a conversion to a sparse data format. Evidently, the design of the Mexican hat function and subsequent sparsification is of utmost importance to keeping the overall number of connections as low as possible. Then, there is the issue of splitting our neuron ensembles onto Loihi cores. Nengo's default splitting behavior is to split ensembles into consecutive blocks of neurons in a row-major order. Given our localized connectivity the default behavior results in

unnecessarily high inter-core communication. Instead, we manually specify the splitting in a way that exploits our connectivity. Per ensemble (of shape $(12, 12, 12)$), 27 neuron blocks of shape $(4, 4, 4)$ are to be distributed onto distinct Loihi cores. To keep the attractor's neural activity in check, i.e. prevent activity packets from dissolving or diverging, some kind of global inhibition is required. To this end, we slightly reduce the neurons' bias current. This brings the desired effect and has two important advantages over e.g. dynamic global inhibition. First, it does not require any additional connections. Second, the inhibition strength does not depend on the number of concurrent activity packets, i.e. pose estimates. What makes designing and optimizing the system so complicated is the interdependence of most parameters. Overall, we utilize Loihi's available resources as well as possible given our goals and system architecture.

We show the system's general stability and its capability of performing path integration from relative tangential and angular velocity inputs, as long as these stay within certain ranges. The system functions in the presence of local uncertainty from noisy velocity inputs and integration errors, as demonstrated in trajectory following experiments where topological features such as sharp or wide-radius turns, and relative distances between such features are captured accurately. Considering that we currently do not include loop closure, reset by visual input, or any other form of supervision, a certain drift of the pose estimate from the actual pose is to be expected. This drift may originate from the mentioned noisy or inaccurate velocity input, from the neural dynamics, from the attractor dynamics, or other factors. Regarding local uncertainty, it might be interesting to test the system with different amounts of noise affecting e.g. the neural activity directly.

Our system can also handle non-local uncertainty due to larger errors caused by outside influence, e.g. repositioning the system, or ambiguous visual inputs. To this end, we demonstrate that our conjunctive representation of planar position and head orientation allows to unambiguously sustain and propagate multiple simultaneous pose estimates. Compared to a disjunctive representation of these quantities, however, we require more neurons to achieve the same single-bump resolution. For example, when separately representing the heading angle, the resolution scales linearly with the number of neurons used in the respective module; whereas in our system an entire layer of neurons needs to be added to the main attractor ensemble as well as to all three shifting ensembles to increase the angular resolution by one discrete step. (Technically, with a fixed spatial resolution, one could still argue that increasing the angular resolution linearly increases the number of neurons, but the factor is prohibitively large.) So, we need to balance the following trade-off: to improve path integration in terms of spatial resolution we want to increase the spatial neuron grid resolution, but would have to decrease the number of θ-layers, which decreases overall path integration performance; and to improve path integration in terms of directional flexibility we want to increase the number of θ-layers, but would have to decrease the spatial neuron grid resolution, which decrease overall path integration performance. This directly explains why we need to work with a comparatively low spatial resolution of 12×12 neurons in order to get a decent angular resolution of 12. That configuration seems to be kind of a sweet spot. Of course, reducing the angular resolution a bit in order to increase the spatial resolution a bit, or vice versa, might work better in certain scenarios.

Overall, the system performance is quite satisfactory and the goals stated in Sec. 2.4 are fulfilled. Still, we want to highlight four issues. First, the simulation is not yet running in real time. For a couple of seconds of simulated time we need a couple of minutes of computation time. As discussed above, we explored numerous optimizations to reduce the system's complexity. We believe that there is not much left to improve on the implementation-side. Instead, we postulate that the runtime issue should be fixed when the system is run on actual Loihi hardware. Then, the computation overhead of simulating neural dynamics would be eliminated, and there is the additional benefit of high parallelism, since the neurons would be run in parallel. Second, our system seems to require precisely tuned connection weights. There is a strong dependence between weight parameters themselves, e.g. excitation strength vs. inhibition strength, or width of excitatory part vs. offset of shifting weights; as well as between weight parameters and other system parameters, e.g. offset of shifting weights vs. synaptic time constant. As a consequence, most system parameters, including the weights, need to be tuned conjunctively. Such strong interdependence is not ideal. Potential reasons are the rela-

tively small number of neurons and the requirement of supporting multiple simultaneous activity packets. Third, the stable states of the attractor network correspond to combinations of poses located at neurons' grid locations. This is different from an ideal continuous attractor. It is due to the rather coarse discrete neuron grid and, more importantly, the low number of neurons involved in representing any single pose estimate. Related to the discrete stable states, the fourth issue concerns the preference of pose estimates to move along grid axes or into a horizontal layer's preferred direction. This can cause serious absolute deviations over longer trajectories. As already mentioned in the paragraphs on optimization and on multiple simultaneous pose estimates, the last two issues could be addressed by increasing the neuron grid resolution. Simply using more neurons would increase the number of stable states and bring the attractor closer to a continuous one. If we additionally widen the neural tuning curves and recurrent connection weight function, more neurons would be active per pose estimate, resulting in smoother trajectories and less drift. Pose estimates could remain longer between dynamically stable attractor states, i.e. between neurons' locations. Then there is also the aspect of increasing the angular resolution, which would make more directions the 'preferred' ones for pose estimates to travel along. Regarding the θ-dimension, it might not only be fruitful to try different resolutions along this dimension (independent of the other two dimensions), but also to try a different variance of the underlying 3D Gaussians in this dimension. This would, however, require retuning the recurrent and shifting connection weights.

More ideas for future tweaks and adaptations are covered in the next chapter.

8 Future Work

In this section we collect potential interesting modifications, extensions, or alternative approaches to various aspects of our system.

Neuron Parameters. As outlined in Ch. 5, we compute distinct encoder vectors per neuron according to it's preferred pose. Together with desired intercepts and nominal firing rates, where we define the same value combination for each neuron, Nengo computes gain and bias per neuron. Unsurprisingly, all gains as well as all biases turn out equal. We then speak of a homogeneous neuron population. This results in a certain regularity in the system's dynamics. Of course, there is still noise stemming from the spiking neuron dynamics, but it is somewhat similar across the neurons, depending on the current activity in the neighborhood. Additionally, all neurons are initialized with a voltage potential of 0, a constraint stemming from the Loihi hardware. These facts altogether mean for example if we input some pose centered at a neuron, then the neural activity over time will be perfectly symmetric according to the neuron grid.

In contrast, the neuron populations observed in real neurons are heterogeneous, i.e. have diverse biophysical parameters. Consequently, the firing rate maps are more irregular and appear more noisy. Conklin and Eliasmith[13], for example, specify ranges for the intercepts and the nominal firing rates for Nengo to randomly choose from when computing the gains and biases for the neurons. They report that the heterogeneity of their neural population and the associated diverse behavior match the ones of biological neurons. It will be interesting to test randomized neuron parameters with our ensembles. The stability and drift behavior can be compared to the current one, and the degree of realism of the neural activity regarding noise and irregularity can be evaluated.

Multiple Spatial Scales. To obtain a unique position estimate within the desired region of operation (or, put differently, to extend the range of operation) and to further increase the precision[40] and spatial resolution of our path integration, multiple independent modules with different spatial scales can be combined. Smaller scales refine the estimate from coarser scales. A module here refers to a complete copy of the current system, i.e. containing the attractor network and three shifting networks. This is akin to the modular organization of grid cells with increasing spatial scale along the dorsoventral axis of the medial entorhinal cortex. The spatial scale of a module is defined in the mapping of the module's domain Domto the real world, e.g. the domain's extent along one axis may be mapped to $0.5\,\mathrm{m}$. The velocity input to the shifting ensembles is scaled according to this mapping.[42] For grid modules with larger scales the velocity input is attenuated more. At some point this may lead to pinning, i.e. the velocity input being too weak to meaningfully affect the attractor state (see also Sec. 5.7). Ideally, the nominal flow of activity (in neurons per unit distance) should be proportional to the nominal velocity input for each grid module.[17]

Stemmler, Mathis, and Herz[58] present a thorough theoretical work on the representation and decoding of position with multiple grid cell modules of different spatial scales. They conclude that for optimal performance the difference in scale between successive modules should be 1.5, as that does reduce the decoding uncertainty without increasing the likelihood of worst-case decoding errors. In fact, it maximizes the distance between the primary and secondary maxima of the posterior probability, making it the most robust choice. The second conclusion is that the orientations of the individual grid cells per module should be equal. Both of these findings are actually confirmed to apply for real grid cell networks. As to their approach, they provide recursive equations to combine the population vector estimates from multiple grid cell modules of different spatial scales. The coarsest module gives a rough position estimate subject to considerable uncertainty, as they visualize by a larger variance of the estimate relative to the modules period. This estimate is refined by the next-smaller-scale module's readout, it's variance being smaller in the first place but subject to ambiguity due to multiple feasible estimates falling into the total considered spatial

domain. With the additional module the uncertainty is reduced, the reduction being greater the more neurons are contained in the finer module. Also, the ambiguity of the smaller module is mitigated by the unique estimate of the larger module. As more and more scales are combined recursively, the variance/uncertainty decreases until the performance of an ideal observer is reached. The multiscale approach can be likened to error correction. They also give an intuitive analogy for combining two scales: let the coarser scale be the hour hand of a clock, and the finer scale the minute hand; then on a high level the readout works like reading the time, i.e. we take the hour value as base and add onto it the minute value. The presented ideas extend to the two-dimensional case. There, the unit cells are hexagons that can be mapped onto tori, just like we do in our work. A coarser scale sort of sets the reference frame for the next finer scale. Similarly to 1D, the coarser scales feature greater localization, but higher uncertainty per estimate, while the finer scales have ambiguous estimates but lower uncertainty. Regarding the clock analogy, for the case of multiple dimensions there are simply multiple clocks, one per dimension.

For the 2D case we give an additional intuitive explanation of how the hexagonally-periodic position estimates of grid cell modules of different scales work together and ambiguity of the smaller-scale grid is resolved. Assume we have two spatial scales. Both estimates start at the origin per module. After some movement the individual estimates are ambiguous when considered separately, according to the respective module's period. Specifically, the ambiguity takes the form of hexagonal grids where the potential true positions are all the grid points. The spacing of the grids directly corresponds to the module's period. When we superimpose the two grids the locations in space where two grid points coincide form a new hexagonal grid of significantly larger spacing than any of the two original grids. Hence, depending on the traveled distances we expect this two-module architecture might already be sufficient to unambiguously decode the current position. Note that the first take on resolving the ambiguity corresponds to the 'Nested' view in Edvardsen[17], while the second one corresponds to the 'Combinatorial' view in the same paper.

Stemmler et al. 's considerations and equations could be adapted for our system. From an implementation perspective, the different modules would likely exist side-by-side without direct interaction. Since the variance of the involved 3D Gaussians is defined relative to the domain it would make sense to keep it the same for each module in order to be able to use the already tuned connection weights and dynamics parameters. The readout would then combine the individual estimates according to the derived equations. Alternatively, it might be possible to decode from multiple attractor networks using spiking neuron ensembles. It remains to identify how much the NEF principles could simplify this approach.

Abstract Cognitive Spaces. Eventually, our developed model is envisioned to extend to non-spatial information, i.e. a general cognitive map. This is based on research suggesting that the hippocampal-entorhinal system of place cells, grid cells, and others, might not just be involved in spatial processing and navigation, i.e. representing Euclidean space, but actually contribute to abstract cognition. If we treat cognitive spaces as geometric multidimensional spaces, with individual dimensions corresponding to the value of a specific feature, then the notion of concepts would emerge as convex regions therein. It has been suggested that the place and grid cells might be capable of encoding these abstractly-interpreted dimensions. Closeness of encoded points intuitively corresponds to similarity of the associated cognitive concepts. The increase of spatial scale from dorsal to ventral areas of the brain areas containing the relevant neurons could be interpreted as representing abstract information at different levels of detail. Grid cells might be used to navigate the cognitive spaces, while place cells would represent the actual information. Movement is not only possible along one encoded dimension at a time, but along any direction in the encoded space. The remapping observed in place cells allows for switching between different dimensions or entire spaces. Importantly, switching back to previous states is possible, i.e. 'previous knowledge' can be retrieved. While the representation and navigation of Euclidean space and cognitive space up to three dimensions can be reasonably explained, it is rather unclear how more complex and higher-dimensional spaces would be encoded.[6,56] For example, place and grid cells in rodents have been found to encode the frequency of sound.[24] We can consider the representation of

abstract concepts from another perspective. Effectively, many continuous quantities other than space itself are typically specified in spatial terms. It seems logical that such quantities can be represented by the spatial system of the brain. So-called social place cells might represent the location of other animals or people, time could be considered a 1D line in space, and audio properties such as pitch and tone can also be taken as 1D lines in space.[53]

Deployment on Loihi and Robot. Eventually, the system should run on an Intel Loihi neuromorphic chip. This entails certain resource limitations, most prominently regarding the number of neurons and the number of connections. However, it also offers highly efficient operation with lower power consumption compared to traditional computers. A strong benefit of using the Nengo library is the availability of NengoLoihi, a backend for running Nengo models on the Loihi hardware. Accordingly, the deployment on the neuromorphic chip would be readily facilitated. During development, we anticipated and incorporated the neuromorphic methodology, e.g. by splitting up the ensembles into smaller chunks that fit onto the available neuron cores, such that major restructurings of the system when deploying onto the chip should be avoided.

Afterwards, to complete the development stack, the system may be ported to the modular biomimetic mouse robot, NeRmo (Neurorobotic Mouse), which mirrors the locomotion of a rodent via tendon-driven actuation.[39] The robot can 'walk' forward, backwards, and left or right. Given our current design, the backward movement will need to be augmented by a $180\,\mathrm{deg}$ rotation, forward movement, and another $180\,\mathrm{deg}$ rotation. Overall, testing with the robot would allow us to verify the correct operation in real-life navigation tasks with actual binocular visual input.

When the robot dynamics are known, it could also be beneficial to implement a time-dependent model that represents the possible movement of the robot and to fuse or validate the raw pose readout with predictions from that model to compute the final readout. This could have e.g. a smoothing effect on the represented pose estimates' trajectories. While such model would likely be implemented in a non-neuromorphic way, it may be fruitful to approximate it by a Nengo ensemble plus decoded connection.

Mapping and Visual Reset. Spatial navigation involves localization, mapping, and route planning.[19] In this project, we focus on the first part, i.e. path integration. For a comprehensive system also visual input and processing need to be considered, and can further be used to create and maintain a spatial map of the environment. So, the mapping component of spatial navigation constitutes an interesting topic for future work. (The third component, route planning, seems to be a bit more separate from the other two.)

Visual input could be incorporated via a new neural population that projects via plastic synapses to the pose ensemble. This new population's neurons are tuned so that visual stimuli are encoded in their collective firing. The subset of active cells in the new population (representing the current visual input) are then associated with the subset of active cells in the pose ensemble (representing the current pose estimate) by modifying the plastic synapses according to Hebbian learning, i.e. strengthening coactive neurons' connections. Over time, poses become associated with visual stimuli. In other words, they are anchored to the environment. When the same visual input is observed again, the associated subset of pose cells receive excitatory input via the now-strengthened connections[43] (see also Sec. 4.16).

For realizing visual reset, it might be feasible to implement some kind of inhibition of the new neural population, akin to the shifting ensembles' control. Also, a constant inhibitory component could be included in the connections from the new population to the pose ensemble. Its purpose is to weaken any activity packets distinct from the (not-yet-existing) activity packet associated with the current visual input.

As of now, there are only grid cell-like neurons in our system (not counting the just-mentioned 'vision population'). We could additionally introduce place cell-like neurons. In biology, grid cells and place cells are interconnected. The latter actually emerge from input of the former (via synaptic plasticity), while the former are (re-)calibrated by the latter (see Ch. 3). These connections may also introduce environment-specific information into the system. In biology, the following observations have been made: place fields often cluster near reward locations, grid fields increase their firing rate near reward locations, and grid fields shift towards reward locations.[56]

As for the hardware, Loihi's learning engine supports pairwise or triplet STDP, reinforcement learning with tag assignments, and more complex rules considering averaged and spike-timing traces.[16]

The previous considerations are not to be confused with episodic memory, which is also involved in navigation, but rather in the path planning part. It is responsible for remembering and recalling experiences, and ordering events. A potential implementation might be based on a recurrent spiking neural network.[63]

Overall, incorporating some form of mapping into our system may stabilize the path integration, especially during long trajectories. Kreiser et al.[37] report considerable performance improvements in terms of accuracy and drift when using reset compared to without it. They demonstrate that the mean square error between target and actual neuron index at certain points of a trajectory is significantly smaller when using certain reset strategies. The error increases with time, but seems to remain within some interval with proper reset.

A Appendix

Table A.1 Velocity calibration raw data of circular movement.

i_{T}	i_{CCW}	v (u/s)	ω (rad/s)	i_{T}	i_{CCW}	v (u/s)	ω (rad/s)
0	0	0.246	1.487	0.04	0	0.113	1.470
0	0.01	0.251	1.358	0.04	0.01	0.106	1.335
0	0.02	0.239	1.230	0.04	0.02	0.103	1.153
0	0.03	0.240	0.932	0.04	0.03	0.117	0.765
0	0.04	0.239	0.661	0.04	0.04	0.094	0.605
0	0.05	0.223	0.433	0.04	0.05	0.092	0.315
0	0.06	0.232	0.210	0.04	0.06	0.080	0.175
0	0.07	0.234	0	0.04	0.07	0.083	0
0.01	0	0.231	1.467	0.05	0	0.064	1.473
0.01	0.01	0.222	1.339	0.05	0.01	0.058	1.301
0.01	0.02	0.231	1.230	0.05	0.02	0.046	1.090
0.01	0.03	0.229	0.971	0.05	0.03	0.053	0.778
0.01	0.04	0.212	0.604	0.05	0.04	0.053	0.614
0.01	0.05	0.212	0.385	0.05	0.05	0.051	0.329
0.01	0.06	0.196	0.191	0.05	0.06	0.017	0.079
0.01	0.07	0.217	0	0.05	0.07	0.018	0
0.02	0	0.205	1.386	0.06	0	0	1.361
0.02	0.01	0.205	1.312	0.06	0.01	0	1.257
0.02	0.02	0.193	1.128	0.06	0.02	0	1.075
0.02	0.03	0.201	0.898	0.06	0.03	0	0.831
0.02	0.04	0.182	0.661	0.06	0.04	0	0.531
0.02	0.05	0.184	0.389	0.06	0.05	0	0.063
0.02	0.06	0.185	0.179	0.06	0.06	0	0
0.02	0.07	0.172	0	0.06	0.07	0	0
0.03	0	0.154	1.404	0.07	0	0	1.257
0.03	0.01	0.157	1.317	0.07	0.01	0	1.208
0.03	0.02	0.157	1.052	0.07	0.02	0	1.037
0.03	0.03	0.146	0.904	0.07	0.03	0	0.820
0.03	0.04	0.143	0.659	0.07	0.04	0	0.510
0.03	0.05	0.141	0.371	0.07	0.05	0	0
0.03	0.06	0.143	0.154	0.07	0.06	0	0
0.03	0.07	0.142	0	0.07	0.07	0	0

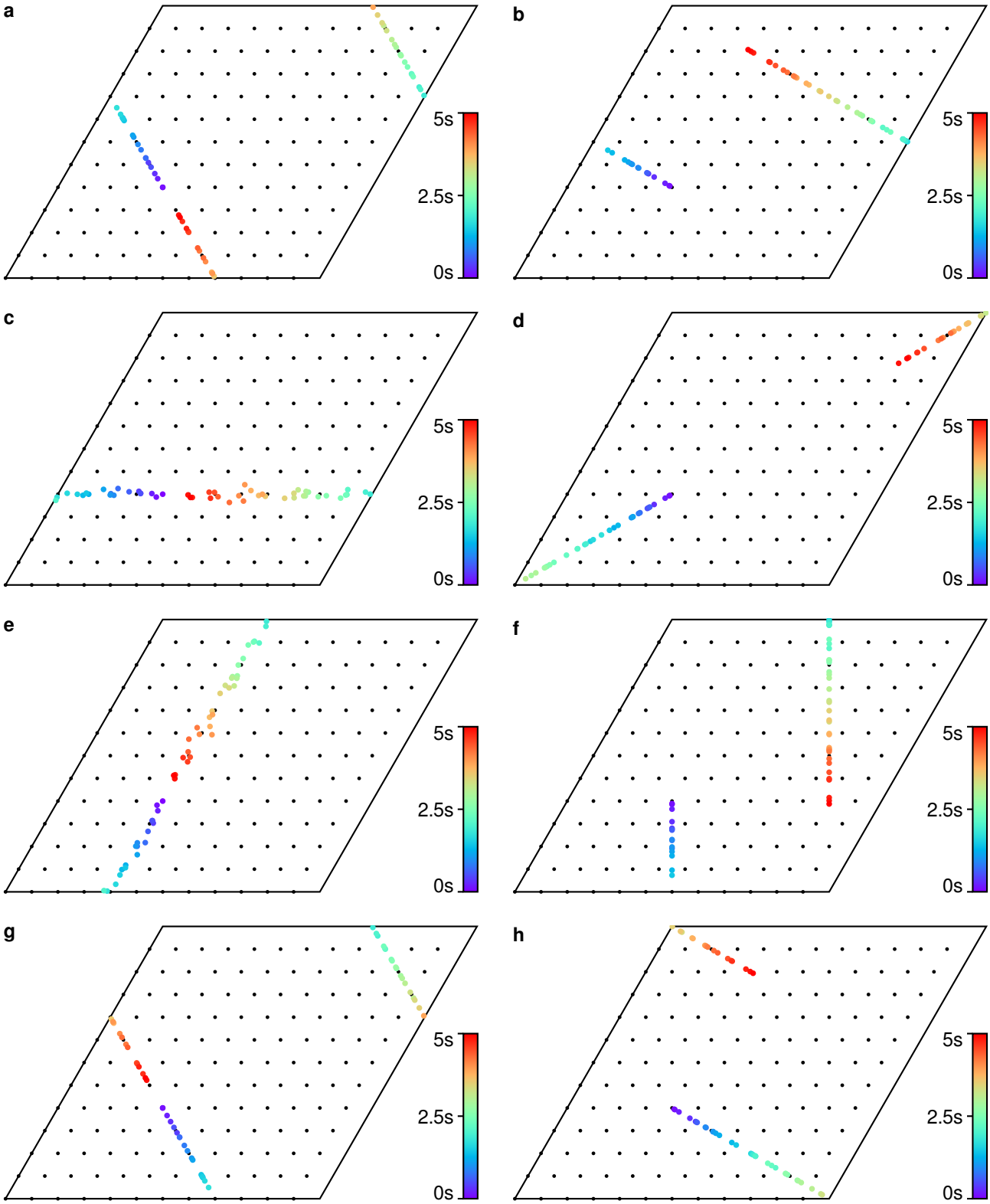

Figure A.1 Pure translational movement into different directions (complements Fig. 6.2). **a**, Initial pose at $t = 0$ is $\mathrm{P}(0) = (4, 4, 4)_{\mathrm{g}}$. **b**, $\mathrm{P}(0) = (4, 4, 5)_{\mathrm{g}}$. **c**, $\mathrm{P}(0) = (4, 4, 6)_{\mathrm{g}}$. **d**, $\mathrm{P}(0) = (4, 4, 7)_{\mathrm{g}}$. **e**, $\mathrm{P}(0) = (4, 4, 8)_{\mathrm{g}}$. **f**, $\mathrm{P}(0) = (4, 4, 9)_{\mathrm{g}}$. **g**, $\mathrm{P}(0) = (4, 4, 10)_{\mathrm{g}}$. **h**, $\mathrm{P}(0) = (4, 4, 11)_{\mathrm{g}}$.

Bibliography

[1] A. Aggarwal. “Neuromorphic VLSI realization of the hippocampal formation”. In: *Neural Networks* 77 (May 1, 2016), pp. 29–40. ISSN: 0893-6080. DOI: 10.1016/j.neunet.2016.01.011.

[2] M. A. Arbib. “From spatial navigation via visual construction to episodic memory and imagination”. In: *Biological Cybernetics* 114.2 (Apr. 2020). Publisher: Springer, pp. 139–167. ISSN: 14320770. DOI: 10.1007/S00422-020-00829-7.

[3] D. Ball et al. “OpenRatSLAM: An open source brain-based SLAM system”. In: *Autonomous Robots* 34.3 (Apr. 2013). Publisher: Springer, pp. 149–176. ISSN: 09295593. DOI: 10.1007/s10514-012-9317-9.

[4] A. Banino et al. “Vector-based navigation using grid-like representations in artificial agents”. In: *Nature 2018 557:7705* 557.7705 (May 2018). Publisher: Nature Publishing Group, pp. 429–433. ISSN: 1476-4687. DOI: 10.1038/s41586-018-0102-6.

[5] T. Bekolay et al. “Nengo: a Python tool for building large-scale functional brain models”. In: *Frontiers in Neuroinformatics* 7 (JAN Jan. 2014). Publisher: Frontiers, p. 48. ISSN: 1662-5196. DOI: 10.3389/fninf.2013.00048.

[6] J. L. Bellmund et al. *Navigating cognition: Spatial codes for human thinking*. ISSN: 10959203 Issue: 6415 Publication Title: Science Volume: 362. Nov. 2018. DOI: 10.1126/science.aat6766.

[7] Z. Bing et al. “A survey of robotics control based on learning-inspired spiking neural networks”. In: *Frontiers in Neurorobotics* 12 (July 2019). Publisher: Frontiers Media S.A., p. 35. ISSN: 16625218. DOI: 10.3389/fnbot.2018.00035.

[8] T. Bonnevie et al. “Grid cells require excitatory drive from the hippocampus”. In: *Nature Neuroscience* 16.3 (Mar. 2013). Number: 3 Publisher: Nature Publishing Group, pp. 309–317. ISSN: 1546-1726. DOI: 10.1038/nn.3311.

[9] E. Bostock, R. U. Muller, and J. L. Kubie. “Experience-dependent Modifications of Hippocampal Place Cell Firing”. In: *Hippocampus* 1.2 (Apr. 1991). Publisher: John Wiley & Sons, Ltd, pp. 193–205. ISSN: 10981063. DOI: 10.1002/hipo.450010207.

[10] Y. Burak and I. R. Fiete. “Accurate Path Integration in Continuous Attractor Network Models of Grid Cells”. In: *PLOS Computational Biology* 5.2 (Feb. 20, 2009). Publisher: Public Library of Science, e1000291. ISSN: 1553-7358. DOI: 10.1371/journal.pcbi.1000291.

[11] N. Burgess, C. Barry, and J. O’Keefe. “An oscillatory interference model of grid cell firing”. In: *Hippocampus* 17.9 (2007), pp. 801–812. ISSN: 1050-9631. DOI: 10.1002/hipo.20327.

[12] J. Cepelewicz. *How Animals Map 3D Spaces Surprises Brain Researchers*. Quanta Magazine. Oct. 14, 2021. URL: https://www.quantamagazine.org/how-animals-map-3d-spaces-surprises-brain-researchers-20211014/ (visited on 02/15/2022).

[13] J. Conklin and C. Eliasmith. “A Controlled Attractor Network Model of Path Integration in the Rat”. In: *Journal of Computational Neuroscience* 18.2 (Mar. 1, 2005), pp. 183–203. ISSN: 1573-6873. DOI: 10.1007/s10827-005-6558-z.

[14] D. Corneil and W. Gerstner. “Attractor Network Dynamics Enable Preplay and Rapid Path Planning in Maze–like Environments”. In: *Advances in Neural Information Processing Systems*. Vol. 28. ISSN: 10495258. Curran Associates, Inc., 2015, pp. 1684–1692. URL: https://proceedings.neurips.cc/paper/2015/file/e515df0d202ae52fcebb14295743063b-Paper.pdf.

[15] J. J. Couey et al. “Recurrent inhibitory circuitry as a mechanism for grid formation”. In: *Nature Neuroscience* 16.3 (Mar. 2013). Number: 3 Publisher: Nature Publishing Group, pp. 318–324. ISSN: 1546-1726. DOI: 10.1038/nn.3310.

[16] M. Davies et al. “Loihi: A Neuromorphic Manycore Processor with On-Chip Learning”. In: *IEEE Micro* 38.1 (Jan. 2018). Publisher: IEEE Computer Society, pp. 82–99. ISSN: 02721732. DOI: 10.1109/MM.2018.112130359.

[17] V. Edvardsen. “Long-range navigation by path integration and decoding of grid cells in a neural network”. In: *2017 International Joint Conference on Neural Networks (IJCNN)*. 2017 International Joint Conference on Neural Networks (IJCNN). ISSN: 2161-4407. May 2017, pp. 4348–4355. DOI: 10.1109/IJCNN.2017.7966406.

[18] C. Eliasmith and C. H. Anderson. *Neural engineering: computation, representation, and dynamics in neurobiological systems*. Computational neuroscience. Cambridge, Mass: MIT Press, 2003. 356 pp. ISBN: 978-0-262-05071-5.

[19] R. A. Epstein et al. *The cognitive map in humans: Spatial navigation and beyond*. ISSN: 15461726 Issue: 11 Pages: 1504–1513 Publication Title: Nature Neuroscience Volume: 20. Nov. 2017. DOI: 10.1038/nn.4656.

[20] T. Evans et al. “How environment and self-motion combine in neural representations of space”. In: *The Journal of Physiology* 594.22 (2016), pp. 6535–6546. ISSN: 1469-7793. DOI: 10.1113/JP270666.

[21] J. M. Fellous, P. Dominey, and A. Weitzenfeld. “Complex spatial navigation in animals, computational models and neuro-inspired robots”. In: *Biological Cybernetics 2020 114:2* 114.2 (Apr. 2020). Publisher: Springer, pp. 137–138. ISSN: 1432-0770. DOI: 10.1007/S00422-020-00832-Y.

[22] M. Fieweger. *Neuromorphic Implementation of Place Cells for Robot Navigation on Intel Loihi*. 2020.

[23] A. Finkelstein et al. “Three-dimensional head-direction coding in the bat brain”. In: *Nature* 517.7533 (Jan. 2015). Number: 7533 Publisher: Nature Publishing Group, pp. 159–164. ISSN: 1476-4687. DOI: 10.1038/nature14031.

[24] V. L. Flanagin. “Spatial Orientation and Navigation”. Lecture: Computational Neuroscience. Technical University of Munich, June 15, 2021.

[25] M. C. Fuhs and D. S. Touretzky. “A Spin Glass Model of Path Integration in Rat Medial Entorhinal Cortex”. In: *Journal of Neuroscience* 26.16 (Apr. 19, 2006). Publisher: Society for Neuroscience Section: Articles, pp. 4266–4276. ISSN: 0270-6474, 1529-2401. DOI: 10.1523/JNEUROSCI.4353-05.2006.

[26] R. Gao et al. *Learning Grid Cells as Vector Representation of Self-Position Coupled with Matrix Representation of Self-Motion*. May 24, 2019. DOI: 10.48550/arXiv.1810.05597.

[27] R. J. Gardner et al. “Toroidal topology of population activity in grid cells”. In: *Nature* 602.7895 (Feb. 2022). Number: 7895 Publisher: Nature Publishing Group, pp. 123–128. ISSN: 1476-4687. DOI: 10.1038/s41586-021-04268-7.

[28] G. Ginosar et al. “Locally ordered representation of 3D space in the entorhinal cortex”. In: *Nature* 596.7872 (Aug. 2021). Number: 7872 Publisher: Nature Publishing Group, pp. 404–409. ISSN: 1476-4687. DOI: 10.1038/s41586-021-03783-x.

[29] L. M. Giocomo, M.-B. Moser, and E. I. Moser. “Computational Models of Grid Cells”. In: *Neuron* 71.4 (Aug. 25, 2011). Publisher: Elsevier, pp. 589–603. ISSN: 0896-6273. DOI: 10.1016/j.neuron.2011.07.023.

[30] R. M. Grieves and K. J. Jeffery. “The representation of space in the brain”. In: *Behavioural Processes* 135 (Feb. 2017). Publisher: Elsevier, pp. 113–131. ISSN: 0376-6357. DOI: 10.1016/J.BEPROC.2016.12.012.

[31] A. Gruening and S. Bohte. “Spiking Neural Networks: Principles and Challenges”. In: (Apr. 1, 2014). URL: https://ir.cwi.nl/pub/22918 (visited on 12/16/2022).

[32] A. Guanella, D. Kiper, and P. Verschure. “A model of grid cells based on a twisted torus topology”. In: *International Journal of Neural Systems* 17.4 (Aug. 2007), pp. 231–240. ISSN: 0129-0657. DOI: 10.1142/S0129065707001093.

[33] T. Hafting et al. “Microstructure of a spatial map in the entorhinal cortex”. In: *Nature* 436.7052 (Aug. 2005). Number: 7052 Publisher: Nature Publishing Group, pp. 801–806. ISSN: 1476-4687. DOI: 10.1038/nature03721.

[34] A. V. Herz, A. Mathis, and M. Stemmler. “Periodic population codes: From a single circular variable to higher dimensions, multiple nested scales, and conceptual spaces”. In: *Current Opinion in Neurobiology* 46 (Oct. 2017). Publisher: Elsevier Current Trends, pp. 99–108. ISSN: 0959-4388. DOI: 10.1016/J.CONB.2017.07.005.

[35] R. Kreiser et al. “A Neuromorphic Approach to Path Integration: A Head-Direction Spiking Neural Network with Vision-driven Reset”. In: *Proceedings - IEEE International Symposium on Circuits and Systems*. Vol. 2018-May. ISSN: 02714310. Institute of Electrical and Electronics Engineers Inc., Apr. 2018. ISBN: 978-1-5386-4881-0. DOI: 10.1109/ISCAS.2018.8351509.

[36] R. Kreiser et al. “An On-chip Spiking Neural Network for Estimation of the Head Pose of the iCub Robot”. In: *Frontiers in Neuroscience* 14 (2020). ISSN: 1662-453X. DOI: 10.3389/fnins.2020.00551.

[37] R. Kreiser et al. “Pose Estimation and Map Formation with Spiking Neural Networks: Towards Neuromorphic SLAM”. In: *IEEE International Conference on Intelligent Robots and Systems*. ISSN: 21530866. Institute of Electrical and Electronics Engineers Inc., Dec. 2018, pp. 2159–2166. ISBN: 978-1-5386-8094-0. DOI: 10.1109/IROS.2018.8594228.

[38] A. Krishna et al. “Biomimetic FPGA-based spatial navigation model with grid cells and place cells”. In: *Neural Networks* 139 (July 2021). Publisher: Pergamon, pp. 45–63. ISSN: 0893-6080. DOI: 10.1016/J.NEUNET.2021.01.028.

[39] P. Lucas et al. *Neurorobotic Mouse (NeRmo) V4.1*. TUM-I2081. 2020. DOI: 10.14459/2020md1540519.

[40] A. Mathis, A. V. Herz, and M. B. Stemmler. “Resolution of Nested Neuronal Representations Can Be Exponential in the Number of Neurons”. In: *Physical Review Letters* 109.1 (July 2012). Publisher: American Physical Society, p. 018103. ISSN: 00319007. DOI: 10.1103/PHYSREVLETT.109.018103.

[41] A. Mathis, M. B. Stemmler, and A. V. Herz. “Probable nature of higher-dimensional symmetries underlying mammalian grid-cell activity patterns”. In: *eLife* 4 (Apr. 24, 2015). Ed. by M. S. Goldman. Publisher: eLife Sciences Publications, Ltd, e05979. ISSN: 2050-084X. DOI: 10.7554/eLife.05979.

[42] B. L. McNaughton et al. “Path integration and the neural basis of the ’cognitive map’”. In: *Nature Reviews Neuroscience 2006 7:8* 7.8 (Aug. 2006). Publisher: Nature Publishing Group, pp. 663–678. ISSN: 1471-0048. DOI: 10.1038/nrn1932.

[43] M. J. Milford, G. F. Wyeth, and D. Prasser. “RatSLAM: A hippocampal model for simultaneous localization and mapping”. In: *Proceedings - IEEE International Conference on Robotics and Automation*. Vol. 2004. ISSN: 10504729 Issue: 1. Institute of Electrical and Electronics Engineers Inc., 2004, pp. 403–408. DOI: 10.1109/robot.2004.1307183.

[44] M. J. Milford, J. Wiles, and G. F. Wyeth. “Solving navigational uncertainty using grid cells on robots”. In: *PLoS computational biology* 6.11 (Nov. 11, 2010), e1000995. ISSN: 1553-7358. DOI: 10.1371/journal.pcbi.1000995.

[45] M. J. Milford and G. F. Wyeth. “Mapping a suburb with a single camera using a biologically inspired SLAM system”. In: *IEEE Transactions on Robotics* 24.5 (2008), pp. 1038–1053. ISSN: 15523098. DOI: 10.1109/TRO.2008.2004520.

[46] J. Monteiro, A. Pedro, and A. J. Silva. “A Gray Code model for the encoding of grid cells in the Entorhinal Cortex”. In: *Neural Computing and Applications* 34.3 (Feb. 1, 2022), pp. 2287–2306. ISSN: 1433-3058. DOI: 10.1007/s00521-021-06482-w.

[47] E. I. Moser, M.-B. Moser, and B. L. McNaughton. “Spatial representation in the hippocampal formation: a history”. In: *Nature Neuroscience* 20.11 (Nov. 2017). Number: 11 Publisher: Nature Publishing Group, pp. 1448–1464. ISSN: 1546-1726. DOI: 10.1038/nn.4653.

[48] *NEF summary — Nengo 3.2.0 docs*. Jan. 27, 2022. URL: https://www.nengo.ai/nengo/v3.2.0/examples/advanced/nef-summary.html (visited on 11/23/2022).

[49] M. F. Nolan. “Neural mechanisms for spatial computation”. In: *The Journal of Physiology* 594.22 (2016), pp. 6487–6488. ISSN: 1469-7793. DOI: 10.1113/JP273087.

[50] H. A. Obenhaus et al. “Functional network topography of the medial entorhinal cortex”. In: *Proceedings of the National Academy of Sciences* 119.7 (Feb. 15, 2022). Publisher: Proceedings of the National Academy of Sciences, e2121655119. DOI: 10.1073/pnas.2121655119.

[51] T. Odland. *Fourier analysis on abelian groups; theory and applications*. Accepted: 2018-03-08T14:43:05Z Publisher: The University of Bergen. 2017. URL: https://bora.uib.no/bora-xmlui/handle/1956/17498 (visited on 04/10/2022).

[52] B. Osgood. “Chapter 8: n-dimensional Fourier Transform”. In: *Lecture Notes for EE* 261 (). URL: https://see.stanford.edu/materials/lsoftaee261/chap8.pdf (visited on 12/05/2022).

[53] G. Rammes. “Hippocampus: Place, Grid, and Head-Direction Cells”. Lecture: Cognitive Neuroscience. Technical University of Munich, June 21, 2021.

[54] F. Sargolini et al. “Conjunctive Representation of Position, Direction, and Velocity in Entorhinal Cortex”. In: *Science* (May 5, 2006). Publisher: American Association for the Advancement of Science. DOI: 10.1126/science.1125572.

[55] P. Scleidorovich et al. “A computational model for spatial cognition combining dorsal and ventral hippocampal place field maps: multiscale navigation”. In: *Biological Cybernetics* 114 (2020), pp. 187–207. DOI: 10.1007/s00422-019-00812-x.

[56] M. Sosa and L. M. Giocomo. “Navigating for reward”. In: *Nature Reviews Neuroscience 2021 22:8* 22.8 (July 2021). ISBN: 0123456789 Publisher: Nature Publishing Group, pp. 472–487. ISSN: 1471-0048. DOI: 10.1038/s41583-021-00479-z.

[57] D. Spalla, A. Treves, and C. N. Boccara. “Angular and linear speed cells in the parahippocampal circuits”. In: *Nature Communications* 13.1 (Apr. 7, 2022). Number: 1 Publisher: Nature Publishing Group, p. 1907. ISSN: 2041-1723. DOI: 10.1038/s41467-022-29583-z.

[58] M. Stemmler, A. Mathis, and A. V. Herz. “Connecting multiple spatial scales to decode the population activity of grid cells”. In: *Science Advances* 1.11 (Dec. 2015). Publisher: American Association for the Advancement of Science. ISSN: 23752548. DOI: 10.1126/SCIENCE.1500816.

[59] H. Stensola et al. “The entorhinal grid map is discretized”. In: *Nature* 492.7427 (Dec. 2012). Number: 7427 Publisher: Nature Publishing Group, pp. 72–78. ISSN: 1476-4687. DOI: 10.1038/nature11649.

[60] J. Sturm et al. “A benchmark for the evaluation of RGB-D SLAM systems”. In: *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*. 2012 IEEE/RSJ International Conference on Intelligent Robots and Systems. ISSN: 2153-0866. Oct. 2012, pp. 573–580. DOI: 10.1109/IROS.2012.6385773.

[61] L. Svenningsson. *Fourier transform of BCC and FCC lattices for MRI applications*. 2015. URL: http://urn.kb.se/resolve?urn=urn:nbn:se:uu:diva-257148 (visited on 04/10/2022).

[62] T. Taketomi, H. Uchiyama, and S. Ikeda. *Visual SLAM algorithms: A survey from 2010 to 2016*. ISSN: 18826695 Pages: 16 Publication Title: IPSJ Transactions on Computer Vision and Applications Volume: 9. 2017. DOI: 10.1186/s41074-017-0027-2.

[63] H. Tang, R. Yan, and K. C. Tan. "Cognitive Navigation by Neuro-Inspired Localization, Mapping, and Episodic Memory". In: *IEEE Transactions on Cognitive and Developmental Systems* 10.3 (Sept. 2018). Conference Name: IEEE Transactions on Cognitive and Developmental Systems, pp. 751–761. ISSN: 2379-8939. DOI: 10.1109/TCDS.2017.2776965.

[64] A. A. Vergani and C. R. Huyck. *Critical Limits in a Bump Attractor Network of Spiking Neurons*. Mar. 30, 2020. DOI: 10.48550/arXiv.2003.13365.

[65] A. Vince and X. Zheng. "Computing the Discrete Fourier Transform on a Hexagonal Lattice". In: *Journal of Mathematical Imaging and Vision* 28.2 (June 1, 2007), pp. 125–133. ISSN: 1573-7683. DOI: 10.1007/s10851-007-0013-x.

[66] J. Wang, R. Yan, and H. Tang. "Grid cell modeling with mapping representation of self-motion for path integration". In: *Neural Computing and Applications* (Feb. 2, 2022). ISSN: 1433-3058. DOI: 10.1007/s00521-021-06039-x.

[67] J. Wang, R. Yan, and H. Tang. "Multi-Scale Extension in an Entorhinal-Hippocampal Model for Cognitive Map Building". In: *Frontiers in Neurorobotics* 0 (2021). Publisher: Frontiers. ISSN: 1662-5218. DOI: 10.3389/fnbot.2020.592057.

[68] P. Yang. "Materials & Solid State Chemistry". Lecture: Chemistry 253. UC Berkeley, 2016. URL: http://nanowires.berkeley.edu/teaching/253a/2016/253A-2016-01.pdf (visited on 12/17/2022).

[69] F. Yu et al. "NeuroSLAM: a brain-inspired SLAM system for 3D environments". In: *Biological Cybernetics* 113.5 (Dec. 2019). Publisher: Springer Verlag, pp. 515–545. ISSN: 14320770. DOI: 10.1007/s00422-019-00806-9.

[70] X. Zheng and F. Gu. "Fast Fourier Transform on FCC and BCC Lattices with Outputs on FCC and BCC Lattices Respectively". In: *Journal of Mathematical Imaging and Vision* 49.3 (July 1, 2014), pp. 530–550. ISSN: 1573-7683. DOI: 10.1007/s10851-013-0485-9.