# Spiking neural networks solve robot planning problems

**Spiking neural networks zum Lösen von Planungsproblemen für Roboter**
Master-Thesis von Daniel Tanneberg
September 2015

TECHNISCHE
UNIVERSITÄT
DARMSTADT

Spiking neural networks solve robot planning problems
Spiking neural networks zum Lösen von Planungsproblemen für Roboter

Vorgelegte Master-Thesis von Daniel Tanneberg

1. Gutachten: Prof. Dr. Jan Peters
2. Gutachten: Dr. Elmar Rueckert

Tag der Einreichung:

# Erklärung zur Master-Thesis

Hiermit versichere ich, die vorliegende Master-Thesis ohne Hilfe Dritter nur mit den angegebenen Quellen und Hilfsmitteln angefertigt zu haben. Alle Stellen, die aus Quellen entnommen wurden, sind als solche kenntlich gemacht. Diese Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen.

Darmstadt, den 21. September 2015

_____

(Daniel Tanneberg)

# Abstract

We propose here a novel approach to solve robot planning problems based on spiking neural network models. The method is motivated by recent neuroscience findings on how rodents create mental plans for maze navigation and is grounded in the framework of planning as probabilistic inference. In this thesis, we demonstrate that the proposed spiking neural network is a suitable alternative to classical approaches and comes with interesting features.

Neural networks can be used in massive parallel computing, e.g., when implemented in neuromorphic hardware. These brain-like chips consist of thousands of memory and processing units operating in parallel. However, we are lacking suitable learning rules and algorithms. The developments in this thesis provide first testable algorithms for real-world robot planning applications.

Arbitrary complex functions can be learned such as dynamic or kinematic models. For that, a spike dependent version of contrastive divergence was derived to learn non-linear functions with kinesthetic teaching.

We show that these models can scale to a six-dimensional KUKA robot system, where in addition to an existing two-dimensional task space planning model two additional models were developed. One of these models can be queried in both directions, enabling that forward and inverse models can be learned at the same time.

Obstacles of arbitrary shape can be encoded in form of repelling forces through synaptic inhibition. Sampling of movement plans is done $4 - 60$ times faster than real-time, which allows for foraging robot control, preparing multiple alternative solutions and deciding online which plan to execute.

With the additionally implemented online rejection sampling, we could achieve target reaching errors of 4% in the modelled operational area. Furthermore, the generated movement trajectories did not require any post processing. Using bidirectional feedback between task and joint space during planning, smooth and goal-directed movements were computed at the same time.

# Zusammenfassung

In dieser Arbeit stellen wir einen neuen Ansatz zum Lösen von Planungsproblemen für Roboter vor, der auf *spiking neural networks* basiert. Die Methode ist inspiriert von neurowissenschaftlichen Erkenntnissen darüber wie Nagetiere mentale Pläne zum Navigieren in Labyrinthen erstellen, und ist eingebettet im Framework *planning as probabilistic inference*. Wir zeigen, dass das vorgeschlagene *spiking neural network* eine geeignete Alternative zu klassischen Planungsmethoden darstellt und interessante Eigenschaften mit sich bringt.

Neuronale Netzwerke können für massiv parallele Berechnungen benutzt werden, z.B., wenn sie auf *neuromorpher* Hardware implementiert werden. Diese Gehirn-artigen Chips bestehen aus tausenden von Speicher- und Recheneinheiten welche parallel arbeiten. Jedoch mangelt es noch an passenden Lernregeln und Algorithmen. Die in dieser Arbeit entwickelten Methoden stellen erste prüfbare Algorithmen für reale Roboter Planungsprobleme zur Verfügung.

Das vorgestellte Modell kann beliebig komplexe Funktionen lernen, wie zum Beispiel *dynamic* oder *kinematic* Modelle. Um nicht-lineare Funktionen mit *kinesthetic teaching* zu Lernen, haben wir eine auf *spikes* basierende Version von *contrastive divergence* entwickelt.

Wir zeigen, dass diese Modelle auf ein sechsdimensionales KUKA Roboter System skalieren und entwickeln zusätzlich zu einem existierenden zweidimensionalem *task space* Planungsmodell zwei weitere Modelle. Eines dieser Modelle kann in zwei Richtungen benutzt werden, was es ermöglicht, Vorwärts- und Rückwärtsmodelle zur gleichen Zeit zu lernen.

Hindernisse von beliebiger Form können durch synaptische Inhibition als abstoßenden Kräfte modelliert werden. Das Samplen von Bewegungsplänen ist $4 - 60$ mal schneller als Realzeit, was vorausschauende Roboterkontrolle ermöglicht indem mehrere Alternativen vorgeschlagen werden können und online entschieden wird, welche ausgeführt werden soll.

Durch zusätzlich implementiertes *rejection sampling* konnten wir einen Abweichungsfehler am Ziel von ungefähr 4% im modellierten Bewegungsgebiet erreichen. Außerdem brauchen die generierten Bewegungspläne keine nachträgliche Bearbeitung. Durch bidirektionales Feedback zwischen *task* und *joint space* während des Planens, können glatte und zielgerichtete Bewegungen gleichzeitig generiert werden.

# Contents

# Figures and Tables

## List of Figures



## List of Tables

# 1 Introduction

## 1.1 Motivation & Goals

Integrating robots more and more into our everyday life and work is a big vision in robotics. This will have a great impact in many areas and will surely change habits in our everyday life and work. For example, autonomous robots may assemble cars or assist in medicine, help in the household, or work in environments which are dangerous for humans. Independent from where the robots will work and what they should do, planning is a fundamental skill that is required in almost all robot tasks.

Up to date, the capabilities of spiking neural networks (SNNs) for planning have been poorly explored, although spiking neural networks have some very useful properties and are computationally powerful models for brain functions. They can model arbitrary complex distributions like multi-modal distributions, which is, for example, required for encoding multiple solutions for planning problems. However, a big drawback is their computational time and complexity, especially in larger networks. Due to their massive parallel computing ability and due to the huge effort that is undertaken in research of neuromorphic hardware, real-time computation of (large and complex) SNNs may become feasible soon. Thus, it is worth to investigate the capabilities of SNNs for robotic problems. Implementations on neuromorphic hardware promise to be able to process large input streams from, for example, visual and tactile sensors using parallel computing[35]. When comparing the energy consumption, event based neural network implementations are more efficient than classical von Neumann architectures[2]. With this work we want to show that it is feasible to adapt biological mechanisms with recurrent spiking neural networks on robot planning problems and that these models can be learned from human demonstrations.

### Robot planning

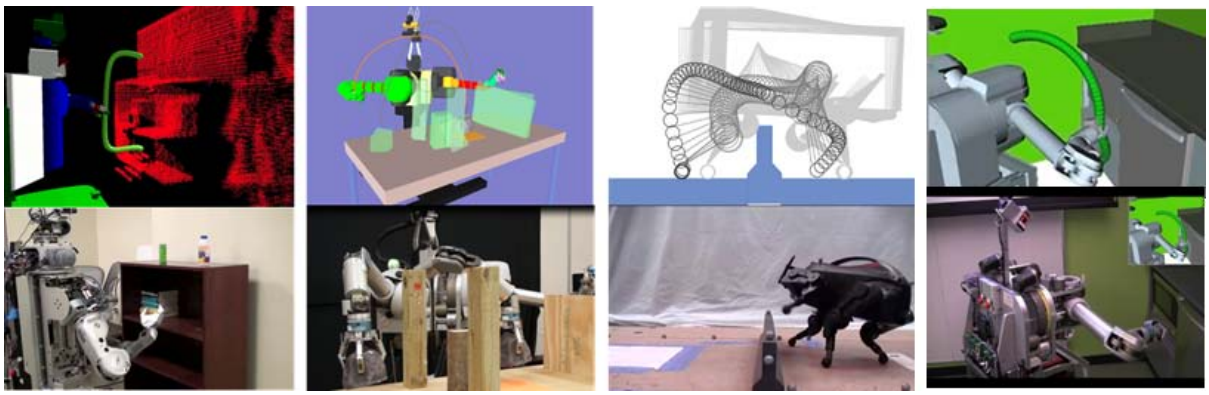

**Figure 1.1.:** Multiple planning tasks in different environments and with diverse robots. The pictures in the upper row show simulation results. The pictures in the lower row show results of real robots. In the presented examples, the robots have to solve different planning tasks, e.g., object grasping and manipulation, avoiding obstacles and motor planning with balancing constraints. Taken from [45].

In order to plan a movement in an environment, it is necessary to have a time dependent description of the environment, referred as the **state**. All possibles situations that can arise are captured by the *state space*. A state could represent different features of the environment and the task to solve, e.g., the

position and orientation of the robot. State spaces can be discrete or continuous, finite or infinite. In most real applications the state space is too large and cannot be represented explicitly. However, in the most simple case, only an *initial* and a *goal* state are given. They define the current state of the robot, where it is or should be at the start, and the desired future target state, where the robot should be at the end of the planned movement.

To reach this target state from the initial state, the robot needs to apply certain **actions** from a set of possible actions. Actions manipulate the state and transform one state into another state. This manipulation, i.e., how the state changes when an action is applied, needs to be specified in the formulation of the planning problem. In some problems, actions are not specified explicitly and the output of the planning algorithm is a sequence of states instead of actions. The actions can then be inferred from the sequence of states as they arise *naturally* from the problem setting.

In many planning problems a **reward** is defined, that specifies how *good* a generated solution is. This reward can take many different features into account, e.g., how precise a movement is, how smooth it is or if it violates any constraints. It can be a simple binary reward, e.g., denoting only if a goal is reached, or a continuous reward, grading the movement. In general the goal is to maximize the expected reward.

The output of a planning algorithm is, in general, a sequence of actions, the **plan**. In a more complicated setup, the future states cannot be predicted and the output specifies actions as a function of state, i.e., a *policy*. There are two major criteria of plans: **feasibility** and **optimality**. Feasibility means, finding any plan that achieves the desired goal and satisfying all constraints. Optimality means, finding a *feasible* plan that additionally is optimal for some specified criterion, e.g., minimizes the required energy to execute it or finding shortest solution.

#### Challenges in robot planning

In most real world problems and applications, robot motion planning takes place in a continuous state space. A *motion plan* consists of motions (i.e., actions) which are appropriate for the robot and which lead the robot to the desired goal state, without hitting any obstacle and satisfying all constraints. Depending on the application, the order of importance may change, i.e., in some applications avoiding obstacles is the primary goal while in other tasks satisfying a constraint is more important.

So far we have only talked about the state space that spans the space in the environment. For robot motion planning there is another important space, i.e., the space in which the robot *lives*. This space is called *joint space*. The dimensionality of this space is dependent on the degrees of freedom of the robot, e.g., the number of joints and motors of the robot. In this configuration space, motion planning can be seen as a search in a high-dimensional space in which obstacles are represented implicitly. The mapping from joint space into task space is called *forward kinematics* and the mapping from task space into joint space is called *inverse kinematics*. While the forward kinematics are well defined and form an one-to-one mapping, i.e., each joint configuration maps to one endeffector position, the inverse kinematics are more complex and difficult. Given a particular position of the endeffector, there are multiple (even up to infinite) joint configurations that lead to that endeffector position. This can be easily visualized for better understanding, e.g., place your hand at a certain position and keep it fixed there. Without changing the position of your hand (the endeffector in this case), you can still move the joints of your arm. As motion planning is often done in task space, computing or learning the inverse kinematics is required to transform the movements into joint trajectories that can be executed by the robot.

As nowadays real-world robots have to act in highly dynamic, unstructured, open and novel environments, they have do deal with different challenges to solve the given tasks. They need to deal with, e.g., noisy observations and singularities, while being able to adapt fast to changes in the environment and maintaining real-time computation to be useful for real problems.

### Biological inspiration

As planning is a fundamental skill robots need to be able to perform, there exist different sophisticated algorithms for solving robot planning problems. The motivation of this work is not just to provide another planning algorithm, but rather to propose a novel biological inspired approach that is inspired by the *natural* way of solving planning problems. Additionally, learning of the proposed model using Imitation Learning through kinesthetic teaching is discussed in this work.

What is meant by the *natural* way of solving planning problems and why do we use spiking neural network models? The proposed approach is motivated by recent results on how rats solve planning problems. More precisely, motivated by the neural activity recorded from hippocampal cells in rats while planning. These neural activities depict the path of the rat during maze navigation experiments. These activities can be observed not only during the execution of the plan, but also during the phase of *mental planning*. While the rat is thinking about its future path to a known target, the recorded neural activity of these specialized cells can be decoded into the future path of the rat. The goal of this work is to adapt this mechanism in an abstract and simplified way and use it to solve planning problems with robots. Finally, we show that the model can be learned from human demonstrations.

The model was first proposed in [38], where the theoretical foundation is introduced and simulation experiments reveal that it can reproduce the neural activity recorded in rat experiments. Parts of this thesis entered that publication, in particular, the adaption of the model for real robot problems, i.e., adding an obstacle avoidance task and learning the model from human demonstrations.

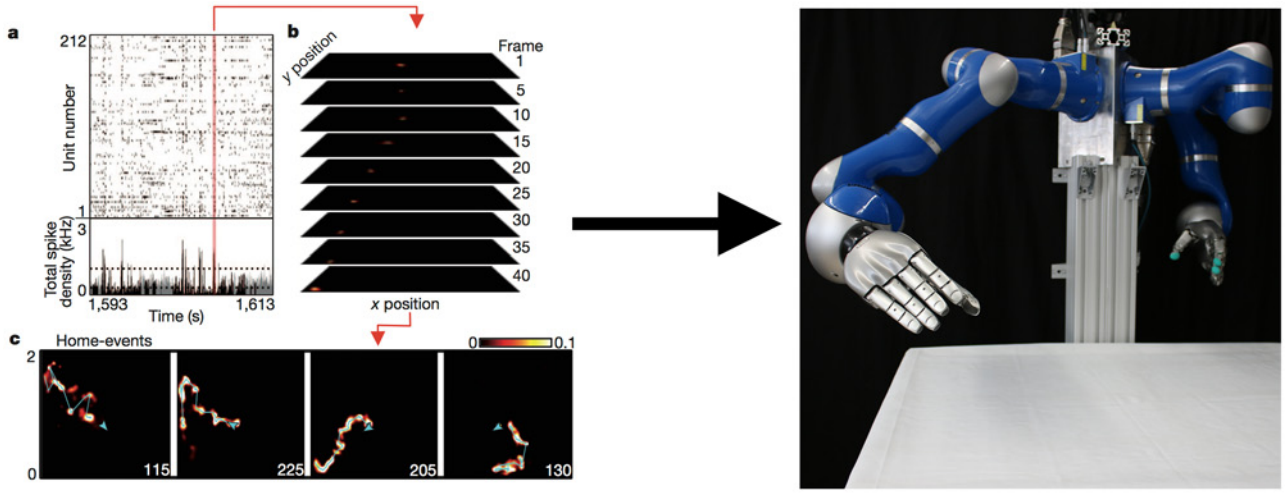


**Figure 1.2.:** Illustration of the goal of this work: transferring the biological concept of motion planning as it was discovered in rat recordings to motion planning for robots. The left picture shows the neural activity of cells from a rat during a spatial navigation task and the decoded movement trajectories. The right picture shows the KUKA lightweight arm we used for the experiments.

**Contribution**

We use a recurrent spiking neural network (SNN) as model structure that is grounded in the theory of probabilistic inference and can reproduce the transient firing of the rat experiments[38]. The SNN is an abstraction of biological neural networks. In planning problems, it is necessary to deal with temporal sequences and the distributions are time-varying. For such time-varying distributions SNNs are able to learn a generative model of temporal sequences, e.g., movement trajectories, through synaptic plasticity rules. After learning, the neural network performs effectively forward sampling from the target distribution. When running freely, the movement trajectories sampled from the neural network represent random walks. By injecting proper task related input to the network through additional task neurons, the samples are drawn from the posterior distribution representing the planning problem.

The basic structure of the models used in this work to solve planning problems consists of two different kinds of neuron populations, state and task neurons. While the state neurons model the planning space (either task or joint space) and encode the state transition model in their synaptic connections, the task neurons encode task related information like, for example, initial and target position or obstacles.

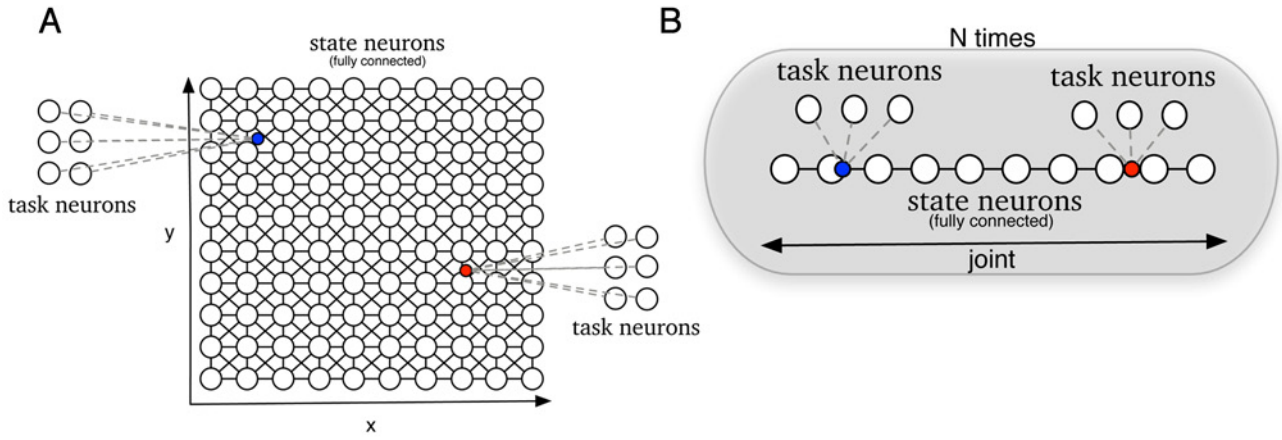


**Figure 1.3.:** (A) Two-dimensional task space planning model. (B) The factorized model for joint space planning consisting of $N$ independent one-dimensional models.

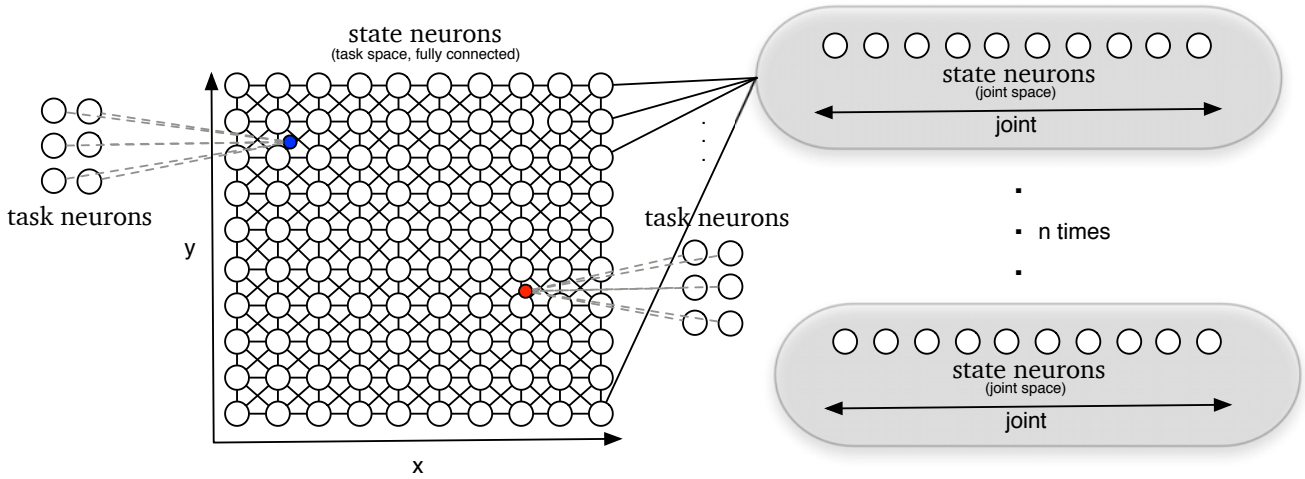


**Figure 1.4.:** Hierarchical model for task space planning and inverse kinematic mapping.

In this work, we propose three different spiking neural network models for planning. The first model is used for planning in a two-dimensional task space, i.e., planning in a Cartesian $xy$ plane spanning the reachable area of the robot endeffector. It uses a *full population code*, meaning that the state neurons fill up the entire space and are fully connected. This architecture suffers from the curse of dimensionality and the model becomes impractical when using more than three dimensions.

To scale up the model to higher dimensional problems, we use an approximation based on factorized models. E.g., for planning in six-dimensional joint space, we learned six independent one-dimensional models and combined their generated trajectories at the end. This makes planning in higher dimensional space feasible, but because the models do not share information during planning, this approach cannot be used in environments with obstacles.

To overcome this, we developed a third model that combines the two previous models. In this hierarchical model, we used the model for planning in two-dimensional task space to solve the given planning problem. Additionally, we extended that model with six factorized models that map the planned task space movement into joint space trajectories during planning.The six factorized models encode the inverse kinematics, which we learned from human demonstrations as well. During sampling of a movement trajectory, the spikes of the task space model are used as input to the inverse kinematic models to map the movement into joint space. The spikes from the inverse kinematic models are used as additional input to the task space planning model to provide feedback from the joints to the planning process. With this approach, we can generate higher dimensional joint space trajectories with a factorized spiking neural network model architecture, while having the advantages of the full population code of the task space model for obstacle avoidance.

## 1.2 Related Work

As motion planning is a fundamental task for robots, there have been a multitude of algorithms proposed and used over the years. Here, we want to give a brief overview of three commonly used algorithms, Rapidly-Exploring Random Trees (**RRTs**)[22], Stochastic Trajectory Optimization for Motion Planning (**STOMP**)[19] and Covariant Hamiltonian Optimization for Motion Planning (**CHOMP**)[36][45].

**RRTs** are a randomized data structure for path planning in some metric space, e.g., joint or task space. Path planning is generally be viewed as a search in that metric space for some continuous path connecting the initial and target states. RRTs are constructed such that all generated nodes and edges lie entirely in the *free* area of the space, i.e., they represent reachable and applicable locations or configurations, respectively paths. A RRT is iteratively built up starting from an initial state. In each iteration, a random valid state is generated and the *closest* (in terms of some distance metric) neighbouring node in the RRT is selected. Next the control input which minimizes the distance between the random state and the nearest node is selected. The new state, which is obtained by applying this control to the nearest node and predefined transformation rules, is added as a new node to the RRT. This new node is connected with a new edge corresponding to the control input to the nearest node. The algorithm converges if the target state or a fixed number of iterations is reached. To improve the convergence of the RRT algorithm, several extensions have been proposed [6][21][24].

In comparison to RTTs, that handles planning as a search, **STOMP** handles motion planning as an optimization problem. A smooth trajectory is found by minimizing some costs that include target reaching, obstacle avoidance and constraints. One big advantage of STOMP is that it can deal with more general cost functions, because it does not rely on gradients. Trajectories have a fixed length $T$ and are discretized over time into $N$ equally spaced waypoints. With that a trajectory can be represented as a simple vector $\theta \in \mathbb{R}^N$. The algorithm starts with an initial randomized, maybe even infeasible, trajectory vector. In each iteration $K$ new noisy trajectories are created by adding normal distributed noise to the trajectory vector. The costs of each of the $K$ trajectories are evaluated and transformed into probabilities. To update the trajectory vector $\theta$ the noisy parameters are combined probability-weighted for each timestep and for the final update step this combined parameter update is smoothed. The algorithm converges if the trajectory cost doesn't change any more.

**CHOMP** treats motion planning as an optimization problem as well, and minimizes a combination of smoothness and obstacle avoidance costs to improve an initial trajectory. In contrast to STOMP it requires gradient information for these objectives. Trajectories are represented as vectors of configurations over time (waypoints) as described before for STOMP, but CHOMP is designed to be invariant to the particular used trajectory representation. In each iteration the trajectory is improved by minimizing a local approximation (first-order Taylor expansion) of the cost function that allows only smooth perturbations to the trajectory. The resulting update rule is a special case of *covariant gradient descent*. The optimization problem in each iteration can be seen as a Lagrangian form of optimization. It tries to maximize the decrease in the objective function while allowing only small changes in the average acceleration of the trajectory.

## 1.3 Outlook

In Section 2, we give an overview of the biological background that motivated our approach. For that, we give a basic introduction into the brain and briefly introduce the cell types that are part of the brain's internal navigation system. We also introduce the machine learning algorithms for planning and model learning that we use, i.e., the framework of planning as probabilistic inference and contrastive divergence for model learning.

In Section 3, we discuss the spiking neural network models used in this work. First, a basic introduction of spiking neural networks is given and how samples can be drawn from those. After that, three different models are described and how the models can adapt to different planning tasks. Finally, we show how contrastive divergence can be used to learn the state transition models and the inverse kinematics from human demonstrations.

In Section 4, the experiments to evaluate our models are presented. First, we describe the learning progress of our models including the gathering of real robot training data. Next, we describe how the models are used to generate smooth movement trajectories. Finally, we evaluate our models on target reaching and obstacle avoidance tasks in simulation and on the KUKA lightweight arm.

In Section 5, we summarize our results and discuss possible future work.

# 2 Background

## 2.1 Planning & cognitive maps in the brain

To understand the motivation of the proposed model and the connection to the biological background, this section gives a brief overview of the concepts and experiments that motivated the spiking neural network model proposed here and used to solve robot planning problems.

### 2.1.1 The brain & neurons - A brief overview

The brain is by far the most complex system we know about and has been optimized through evolution over thousands of years. Although a huge effort was and still is undertaken, e.g., the *Human Brain Project*[1], the brain and its functionality are not well understood yet. This overview given here should be considered as a very rough and high-level description of the brain and its main units, the neurons. The given description should provide a basic knowledge to understand the biological inspiration of the proposed work.

The brain is the center of the (central) nervous system. It acts as the highest control and relay station, preprocessing sensory input, producing motor commands and is responsible for almost all functions of the body. Located in and protected by the skull the brain is close to the primary sensory organs for vision, hearing, balance, taste and smell. A typical human brain roughly consists of 16 billion neurons and each neuron has connections to up several thousand other neurons, building a huge, complex network. The brain is divided in different structural parts as well as in functional parts. Different brain regions are highly optimized for certain tasks, although a clear assignment of, e.g., cognitive tasks to a single region can't be made due to long distance synapses.

Neurons are the information processing units inside the brain (and the central nervous system) and are linked through synapse to other neurons. Synapses are the location where neurons exchange information through electrical or chemical processes changing the neurons environment. A synapse consists of two parts, the *presynapse* where the presynaptic neuron releases its information and the *postsynapse* where the postsynaptic neuron receives the released information. The exchange of information between the neurons is based on signal pulses called action potentials or *spikes*. They result in the release of *neurotransmitters* which are chemicals that carry the information from the presynaptic neuron through the synaptic cleft to the postsynaptic neuron. As a neuron has up to several thousand synapses it receives information from many other neurons at the same time and at different locations. The information received is spatio-temporal integrated and the result determines if the neuron should spike to propagate the information to the other postsynaptic neurons. Populations of interconnected neurons form a dynamic and complex network.

One of the above mentioned specialized areas inside the brain is the *hippocampus*, respectively mammals mostly have two hippocampi located in the medial temporal lobe of each side (hemisphere) of the brain. The temporal lobe is a part of the neocortex, the most outer part of the brain, and is located at the sides of the brain. The neocortex is folded very strong and medial means that it is located *to the inside*. The hippocampus plays an important role for memory and navigation, e.g., to transfer information from the short-term memory into the long-term memory, the connection of different spatial locations or for navigational tasks.

[1] `https://www.humanbrainproject.eu/`

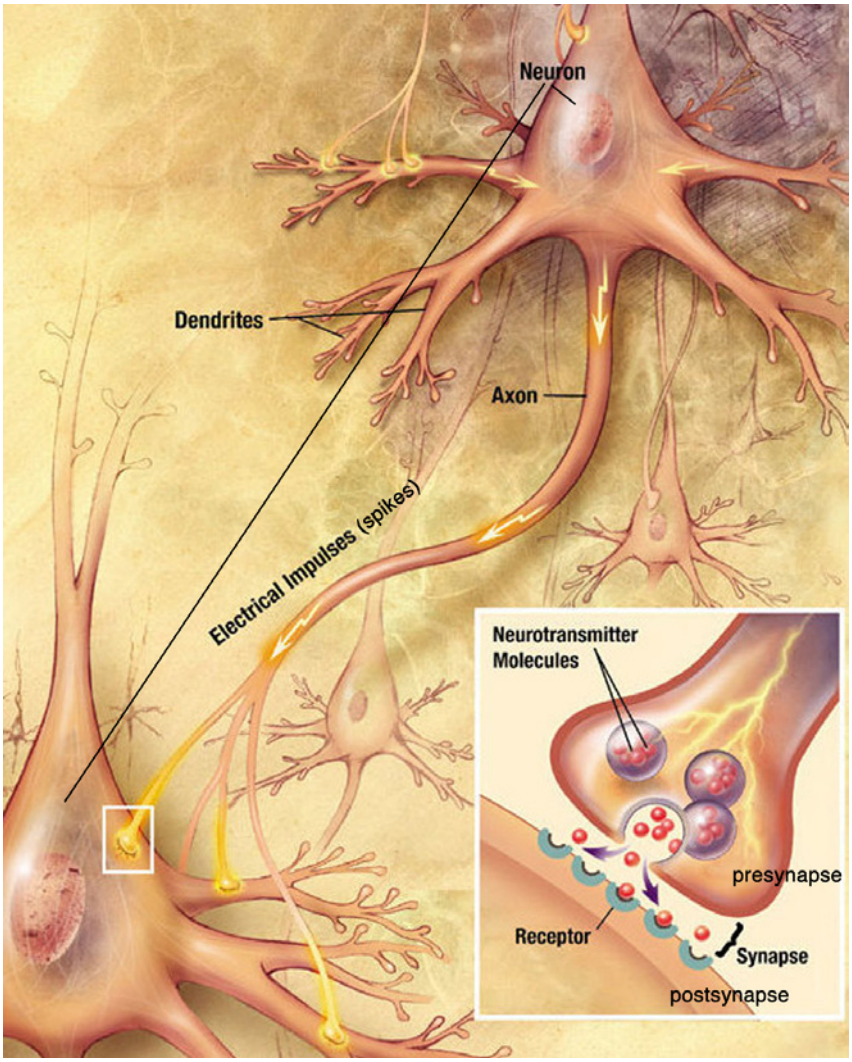


**Figure 2.1.:** Sketch of a synapse, neurons and their interaction. Picture from Wikipedia[1]

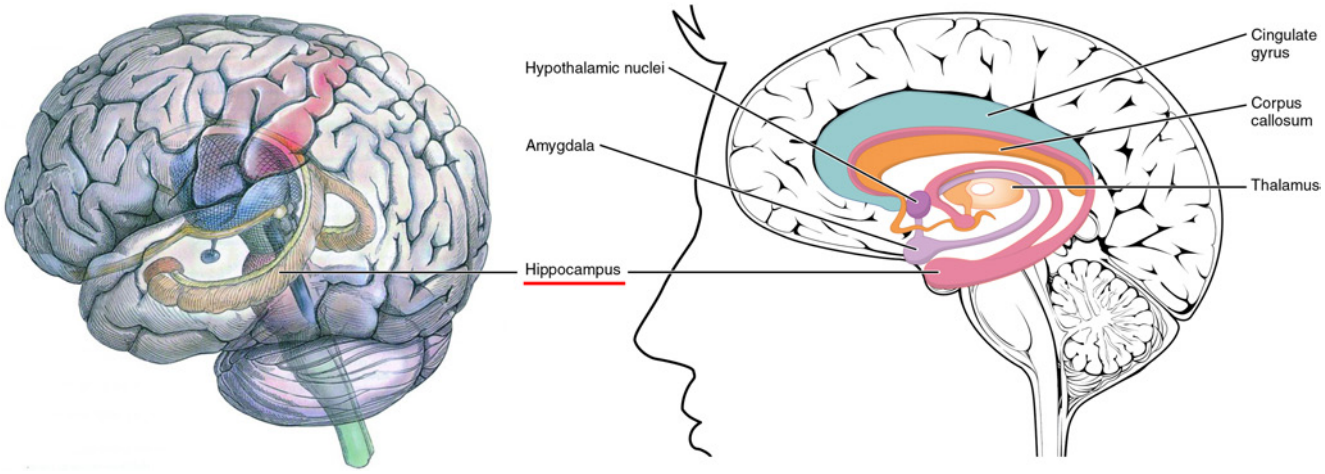


**Figure 2.2.:** Sketch of the anatomical position of the hippocampus. Pictures from Wikipedia[2] and NYU School of Medicine[3]

[1] https://en.wikipedia.org/wiki/Synapse
[2] https://en.wikipedia.org/wiki/Hippocampus
[3] http://www.med.nyu.edu/adc/participate-research/brain-donation

### 2.1.2 Place cells & the brain's internal navigation system

The brain's internal navigation system is a complex dynamic system and consists of different specialized parts and cells types, which are still under research and not completely understood. But so far, several different cell types that play important roles were identified. These different cell types are called border cells, head direction cells, grid cells and place cells.

**Border cells**[39] show an increased activity if there are boundaries in the environment at a particular distance and direction from the animals actual location. They help to form a model of the environment which is used during navigation tasks and movement planning. As the border cells model parts of the environment, they are dependent on the environment.

In contrast, the **head directions cells**[41][42] are independent from a certain environment and are even (mostly but not completely) independent from sensory input. They are mainly driven by the vestibular[1] system and show an increased activity for particular directions of the head. Thus, they help to perform self localization in the environment and are involved in planning movements as well.

A very important role within that complex system is played by the **grid cells**[11][15]. Grid cells basically encode the Euclidean space in a cognitive representation and by this they can model distance measurements. The grid cells show an increased activity if the animal enters small regions in the environment, but in contrast to the place cells described next, each grid cell is associated with multiple locations. These locations or small areas, called *firing fields*, are roughly equal in size and are arranged in a regular pattern of triangular arrays. Different grid cells have different firing fields, e.g., they can have the same size and spacing of the triangular arrays but are displaced from each other or they may have a different size and spacing of the triangular arrays. By this they can encode the Euclidean space, measure distances and play a very important role within the brains navigation system.

The last cell type we want to look at are the **place cells**[14][28][32] and their functionality. Place cells build the foundation of the model used in this work. Place cells are specialized neurons and are only located in the hippocampus. These cells have an increased activity if the animal enters a certain location in the environment. These locations are called *place fields* and denote the preferred location of the associated place cell. Depending on the size of the current environment, each place cell has only one or up to a few associated place fields. The population of place cells form a cognitive representation of the environment, known as a *cognitive map* [40]. In an abstract and simplified view, a place cell can be seen as a landmark to a certain location in the world. If the animal enters this location while walking or during *mental planning* the activity of the associated place cell is increasing. The spike pattern of the place cells are determined by the internal belief of the current state in the environment and external preprocessed sensory input. To adapt to changing environments, e.g., the rat is placed inside a new maze, place cells can change their spike pattern and place fields, which is called *remapping*.

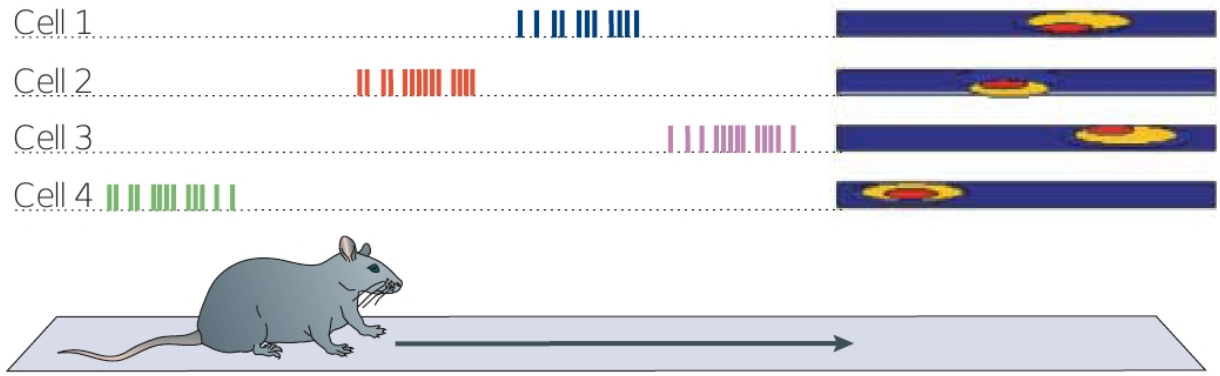


**Figure 2.3.:** Sketch of place cells activities and their related place fields as a rat moves along a track. At different locations different place cells have increased activity. Taken from [30].

[1] the sensory system that provides information about balance and spatial orientation

## 2.2 Transient firing in rats while planning

Recent results [8][33] showed the correlation of observed transient firing in behaving animals in phases of mental planning while solving different tasks and the resulting movements. The experiment in [33] is described here briefly, to show that the neural activity of hippocampal place cells is able to support the generation of goal-directed movement trajectories.

The environment of the experiment was a $2 \times 2m$ open-field arena in which there were 36 predefined and clearly separated locations. While the rat was solving the tasks of finding the liquid chocolate inside this arena, up to 250 hippocampal cells were recorded using an implanted miniaturized lightweight microdrive with 40 adjustable tetrodes. 20 tetrodes were targeted towards each dorsal hippocampal area CA1 . The setup of the tasks the rat had so solve was the following: each trial of the experiment was divided into two different phases. In the first phase the reward, in form of liquid chocolate, was located at a random and unknown location. So in order to get that reward the rat has to explore its environment. After the rat obtained the reward, a new reward was automatically placed at the home location of the rat. Such that to obtain the new reward the rat had to memorize the home location and plan a path back to this location. The recorded spike trains were decoded into a sequence of spatial locations which reflect the path of the rat. The neural activity of these specialized place cells depict a movement path to a given location during mental planning.

With our spiking neural network model we want to reproduce these observed neural activity of hippocampal place cells encoding future paths to use it for robot motion planning.

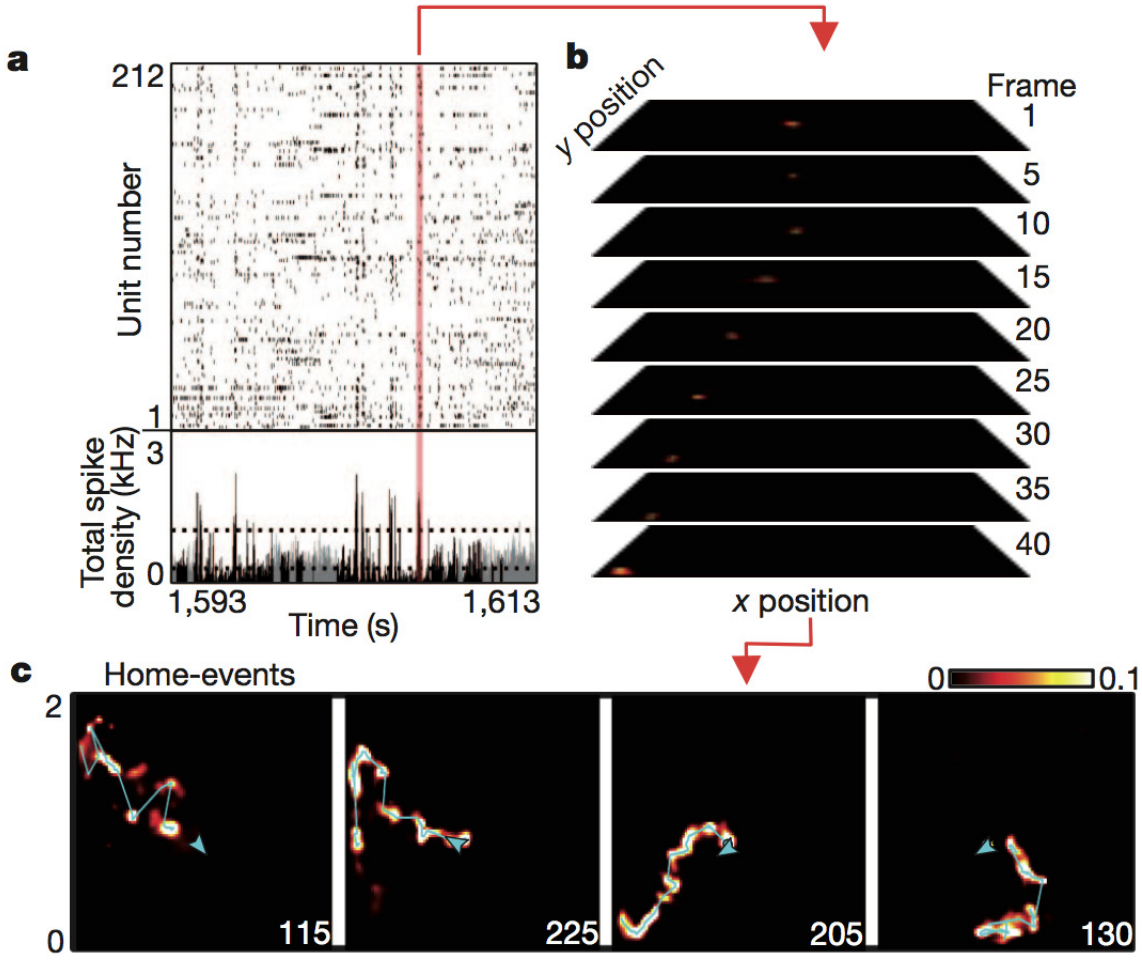


**Figure 2.4.:** The recorded neural activity (a) gets decoded into a sequence of spatial locations (b), which results in movement plan (c) reflecting the path of the rat. Taken from [33]

## 2.3 Machine learning algorithms for planning & model learning

This section gives a brief introduction and overview of the basic techniques used in this work like *planning as probabilistic inference* and a general description of the used model learning algorithm *contrastive divergence*.

### 2.3.1 Planning as probabilistic inference

In planning as probabilistic inference (PAI) [4], an agent makes use of an internal generative model that represents the future as a joint probability distribution over states, actions and rewards. The agent can infer a probability of any sequence of states and actions. To perform planning it can sample sequences from the model. But regarding that the agent has an internal generative model representing a probability distribution, it can do even better by conditioning on rewards. The agent assumes that its actions lead to a reward in each timestep and uses inverse inference to find the sequence of actions that explains this assumption best. Taking a more technical view, the agents action policy can be seen as a set of parameters which define a probability distribution over actions for each state. Planning as probabilistic inference then finds the maximum likelihood estimate of those actions conditioned on receiving reward.

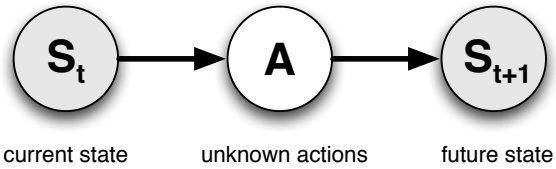


**Figure 2.5.:** Sketch of the idea behind probabilistic inference for planning. The current state denoted by $\mathbf{S_t}$, the desired target or future state denoted by $\mathbf{S_{t+1}}$ and the unknown intermediate actions denoted by **A**.

Let's consider a simple example[37] to illustrate the idea of PAI. A sketch of the example is shown in Figure 2.5, where **S** denotes the state and **A** the actions. These states and actions are modelled by random variables. In this simple planning example there is no reward. The current state at time $t$ $\mathbf{S}_t$ and the desired target (future) state $\mathbf{S}_{t+1}$ are known, indicated by the shaded node, whereas the intermediate actions **A** are unknown. To calculate the probabilities of the intermediate actions, which are required to reach the desired state from the current state, we condition on the known states $\mathbf{S}_t$ and $\mathbf{S}_{t+1}$. To denote a particular value of one of the random variables lower case letters are used, e.g., **a** for an instance from the actions **A**. Its probability is denoted by $P(A = a)$, where $P(a)$ is used as shorthand. Now we can formulate a mathematical description of this simple planning problem example. This involves two steps, first *modelling* of the interaction between the random variables and second *inference* to solve the defined planning problem. The interaction between the random variables is modelled by a joint distribution over all random variables, $p(s_t, a, s_{t+1}) = p(s_t)p(a|s_t)p(s_{t+1}|a)$, where the dependencies are visualized in the graphical model in Figure 2.5. In a graphical model, nodes represent random variables and edges represent dependencies between the random variables. Learning the parameters of the model can be done by different sophisticated methods, e.g., Expectation-Maximization.

Having defined the model, we can now answer the questions we are interested in by performing inference in the model. So the probability of a certain action is given by

$$p(a|s_t, s_{t+1}) = \frac{1}{Z} p(s_t)p(a|s_t)p(s_{t+1}|a), \tag{2.1}$$

where $Z$ denotes the normalization constant that ensures proper probabilities and is given by the marginal distribution over all possible actions, i.e., $Z = \sum_{a' \in A} p(s_t)p(a'|s_t)p(s_{t+1}|a')$. With that the

most probable action $a$ that solves the given planning problem can be inferred. The planning problem can easily be extended to more complex multi-step planning problems by replacing the action $a$ with a sequence of actions $< a_1, a_2, .., a_T >$ or a sequence of states $< \mathbf{s}_1, \mathbf{s}_2, .., \mathbf{s}_T >$ of length $T$.

An advantage of the probabilistic inference for planning perspective is that we can include a multitude of different random variables into the model. These random variables can represent arbitrary information, e.g., features of the states or goals and constraints of the given planning problem. Additionally, random variables can also be used to represent prior knowledge about the problem in the model.

Solving the inference problem in graphical models can be done by different techniques, e.g., message passing, variational inference or sampling methods. We will use a sampling based method in this work to solve the inference problem for robot movement planning.

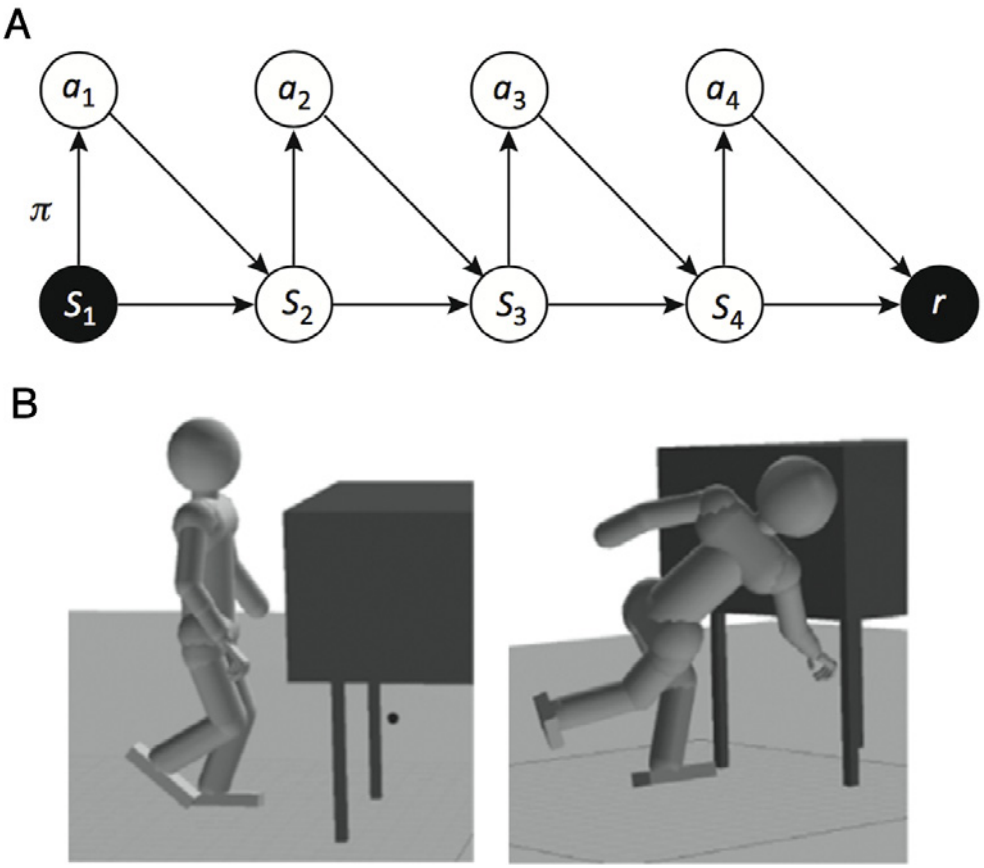


**Figure 2.6.:** A more complex example for PAI. (A) The generative model is represented as a Bayesian Network where nodes represent random variables; states $s$, actions $a$ and reward $r$. Edges represent conditional dependencies. The connections from a state to an action contain the policy $\pi$. A block node is known or clamped to a known value. PAI assumes to receive reward and searches for the most probable policy under that assumption. Thus, PAI finds the most probable sequence of actions, respectively states. (B) An example of PAI used in robotics where it has been applied, for example, to motor control problems. In the example here, PAI uses approximate inference to discover a feasible motion for grasping an occluded object under a desk. Images taken and adapted from [4].

### 2.3.2 Model learning using contrastive divergence

Contrastive divergence (CD) is a model learning algorithm proposed by Geoffrey Hinton [16] that approximates the maximum likelihood gradient by drawing samples from a proposed distribution. In this section a higher level description and derivation is shown based on the notes from Oliver Woodford [44].

Let us assume that the probability of a data point $x$ is modelled by a function $f(x;\Theta)$, with $\Theta$ as a vector of the model parameters. The probability of the data $p(x;\Theta)$ must integrate to 1 over all $x$ to be a valid probability distribution. Thereby the probability of $x$ is given by

$$p(x;\Theta) = \frac{1}{Z(\Theta)} f(x;\Theta) \,, \tag{2.2}$$

where $Z(\Theta)$ is the partition function that normalizes the probabilities and is defined as

$$Z(\Theta) = \int f(x;\Theta) \;\; dx \;. \tag{2.3}$$

The goal of the learning process is to maximize the probability of the training data, i.e., to find the *optimal* parameters $\Theta^*$ that explain the data best. We denote the set of independent and identically distributed (i.i.d.) training examples as $\mathbf{X} = x_{1,..,K}$, with $K$ examples. Thus, the probability of the training data is given by

$$p(\mathbf{X};\Theta) = \prod_{k=1}^{K} \frac{1}{Z(\Theta)} f(x_k;\Theta) \,. \tag{2.4}$$

This formulation is equivalent to the minimization of the negative logarithm of $p(\mathbf{X};\Theta)$, which is commonly denoted as the energy. It is defined as

$$E(\mathbf{X};\Theta) = \log Z(\Theta) - \frac{1}{K} \sum_{k=1}^{K} \log f(x_k;\Theta) \,. \tag{2.5}$$

To minimize the energy, in order to find the optimal parameters $\Theta^*$, we differentiate Equation (2.5) w.r.t. the parameters $\Theta$. The gradient of the energy function is given by

$$\frac{\partial E(\mathbf{X};\Theta)}{\partial \Theta} = \frac{\partial}{\partial \Theta} \left( \log Z(\Theta) \right) - \frac{\partial}{\partial \Theta} \left( \frac{1}{K} \sum_{k=1}^{K} \log f(x_k;\Theta) \right) \,. \tag{2.6}$$

Differentiating the second term on the right hand side of Equation 2.6 gives

$$\begin{aligned} \frac{\partial}{\partial \Theta} \left( \frac{1}{K} \sum_{k=1}^{K} \log f(x_k;\Theta) \right) &= \frac{1}{K} \sum_{k=1}^{K} \frac{\partial \log f(x_k;\Theta)}{\partial \Theta} \\ &= \left\langle \frac{\partial \log f(x;\Theta)}{\partial \Theta} \right\rangle_{\mathrm{x}} \,. \end{aligned} \tag{2.7}$$

The symbol $\langle .. \rangle_{\mathbf{X}}$ denotes the expectation over the set of training data points $\mathbf{X}$. Next, we differentiate the first term on the right hand side of Equation 2.6, i.e., we differentiate the partition function $Z(\Theta)$ w.r.t. the parameters $\Theta$,

$$\begin{aligned}
\frac{\partial}{\partial \Theta}\left(\log Z(\Theta)\right) &= \frac{1}{Z(\Theta)}\frac{\partial Z(\Theta)}{\partial \Theta} \\
&= \frac{1}{Z(\Theta)}\frac{\partial}{\partial \Theta}\int f(x;\Theta)\ dx \\
&= \frac{1}{Z(\Theta)}\int \frac{\partial f(x;\Theta)}{\partial \Theta}\ dx \\
&= \frac{1}{Z(\Theta)}\int f(x;\Theta)\frac{\partial \log f(x;\Theta)}{\partial \Theta}\ dx \\
&= \int \frac{1}{Z(\Theta)} f(x;\Theta)\frac{\partial \log f(x;\Theta)}{\partial \Theta}\ dx \\
&= \int p(x;\Theta)\frac{\partial \log f(x;\Theta)}{\partial \Theta}\ dx \\
&= \left\langle \frac{\partial \log f(x;\Theta)}{\partial \Theta} \right\rangle_{p(x;\Theta)} . \qquad (2.8)
\end{aligned}$$

The expectations in Equation (2.7) and (2.8) are the same despite that in Equation (2.7) it is taken over the training data and in Equation (2.8) it is taken over the model distribution $p(x;\Theta)$. This calculation is in general intractable as it depends on the partition function $Z(\Theta)$. However, the expectation can be approximated by drawing samples from the model distribution $p(x;\Theta)$. This approximation is done by performing Markov Chain Monte Carlo (MCMC) sampling starting from Markov Chains initialized with the training data. As the Markov Chains iterate the sampled data get closer to samples from the proposed distribution. Denoting the state of the Markov Chains after $n$ cycles with $\mathbf{X}^n$, we have all ingredients to formulate the parameter update rule. Therefore we plug the derivatives in Equation (2.7) and (2.8) back into Equation (2.6) which leads to

$$\begin{aligned}
\frac{\partial E(\mathbf{X};\Theta)}{\partial \Theta} &= \left\langle \frac{\partial \log f(x;\Theta)}{\partial \Theta} \right\rangle_{p(x;\Theta)} - \left\langle \frac{\partial \log f(x;\Theta)}{\partial \Theta} \right\rangle_{\mathbf{X}} \\
&= \left\langle \frac{\partial \log f(x;\Theta)}{\partial \Theta} \right\rangle_{\mathbf{X}^\infty} - \left\langle \frac{\partial \log f(x;\Theta)}{\partial \Theta} \right\rangle_{\mathbf{X}^0} . \qquad (2.9)
\end{aligned}$$

To compute accurate approximations of the gradient many MCMC cycles are required. Such computations are expensive and therefore it is often impractical to compute accurate approximations. In [16] it was suggested that only a few MCMC cycles are needed to obtain useful approximations of the expectations in practice. Already after very few cycles the sampled data has moved from the training data towards the model distribution which indicates the direction towards which the model distribution should change in order to model the training data more accurate. Empirically, it was shown in [16] that even a single MCMC cycle has sufficient information to let the learning process converge to the maximum-likelihood solution.

Using this insight we can write the general parameter update equation as

$$\Theta_{t+1} = \Theta_t + \alpha\left(\left\langle \frac{\partial \log f(x;\Theta)}{\partial \Theta} \right\rangle_{\mathbf{X}^0} - \left\langle \frac{\partial \log f(x;\Theta)}{\partial \Theta} \right\rangle_{\mathbf{X}^1}\right) , \qquad (2.10)$$

where $\alpha$ denotes the learning rate. This update rule defines the iterative learning algorithm *contrastive divergence* which is an efficient learning algorithm for complex models based on approximations through sampling.

# 3 Path planning with spiking neural networks

This section describes the basic structure of the proposed spiking neural network models, the different kinds of neuron populations, their relationships and how movement planning and learning of the state transition models is done. The proposed models are not a direct one-to-one implementation of the biological foundation described in Subsection 2.1, but rather try to abstract and adapt the basic mechanisms to the problem of path planning for robots. We introduce three different models here. First, a two-dimensional model for planning in task space. Second, a factorized model for planning in six-dimensional joint space. Third, a hierarchical model for planning in two-dimensional task space with on-the-fly mapping into joint space using a learned factorized inverse kinematic model.

## 3.1 Spiking neural networks

Artificial neural networks are built by a set of interconnected simple processing units, the neurons, and are inspired by the observations of biological learning systems built in that way. Information-processing in such systems is highly parallel and depends on information representations distributed over many neurons.

Spiking neural networks (SNNs) are an advanced generation of artificial neural network models which add a new layer of realism to neural simulations. They are implemented as recurrent neural networks and communication in the model is done by sequences of spikes. It has been shown that SNNs can solve all problems that are solvable by classic neural networks and that SNNs are computationally even more powerful[26].

Basically SNNs work very similar to classic artificial neural networks where neurons are connected through weighted synapses to exchange information. The most important difference is that they incorporate the concept of time. This means the output (spike) of a presynaptic neuron is only considered as input for the postsynaptic neuron if that spike arrives in a certain time window. In the context of learning this time window is called the Spike Timing Dependent Plasticity (STDP) window and determines the time period in which spikes are considered as correlated. Timing is very crucial for the propagation of information in such models. The generation of a spike is determined by the membrane potential, which relates to the electrical charge of a neurons membrane. Incoming spikes increase or decrease this value depending on arriving spikes at excitatory or inhibitory synapses.

Considering the simplest neuron model, i.e., so called (deterministic) integrate-and-fire neuron, the neuron spikes if a certain threshold is reached and the information is propagated to its neighbours. In a probabilistic model, like ours, the membrane potential is used as a probability and a spike is determined based on that probability. The output of a SNN model is a spike train, which is the neural activity of the neurons over time. This spike train can be the activity of some output neurons or of the whole population of neurons, depending on the modelled problem.

SNNs are implemented as recurrent neural networks that can model and learn temporal dynamics. During learning of the model the synaptic weights are adapted. A broad and nice introduction to SNNs can be found in [34].

## 3.2 Neural dynamics as sampling

Sampling can solve any inference task with arbitrary precision, assuming we can sample from the desired distribution and have enough computational resources and time. As we are using the framework of probabilistic inference to solve planning problems (Subsection 2.3.1), we need to take a look at how samples can be drawn from a spiking neural network model (Subsection 3.1) in discrete time.

By simulating the inherent stochastic dynamics of the spiking neural network model, samples can be drawn from the modelled distribution[7], as the dynamics implement forward sampling. The sample is drawn iteratively, i.e., the sample is not drawn at once but rather is built up over time. Values of the sample at time $t$ are determined by the values at time $t-1$. Clamping some neurons, which model variables, to fixed values, observations or assumed rewards can be taken into account and the spiking neural network can produce samples from the conditional distribution. This is used in the planning as probabilistic inference framework. The neurons that model the concrete planning task are clamped to a certain probabilistic spiking pattern and the movement trajectory samples are drawn using this input as condition (see Subsection 3.3 for details of the different neurons, membrane potential and model structure). Operating in discrete time and using a fixed refractory period $\tau$ that decays linearly, the neurons spike in each timestep with a probability determined by their membrane potential. This membrane potential is determined by the synaptic connections of the neurons and the activity of the connected neurons in the previous timestep. All spikes from connected presynaptic neurons get weighted by the corresponding synaptic weight and are integrated to an overall postsynaptic potential (PSP). In a more *realistic* model not only the previous timestep $t-1$ is used as presynaptic input, but a spike has influence on the postsynaptic neuron for a certain time. In the case of a fixed influence time that decays linearly over time, this is called *rectangular window*. If this window size is not larger than the refractory period $\tau$, the PSPs are non-additive, since two spikes never can occur faster than $\tau$.

This procedure generates a sequence of the states of the neurons, i.e., a sequence of spiking activity. The whole spike train generated by this represents the sample drawn from the modelled distribution.

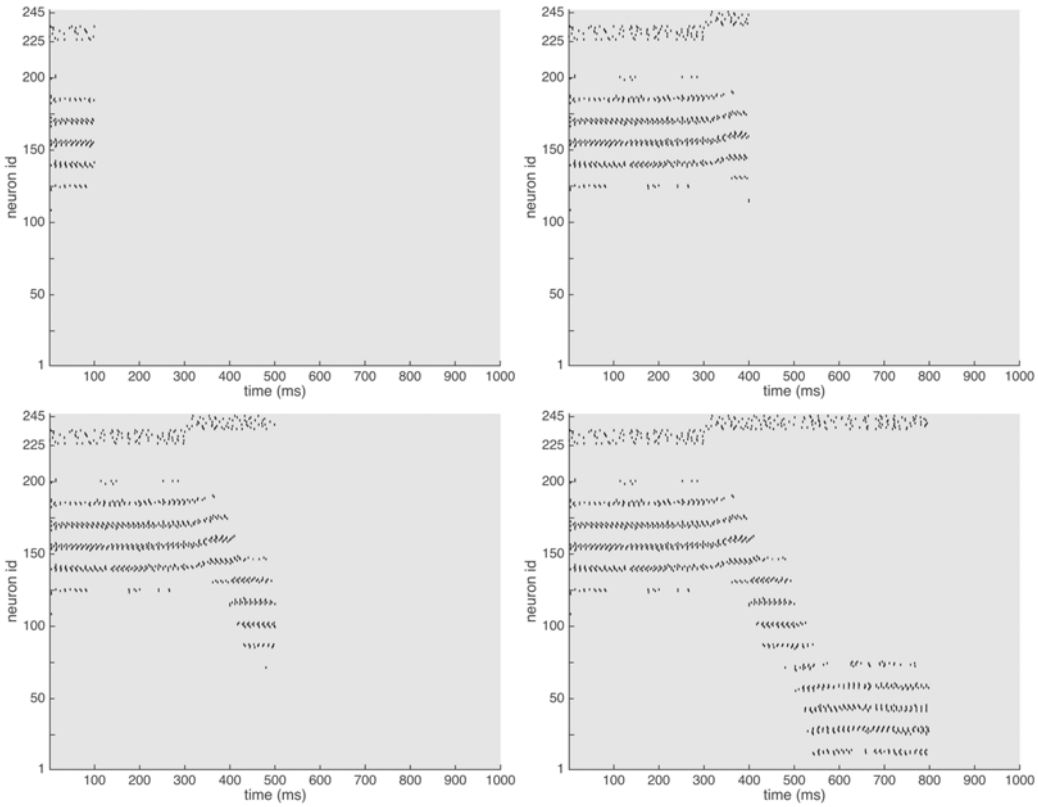


**Figure 3.1.:** Snapshots at different timesteps during sampling from the proposed SNN.

## 3.3 Two-dimensional task space planning model

The spiking neural network models used in this work are realized as recurrent neural networks and basically consist of two different populations of neurons; a layer of *K state* neurons and a layer of *N task* neurons. State neurons encode locations and task neurons encode desired goals or obstacles to avoid. First, the two-dimensional model for task space planning is introduced in detail here. The factorized joint space model and the hierarchical model are modelled by the same mechanisms.

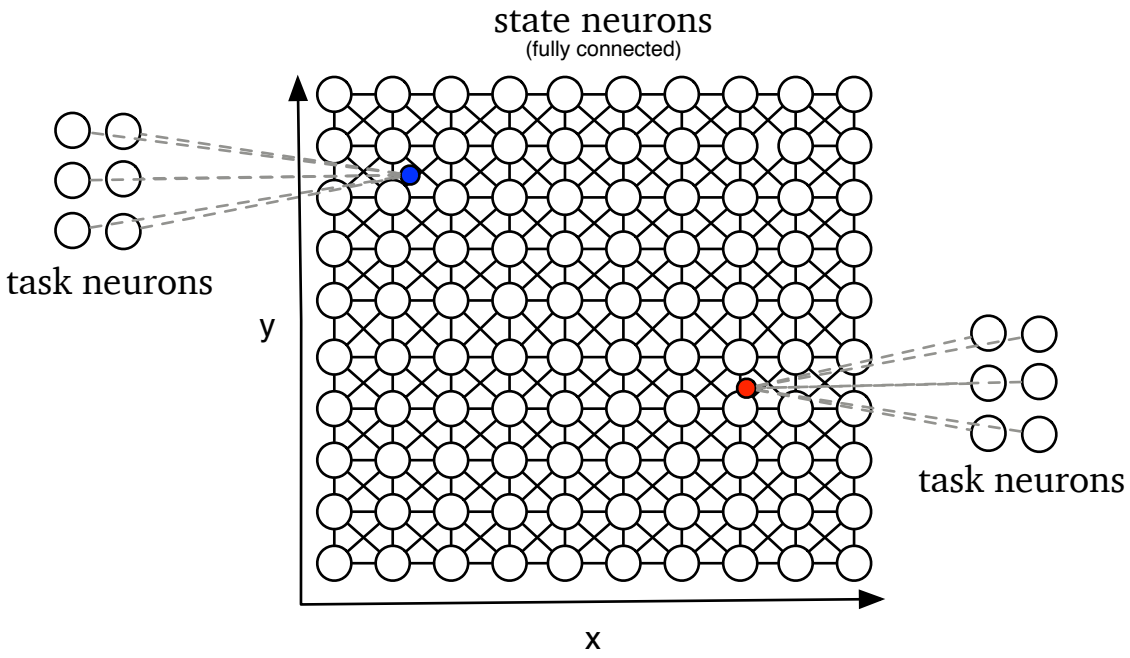


**Figure 3.2.:** Sketch of the two-dimensional task space planning model.

Each constraint or any task related information is modelled by a population of task neurons, e.g., for the simplest planning problem being at position A and going to position B, there are two task neuron populations, each encoding one position.

In Figure 3.2, a simplified sketch of a two-dimensional model with two constraints is shown. There are six task neurons per constraint, which are located around the positions illustrated as the blue and red dots. The dashed lines do not show synaptic connections but should indicate the position of the task neurons. To keep the sketch clear the connections between the task and state neurons are not shown. Only the nearest neighbour connections out of the fully connected state neurons are shown.

All neurons have a position in a defined coordinate system. In our models, the state neurons are equally aligned within the desired state space. The task neurons are uniformly distributed around the constraint (a goal or an obstacle) that they encode with a very small variance. There are no connections in between the task neuron population, but each task neuron is connect to *all* state neurons by the synaptic weights $\theta_{j,k}$, i.e., connecting task neuron $j$ to state neuron $k$. The state neuron population is fully connected, i.e., each state neuron has a connection to every other state neuron by the synaptic weight $w_{i,k}$, i.e., connecting state neuron $i$ to state neuron $k$. We denote the activity of the state neurons by $\mathbf{v}_t = (v_{t,1}, .., v_{t,K})$, where $v_{t,k} = 1$ if state neuron $k$ spiked at time t and $v_{t,k} = 0$ else. The activity of the task neurons is denoted the same way by $\mathbf{y}_t$. The synaptic weights between the state neurons model the state transition model $T(\mathbf{v}_t|\mathbf{v}_{t-1})$ that encodes how likely it is to end up in state $\mathbf{v}_t$ given state $\mathbf{v}_{t-1}$. The state neurons can be seen as an abstract and simplified version of the place cells described in Subsection 2.1.2. By this they model a cognitive map of the environment.

The described spiking network model defines a multi-modal distribution over state sequences, i.e., over movement trajectories. State neurons are modelled by stochastic neurons which build up a membrane potential based on the weighted neural activity of the state neurons in the previous timestep and the neural activity of the task neurons. Task neurons have no afferent connections and spike with a certain, fixed probability. All neurons have a defined refractory period $\tau$ in which they cannot spike again after the last spike. Assuming linear dendritic dynamics the membrane potential of the state neurons is given by

$$u_{t,k} = \sum_{i=1}^{K} w_{i,k} \tilde{v}_i(t) + \sum_{j=1}^{N} \theta_{j,k} \tilde{y}_j(t) \,, \tag{3.1}$$

where $u_{t,k}$ denotes the membrane potential for neuron $k$ at time $t$ and Equation (3.1) defines a simple stochastic version of the spike response model[13]. The first sum is over the $K$ state neurons and models the influence of the state transition model, $w_{i,k}$ denotes the synaptic weight between state neuron $i$ and $k$, and $\tilde{v}_i(t)$ denotes the effects of PSPs from neuron $i$. The second sum is over the $N$ task neurons and models the influence of task related input, where $\theta_{j,k}$ encodes the synaptic weight between task neuron $j$ and state neuron $k$, and $\tilde{y}_j(t)$ denotes the effects of PSPs from neuron $j$.

Using the membrane potential, a probability to spike for the state neurons can be defined by

$$\rho_{t,k} = p(v_{t,k} = 1) = f(u_{t,k}) \,, \tag{3.2}$$

where $f(u_{t,k})$ denotes the activation function, that is required to be differentiable. Now the probability for generating a state sequence $\mathbf{v}_{1:T}$ of length $T$ and starting from a given initial state $\mathbf{v}_0$ is defined by the model distribution

$$q(\mathbf{v}_{1:T}|\theta) = p(\mathbf{v}_0) \prod_{t=1}^{T} \prod_{k=1}^{K} \rho_{t,k}^{v_{t,k}} (1 - \rho_{t,k})^{1 - v_{t,k}} \tag{3.3}$$

$$= p(\mathbf{v}_0) \prod_{t=1}^{T} T(\mathbf{v}_t|\mathbf{v}_{t-1}) \phi_t(\mathbf{v}_t; \theta) \,. \tag{3.4}$$

In this work we focus on model learning, so we are mainly interested in learning $T(\mathbf{v}_t|\mathbf{v}_{t-1})$ encoded in the synaptic weights $w_{i,k}$. The synaptic weights $\theta_{j,k}$ between task and state neurons are handcrafted, e.g., based on the Euclidean distance between the desired goal and the preferred location of the state neuron. This is the location that is encoded by the state neuron, i.e., the coordinates of the state neuron, and refers to its place field. The neural activity $\mathbf{y}_t$ of the task neurons is determined by a probability depending on the kind of constraints the task neurons population models, i.e., task neurons encoding the initial state only spike at the beginning of the planning process, target encoding task neurons are active after that initial phase and obstacle task neurons are active the whole time. This task related input can be learned using reinforcement learning techniques that maximize the probability of generating rewarding (successful) trajectories. This reinforcement learning was studied in recent prior work [38]. As the task neurons model task related constraints they are only activated for solving a particular task. The state transition model is task independent and is used in the unconstrained stochastic process for planning,

$$q(\mathbf{v}_{1:T}) = p(\mathbf{v}_0) \prod_{t=1}^{T} T(\mathbf{v}_t|\mathbf{v}_{t-1}) \,, \tag{3.5}$$

which defines a freely moving agent. The state neurons membrane potential simplifies to

$$u_{t,k} = \sum_{i=1}^{K} w_{i,k} \tilde{v}_i(t) \,. \tag{3.6}$$

Sampling from Equation (3.5) can be implemented by a recurrent spiking neural network (e.g., ideas from [5][17][20]) and generates random walk trajectories of length $T$ in the modelled state space. Injected task related input modulates these random walks towards goal-directed movement plans.

## 3.4 Decoding continuous states with spike patterns

The neurons encode binary random variables and the solution to a given planning problem found by the proposed model is the spike train of the state neurons. This spike train is a sequence of binary activity vectors $\mathbf{v}_t$. To execute the generated movement trajectory on a robot, the spike train needs to be decoded into a sequence of spatial locations. This decoding of the real world state $\mathbf{x}_t$ (e.g., the position in task or joint space) from the neural activity of the state neurons is done by a simple decoding scheme[12][25],

$$\mathbf{x}_t = \frac{1}{|\hat{\mathbf{v}}_t|} \sum_{k=1}^{K} \hat{v}_{t,k} \mathbf{p}_k \qquad \text{and} \qquad |\hat{\mathbf{v}}_t| = \sum_{k=1}^{K} \hat{v}_{t,k} \,, \tag{3.7}$$

with $\mathbf{p}_k$ denoting the preferred position of state neuron $k$ and $\hat{v}_{t,k}$ is the filtered neural activity of state neuron $k$ at time $t$. The filtered neural activity is determined by applying a Gaussian window filter to the binary spiking activity of the state neurons. We used a window size of $100ms$. With this decoding scheme, continuous random variables can be modelled by binary activity patterns.

## 3.5 Factorized population codes for high-dimensional models

The proposed spiking neural network model suffers from the curse of dimensionality and more than three dimensions cannot be efficiently modelled. To use spiking neural networks for higher dimensional planning problems, we propose a factorized approximation using $N$ independent one-dimensional models.

For each dimension a SNN model is learned and samples are drawn from these models, which are combined afterwards. The model structure of each independent model is the same as described in detail for the two-dimensional task space planning model (Subsection 3.3), i.e., the state and task neurons are build the same way. By this we can learn models dealing with seven-dimensional planning problems in task space (Cartesian $x, y, z$ and four-dimensional quaternion for endeffector orientation) or $n$-dimensional planning problem in joint space (the KUKA arm has seven dimensions in joint space as well, although only six joints are used in this work). These models can generate smooth movement trajectories together for high dimensional state spaces, but as they are independent and cannot exchange information during sampling, they cannot deal with obstacle occurring in the environment. Dealing with obstacles in the independent models is an open question so far. Using this factorized approximation approach makes spiking neural network models feasible for planning in higher dimensional spaces. In general, each model consists of relatively few neurons, such that sampling from them can be done very fast. Sampling and generating a solution movement trajectory in each model is independent and therefore can be done in parallel. This speeds up the whole planning progress even more.

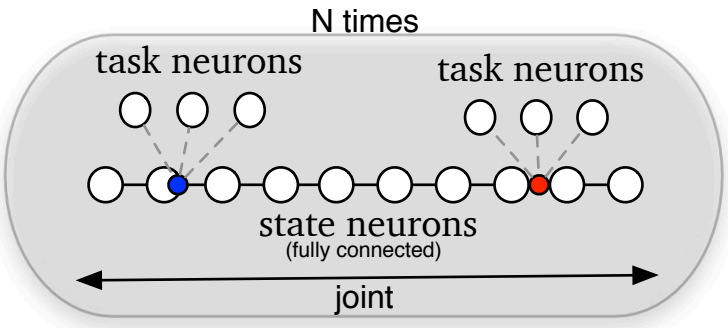


**Figure 3.3.:** Sketch of the factorized model consisting of $N$ independent one-dimensional models.

## 3.6 Hierarchical models for high-dimensional planning problems

One possible solution to overcome the missing ability to deal with obstacles of the independent models, is to join a lower dimensional model for planning in task space that can deal with obstacles with $N$ independent models that map to joint space. This mapping is known as the *inverse kinematics*.

Thus, inverse kinematics are a mapping from task space to joint space in the form of $\mathbf{q} = f(\mathbf{x})$, e.g., a mapping from positions in task space $\mathbf{x}$ (like Cartesian $x, y$ coordinates of the endeffector) to the joint configuration $\mathbf{q}$ that is required to get the endeffector into that position. In order to learn the inverse kinematics, we recorded additionally to the seven-dimensional task space data the seven-dimensional joint space trajectories of the KUKA lightweight arm during kinesthetic teaching (Subsection 4.1).

The hierarchical model is built by combining the two-dimensional task space planning model (Subsection 3.3) with the factorized inverse kinematic models. This model can solve a given planning task in task space and transforms the movement plan into joint angle trajectories at the same time.

The state neurons of the inverse kinematic models do not have lateral connections. In contrast, the state neurons of the task space planning model are fully connected, and project to all state neurons of the inverse kinematic models through symmetric weights. This connectivity is sketched in Figure 3.4.

The symmetric connections have an interesting effect. Feedback can be projected from the task space to the joint space and vice versa. It has been shown that such links facilitate smooth trajectories[43].

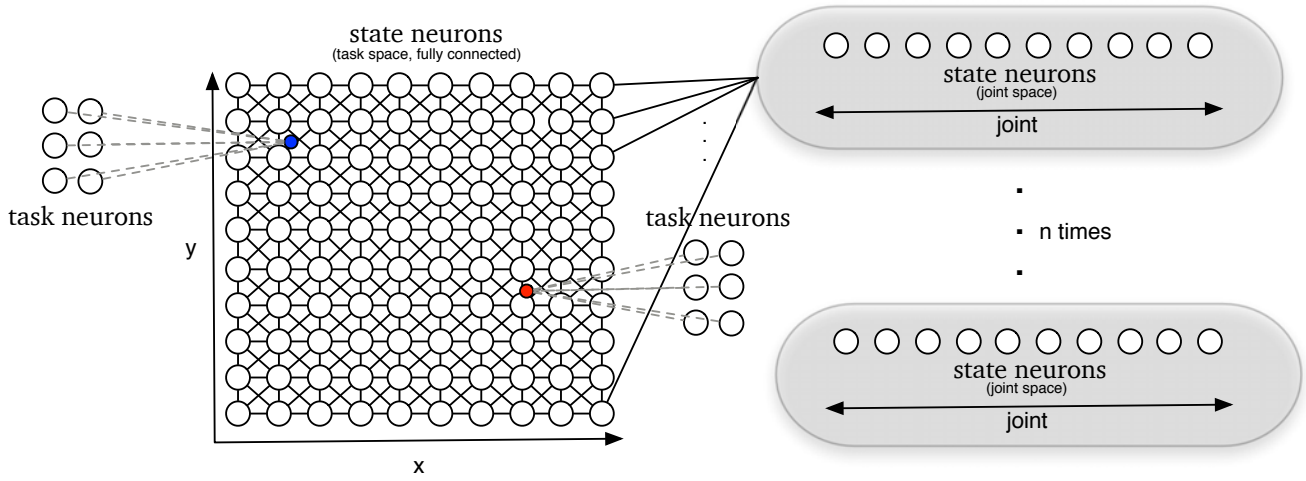


**Figure 3.4.:** Sketch of the hierarchical model with the two-dimensional task space planning model on the left and the independent joint space models on the right.

## 3.7 Task adaption through task neurons

In typical robot planning problems, the planning algorithm has to consider a number of constraints that may also change dynamically, such that the algorithm has to adapt to new tasks online. The architecture of the proposed spiking neural network implements a simple task adaption mechanism without learning of the transition model. Task constraints are modelled by task neurons, which are independent from the transition model. To define a new task, simply a set of task neurons with proper synaptic weights and activity patterns have to be added. Positions that should be reached are modelled by task neurons with an excitatory connection to the state neurons. The activity pattern depends on when a position should be reached. Obstacles on the other hand are modelled by task neurons with inhibitory connections. State neurons in the area of the obstacle are inhibited and no sampled movement will cross this area.

## 3.8 Model learning with spiking networks

### Learning state transition models

The state transition model is encoded in the synaptic weights $w_{k,i}$ between the state neurons of the proposed SNN model. We want to learn this transition model with training data recorded from the real robot through kinesthetic teaching, in order to get a realistic transition model according to the covered area of the robot. For this purpose we use *contrastive divergence* as described in Subsection 2.3.2 which is a very efficient learning algorithm for large and complex models based on approximation through sampling. This section shows how contrastive divergence is used to learn the transition model from real robot training data.

Therefore, we derive a spike dependent version of contrastive divergence to learn the synaptic weights, similar to results in [31]. The transition model of our spiking neural network model is given by

$$T(\mathbf{v}_t|\mathbf{v}_{t-1}) = \exp\left(\sum_{k=1}^{K} w_{k,i}\, v_{t-1,k}\, v_{t,i}\right) , \tag{3.8}$$

where we consider w.l.o.g. a resetting rectangular PSP kernel of one time step length ($v_{t-1,k}$). Using instead a PSP kernel of $\tau$ time step length ($\tilde{v}_k(t)$) follows the same derivation.

We start by differentiating the transition model ($\log T(\mathbf{v}_t|\mathbf{v}_{t-1})$) w.r.t $w_{k,i}$. This results in

$$\begin{aligned}\frac{\partial \log T(\mathbf{v}_t|\mathbf{v}_{t-1})}{\partial w_{k,i}} &= \frac{\partial}{\partial w_{k,i}} \log \exp\left(\sum_{k=1}^{K} w_{k,i}\, v_{t-1,k}\, v_{t,i}\right) \\ &= \sum_{k=1}^{K} \frac{\partial}{\partial w_{k,i}} w_{k,i}\, v_{t-1,k}\, v_{t,i} \\ &= v_{t-1,k}\, v_{t,i} \ . \end{aligned} \tag{3.9}$$

Now inserting (3.9) into the general contrastive divergence update rule in Equation (2.10) leads to

$$\Delta w_{k,i} = \alpha \left( \left\langle v_{t-1,k}\, v_{t,i} \right\rangle_{\mathbf{X}^0} - \left\langle v_{t-1,k}\, v_{t,i} \right\rangle_{\mathbf{X}^1} \right) \ . \tag{3.10}$$

Learning is done by replaying the human demonstrations such that for each timestep there is only one training data for each synaptic weight in form of activity pairs ($\tilde{v}_{t-1,k}, \tilde{v}_{t,i}$). This training spike pattern, denoted by $\tilde{v}$, is compared with the sample drawn from the current transition model (denoted by $v$). With the presynaptic training data as input this leads to the final synaptic update rule

$$\Delta w_{k,i} = \alpha \left( \tilde{v}_{t-1,k}\, \tilde{v}_{t,i} - \tilde{v}_{t-1,k}\, v_{t,i} \right) \ . \tag{3.11}$$

This parameter update rule can be seen as a Hebbian like learning rule. State neurons that spike successively will increase their synaptic weights and state neurons that do not spike successively will decrease their synaptic weights. By this the learned model will prevent jumps in the state space and will only allow small movements per timestep. Note that only closely neighboured state neurons receive excitatory lateral inputs and distant neurons are inhibited.

### Learning inverse kinematic models

To learn the inverse kinematic models in the hierarchical model from the recorded human demonstrations, we basically use the same learning procedure as for the transition model described before. The difference is that synaptic weights between state neurons in task space and the state neurons in joint space are learned. Thus, we want to learn a mapping from task space to joint space. This transition model between task and joint space is given by

$$T(\mathbf{v}_t^{joint}|\mathbf{v}_{t-1}^{task}) = \exp\left(\sum_{k=1}^{K} w_{k,i}\, v_{t-1,k}^{task}\, v_{t,i}^{joint}\right) . \tag{3.12}$$

We derive the synaptic weight update rule in the same way as for the state transition model shown before. This results in the following synaptic weight update rule,

$$\Delta w_{k,i} = \alpha\left(\tilde{v}_{t-1,k}^{task}\,\tilde{v}_{t,i}^{joint} - \tilde{v}_{t-1,k}^{task}\, v_{t,i}^{joint}\right) . \tag{3.13}$$

Using this learning rule we can learn the mapping from task space to joint space by replaying the spike patterns from both spaces to the model. Moreover, the independent inverse kinematic models can be learned simultaneously.

# 4 Robot experiments

For all of our experiments we used a KUKA lightweight arm. For the two-dimensional task space planning model, the control problem is absorbed by a built-in Cartesian tracking controller. The trajectories are executed using inverse kinematics to obtain a referent joint trajectory and inverse dynamics to execute it. The trajectories generated by the factorized model for planning in the six-dimensional joint space are directly executed using inverse dynamics control. The output of our hierarchical model are also joint trajectories and get executed the same way. For evaluating our models, we defined two different tasks with different configurations. The first task is a simple target reaching task, where an initial and desired target position are given, in task or joint space respectively. In the second task the difficulty is increased by adding obstacles.

## 4.1 Gathering training data through kinesthetic teaching

As stated in Subsection 3.8, we want to use real robot data to learn a realistic transition model of the robot and its task space. To collect proper training data we used *kinesthetic teaching*. Thus, we took the robot by the hand and showed it the task it should learn (e.g., imagine a tennis or golf coach). In our particular setup, we guided the robot through the whole task space.

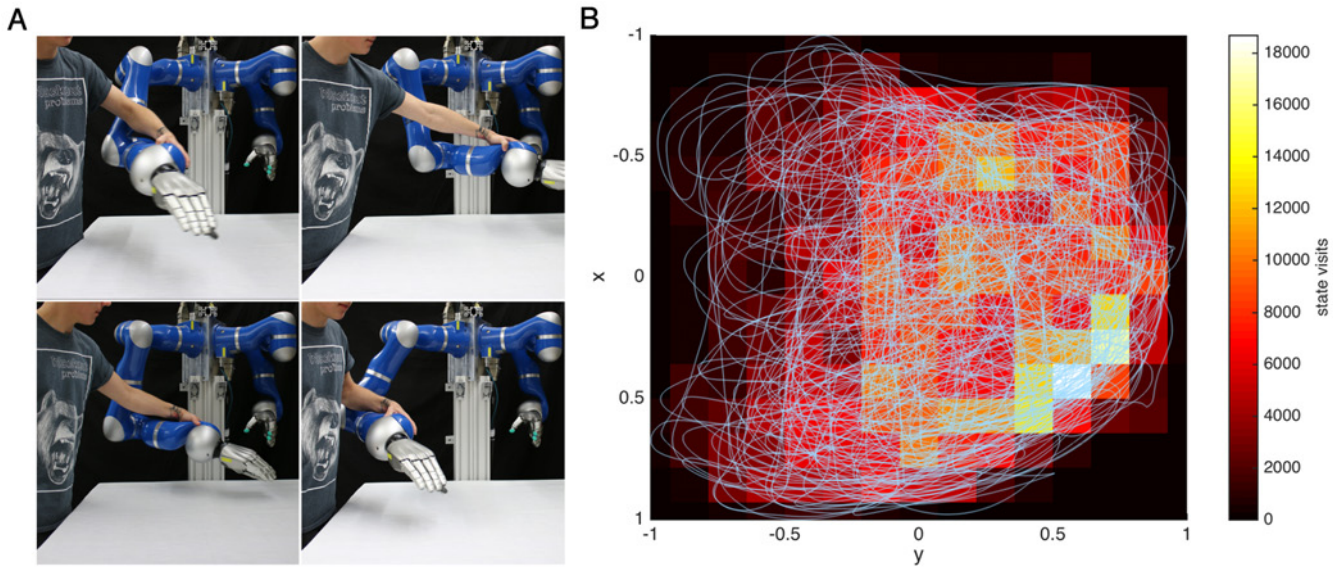


**Figure 4.1.:** Snapshots of the kinesthetic teaching (A) and the recorded task space training data plotted over a heatmap of the visited states (B).

In total, we recorded about 15 minutes of arbitrary movements, sampled at $1ms$ which resulted in about 900.000 state transitions $(\tilde{\mathbf{v}}_{t-1}, \tilde{\mathbf{v}}_t)$ to learn from. Figure 4.1 shows snapshots of the training data recording and recorded movements plotted over a heatmap of the visited states in the task space. A visited state is determined by the nearest state neuron according to Euclidean distance. The covered task space of the robot in $x$ and $y$ coordinates spans a grid of $70 \times 70cm$.

Additionally to the $x$ and $y$ coordinates, the third Cartesian dimension $z$ and the four-dimensional quaternion based orientation of the end-effector were recorded. This data was used to solve the higher dimensional planning problems. In order to learn a model for planning in joint space and the inverse kinematic models, the trajectories of all seven joints of the KUKA arm were recorded as well. The recorded training data had 14 dimensions consisting of task and joint space trajectories.

The recorded demonstrations of the six joints are shown in Figure 4.2. They are plotted next to histograms representing the frequency of the visited states. Only six joints are shown here due to we did not use the seventh joint (the wrist joint) in the demonstrations and kept it fixed. The seven trajectories plotted represent seven separate trials.

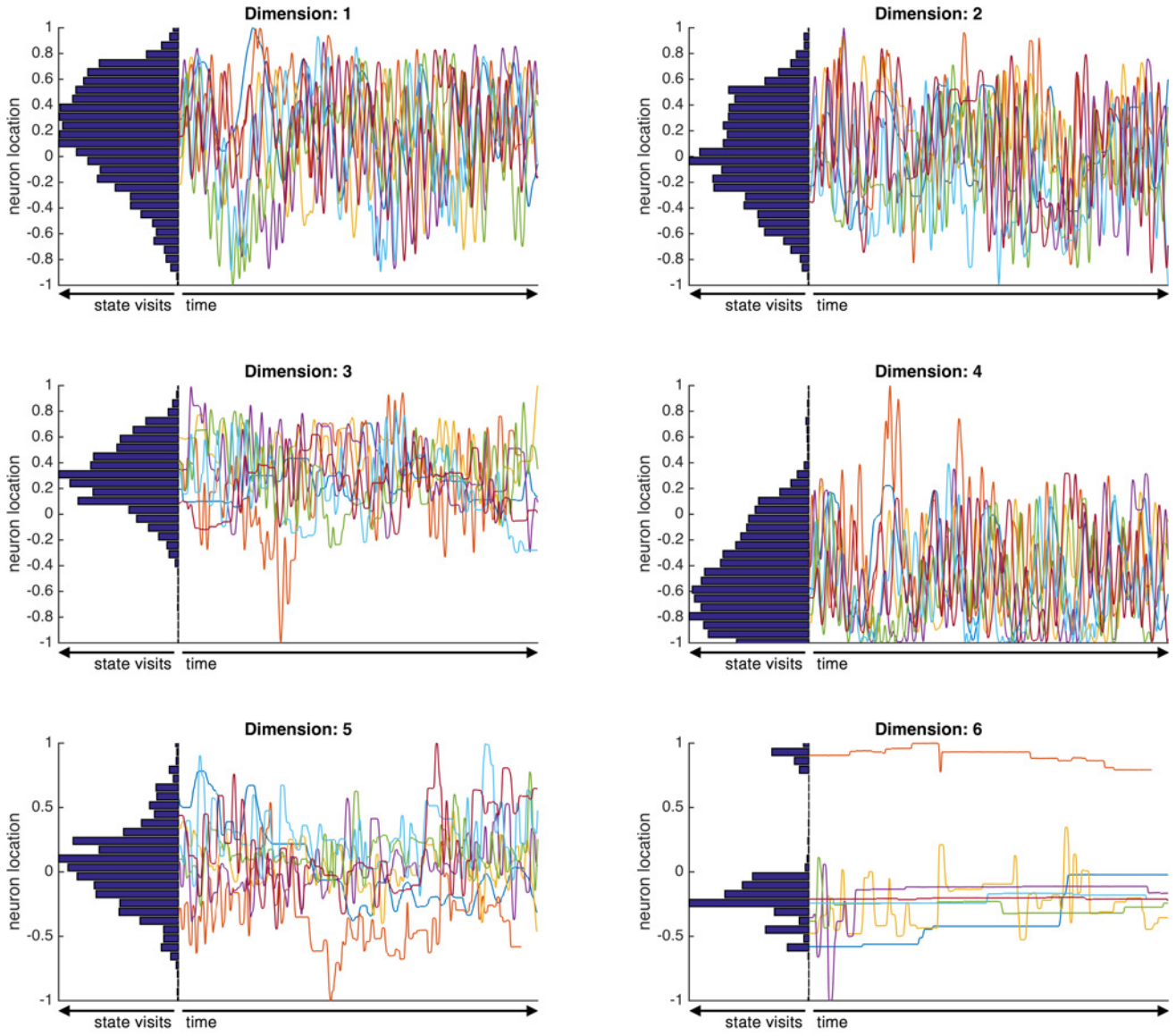


**Figure 4.2.:** The recorded joint space training data. The trajectories of the used six joints are plotted next to a histogram representing the frequency of the visited states.

## 4.2 Preparing the training data

In order to use the recorded training data, it has to be transformed from continuous movement trajectories into binary spike patterns of the state neurons, i.e., we need to compute the vectors $\tilde{\mathbf{v}}_{t-1}, \tilde{\mathbf{v}}_t$.

For the two-dimensional task space planning model, we learned a state transition model with 15 state neurons per dimension. For that purpose the training data of the endeffector movement in task space is transformed using 15 Gaussian basis functions per dimension aligned uniformly between $[-1, 1]$. This results in a grid of 225 state neurons covering the task space in the $x$ and $y$ dimension. Continuous activity values of the state neurons are then determined by the distance of the state neuron to the current movement and its Gaussian basis function. These continuous activity values are transformed into binary spike patterns by using them as input to an inhomogeneous Poisson process. All recorded demonstrations were also mapped into $[-1, 1]$ before they get transformed into binary spike patterns.

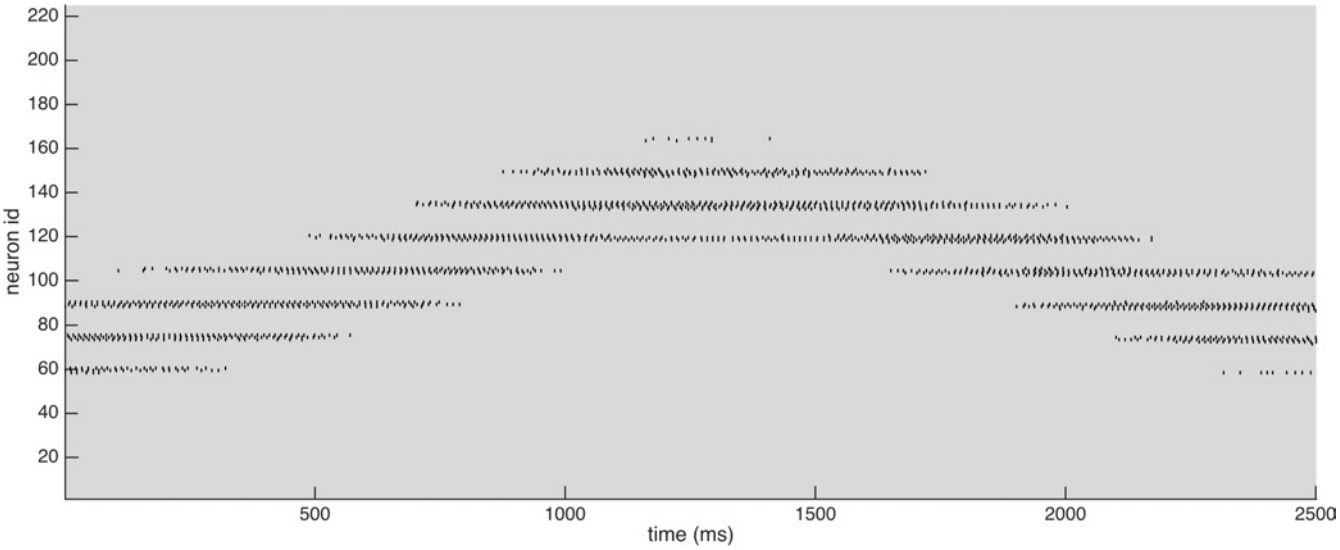


**Figure 4.3.:** 2.5$s$ snippet of transformed training data in two-dimensional task space ($xy$) using $15 \times 15$ state neurons.

In our experiments we used the Gaussian bandwidth parameter $\sigma^2 = 0.03$, an additional Poisson rate offset $\lambda_{off} = -1.5$ to fine tune the spike pattern and a refractory period of $10ms$. Figure 4.3 shows a 2.5$s$ snippet of the transformed training data used for learning of the state transition model of the two-dimensional task space planning model.

For learning the factorized joint space models and the hierarchical model, the recorded trajectories of each joint are transformed into spike patterns the same way as the cartesian trajectories, with the difference that 30 Gaussian basis functions were used, i.e., 30 state neurons per dimension. Therefore we get six spike trains describing the one-dimensional movement of each joint. With those spike trains we learned the factorized joint space model. Having the spike trains of the recorded demonstrations of the task space endeffector movements in $x, y$ and the corresponding spike patterns of the movements of the six joints, we can learn the inverse kinematics using those spike trains simultaneously. Each of the used six joints is modelled by a SNN that maps the spike pattern of the movement trajectory in task space (spikes of the 225 state neurons in task space) into a spike pattern of that joint (spikes of the 30 state neurons in each joint space).

## 4.3 Learning the transition models

With all training data recorded and transformed as described in Subsections 4.1 and 4.2, the state transition model $T(\mathbf{v}_t|\mathbf{v}_{t-1})$ is learned as described in Subsection 3.8 using Equation (3.11). All synaptic weights $w_{k,i}$ were initialized with strong uniformly distributed inhibition, $w_{k,i} \sim \mathcal{U}(-0.99, -0.95)$. Learning is done by replaying the transformed demonstrations in task space (see Figure 4.1) to the model while updating the synaptic weights $w_{k,i}$, i.e., each timestep is successively used as postsynaptic activity $\tilde{v}_{t,i}$ with the previous timesteps $\tilde{v}_{t-1,k}$, determined by the refractory period of $10ms$, as presynaptic activity. The training data, $(\tilde{v}_{t-1,k}, \tilde{v}_{t,i})$, is compared to the state transition $(\tilde{v}_{t-1,k}, v_{t,i})$ sampled from the current model using the presynaptic activity $\tilde{v}_{t-1,k}$ as input. The synaptic weight $w_{k,i}$ of the two-dimensional task space model is updated according to Equation (3.11). A learning rate of $\alpha = 0.05$ was used.

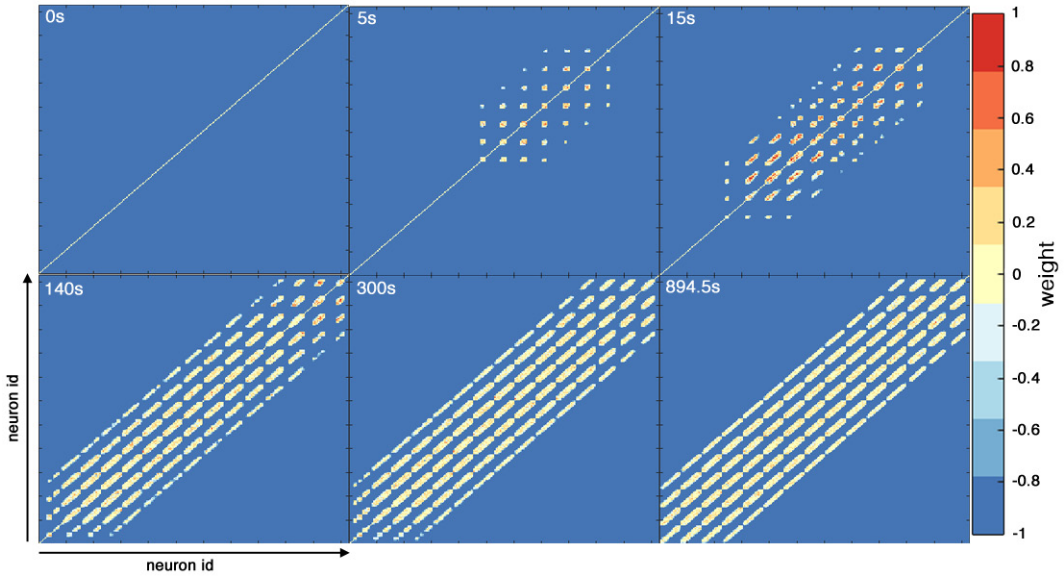


**Figure 4.4.:** Snapshots of the two-dimensional task space state transition model (consisting of 225 state neurons) learning progress over the replay time of the demonstrations. The final model is shown in the lower right area (894.5s replay time).

Figure 4.4 shows snapshots of the learning progress over the replay time of the demonstrations. Starting with randomly distributed inhibitory weights, the learning progress follows the demonstrations by updating synaptic weights between state neurons according to the position of the demonstration. After replaying of all demonstrations, the final learned state transition model is shown in the last panel in Figure 4.4. The training data is only replayed once for learning, because the arbitrary movements of the demonstrations cover all states multiple times, such that there is enough training data for all synaptic weights in a single replay. Replaying the training data multiple times did not improve the state transition model and the synaptic weights did not change any more. To improve the state transition model, more and diverse demonstrations are required that include new unseen state transitions. As one can see in Figure 4.1, not all areas of the state space are covered equally well and thus the learned state transition model is not equally accurate in all areas. Learning of about 15 minutes of demonstrations took less than 15 minutes on a standard computer[1] in MATLAB. Analysing the learned model reveals that only *close* neighbouring state neurons have an excitatory connection and the closer they are the stronger the synaptic weights are. More distant neurons have inhibitory synaptic weights and with this model structure only *small* steps in each timestep are allowed. This prevents to generate movements that jump in the space.

[1] 2.7 GHz Intel Core i7, 8 GB 1333 MHz DDR3

Learning of the six independent models in joint space, works the same way as the learning of the two-dimensional task space planning model. The training data for each model were the movement trajectories in each dimension (see Figure 4.2). The synaptic weights were initialized with strong random inhibition again and were updated during replaying of the training data according to Equation (3.11). After learning, each model encodes the state transition model of the corresponding dimension in the joint angle space.

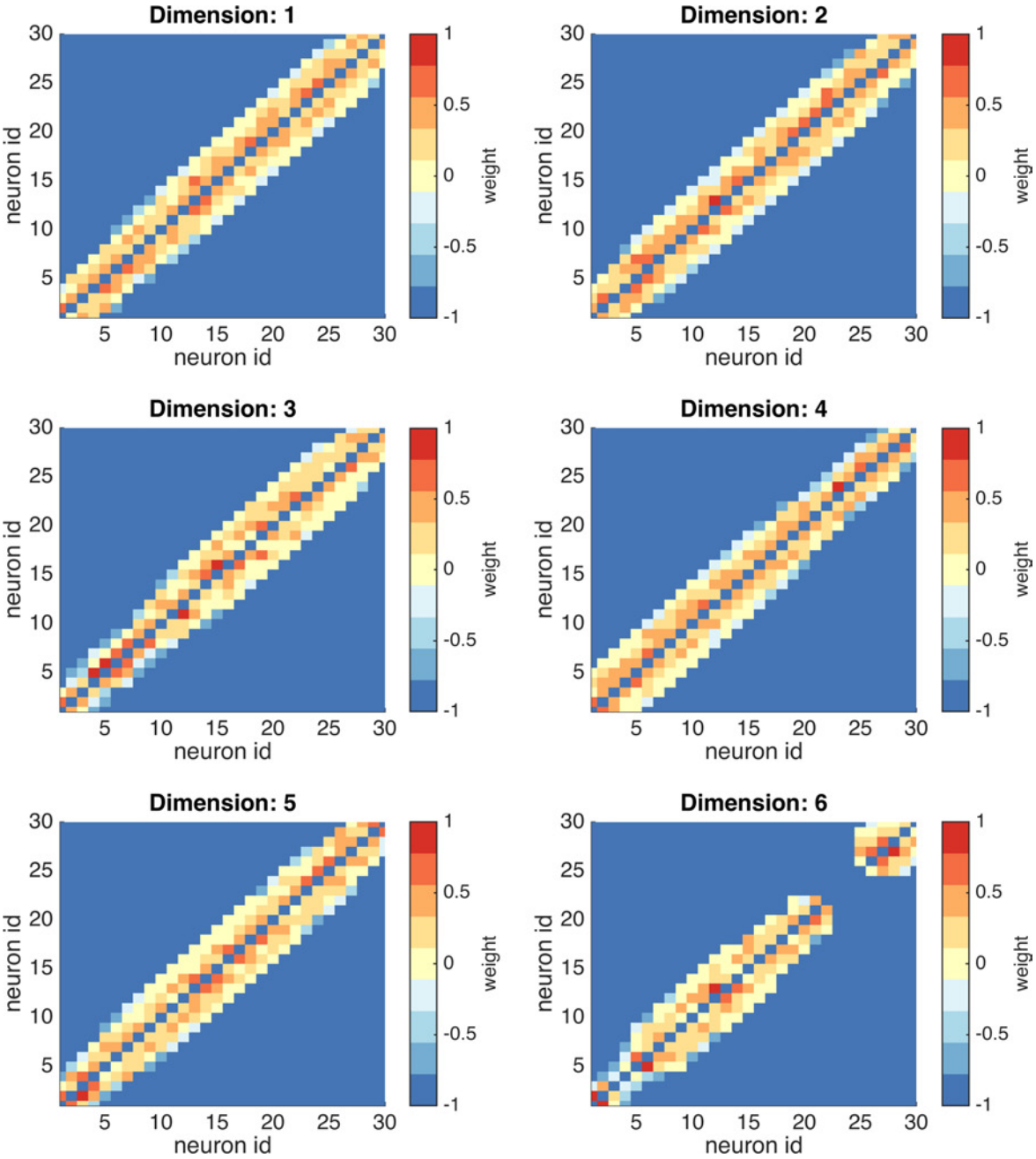


**Figure 4.5.:** The synaptic weights of the six independent models encoding the state transitions in the six joint spaces after learning.

In Figure 4.5 the state transition models of each dimension are shown after learning converged. Analysing the structure of the learned transition models reveals the same structure as in the two-dimensional task space model. Close neighbours have an excitatory synaptic connection that decreases with increased distance until the synaptic weights become an inhibitory connection for distant neurons. Again, this structure prevents to generate trajectories that jump, but now in the joint space.

The hierarchical model consists of two different types of models. First, a two-dimensional task space planning model using the full population code and second, six independent models for the joint space mapping. For task space planning, the two-dimensional model that was learned before was used. Only the six independent models have to be learned additionally. The six models of the inverse kinematics were learned by replaying the transformed demonstrations in task space and the corresponding transformed movements in the joint space. This was done simultaneously and using Equation (3.13) to update the weights. All synaptic weights were initialized with strong random inhibition. After replaying the 15 minutes of training data, the weights of the six learned SNNs modelling the inverse kinematics converged and are shown in Figure 4.6.

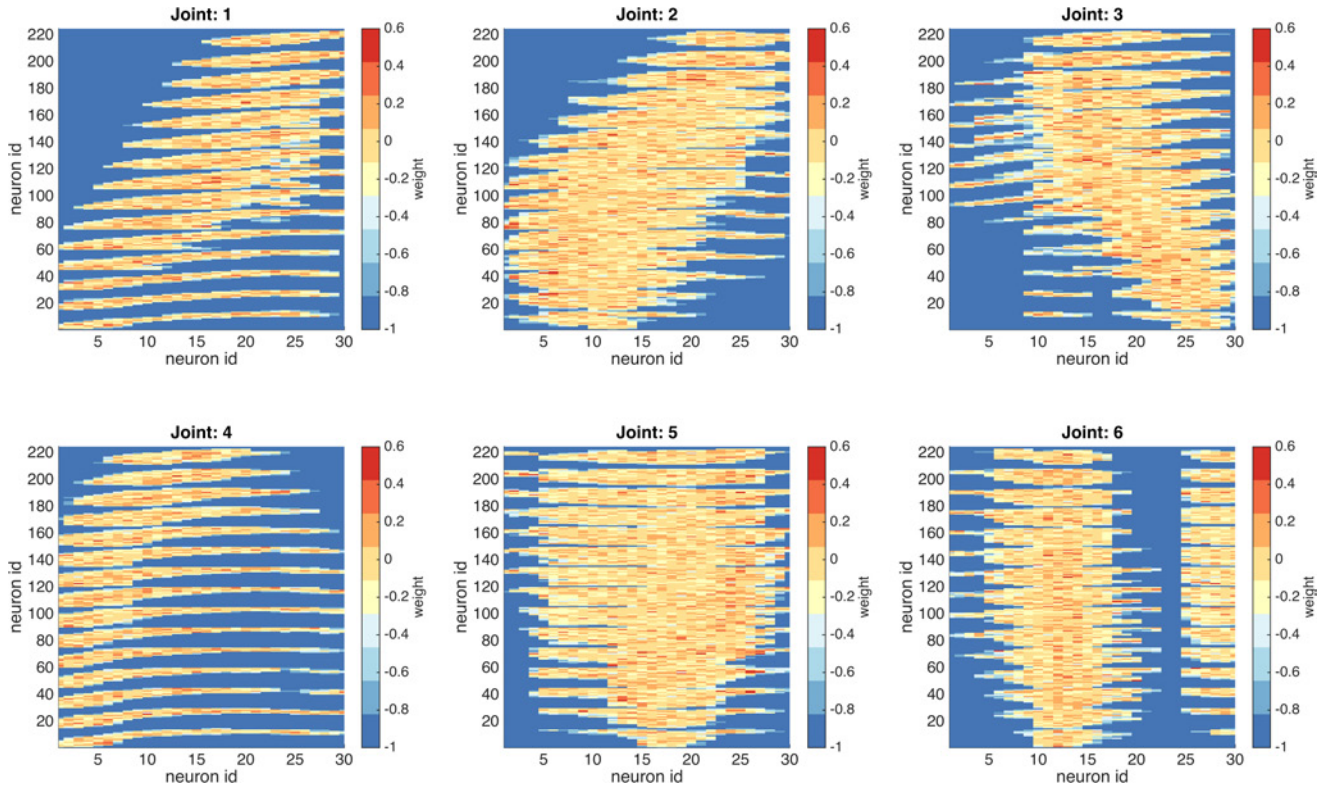


**Figure 4.6.:** The synaptic weights of the SNNs modelling the inverse kinematics of the used six joints of the KUKA lightweight arm after learning.

The hierarchical model is now built of the learned two-dimensional task space planning model and the learned six independent SNNs modelling the inverse kinematics. Each of those six SNNs encodes a mapping from task space to the corresponding joint space. Because the two-dimensional task space planning model consists of 225 state neurons and each one-dimensional movement in joint space is encoded by 30 state neurons, each of the inverse kinematic models is of size $225 \times 30$. During sampling of a movement the output of the two-dimensional task space planning model, i.e., the spiking activity of the 225 task space state neurons, is used as input to determine the spiking activity of the 30 state neurons encoding the joint spaces. The spikes of the joint space state neurons are used as additional input to the task space model to provide feedback of the current joint states to the task space planning model.

## 4.4 Generating smooth movement trajectories

In order to get a smooth movement trajectory that is executable on the robot several steps were performed. First, samples are drawn by simulating the spiking neural network dynamics which implement forward sampling (see Subsection 3.2). This means, the neural activity of the state neurons in each timestep is determined by the neural activity of the state neurons in the previous timestep and task related input from the task neurons. Starting at time $t = 1$, each timestep $t$ of the sample is successively drawn until the predefined sample length with $t = T$ is reached. Next, the binary spike pattern $\mathbf{v}_t$ of the sample is decoded into a sequence of spatial locations $\mathbf{x}_t$ using a Gaussian window filter and the simple decoding scheme described in Subsection 3.4.

Due to the probabilistic model and the built in exploration, not all samples are equally good. To reject the *bad* samples three criteria were used. First, the filtered Euclidean distance using a Gaussian window filter over a small time window to the initial position and second, to the target position. Third, the maximum jerk[1] between timesteps with a small step size was used. All samples that do not reach the initial and target position were rejected, i.e., a sample gets rejected if its average distance is greater than a given precision threshold ($5 - 7\%$ of the modelled space, e.g., $5cm$ in the task space). Additionally, samples that are *too fast* get rejected based on their maximum jerk, i.e., if the maximum jerk is greater than a given threshold ($0.12 - 0.15ms^{-3}$). The jerk is calculated in timesteps of $20ms$.

The set of accepted samples was used to build the final movement trajectory. In our experiments, we used the average position of all accepted samples in each timestep to combine the samples, i.e., using the mean trajectory of the accepted samples. By this the resulting movement trajectory is much smoother than the trajectories from the single samples.

For the obstacle avoidance task, we used only a single sample in the real robot. Simple averaging would not work in that task. There are mostly multiple solutions in an obstacle avoidance task and averaging over different solutions will result in (possibly) infeasible trajectories. To solve this problem, there are different possibilities, e.g., perform clustering on the samples to separate the different solutions and average the trajectories in each cluster or simply use a single sample that is more wiggly.

Before we execute the generated trajectories on the real robot, we shortened the movement trajectory by extracting the part of the movement from where it is closest to the initial position to where it is closest to the target position.

Finally, we had to map the trajectory back from the space of our model $[-1, 1]$ to the space of the robot. Additionally, the movement speed was decreased to execute the sampled trajectories on the real robot. For that we stretched the trajectories from approx. $1s$ to $5s$ using linear interpolation.

[1] rate of change of acceleration

## 4.5 Target reaching task

To evaluate the learned models from Subsection 4.3, we used them to plan movement trajectories for real robot planning problems. Therefore we define different planning tasks involving different task settings and difficulties. We start with a simple target reaching task without any obstacles, where the robot has to move from a given initial position to a known target position.

For this purpose we defined the two positions and model each of it with 10 task neurons. These task neurons were randomly distributed around the position their encode with a very small variance and have fixed probability to spike. They have the same refractory period of $10ms$ as the state neurons. The timing of their spikes is defined by the kind of position they encode, i.e., for the population that encodes the initial position, we chose activity patterns of $300ms$. This allows the network to initialize itself at the given initial position. After this time the task neurons encoding the initial position stopped spiking and the task neurons encoding the target position started to spike until the predefined sample length was reached, here $T = 1300ms$ (resp. $1000ms$ for the factorized model and the hierarchical model).

We evaluated the models with several different initial and target positions. We calculated the acceptance rate of the rejection sampling, the time to generate one movement trajectory (including sampling and decoding), the average smoothness and a target reaching error. All three models were able to generate smooth goal directed movement trajectories that could be executed on the real robot.

Table 4.1 shows the results for these criteria. In total 1000 trajectories were generated with each model. These were split into 10 different tasks, i.e., different combinations of initial and target positions, and 100 trajectories were generated for each task. The acceptance rate is calculated as the ratio of accepted samples to total samples generated. The computational time is the time required to sample and decode a movement trajectory[1]. The smoothness criteria is calculated as the average jerk of the trajectories. For this calculation not every $1ms$ timestep of the trajectory was used. The jerk is calculated in timesteps of $20ms$. The target error denotes the smallest distance of the trajectories to the desired target position.

As the hierarchical model generates a movement trajectory in task space (T) and simultaneously the corresponding trajectories in joint space (J), we evaluated both results. The acceptance rate and the computational times are only calculated once, as rejection sampling is only executed in task space sampling and the computational time includes sampling and decoding of the trajectories in both spaces. For calculating the smoothness and target error criteria of the joint trajectories of the hierarchical model they were mapped into task space using forward kinematics.

| Model | Acceptance rate | Computational time | Smoothness ($ms^{-3}$) | Target error |
|---|---|---|---|---|
| 2D task space | 0.89 | $680 \pm 40ms$ | $0.87 \pm 0.15 \times 10^{-2}$ | $2.24 \pm 1.93cm$ |
| Factorized | 0.96 | $82ms \pm 2ms$ | $1.73 \pm 0.68 \times 10^{-2}$ | $0.11 \pm 0.13rad$ |
| Hierarchical (T) | 0.89 | $1370 \pm 72ms$ | $0.96 \pm 0.19 \times 10^{-2}$ | $2.37 \pm 1.53cm$ |
| Hierarchical (J) | 0.89 | $1370 \pm 72ms$ | $0.87 \pm 0.18 \times 10^{-2}$ | $4.22 \pm 2.79cm$ |

**Table 4.1.:** Evaluation of the three models on simple target reaching tasks. The hierarchical model generates a movement in task space (T) and the corresponding movement in joint space (J). The state space had the dimension $70 \times 70cm$.

[1] using MATLAB on a 8 Core i7 3.4 GHz, 32 GB RAM machine

The factorized model is the fastest, however, the trajectories are less smooth compared to the other models. Note that, we did not use any parallelization. Due to the independence of the joint models in the hierarchical model, their sampling could be done in parallel to speed up the computational time. The task space trajectories of the two-dimensional task space model and of the hierarchical model achieve equally good results on the target error. However, the joint space trajectories of the hierarchical model are slightly worse. This may reflect that the learned inverse kinematic models cannot map the movements perfectly from task to joint space.

### 4.5.1 Inspecting the sampled trajectories

Next we take a closer look at the results of the three models for a simple target reaching task.

We start with the model for two-dimensional task space planning. To generate a smooth movement we sampled 30 trajectories from the learned model, filtered and combined them as described in Subsection 4.4. Figure 4.7 shows the sampled and combined movement trajectories for a particular planning problem. It illustrates the exploration of the different samples and that by combining multiple samples, a much smoother and more goal-directed trajectory can be obtained. Sampling and decoding one trajectory of length $1300ms$ from the model took $680 \pm 40ms$ (see Table 4.1).

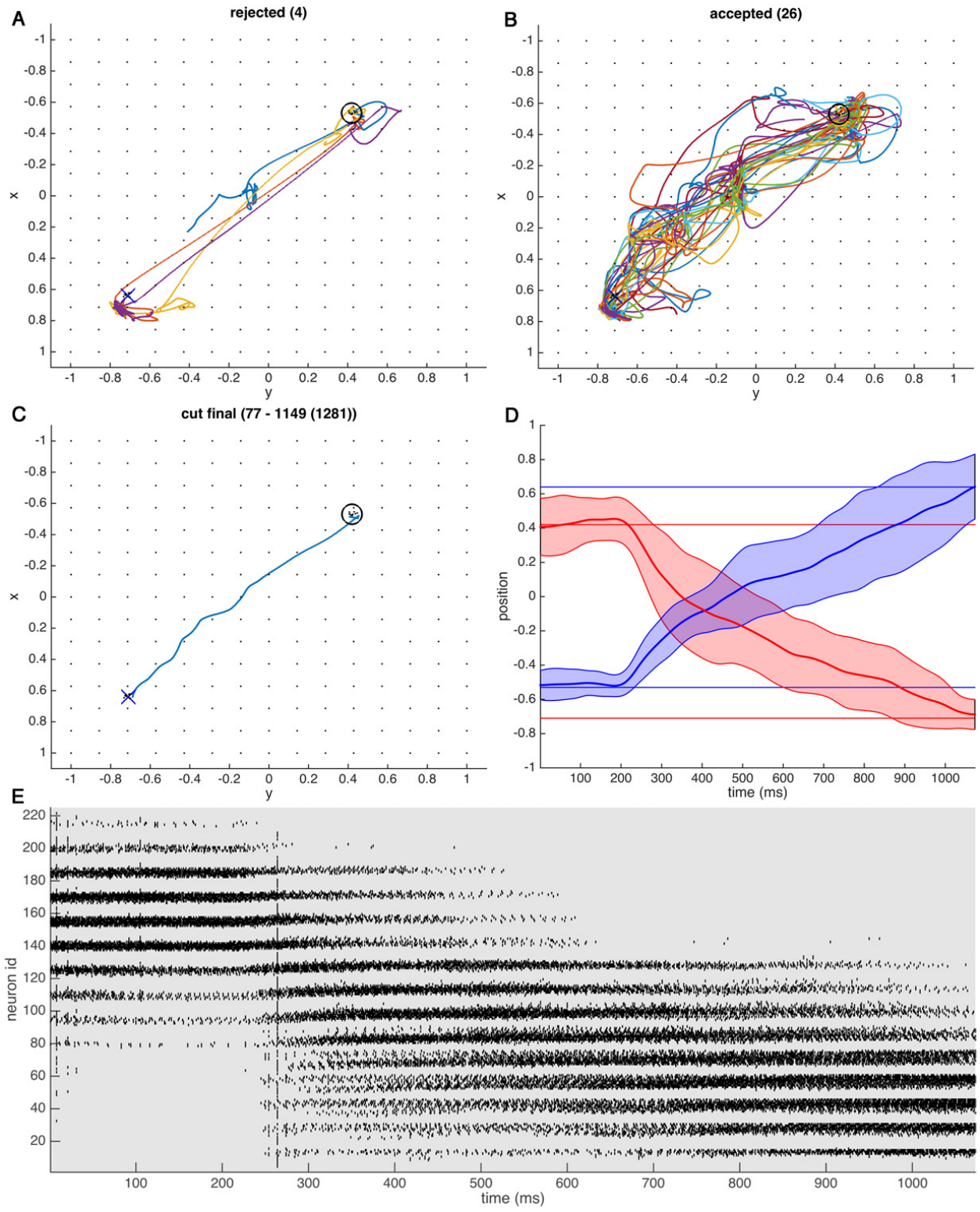


**Figure 4.7.:** Target reaching task solved with the two-dimensional task space planning model. (A-C) Show the movement trajectories in $xy$ space with the underlying state neurons (black dots); initial position denoted by black circle, target position denoted by blue cross. (A) The samples that got rejected. (B) The accepted samples that are used to generate the smooth movement. (C) The smoothed and shortened final movement trajectory. The numbers indicate the start and end cut positions and the total length in $ms$. (D) The smoothed and shortened (mean) trajectory and the standard deviation of the accepted samples plotted for $x$ and $y$; horizontal lines indicate the initial and target positions. (E) The cumulated spike train of the 225 state neurons of the accepted samples.

For the evaluation of the factorized joint space planning model we defined a simple target reaching task in the joint space instead of the task space. The six independent models sampled movement trajectories solving the target reaching task in each dimension independently. We used the same techniques for rejecting and smoothing as before, however with one-dimensional movement trajectories and in the joint space. For generating the final movement trajectories, 30 samples with each model were generated.

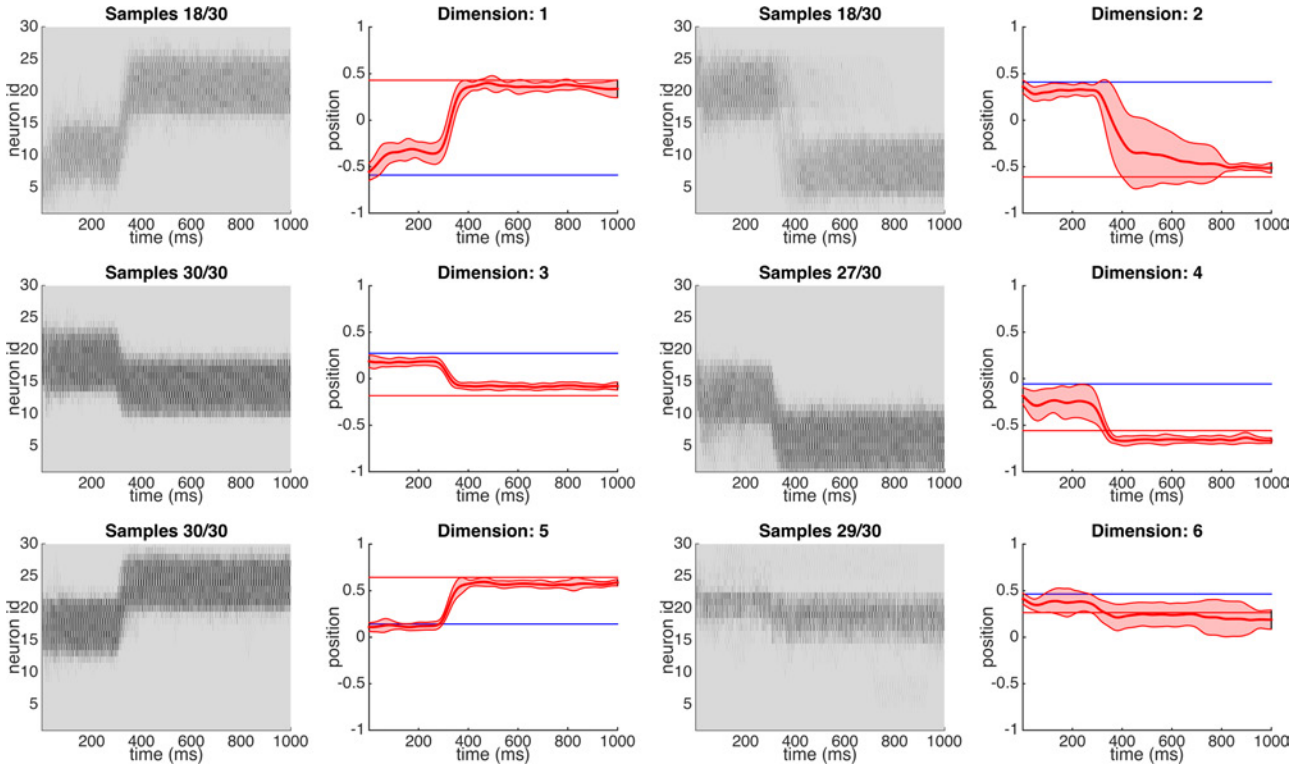


**Figure 4.8.:** Shown here are the cumulated spike trains of the 30 state neurons of each model of the accepted samples. The fraction above the spike trains indicate the fraction of accepted samples. Next to the spike trains the associated smoothed movement trajectories (means) in each dimension are plotted. Horizontal lines indicate initial (blue) and target (red) positions.

Sampling and decoding one trajectory of length $1000ms$ from one independent model took $82ms \pm 2ms$ (see Table 4.1). Due to the independence of the single models, the samples of each model for one complete joint trajectory can be sampled in parallel. Compared to the full population code model in two-dimensional task space, the factorized model is much faster. Due to the independence of the single models, they cannot exchange any information during sampling. Because of this, the factorized model cannot deal with obstacles directly. Another drawback resulting from this lack of information exchange is that the final movement of the six joints are not synchronized. The movement in each dimension is generated by taking the mean of the samples and are completely independent. Therefore, after putting together the single one-dimensional movements into one movement in higher dimensional space, the resulting movement may not be *natural*. This means the movement speed in each dimension may vary. Each joint will try to reach its target position as fast as possible independent of the others. In the extreme case the joints reach their target position one after another instead of reaching the *overall* target position of all joints simultaneously.

As the factorized model generates movement trajectories in joint space, the movements can be executed directly on the robot, i.e., no inverse kinematic model is required.

Now in order to evaluate the hierarchical model, we defined a planning task in task space. The task space model samples the movement trajectory. The generated spikes of the state neurons were used as input for the inverse kinematic models to simultaneously translate the task space movement into joint space movements. The activity of the state neurons of the inverse kinematic models are used as feedback in each timestep and are considered as additional input to the task space planning model during planning. Sampling and decoding of one trajectory in the hierarchical model took $1370 \pm 72ms$ (see Table 4.1).

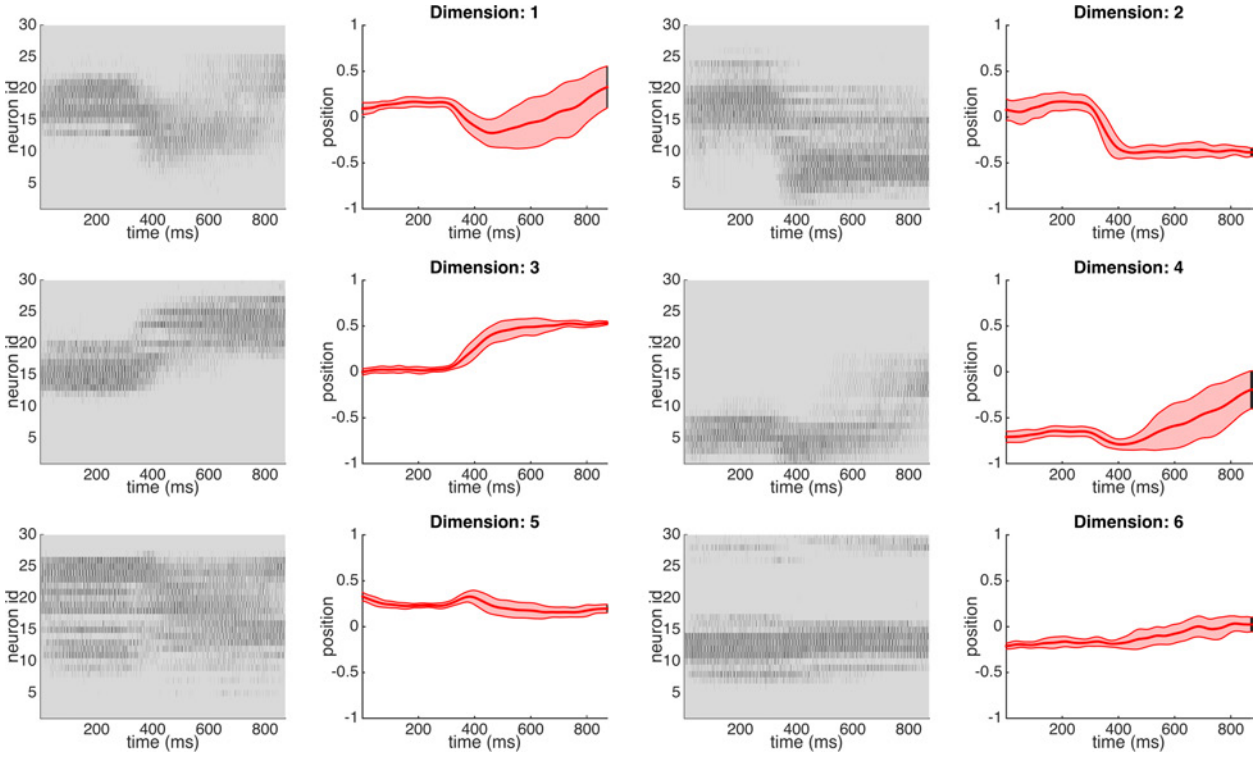


**Figure 4.9.:** Shown here are the cumulated spike trains of the 30 state neurons of each model of the accepted samples. Samples are rejected in task space like with the two-dimensional task space model. Next to the spike trains the associated smoothed movement trajectories (means) in each dimension are plotted.

Figure 4.9 and Figure 4.10 show the results of the hierarchical model. In Figure 4.9, the cumulated spike trains of the inverse kinematic models are plotted for each joint. Next to the spike train the movement trajectory of the corresponding joint is plotted. For building the smooth trajectory the mean of the accepted samples is taken. As planning is done in task space there are no initial and target positions in joint space. In comparison to Figure 4.8 (the independent SNNs results), the trajectories smoothly converge to a target value. Thus, the correlation between the task space and the joint space prevents jumps.

We used the same techniques for rejecting samples and combining multiple samples into a smooth movement as described before. Figure 4.10 shows the cumulated spike train of the task space neurons of the accepted samples and the movement trajectories.

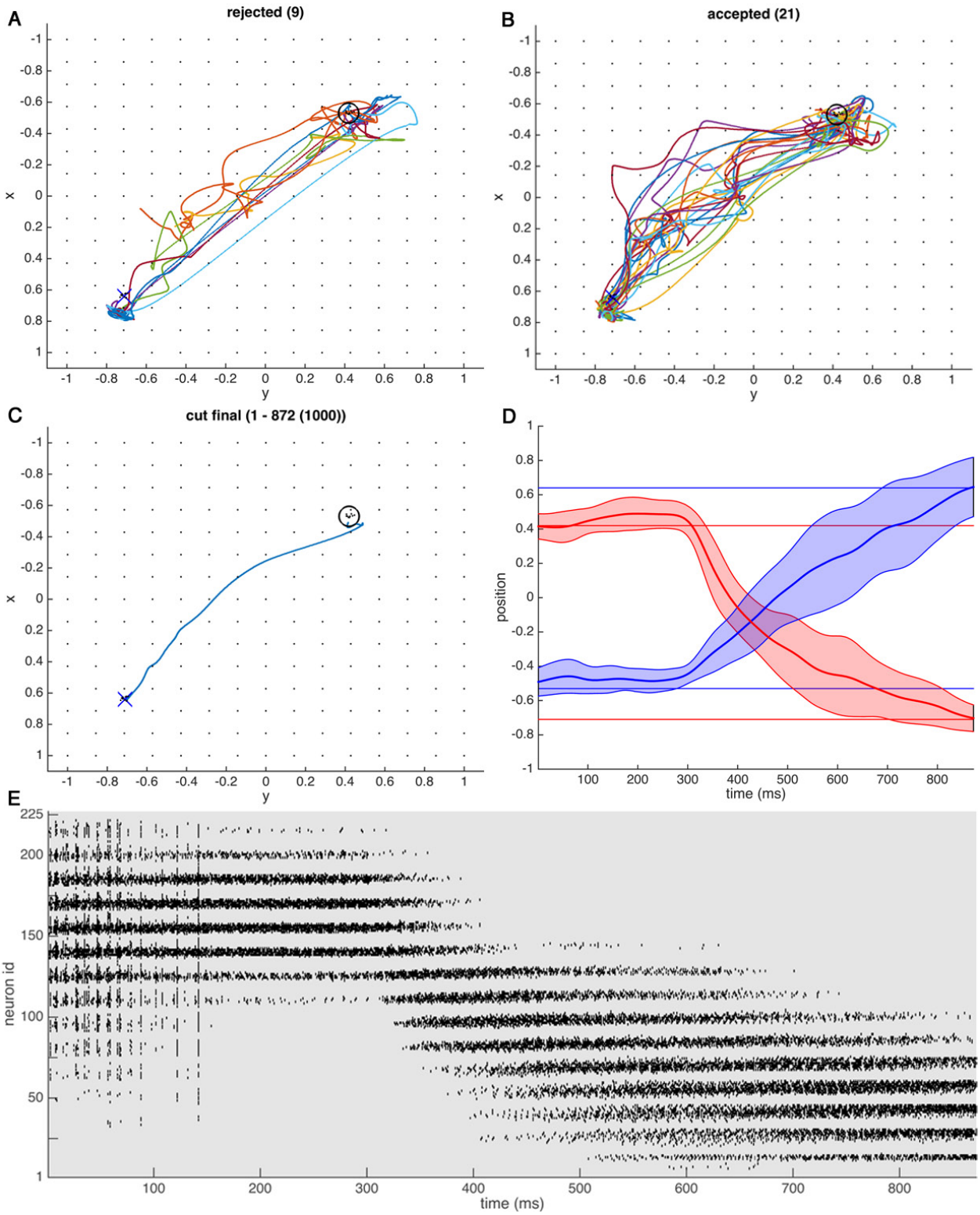


**Figure 4.10.:** Target reaching task solved with the hierarchical planning model. (A-C) Show the movement trajectories in $xy$ space with the underlying state neurons (black dots); initial position denoted by black circle, target position denoted by blue cross. (A) The samples that got rejected. (B) The accepted samples that are used to generate the smooth movement. (C) The smoothed and shortened final movement trajectory. The numbers indicate the start and end cut positions and the total length in $ms$. (D) The smoothed and shortened (mean) trajectory and the standard deviation of the accepted samples plotted for $x$ and $y$; horizontal lines indicate the initial and target positions. (E) The cumulated spike train of the $225$ task space state neurons of the accepted samples.

## 4.6 Obstacle avoidance task

In Subsection 4.5 we showed that we can use the transition models learned from demonstrations to generate smooth and goal directed movement trajectories. Although that those results show that the biological inspired approach based on spiking neural network models are applicable and useful for real robot problems, the experiment setup was quite simple. Given an initial position and a target position the optimal solution is just a straight line. In contrast, real world robot problems are rarely that easy. So in order to evaluate our models on more difficult problems we increased the complexity of the task by adding obstacles.

As described in Subsections 3.3 and 3.7, the proposed spiking neural networks can model multi-modal distributions to encode multiple solutions. Obstacles can be modelled easily by putting a strong inhibition on the state neurons covered by an obstacle. By this these covered state neurons cannot be active any more and no sampled movement trajectory will cross the area covered by an obstacle. As mentioned in Subsection 3.5 the factorized model consisting of $N$ independent one-dimensional models cannot deal with obstacle directly, thus we only considered the two-dimensional task space planning model and the hierarchical model for the obstacle avoidance task.

Additionally to the initial and target positions, each modelled by 10 context neurons, we added two obstacles that were modelled by 20 context neurons. These two obstacles had different size and shape and blocked the *direct* way to the target position, i.e., the solutions from the target reaching task became infeasible.

Without changing the parameters of the model, especially the spiking activity and weights of the task neurons, the acceptance rate for the obstacle avoidance task is much lower than for the simpler target reaching task. This rate can be improved by finding better parameters, which can for example be done using reinforcement learning as in [38]. However, the focus of this work was the learning of the transition model and building an advanced model including the inverse kinematics. So we kept the task neurons activity and weights fixed. In total, we drew 200 samples for the obstacle avoidance task and rejection sampling was done as in the previous experiment, i.e., based on the accuracy of a trajectory and its speed.

For a statistical comparison of the two models, we evaluated the same four criteria as in the target reaching task. For both models we averaged over 10 tasks with 100 trajectories each.

| Model | Acceptance rate | Computational time | Smoothness ($ms^{-3}$) | Target error |
|---|---|---|---|---|
| 2D task space | 0.45 | $635 \pm 8ms$ | $0.85 \pm 0.16 \times 10^{-2}$ | $2.71 \pm 2.24cm$ |
| Hierarchical (T) | 0.64 | $1361 \pm 73ms$ | $0.96 \pm 0.18 \times 10^{-2}$ | $2.21 \pm 1.48cm$ |
| Hierarchical (J) | 0.64 | $1361 \pm 73ms$ | $0.87 \pm 0.20 \times 10^{-2}$ | $4.04 \pm 2.35cm$ |

**Table 4.2.:** Evaluation of the two-dimensional task space model and the hierarchical model on obstacle avoidance tasks. The hierarchical model generates a movement in task space (T) and the corresponding movement in joint space (J). The state space had the dimension $70 \times 70cm$.

The computational time and the smoothness do not change for both models when obstacles are added. However, the acceptance rate drops for both models. The hierarchical model has the higher acceptance rate. This indicates that the correlation between task and joint space helps to generate rewarding trajectories. The two-dimensional task space model performs slightly worse at the target error while the hierarchical model can slightly increase its accuracy. In total, however, the target error of both models stays roughly the same compared to the results of the target reaching tasks.

### 4.6.1 Obstacle avoidance with the task space model

Figure 4.11 shows two example trajectories of an obstacle avoidance task solved with the two-dimensional task space planning model. Both trajectories reach the target position precisely while avoiding the two obstacles with different movements. Because we just used a single sample here instead of combining multiple samples, the resulting movements are less smooth than in the previous experiment.

Figure 4.12 shows another two examples, where the neural activity of the state neurons are contrasted.

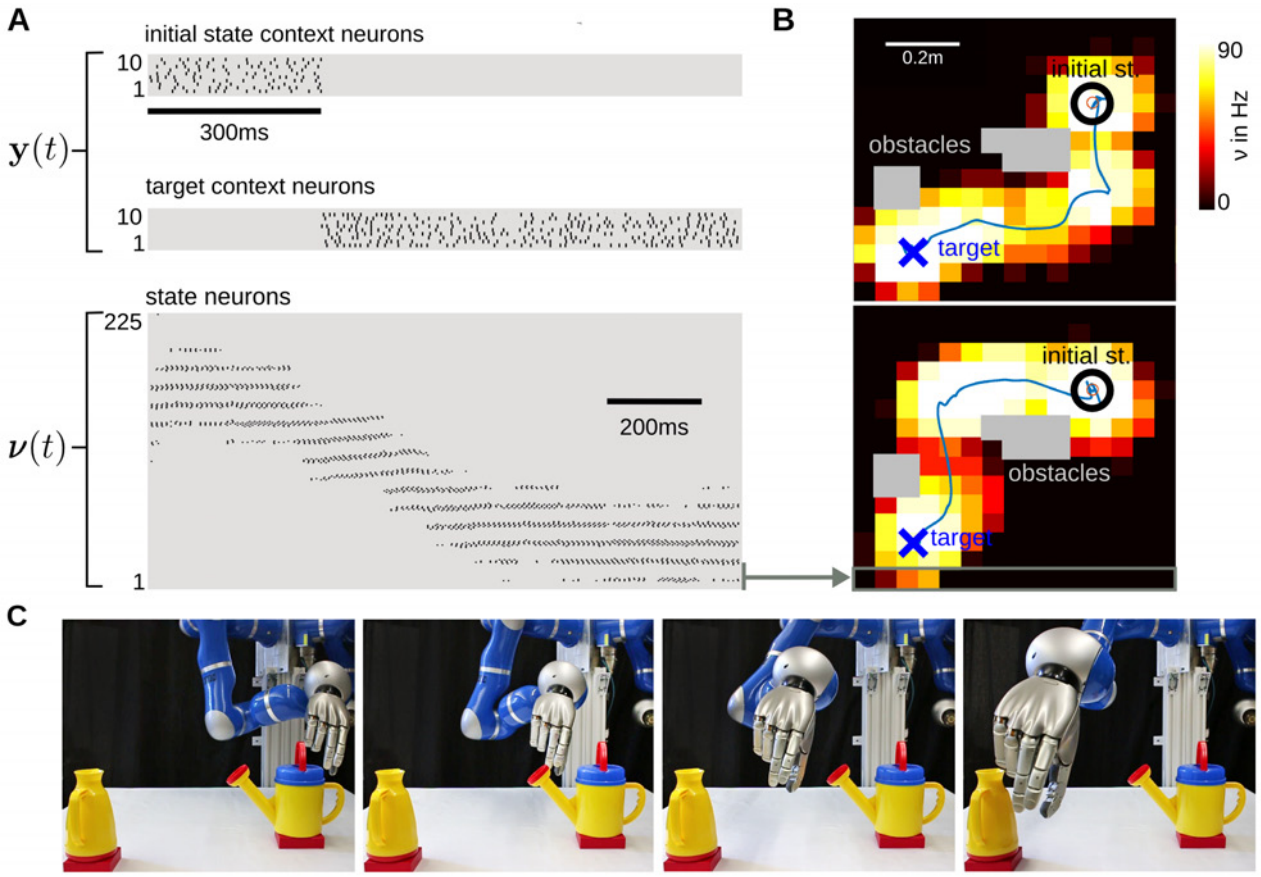


**Figure 4.11.:** (A) Raw sample drawn from the learned model with the activity of the task (context) neurons in the upper part and the state neurons in the lower part. (B) The decoded movement trajectories plotted over the integrated neural activity of the state neurons for two different solutions to the obstacle avoidance task sampled from the learned model. (C) Snapshots of the execution of the second solution on the KUKA lightweight arm.

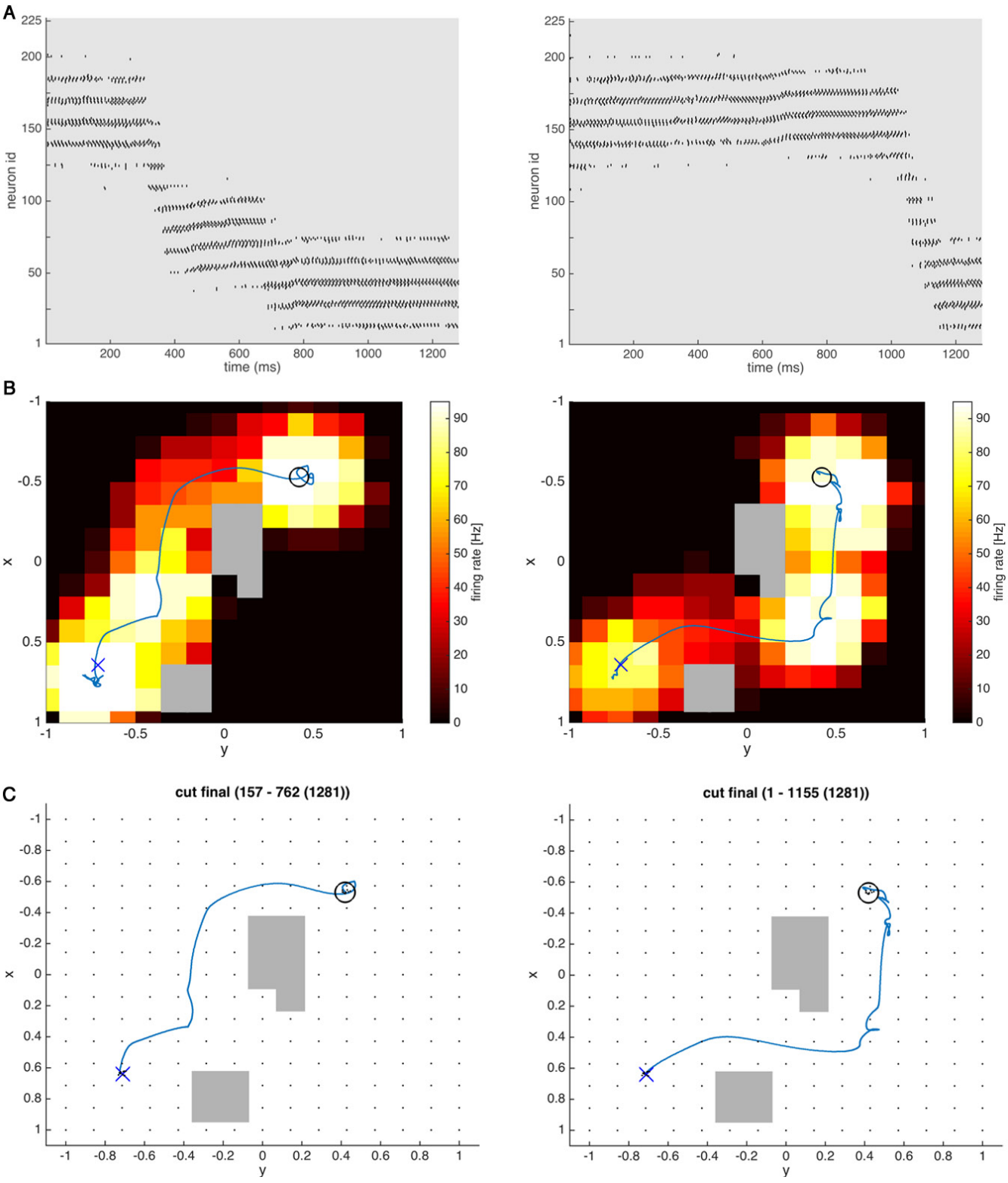


**Figure 4.12.:** Two solutions to the given obstacle avoidance task sampled from the two-dimensional task space planning model. (A) The raw samples drawn from the two-dimensional task space planning model for the given task. Shown are the spiking activities of the state neurons encoding the movements. (B) The decoded movement trajectories from the raw samples plotted over the integrated neural activity of the state neurons. (C) The final movement trajectories. The numbers indicate the start and end cut positions and the total length in $ms$.

### 4.6.2 Obstacle avoidance with the hierarchical model

Sampling, decoding, rejecting and executing of the sampled movements is done in the same way as described in detail in the target reaching task description of the hierarchical model (Subsection 4.5).

In Figure 4.13, the spike trains of the inverse kinematic models are plotted for each joint and next to them the associated movements in joint space are shown. The rejection step is done in task space. Figure 4.14 shows two sampled solutions in task space with the associated movements and activities of the task space state neurons.

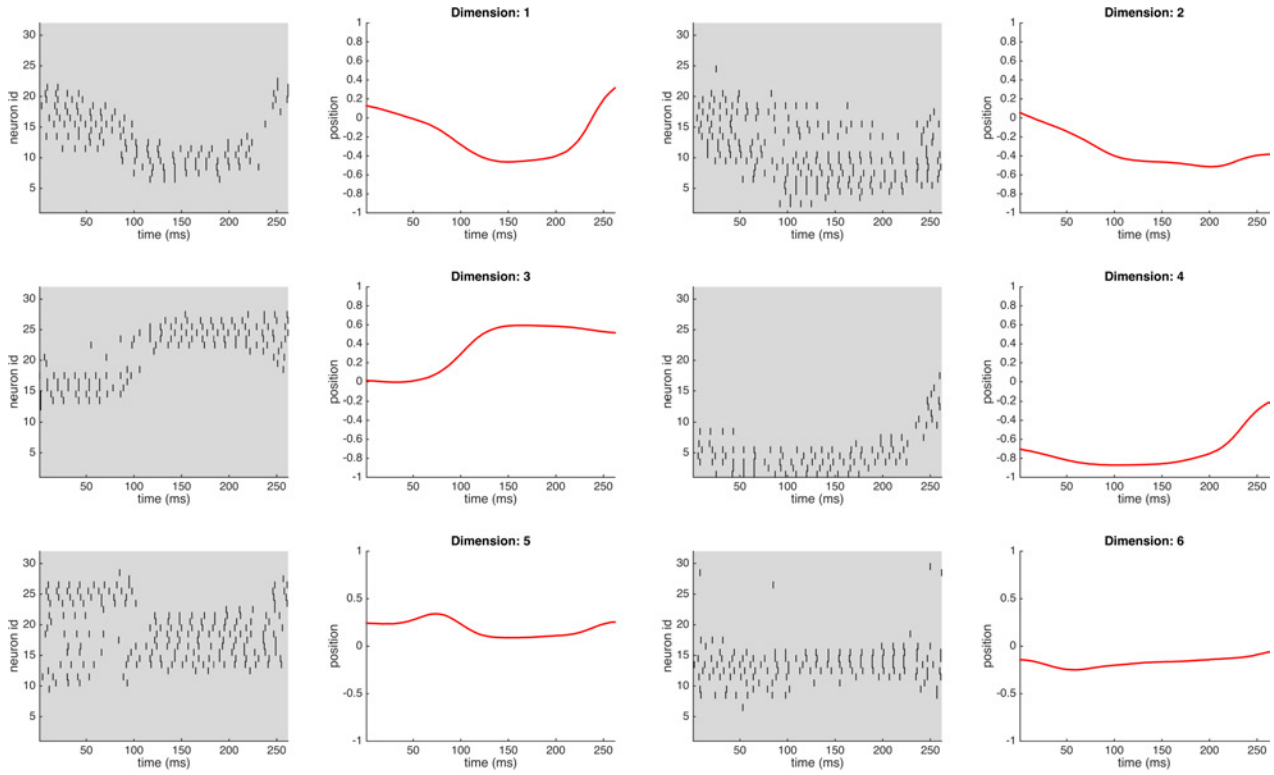


**Figure 4.13.:** Shown here are the spike trains of the 30 joint space state neurons of each inverse kinematic model of one accepted sample. Samples are rejected in task space like with the two-dimensional task space model. Next to the spike trains the associated joint space trajectories generated by the inverse kinematic model in each dimension are plotted.

If we compare the movements of the two-dimensional task space planning model to the hierarchical model, we clearly see that the movements of the hierarchical model are *better*. They are more goal-directed, do not have loops and when executed on the real robot they look more natural. This may come from the feedback of the joint states during the planning progress in the hierarchical model. Note that the resulting joint space trajectories are smooth enough to be executed on the real robot after the stretching procedure.

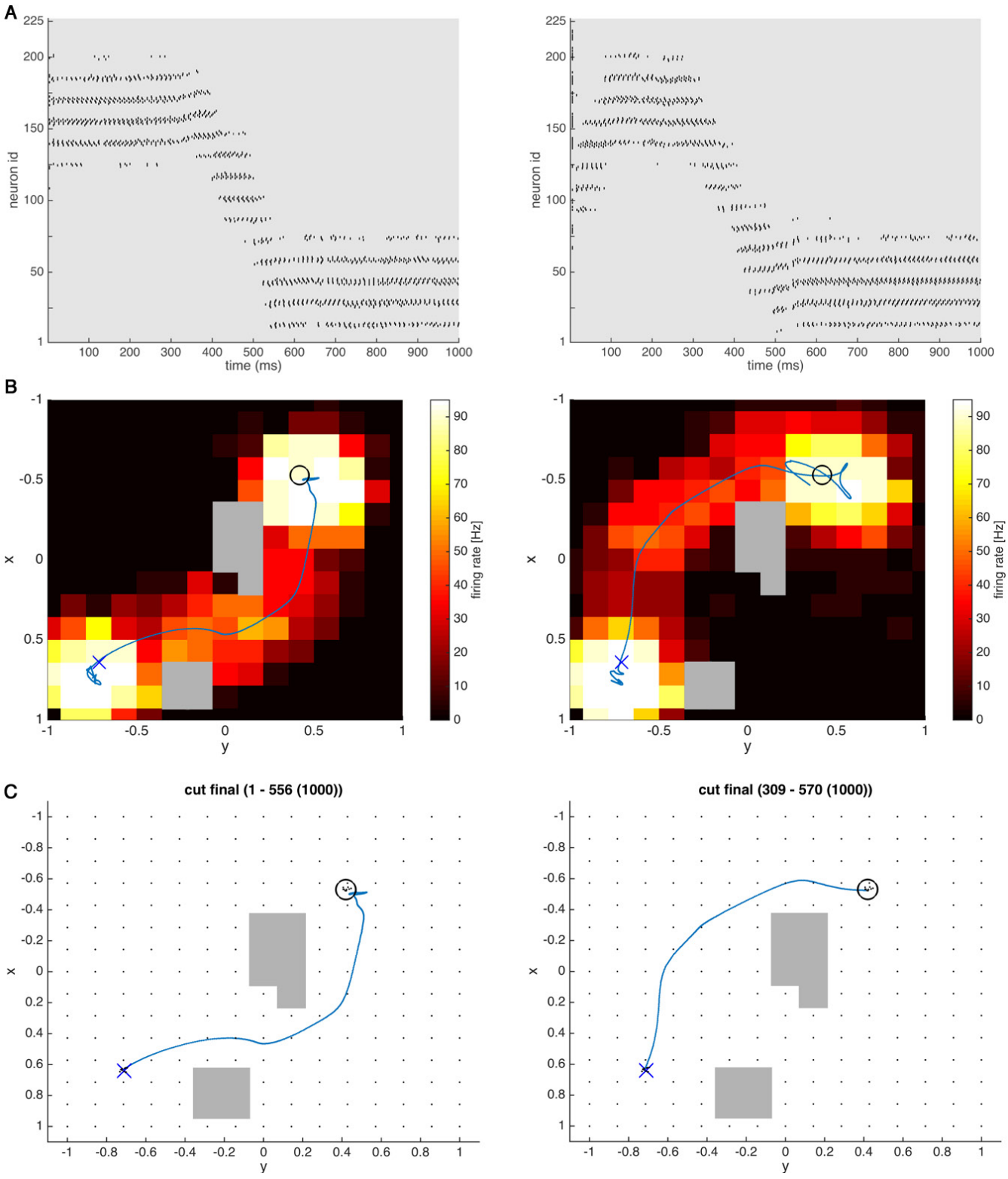


**Figure 4.14.:** Two solutions to the given obstacle avoidance task sampled from the hierarchical model directly in task space. (A) The raw samples drawn from the hierarchical model for the given task. Shown are the spiking activities of the task space state neurons encoding the movements in task space. (B) The decoded movement trajectories from the raw samples plotted over the integrated neural activity of the task space state neurons. (C) The final movement trajectories, numbers indicate the start and end cut positions and the original length.

# 5 Conclusion & Future Work

### Conclusion

In this thesis, we demonstrated that spiking neural networks (SNN) are suitable robot control models with interesting features compared to classical planning methods. The approach is motivated by recent findings in the neural activity in rats during phases of mental planning.

We showed that with the proposed SNN **model learning of arbitrary complex functions** can be realized, i.e., the proposed state transition and kinematics models can represent non-linear and non-unique mappings. One of the learned models can be queried in both directions. Thus, forward and inverse models can be learned at the same time. For learning of the models, we used kinesthetic teaching and derived a spike dependent learning rule based on contrastive divergence.

Using factorized population codes we **scaled a neural controller to high-dimensional systems** and evaluated it on a six-dimensional planning problem. In addition to an existing task space planner, two models were developed. First, an independent factorized joint space control model and second, a hierarchical model operating in task and joint space simultaneously.

This hierarchical model combines the benefits of the task space and the factorized joint space control approaches, i.e., the ability to scale to high-dimensional systems and the feature of encoding obstacles as repelling forces. The hierarchical model generates movement plans in task space and joint angles trajectories at the same time. **Encoding obstacles of arbitrary shape** is implemented through synaptic inhibition of the state neurons such that no sampled movement will cross this area. The state neurons that are *deactivated* by this inhibition mechanism act like repelling forces that prevent the movements to cross the obstacle areas.

The proposed models sampled movement plans in 80 to 1300$ms$. For an executed robot movement of 5$s$, the plans were generated 4 to 60 times faster than real-time. This buffer can be **used for foraging robot control** by preparing multiple alternatives to planning problems and deciding online which plan to execute. Online rejection sampling was implemented to achieve target reaching errors of less than 3$cm$ in a $70 \times 70cm$ operational area, corresponding to 4%.

The learned transition models had a Gaussian-like shape and resulted in **smooth trajectories**. No post processing as in RRT was required, the trajectories could be directly executed on the robot. Best results were achieved with the hierarchical model due to the bidirectional feedback between task space and joint space neurons.

### Future Work

During all of our experiments we kept the properties of the task neuron populations fixed, i.e., the synaptic connections to the state neurons and their activity patterns. As it was shown in [38], this task related input can be learned through reinforcement learning. A promising robot controller could be developed by combining the learning rules and results of both studies.

Another interesting research direction is the grid representation of the state neurons. The number of state neurons and their positions are manually set and define the complexity of the model. These parameters may be learned as well[8]. Using such a dynamic approach, the effect of the curse of dimensionality may be reduced as the number of neurons is only relatively high in areas where high precision is required and low anywhere else. Additionally, the model may adapt to dynamically changing task requirements.

Furthermore, the model provides some interesting possible extensions for planning or neuroscientific studies, e.g., additionally encoding actions[10], installing multiple cognitive maps[1] or simulating forward and backward replays[9][18].

# Bibliography


[1] A. H. Azizi, L. Wiskott, and S. Cheng. A computational model for preplay in the hippocampus. *Frontiers in computational neuroscience*, 7, 2013.

[2] J. Bill and R. Legenstein. A compound memristive synapse model for statistical learning through stdp in spiking neural networks. *Frontiers in neuroscience*, 8, 2014.

[3] C. M. Bishop. *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2006.

[4] M. Botvinick and M. Toussaint. Planning as inference. *Trends in Cognitive Sciences*, 16(10):485 – 488, 2012.

[5] J. Brea, W. Senn, and J. pascal Pfister. Sequence learning with hidden units in spiking neural networks. In *Advances in Neural Information Processing Systems 24*, pages 1422–1430. Curran Associates, Inc., 2011.

[6] J. Bruce and M. Veloso. Real-time randomized path planning for robot navigation. In *Intelligent Robots and Systems, 2002. IEEE/RSJ International Conference on*, volume 3, pages 2383–2388 vol.3, 2002.

[7] L. Buesing, J. Bill, B. Nessler, and W. Maass. Neural dynamics as sampling: A model for stochastic computation in recurrent networks of spiking neurons. *PLoS Comput Biol*, 7(11):e1002211, 11 2011.

[8] U. M. Erdem and M. E. Hasselmo. A goal-directed spatial navigation model using forward trajectory planning based on grid cells. *The European Journal of Neuroscience*, 35(6):916–931, 2012.

[9] D. J. Foster and M. A. Wilson. Reverse replay of behavioural sequences in hippocampal place cells during the awake state. *Nature*, 440(7084):680–683, 2006.

[10] N. Frémaux, H. Sprekeler, and W. Gerstner. Reinforcement learning using a continuous time actor-critic framework with spiking neurons. *PLoS Comput. Biol*, 9(4), 2013.

[11] M. Fyhn, T. Hafting, M. P. Witter, E. I. Moser, and M.-B. Moser. Grid cells in mice. *Hippocampus*, 18(12):1230–1238, 2008.

[12] A. P. Georgopoulos, A. B. Schwartz, and R. E. Kettner. Neuronal population coding of movement direction. *Science*, 233(4771):1416–1419, 1986.

[13] W. Gerstner and W. M. Kistler. *Spiking neuron models: Single neurons, populations, plasticity*. Cambridge university press, 2002.

[14] N. J. Gustafson and N. D. Daw. Grid cells, place cells, and geodesic generalization for spatial reinforcement learning. *PLoS computational biology*, 7(10):e1002235, 2011.

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

[16] G. E. Hinton. Training products of experts by minimizing contrastive divergence. *Neural Comput.*, 14(8):1771–1800, Aug. 2002.

[17] Y. Huang and R. P. Rao. Neurons as monte carlo samplers: Bayesian inference and learning in spiking networks. In *Advances in Neural Information Processing Systems 27*, pages 1943–1951. Curran Associates, Inc., 2014.

[18] A. Johnson and A. D. Redish. Neural ensembles in ca3 transiently encode paths forward of the animal at a decision point. *The Journal of neuroscience*, 27(45):12176–12189, 2007.

[19] M. Kalakrishnan, S. Chitta, E. Theodorou, P. Pastor, and S. Schaal. Stomp: Stochastic trajectory optimization for motion planning. In *Robotics and Automation (ICRA), 2011 IEEE International Conference on*, pages 4569–4574, May 2011.

[20] D. Kappel, B. Nessler, and W. Maass. Stdp installs in winner-take-all circuits an online approximation to hidden markov model learning. *PLoS Comput. Biol*, 10:e1003511, 2014.

[21] J. Kuffner and S. LaValle. Rrt-connect: An efficient approach to single-query path planning. In *Robotics and Automation, 2000. Proceedings. ICRA '00. IEEE International Conference on*, volume 2, pages 995–1001 vol.2, 2000.

[22] S. M. LaValle. Rapidly-exploring random trees: A new tool for path planning. Technical report, 1998.

[23] S. M. LaValle. *Planning Algorithms*. Cambridge University Press, New York, NY, USA, 2006.

[24] S. M. Lavalle, J. J. Kuffner, and Jr. Rapidly-exploring random trees: Progress and prospects. In *Algorithmic and Computational Robotics: New Directions*, pages 293–308, 2000.

[25] W. J. Ma, J. M. Beck, P. E. Latham, and A. Pouget. Bayesian inference with probabilistic population codes. *Nature neuroscience*, 9(11):1432–1438, 2006.

[26] W. Maass. Networks of spiking neurons: the third generation of neural network models. *Neural networks*, 10(9):1659–1671, 1997.

[27] T. M. Mitchell. *Machine Learning*. McGraw-Hill, Inc., New York, NY, USA, 1 edition, 1997.

[28] E. I. Moser, E. Kropff, and M.-B. Moser. Place cells, grid cells, and the brain's spatial representation system. *Annu. Rev. Neurosci.*, 31:69–89, 2008.

[29] K. P. Murphy. *Machine Learning: A Probabilistic Perspective*. The MIT Press, 2012.

[30] K. Nakazawa, T. J. McHugh, M. A. Wilson, and S. Tonegawa. Nmda receptors, place cells and hippocampal spatial memory. *Nature Reviews Neuroscience*, 5(5):361–372, 05 2004.

[31] E. Neftci, S. Das, B. Pedroni, K. Kreutz-Delgado, and G. Cauwenberghs. Event-driven contrastive divergence for spiking neuromorphic systems. *Frontiers in Neuroscience*, 7(272), 2014.

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

[33] B. E. Pfeiffer and D. J. Foster. Hippocampal place-cell sequences depict future paths to remembered goals. *Nature*, 497(7447):74–79, 05 2013.

[34] F. Ponulak and A. Kasinski. Introduction to spiking neural networks: Information processing, learning and applications. *Acta Neurobiol Exp (Wars)*, 71(4):409–433, 2011.

[35] M. Prezioso, F. Merrikh-Bayat, B. Hoskins, G. Adam, K. K. Likharev, and D. B. Strukov. Training and operation of an integrated neuromorphic network based on metal-oxide memristors. *Nature*, 521(7550):61–64, 2015.

[36] N. Ratliff, M. Zucker, J. A. D. Bagnell, and S. Srinivasa. Chomp: Gradient optimization techniques for efficient motion planning. In *IEEE International Conference on Robotics and Automation (ICRA)*, May 2009.

[37] E. Rueckert. On biologically inspired motor skill learning in robotics through probabilistic inference. Technical report, Ph.D. Thesis, University of Technology, Graz, Austria, 2014.

[38] E. Rueckert, D. Kappel, D. Tanneberg, D. Pecevski, and J. Peters. Recurrent spiking networks solve planning tasks. *under review*, 2015.

[39] T. Solstad, C. N. Boccara, E. Kropff, M.-B. Moser, and E. I. Moser. Representation of geometric borders in the entorhinal cortex. *Science*, 322(5909):1865–1868, 2008.

[40] K. L. Stachenfeld, M. Botvinick, and S. J. Gershman. Design principles of the hippocampal cognitive map. In *Advances in Neural Information Processing Systems 27*, pages 2528–2536. Curran Associates, Inc., 2014.

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

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

[43] M. Toussaint and C. Goerick. A bayesian view on motor control and planning. In *From Motor Learning to Interaction Learning in Robots*, pages 227–252. Springer, 2010.

[44] O. Woodford. Notes on contrastive divergence.

[45] M. Zucker, N. Ratliff, A. Dragan, M. Pivtoraiko, M. Klingensmith, C. Dellin, J. A. D. Bagnell, and S. Srinivasa. Chomp: Covariant hamiltonian optimization for motion planning. *International Journal of Robotics Research*, May 2013.

# A List of publications

## A.1 Comments and Contributions to Publications

The paper "Recurrent spiking networks solve planning tasks"[38] presents parts of the results developed in this thesis. In particular, using the two dimensional planning model on the real robot and the model learning results were presented.

The paper "Spiking deep networks for robot control" by Daniel Tanneberg, Jan Peters and Elmar Rueckert is in the progress of writing (2015). Section 3.6 and Secition 4 provide the foundation for this manuscript.