Integration of Sensorimotor Mappings by Making Use of Redundancies

Nikolas J. Hemion
Emergentist Semantics
CoR-Lab
Bielefeld University
Email: nhemion@cor-lab.uni-bielefeld.de

Frank Joublin
Honda Research Institute Europe GmbH
Email: Frank.Joublin@honda-ri.de

Katharina J. Rohlfing
Emergentist Semantics
CoR-Lab
Bielefeld University
Email: kjr@uni-bielefeld.de

***Abstract*—We present a novel approach to learn and combine multiple input to output mappings. Our system can employ the mappings to find solutions that satisfy multiple task constraints simultaneously. This is done by training a network for each mapping independently and maintaining all solutions to multivalued mappings. Redundancies are resolved online through dynamic competitions in neural fields. The performance of the approach is demonstrated in the example application of inverse kinematics learning. We show simulation results for the humanoid robot iCub where we trained two networks: One to learn the kinematics of the robot's arm and one to learn which postures are close to joint limits. We show how our approach can be used to easily integrate multiple mappings that have been learned separately from each other. When multiple goals are given to the system, such as reaching for a target location and avoiding joint limits, it dynamically selects a solution that satisfies as many goals as possible.**

I. Introduction

Autonomous robots that should acquire ever new skills by exploring their physical and social environment autonomously need to solve many problems that involve the capability of learning sensorimotor mappings. A well studied example is the learning of the own kinematics (e.g. [1], [2], [3]). Here the robot is faced with the problem to find correspondences between the variables that it can directly control (for example the angular positions of rotational joints) and the – in some way – measured positions of important parts of its body (for example its hand). This correspondence is usually not trivial and non-linear. For simple kinematic structures of the robot, it can be expressed as an equation by the designer. However the more complex the robot's body is, the more do external factors (such as gravity and friction) play a role. There are cases where it might even be impossible to a priori capture the robot's kinematics in an equation [4] (for example when using pneumatic joints or when there can occur physical change to the robot's body, as through damage or ware-off). Here, the kinematics of the robot has to be learned.

Also the acquisition of more complex skills requires the learning of sensorimotor mappings. Some examples from the robotics literature are learning the "affordances" of objects through physical interaction [5], learning a baseball swing [6], learning to play ping pong [7] and learning to shoot with bow and arrow [8].

Most approaches for the learning of sensorimotor mappings focus on accuracy of the resulting mapping, the generalization capability of the learning method from as few training examples as possible, and its suitability for online-learning. While all of these aspects are very important, often the learning of only one single new skill is studied in a special experimental setup in isolation. Humans constantly perform many of their learned skills in parallel. We can sip from a cup of coffee while walking, we can read in a magazine while stirring in a casserole, we can talk to another person while we are driving a car, etc. Thus, if we want to build robots that can autonomously learn new skills, we need to specify how the learned skills can be organized once the robot has learned more than just one skill, how they interact with each other and how they can be used in conjunction to learn even more complex skills.

In state-of-the-art robotics approaches orchestrating multiple skills in a task, usually the task is segmented into a sequence of basic actions and symbolic planning is used to find sequences of actions suitable to solve the task. In the work of Gienger et al. [9], the task of picking up an object is translated into the execution of an action sequence of the kind "walk to position x in front of the table," "find the object," "determine a good way to grasp and perform it," etc. Most of these basic actions are implemented as *whole-body controllers*, i.e. each skill takes over exclusive control of the entire body of the robot or at least entire parts of the body, such as limbs. Some of the actions can also be performed in parallel, for example visual search only requires using the head motors while the head movement is rather negligible for the behavior of walking to some position. However, this knowledge has to be carefully considered by the designer and is put into the system a-priori.

Similarly, reinforcement learning is concerned with finding sequences of actions to maximize reinforcement signals [10]. The learner has to find a sequence of actions that optimizes the gain of a target signal. Most approaches treat actions as discrete units, and even methods that can in principle deal with continuous state spaces treat actions as transitions from one robot "state" to another [11]. Thus, the parallel execution of several skills is not considered.

Humans are able to execute skills in parallel because our bodies are highly *redundant*, which means that we have many

ways to solve a single task. When you put the index finger of one of your hands on a spot on a table, you can still move your elbow around quite freely without lifting your finger. Thus, there are several solutions of arm configurations allowing to fulfill a task of keeping your finger on the same spot on the table. With your other hand you could now still reach most points on the table for the additional task to pick up an object, even if it meant to bend over or step to the side a bit. This redundancy, or flexibility in fulfilling a skill, allows us to perform several tasks in parallel.

Many robots also have redundant kinematic setups, for example humanoid robots mimic the structure of the human body. These robots in principle have the same, or at least some of the flexibility that we have in performing skills. In control theory, the subspace of the joint-space which allows for reaching a task goal is termed a *null-space* of a task [12]. Knowledge and control of the null-space offers advantages [13]: When controllers are designed by the system developer, null-space control allows for example to avoid self-collisions while reaching. The information necessary has again to be carefully considered by the human developer when implementing the controllers.

In contrast, in the machine learning literature redundancy is often discussed as a problem, the *non-convexity problem* [1]. The problem states that averaging between multiple solutions for one task does not necessarily yield a valid solution. Averaging between solutions is done in learning approaches that use function approximation when training from data points that originate from a *multivalued* function. Thus, these methods try to learn a *many-to-one* mapping yielding incorrect solutions where actually a *many-to-many* mapping should be learned. The most illustrative example is that of a simple robotic arm with a fixed base and two rotational joints. The robot can bring the end of its arm to most points in its workspace in two ways, which are the "elbow-up" and the "elbow-down" solutions. Learning methods that average between solutions will average these two solutions, ending up with associating a fully extended arm posture to every target point, which is incorrect.

To overcome this problem, "redundancy resolution schemes" are applied to the training set, which sort out training samples to guarantee that effectively only training samples from a single-valued function remain [14]. One consequence is of course, that the resulting learned mapping only stores a single solution for each target. Thus, there is a great loss of information and the robot cannot know how to move around in the null-space.

There are some learning methods that can also deal with multivalued functions, for example using locally linear models [2], through manifold learning [15] or using reservoir networks [16]. However it has not been shown, how multiple mappings can be combined to perform several tasks in parallel.

In this work, we therefore propose a system that is able to learn multivalued functions in a way that preserves information about the redundancy and further show how multiple mappings that are learned separately from each other can be integrated to find solutions that comply with several tasks at once.

II. Making Use of Redundancies

As we have just discussed, there exist more than one solution for individual problems in many sensorimotor tasks, as for example with a redundant robot arm that reaches for a point in space. If the robot learned about all the possible solutions, it could freely choose any one of them to reach its goal. This means that it would have a good deal of flexibility in choice and could also freely navigate from one of the solutions to another without having to leave the goal state.

We are therefore interested in finding a way to integrate multiple learned sensorimotor mappings into a system, such that the robot can find solutions that satisfy several task specifications simultaneously. Given that the robot can solve one task in many different ways, it would be convenient to let the system choose one among the possible solutions that also satisfies other tasks. Moreover, our focus lies on finding a generic way of integrating sensorimotor mappings, so that in the long run a robot could acquire ever new sensorimotor mappings online and orchestrate all of its learned mappings in parallel.

The system can employ sensorimotor mappings to generate sets of solutions for a given task. Consider the example of a planar robot with three rotational joints. To bring its end-effector to a point in its two-dimensional workspace, the robot has infinitely many solutions for all points that do not lie at the extreme ends of the workspace. For each target point, the solutions lie on a two-dimensional *manifold* in the three-dimensional space spanning the possible angular configurations of the robot. We will call these "solution manifolds".

If the robot now has learned two sensorimotor mappings, both of which use the robot's arm motors, then solving tasks involving both of these mappings will correspond to two solution manifolds lying in the three-dimensional arm angular space. A favorable solution would be one that solves both tasks, i.e. lies on both solution manifolds. If any such solution exists, which is the case if the two solution manifolds intersect, then such a solution should be selected. Otherwise, that is if the solution manifolds do not intersect, the system should select a solution that lies on either one of the manifolds and minimizes the distance to the other manifold, because then the robot has solved one task while being as close as possible to solving the other, given that the mappings have gradually changing values. Likewise, when having learned several skills and performing multiple tasks simultaneously, the robot should select solutions that lie on as many solution manifolds as possible, and as close as possible to the remaining ones.

Also, one of the tasks could have a higher priority for the robot than others. For example, actually brining the end-effector to the target point might be crucial, while it would be nice to avoid having joints in their limits while doing so. Thus, the former task would have a higher priority than the latter.

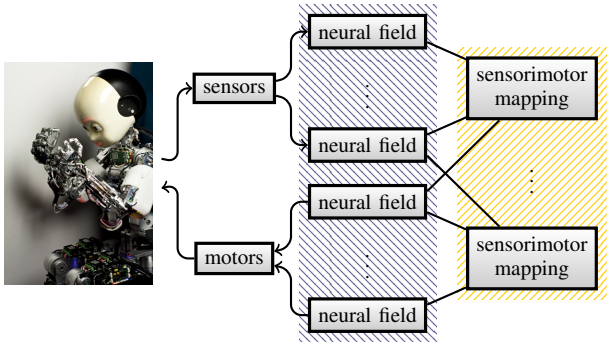


Fig. 1. A schematic overview of the proposed system. Input and output vectors are represented in a set of neural fields of receptive field units. Sensorimotor mappings are trained to store information about correlations between different in- and outputs. The neural fields are shared among all learned mappings, i.e. the field activations are used as inputs for all connected mappings. The responses of the mappings are then combined to generate a coherent system response.

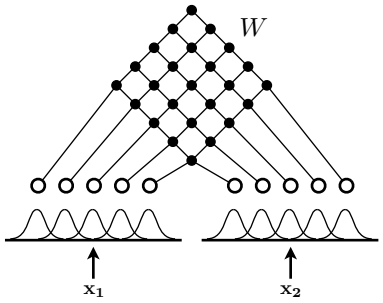


Fig. 2. Schematic view of the sigma-pi network, for simplicity in the case of two one-dimensional inputs. Inputs x_1 and x_2 first activate receptive field units in the neural fields, their activation is then fed into the network. The grid represents the possible synaptic connections with associated sigma-pi weights. Each unit in the neural fields can be connected to all units in the other field. Note that there are no lateral connection, but knots in the grid only represent multiplicative connections for which the network can store sigma-pi weights.

III. Computational Model

Figure 1 is a schematic overview of our proposed system. We use a set of neural fields to represent values of variables encoding for sensor inputs and motor outputs. As we will describe below, the neural fields are composed of receptive field units that cover the corresponding domains of the input and output variables. Based on the activations in the neural fields, we let the system acquire sensorimotor mappings that should learn about correlations in the input and output variables. Tasks are given to the system as target values for the input variables, i.e. activation landscapes in the neural fields. The system should then employ its learned sensorimotor mappings to control the output variables (e.g. angular joint positions) such that it will bring the input variables to the target values.

A. Learning Redundant Sensorimotor Mappings

We want the robot to be able to learn skills autonomously from its sensorimotor experiences. To represent solution manifolds, our approach requires us to be able to retrieve many solutions upon a query from the learned mappings, instead of just a single solution. In the following we will discuss how this can be done using networks of *sigma-pi units*. Sigma-pi units have originally been studied in the 1980s (e.g. [17], [18]). More recently, Weber and Wermter have proposed to use sigma-pi units in conjunction with a SOM-like learning algorithm to learn invariances in input signals in an unsupervised fashion [19].

Our choice of using sigma-pi units is based on considerations of simplicity, as the sigma-pi weights very naturally implement the properties that we require. Thus, using sigma-pi networks allows us to more easily study the properties of our system. However, it should be noted that our approach does not depend on the sigma-pi network, but that the network is merely a building block in our system and could in principle also be replaced by other learning methods, such as the LWPR [2] or manifold learning [15], combined with a sampling-based readout method.

We begin with a formal definition of the network. Figure 2 is a schematic representation of the structure of the sigma-pi network. As described above, values of input and output variables are encoded in the system in a set of neural fields. Each of the fields is composed of receptive field units, implementing radial basis functions with centers $\mathbf{m}_i$. The response of the i-th input unit to the input vector $\mathbf{x}$ is computed as

$$u_i = g(\mathbf{x} - \mathbf{m}_i), \tag{1}$$

where g is a Gaussian function according to

$$g(\mathbf{d}) = \exp\left(-\nu \cdot \frac{\mathbf{d}^T\mathbf{d}}{2\sigma^2}\right). \tag{2}$$

We use a simple static layout for the centers $\mathbf{m}_i$ on a regular grid, such that the domain of the corresponding vector $\mathbf{x}$ is covered by the receptive field units in the neural field.

Using the receptive field units, a manifold can be represented by giving a high activation value to all those units of which the receptive fields coincide with parts of the actual manifold, and a low activation value to all other units.

We want to be able to query the network by presenting it with one or more input variables, and it should in response activate solution manifolds in the remaining input variables. For example, consider a network that has learned a mapping between arm angular positions and cartesian end-effector positions. If we query that network by presenting it with a target position in cartesian space, it should respond by showing us all possible angular positions that bring the end-effector to the target position. Also, the other way around, we would like to let the network provide us with the predicted end-effector position, if we query it by presenting an angular position vector.

This can be achieved by learning associations between the receptive field units, and upon a query activating all units that are associated with the activated input units. Sigma-pi units have been proposed as a model for the synaptic basis of associative learning in cerebral cortex [18]. The idea behind using the sigma-pi unit for associative learning is that it learns about co-activation of input units. Networks of sigma-pi units

belong to the class of "higher order" neural networks, as the simple additive units of linear feed-forward neural networks are extended by multiplicative connections. Thus, whereas in first-order neural networks the net input to a unit i is given by

$$net_i = \sum w_{ij} u_j, \tag{3}$$

for the sigma-pi units the activation function includes the multiplication of inputs,

$$net_s = \sum w_s u_{s_1} u_{s_2} \dots u_{s_n} \tag{4}$$

$$= \sum w_s \prod_{m=1}^{n} u_{s_m}, \tag{5}$$

where $s = (s_1, \dots, s_n)$ is a vector of indices. The introduction of these multiplicative connections allows units to *gate* one another [17]: If one unit has zero activation, then the activation of other units in the multiplicative connection have no effect.

In our implementation, we replaced the sum and product operators by the $\max$ and $\min$ operators, respectively, to avoid the need for normalization of network responses. Thus, the modified net input to a unit is

$$net_s = \max \left(w_s \cdot \min_{m=1}^{n} (u_{s_m}) \right). \tag{6}$$

We want to query the network by presenting it with an arbitrary activation pattern across the input neurons, and expect the network to respond by giving us solution manifolds for the missing inputs.

We can achieve this behavior of the network in the following way, using the gating property of the multiplicative connections. We formulate constraints on the values of the variables in the query by forcing the activation of those receptive field units that represent allowed values to be high, while forcing the activation of all other units in that neural field to be low.

The network stores information about co-activation of input units in the connection weights of the multiplicative connections. Let us refer to the activation level of the j-th unit in the i-th neural field as $u_{i,j}$. More generally speaking, if t_i denotes the number of neurons in the i-th neural field, then $S_i = \{1, 2, \dots, t_i\}$ is the set of possible values for the index j. Weights can be trained for all possible combinations of taking a single unit from each of the inputs. This can be formalized as taking the Cartesian product of the sets S_i,

$$S = S_1 \times S_2 \times \dots \times S_n \tag{7}$$

$$= \{s = (s_1, s_2, \dots, s_n) \mid s_i \in S_i\}, \tag{8}$$

which gives us combinations of indices for all possibilities to combine exactly one input unit from each input domain.

We want the network weights $w_s, s \in S$ to reflect the amount to which the input units determined by s tend to be co-active during training. Thus, if the neurons are always activated together, then the weight should adopt a high value, and if one or more units is never active along with the others during training, then the weight should be zero (i.e. there is no connection). If whenever one of the neurons was active, only half of the time also all the other neurons determined by s were also co-activated, then the connection weight should have a value around 0.5.

To achieve this, we can use a simple Hebbian learning rule. As training samples we use tuples of input vectors $(\mathbf{v}_1, \mathbf{v}_2, \dots, \mathbf{v}_n)$ to activate the input neurons according to Equation 1. The resulting activations $u_{i,j}$ are used to compute the network activation for all $s \in S$ (cf. Equation 5), which is then used to update the network weights as

$$\delta w_s = \lambda \cdot net_s \tag{9}$$

with learning rate λ.

B. Querying the Network

This section describes how the network can be used in a query by specifying a task description in terms of target sensorimotor input values to retrieve possible values for the missing variables.

We specify which variables to constrain by giving a set $Q \subseteq \{1, \dots, n\}$ of corresponding indices of neural fields. Given the notation of S in Equation 8 and the net input in Equation 5, we can formulate the network query as

$$\widetilde{u}_{i,j} = \sum_{r \in \{s \in S \mid s_j = i\}} w_r \prod_{m \in Q} u_{m,r_m}, \tag{10}$$

where $u_{i,j}$ is the input activation that was specifies for the query for all units in the neural fields given by Q, and $\widetilde{u}_{i,j}$ is the activation value that was retrieved from the network in response to the query. The activation of a unit is a sum over the activation of all elements r of the Cartesian product S in which that unit is itself a member, i.e. $s_j = i$. For all of these elements we compute the product of the activations of the members that were specified in the query, weighted by the connection weight w_r. The modified version that we used for our implementation is

$$\widetilde{u}_{i,j} = \max_{r \in \{s \in S \mid s_j = i\}} \left(w_r \cdot \min_{m \in Q} (u_{m,r_m}) \right). \tag{11}$$

There is an intuitive graphical interpretation for this query, see Figure 3. In the simple case where there are only two sets of input neurons, the network weights can be arranged in a planar grid, such that each knot in the grid represents the multiplication of two neurons (cf. also Figure 2). Thus, all knots in the grid together represent the Cartesian product of the sets of input neurons. If in a query we specify activations of input neurons in one domain and compute the product of the activations with the associated weights, we get the network activation, such that we have one value for each of the knots in the grid. In the next step, we accumulate for each unit all the values along the line of knots that are connected to that unit, which gives us the retrieved activation value of the unit as the response of the query. The same picture can easily be extended to the higher dimensional case, in which there are more than two inputs or where the inputs have more than a single dimension. Here, the network weights are arranged in a hyper-cube instead of a grid, and the line of knots corresponds to a slice through the hyper-cube.

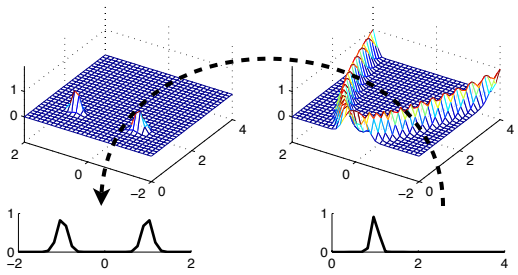


Fig. 3. Graphical interpretation for the network query using the example of a network that has learned the function $y = x^2$ in the interval $x \in [-2, 2]$. To retrieve all solutions for $x^2 = 1$, the population of input neurons in the y domain are activated for $y = 1$, see bottom right plot. This activation is fed into the network and after the multiplication with the synaptic weights gives the net activation that can be seen in the top left plot. In the readout, each input neuron accumulates the net activity from all points on the grid to which it is connected.

C. Decision Making, Control and Skill Combination

So far we have proposed to use sigma-pi networks to associatively learn multi-valued mappings between any number of input variables. We have further formulated a way to query the network for solution manifolds by presenting it with target values for the input variables. We now want to turn to the following questions:

- How can we pick a solution from the solution manifold to reach the target sensorimotor state?
- How can we use the network to generate motor commands?
- How can we combine several learned sensorimotor mappings in such a way, that we can still simply specify target values for the input variables, and have the system come up with a coherent solution?

1) Dynamic Decision Making: In principle the system could pick any solution from the manifold. However, when the target value for the input variable changes, the solution manifold changes with it. In most sensorimotor tasks, gradual changes in the target value also correspond to gradual changes in the associated variables. Thus, when for example the target position for the robot's end-effector position is moved gradually, also the solution manifold moves continuously, not abruptly. We would like the selected solution to dynamically follow the solution manifold to avoid unnecessary large movements. Also, sensorimotor inputs are usually noisy, thus the selection mechanism should be robust against noise to a sufficient degree.

To avoid having to implement an elaborate heuristic to manage the decision making process of picking a new solution, we propose to use *dynamic neural fields* as competition mechanisms. Dynamic neural fields are neural fields of laterally connected neurons, where connections are excitatory for short distances and inhibitory for longer distances between neurons (comparable to a "mexican hat" function). The field dynamics is based on a dynamic update rule, which was first studied by Amari [20] and is therefore also referred to as the "Amari dynamics". Amari investigated the properties of such fields with respect to dynamic pattern formation and stability. One simple, but for us important case is the formation of a single peak solution in the neural field, such that the inhibitory connections prevent the formation of any other activation peaks. The peak remains stable as long as it is provided with input, is robust against noise as the Amari dynamics implements a low-pass filtering of the input signal, and can also follow the input signal when the location of strong input is shifted gradually. Thus, the Amari dynamics nicely complies with the requirements for the decision making process that we have stated above.

In our implementation of the dynamic neural field we follow that of Erlhagen and Schöner [21] as a discrete time implementation, which was also adapted by Toussaint [22]. The dynamic equation for the activation u_i of the neurons in the neural field is

$$\tau \dot{u}_i = -u_i + h + S_i + \sum_{j=1}^{m} w_{ij} f(u_j), \tag{12}$$

where τ is the time constant of the dynamics, h is a parameter for global self-inhibition and specifies the resting level, S_i is the input to the i-th unit in the neural field, w_{ij} is a distance weighting that effectively implements the excitation and inhibition property, and f is a non-linearity.

In our implementation, we used a "ramp" function for the non-linearity f as

$$f(u) = \begin{cases} 0 & u \leq 0 \\ u & 0 < u < 1 \\ 1 & u \geq 1 \end{cases} \tag{13}$$

and for the distance weighting w_{ij} a Gaussian function shifted by a constant w_I to achieve inhibition across the whole neural field,

$$w_{ij} = w_E \cdot \exp\left(-\frac{d(i,j)^2}{2\sigma_E^2}\right) - w_I. \tag{14}$$

Here, $d(i, j)$ is the Euclidean distance between the units i and j in the neural field, σ_E determines the excitatory range and w_E determines the strength of the excitation.

2) Local Gradient Descend: The output of the dynamic neural field is a single-peak activation lying on the solution manifold. We want to generate a motor command from this activation through a population readout mechanism as a linear combination of the centers $\mathbf{m}_i$. However directly using the output activation of the dynamic neural field units as weights for the centers does not give us a precise motor command to reach the target value. The output of the dynamic neural field only results in a pre-selection of units that should be used in the population readout, but the activation levels of the dynamic neural field units do not reflect the actual coefficients for the linear combination. We therefore use a gradient descend method to control the target vector.

As an example let us again consider the learning of the kinematics of a robot arm, where we encode the controllable angular positions θ and the corresponding Cartesian position of the end-effector $\mathbf{x}$ by two neural fields with activations

$u_{\theta,j}$ and $u_{\mathbf{x},j}$ respectively. To retrieve a solution to the inverse kinematics, the neurons $u_{\mathbf{x},j}$ for the target position $\mathbf{x}^*$ of the end-effector are activated according to Equation 1 as

$$\mathbf{u}_{\mathbf{x}}(\mathbf{x}^*) = (u_{\mathbf{x},1}, \ldots, u_{\mathbf{x},t_{\mathbf{x}}}), \tag{15}$$

which is a single-peak activation. This is then used for the query as described in Section III-B to retrieve a solution manifold encoded in the activations

$$\widetilde{\mathbf{u}}_{\theta} = (\widetilde{u}_{\theta,1}, \ldots, \widetilde{u}_{\theta,t_\theta}), \tag{16}$$

which is the input to the dynamic neural field. The output of the dynamic neural field is a localized activation pattern of the units in the domain of θ.

We readout a target posture from the output of the dynamic neural field as a sum of the centers $\mathbf{m}_j$ weighted by the activations of the corresponding units,

$$\widetilde{\theta} = \sum_{j=1}^{t_\theta} \widetilde{u}_{\theta,j} \cdot \mathbf{m}_j \tag{17}$$

The resulting end-effector position lies close, but is not necessarily equal to the target position. However, we can compute a gradient for the control variable θ by using the sigma-pi weights. For this we first compute the difference in activation between the target position $\mathbf{x}^*$ and the actually observed end-effector position $\mathbf{x}$ as

$$\delta\mathbf{u}_{\mathbf{x}} = \mathbf{u}_{\mathbf{x}}(\mathbf{x}^*) - \mathbf{u}_{\mathbf{x}}(\mathbf{x}) \tag{18}$$

and then use it along with the output of the dynamic neural field in another query of the network according to equation 11. The query gives us a change $\delta\widetilde{\mathbf{u}}_{\theta}$ in the activation of the units, which we can use in an update step. Iterating this process effectively implements a gradient descend that we can use to control the motors and bring the end-effector to the target location.

3) Skill Combination: In Section II we have already described a few properties that we want the solution selection process to have, when we combine sensorimotor mappings to obtain multiple solution manifolds in the domains of input variables:

- The solution should lie on at least one solution manifold.
- If two or more manifolds intersect, then it should lie on an intersection point of as many manifolds as possible.
- We would like to be able to assign priorities to tasks.

The competition dynamics of the dynamic neural fields can be exploited to achieve these criteria: To combine solution manifolds, we can use a superposition of the network responses,

$$\widetilde{\mathbf{u}}_i = \sum_{k \in \{l | i \in Q_l\}} a_k \cdot \sigma\left(\widetilde{\mathbf{u}}_i^k\right). \tag{19}$$

Here, Q_k is a set of indices that defines for the k-th sensorymotor mapping to which neural fields it is connected, $\widetilde{\mathbf{u}}_i^k$ is the response of the network for the i-th field and σ is a nonlinearity to transform the gradual network responses to a plateau of uniform activation. The factors a_k are normalized so that their sum is bound to be below or equal to 1.

The resulting activation of the neurons in the input maps is used as the input for the dynamic neural field. The competition dynamics of the field then functions as the desired decision making mechanism as follows.

In cases where the robot only has one task, either because it only has acquired a single mapping or because only one network has neurons with activations for target sensorimotor values, the field behavior reduces to that described in the previous section. The properties of the field dynamics guarantee that an activation pattern resembling a peak can only develop where there is activation present in the input. Thus, in this simple case, it is obvious that we will always end up with a valid solution that lies on a manifold. If there is no input at all, then the field output goes to the resting level for all the neurons in the field. This can easily be detected so that no command for the motors is generated at all.

The same is also true for the combined network responses. When there are more than one network responses, then the additive combination of the activation levels will give us an activity landscape, where neurons that are solutions for more than one task are activated more than those that are only solutions for a single task. The dynamics of the neural field will always favor the strongest input, as soon as it surmounts the input activation at a current peak by a given amount.

The last property of the decision making process, being able to assign priorities to tasks, can easily be achieved by controlling the factors a_k in Equation 19. If there exists a point where all solution manifolds intersect, this point will have the highest activation value in the neural field. If however there is no such point, the manifold with the higher priority will have a higher activation value in the superposition and will thus be selected by the field dynamics.

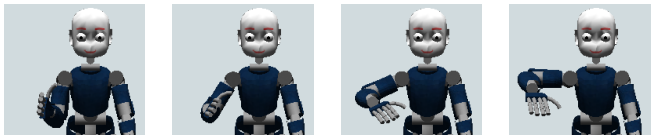
IV. Results

To test the proposed approach we performed two experiments of kinematics learning, using a simulation of the humanoid robot iCub [23].

A. Experiment 1

At first we trained a sigma-pi network to learn the forward and inverse kinematic mappings, which are transformations between a vector of arm angular values θ and the Cartesian end-effector position $\mathbf{x}$. We used the robot's right shoulder and elbow joints, thus having a four-dimensional angular posture space. We sampled this space on a regular grid by placing in each dimension 10 equally distributed grid points, resulting in $T = 10000$ configurations. We let the robot move its motors into the joint configurations and recorded a pair of vectors $(\theta, \mathbf{x})$ for each of them.

To represent the Cartesian input vector of end-effector coordinates, we used a three-dimensional field of $42 \times 36 \times 41$ input neurons. We initialized the centers of their receptive fields on a regular grid of positions that were equally distributed and covered the whole workspace of the robot. Similarly, we used a four-dimensional field of $10 \times 10 \times 10 \times 10$ input



(a) (b) (c) (d)

Fig. 4. Several postures that were retrieved by the system for the task of bringing the robot's end-effector to one point while constraining one of the shoulder joints. The position of iCub's end-effector is at the center of its palm, which is kept at the target point in the queries shown in (a)-(c) In each new query, the constrained postural value for the shoulder joint is increased, until in the final query that is shown in (d) iCub cannot reach the target point with the constrained posture, because it has reached its joint limits.

neurons with receptive fields arranged on a grid that covered the domain of the arm angular position vector.

We used a sigma-pi network to learn associations between joint configuration vectors and Cartesian end-effector coordinates. For each training sample $(\theta^t, \mathbf{x}^t)$, we first computed the activation of the input units according to their receptive fields (cf. Equation 1) and then computed the activation of the product units net_s^t according to Equation 6 and updated the weights using the $\max$ operator as

$$w_s^{t+1} = \max\left(w_s^t,\ net_s^{t+1}\right), \tag{20}$$

where w_s^t is the resulting weight after having processed the t-th training sample. Note that we omitted the learning rate parameter λ implicitly setting it to 1 to implement one-shot learning.

We initialized a dynamic neural field to match the configuration of the map of input neurons for the arm angular position vector (i.e. $10 \times 10 \times 10 \times 10$ equidistant neurons). Our choice of parameters for the neural field dynamics was $\tau = 15$, $h = -0.1$, $w_E = 1.26$, $w_I = 0.004$ and $\sigma = 0.6$, where the distance between two neighboring neurons i and j in the dynamic neural field was $d(i, j) = 1$.

To test the decision making process and the proposed method to combine multiple learned mappings, we used a second network that should learn the mapping

$$f(\theta) = \theta_2, \tag{21}$$

which simply stores for every posture θ only the angular position of the second shoulder joint. Thus when querying the learned mapping for candidate postures that use a given angular position for the second shoulder joint, the network response will be an activation representing the axis-parallel hyper plane for all postures including θ_2. We used a one-dimensional input map of 10 neurons with receptive fields covering the possible angular positions for the joint.

We tested the method by giving a position in front of iCub as target and querying the first network for solutions. We simultaneously used the second network to tell the system that it should drive the second shoulder joint into different target positions. We also set the priority of the second network to be higher than that of the first network. Thus, we expect the system to reach for the target point while using a posture that satisfies the constraint for the shoulder joint. Figure 4 shows example postures that were generated by the system. The system was able to generate solutions that satisfied both of the goals that it was given, i.e. it successfully moved the end-effector to the target point while staying in the specified angular position with the second shoulder joint.

B. Experiment 2

In the second experiment, we replaced the second input by one that should allow the robot to avoid joint limits. For this, we introduced a virtual proprioceptive sensor that should give the robot feedback about the "quality" of a posture, where postures that had one or more joints in their limits produced a value of 0, whereas postures in the center of the angular space should have values of 1. We used a modified cosine to implement this function.

We then retrained the second network with input from this virtual sensor, thus associating the sensor output to each arm posture. In combination with the network that has learned the robot's kinematics, the two networks perform as a reaching controller that allows to reach for all points in the robot's workspace while avoiding joint limits whenever possible. By setting the priority of the target point higher than that of the joint limit avoidance, we can ensure that iCub actually reaches for all points, even if this is only possible with a configuration that has one or more joints in their limits.

Figure IV-B shows the result of querying the system for a posture to reach for a target position in iCub's workspace, on the one hand only using the network that has learned the robot's kinematics, and on the other hand also using the join limit avoidance network.

C. A Note on Computational Performance

As described in Section IV-A, the first sigma-pi network that was trained for the robot's kinematics has as inputs a four-dimensional neural field representing the angular motor space and a three-dimensional neural field representing the position of the end-effector. Therefore the synaptic weight matrix of the network is seven-dimensional, which results in a huge amount of possible synaptic connections. However, since we use a simple Hebbian learning rule for the training of the network and do not have to initialize the network weights randomly but start with an "empty" network, the number of weights above zero that the trained network stores remains relatively small. Of the $10 \times 10 \times 10 \times 10 \times 42 \times 36 \times 41$ possible weights, only 1.24% have a value above zero in our trained network. Implementing the sigma-pi networks using a sparse matrix representation therefore allows to dramatically increase the computational performance.

V. Conclusion

In this work we have proposed a novel approach to combine multiple learned sensorimotor mappings by making use of the robots redundancies. In contrast to approaches to sensorimotor learning that discard redundant solutions and only store one representative solution for each target value, we have introduced a learning mechanism that is capable of learning many-to-many mappings, which allows us to simultaneously retrieve

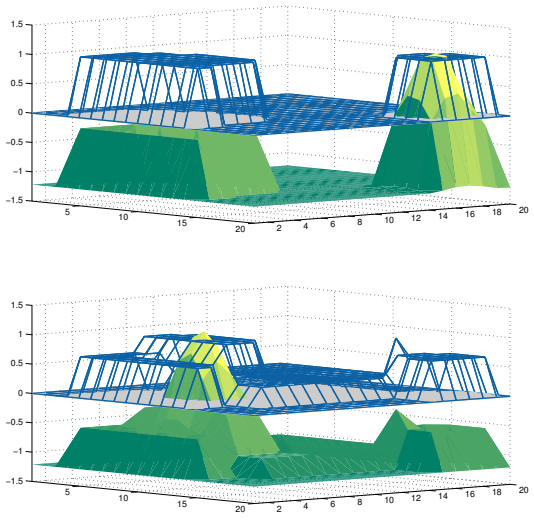


Fig. 5. The two plots show the activation that was generated by the system when queried for a solution to reach a target point in space. The activation of the neural field representing the robot's joint configuration is shown, representatively only for two dimensions. To give a more detailed view, we used a neural field with 20 neurons in each dimension to produce the plot. Blue lines show the activation level of neurons in the neural field, which is also the input to the dynamic neural field. The activation of the neurons in the dynamic neural field can be seen as green activation landscapes. The grey plane indicates the zero level. When using only the network that has learned the robots kinematics (upper plot), the system generates a solution manifold that is split into two regions, as can be seen by the two plateaus in the activation of the neural field. The field dynamics then produces a localized peak that represents the solution. In this case, a solution close to the joint limits was selected. When using the combined response from both networks (lower plot), postures that are away from joint limits are favored. This can be seen as the superposition of the two network responses produces an activation landscapes with three activation levels: One (the lowest) for all solutions that are away from joint limits, one for solutions that bring the end-effector to the target location, and one (the highest) that combines both tasks, i.e. represents the intersection of the two solution manifolds. The field dynamics consequently generates a peak at the highest plateau level.

all solutions to reach a target value. We have shown that the solutions from several learned mappings can easily be combined, and our decision making process is able to dynamically select the favorable solution online. When considering this in the context of autonomous robots, which should be able to extend their repertoire of skills by learning new ones through exploration, this presents a straightforward way of integrating multiple learned sensorimotor mappings, because there is no need for an arbitration mechanism that assigns motor resources to individual skills. Our method simply combines the different solution manifolds stemming from the queries of multiple networks into a combined representation. Priorities can easily be assigned to each task, so that it can for example be ensured that the main task is fulfilled and other tasks are only pursued in the null-space of that main task.

Acknowledgment

Nikolas J. Hemion gratefully acknowledges the financial support from Honda Research Institute Europe.

References

[1] M. I. Jordan and D. E. Rumelhart, "Forward models: Supervised learning with a distal teacher," *Cognitive Science*, vol. 16, no. 3, pp. 307–354, 1992.

[2] A. D'Souza, S. Vijayakumar, and S. Schaal, "Learning inverse kinematics," in *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, 2001, pp. 298–303.

[3] J. Peters and D. Nguyen-Tuong, "Incremental online sparsification for model learning in real-time robot control," *Neurocomputing*, vol. 74, no. 11, pp. 1859–1867, 2011.

[4] M. Hoffmann, H. Marques, A. Arieta, H. Sumioka, M. Lungarella, and R. Pfeifer, "Body schema in robotics: A review," *IEEE Transactions on Autonomous Mental Development*, vol. 2, no. 4, pp. 304–324, 2010.

[5] L. Montesano, M. Lopes, A. Bernardino, and J. Santos-Victor, "Learning object affordances: From Sensory–Motor coordination to imitation," *IEEE Transactions on Robotics*, vol. 24, no. 1, pp. 15–26, 2008.

[6] J. Peters and S. Schaal, "Reinforcement learning of motor skills with policy gradients," *Neural Networks*, vol. 21, no. 4, pp. 682–697, 2008.

[7] K. Mülling, J. Kober, and J. Peters, "A biomimetic approach to robot table tennis," *Adaptive Behavior*, vol. 19, no. 5, pp. 359–376, 2011.

[8] P. Kormushev, S. Calinon, R. Saegusa, and G. Metta, "Learning the skill of archery by a humanoid robot iCub," in *10th IEEE-RAS International Conference on Humanoid Robots*. IEEE, 2010, pp. 417–423.

[9] M. Gienger, M. Toussaint, and C. Goerick, "Whole-body motion planning building blocks for intelligent systems," in *Motion Planning for Humanoid Robots*, K. Harada, E. Yoshida, and K. Yokoi, Eds. Springer, 2010.

[10] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*. Cambridge, MA: MIT Press, 1998.

[11] K. Doya, "Reinforcement learning in continuous time and space," *Neural Computation*, vol. 12, no. 1, pp. 219–245, 2000.

[12] A. Ligeois, "Automatic supervisory control of the configuration and behavior of multibody mechanisms," *IEEE Transactions on Systems, Man and Cybernetics*, vol. 7, no. 12, pp. 868–871, 1977.

[13] M. Gienger, H. Janssen, and C. Goerick, "Task-oriented whole body motion for humanoid robots," in *5th IEEE-RAS International Conference on Humanoid Robots*. IEEE, 2005, pp. 238–244.

[14] M. Rolf, J. Steil, and M. Gienger, "Goal babbling permits direct learning of inverse kinematics," *IEEE Transactions on Autonomous Mental Development*, vol. 2, no. 3, pp. 216–229, 2010.

[15] M. Lopes and B. Damas, "A learning framework for generic sensory-motor maps," in *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*. San Diego, USA: IEEE, 2007, pp. 1533–1538.

[16] R. F. Reinhart and J. J. Steil, "Neural learning and dynamical selection of redundant solutions for inverse kinematic control," in *11th IEEE-RAS International Conference on Humanoid Robots*. IEEE, 2011, pp. 564–569.

[17] D. E. Rumelhart and J. L. Mcclelland, *Parallel Distributed Processing: Explorations in the Microstructure of Cognition. Vol. 1: Foundations*. MIT Press, 1986.

[18] B. W. Mel and C. Koch, "Sigma-Pi learning: On radial basis functions and cortical associative learning," *Advances in Neural Information Processing Systems*, vol. 2, p. 474481, 1989.

[19] C. Weber and S. Wermter, "A self-organizing map of sigma-pi units," *Neurocomputing*, vol. 70, no. 13-15, pp. 2552–2560, 2007.

[20] S.-i. Amari, "Dynamics of pattern formation in lateral-inhibition type neural fields," *Biological Cybernetics*, vol. 27, no. 2, pp. 77–87, 1977.

[21] W. Erlhagen and G. Schöner, "Dynamic field theory of movement preparation," *Psychological Review*, vol. 109, no. 3, pp. 545–572, 2002.

[22] M. Toussaint, "A sensorimotor map: Modulating lateral interactions for anticipation and planning," *Neural Computation*, vol. 18, no. 5, pp. 1132–1155, 2006.

[23] V. Tikhanoff, A. Cangelosi, P. Fitzpatrick, G. Metta, L. Natale, and F. Nori, "An open-source simulator for cognitive robotics research: the prototype of the iCub humanoid robot simulator," in *Proceedings of the 8th Workshop on Performance Metrics for Intelligent Systems*. ACM, 2008, p. 5761.