Combination of Movement Primitives for Robotics

Kombination von Movement Primitives in der Robotik

Master-Thesis von Johannes Ringwald aus Mannheim

July 21, 2014

TECHNISCHE UNIVERSITÄT DARMSTADT

Fachbereich Informatik
Institut Autonomous Systems

Combination of Movement Primitives for Robotics
Kombination von Movement Primitives in der Robotik

Vorgelegte Master-Thesis von Johannes Ringwald aus Mannheim

1. Gutachten: Prof. Dr. Jan Peters
2. Gutachten: Dr. Gerhard Neumann und Alexandros Paraschos

Tag der Einreichung:

Erklärung zur Abschlussarbeit gemäß § 22 Abs. 7 APB der TU Darmstadt

Hiermit versichere ich, Johannes Ringwald, die vorliegende Master-Thesis ohne Hilfe Dritter und nur mit den angegebenen Quellen und Hilfsmitteln angefertigt zu haben. Alle Stellen, die Quellen entnommen wurden, sind als solche kenntlich gemacht worden. Diese Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen.
In der abgegebenen Thesis stimmen die schriftliche und elektronische Fassung überein.

Datum: Unterschrift:

Thesis Statement pursuant to § 22 paragraph 7 of APB TU Darmstadt

I herewith formally declare that I have written the submitted thesis independently. I did not use any outside support except for the quoted literature and other sources mentioned in the paper. I clearly marked and separately listed all of the literature and all of the other sources which I employed when producing this academic work, either literally or in content. This thesis has not been handed in or published before in the same or similar form.
In the submitted thesis the written copies and the electronic version are identical in content.

Date: Signature:

Abstract

Probabilistic Movement Primitives (ProMPs) are a promising approach to represent movements. ProMPs can capture the variance of the movements and can be trained by immitation learning. Additionally, they enabled new operators on the primitives, such as modulation of the movement, as well as combination and blending between primitives. We modify the probabilistic structure to generate movement primitives dependent on a certain task (TaskMP), like reaching a specific position or orientation. In this work, we focus on the combination of these task dependent probabilistic movement primitives.
In the first step toward our goal, we focus on the modelling of TaskMPs by Gaussian mixture models with linear experts. This approach is able to capture non-linear dependencies between a task and the movement representation, in the primitive parameters. The proposed system were trained by an Expectation Maximization (EM) algorithm.
We analyse the combination of two individually trained TaskMPs for two different tasks. The combination of primitives enables us to solve the combination of two tasks by reusing trained primitives. We demonstrate that if the primitives A and B are able to solve two separate tasks, it is possible to solve all combinations of these single tasks by the combination of the two primitives A and B. We model the combination of primitives by a mixture of experts model.
To further improve the results of this combination, an learning approach is derived, which trains the model-parameters of a mixture combination at once. This product of a mixture of experts is able to solve a combination of tasks with less mixture components and less training data than a single TaskMP. We compare the result with the previous combination approach and a single task dependent primitive.

Abstract

Probabilistic Movement Primitives (ProMPs) stellen einen vielversprechenden Ansatz zur Repräsentation von Bewegungen dar. ProMPs können durch Immitations-Lernen trainiert werden und sind fähig die Varianz von Bewegungen abzubilden. Desweiteren ermöglichen sie die Anwendungen von neuen Operatoren wie die Modulation einer Bewegung oder die Kombination und das Blenden zwischen Primitiven. Der Strukturansatz über Wahrscheinlichkeitsverteilungen wird so modifiziert, dass die Generierung von Bewegungs-Primitiven, die von bestimmten Aufgaben bzw. Parametern abhängen (TaskMPs), ermöglicht wird. Beispiele hierfür sind das Einnehmen einer bestimmten Position oder Orientierung. Diese Arbeit legt den Schwerpunkt auf die Kombination solcher aufgabenorientierten Bewegungs-Primitive.
Der erste Schritt beinhaltet die Modellierung von TaskMPs, über Gaußsche Mischverteilungsmodelle mit linearen Experten. Durch diesen Ansatz ist es möglich, nicht lineare Abhängigkeiten zwischen Ziel und Bewegungsrepräsentation bzw. den Primitive-Parametern mit ein zu beziehen.
Auf Basis dieser Schritte wird die Kombination zweier unabhängig (für zwei unterschiedliche Ziele) trainierten TaskMPs analysiert. Eine solche Kombination ermöglicht es, eine Zusammenstellung aus zwei Zielen durch wiederholten Einsatz bereits trainierter Primitive zu lösen. Sind die Primitve A und B fähig zwei separate Aufgaben zu lösen, ist es ebenfalls möglich, alle Kombinationen dieser einzelnen Ziele mittels einer Kombination der Primitive A und B zu lösen. Diese Kombination aus zwei Bewegungs-Primitiven wird durch ein Mischverteilungs-Experten-Modell abgebildet.
Um diese Ergebnisse weiter zu verbessern, wurde ein Trainingsansatz aufgebaut, welcher alle Parameter einer Mischmodell-Kombination gleichzeitig lernt. Dieses Produkt aus Misch-Experten-Modellen ist fähig, eine Ziel-Kombination mit weniger Modell-Komponenten und dadurch weniger Trainings-Daten zu lösen, als es ein einzelnes TaskMP im Stande wäre. Die Ergebnisse werden mit dem vorherigen Kombinations-Ansatz und einzelnen TaskMPs verglichen.

Contents

1 Introduction **4**

2 Preliminaries and related Works **5**
2.1 Movement Primitives . . . 5
2.1.1 Biological Background . . . 5
2.1.2 Forms of Movement Primitives and Policy Representations . . . 6
2.1.3 Combination of MPs . . . 6
2.2 Task Dependency and Decomposition . . . 7
2.3 Probabilistic Movement Primitives . . . 7
2.3.1 Principle Form . . . 8
2.3.2 New Operators of ProMPs . . . 9

3 Task-Dependent Probabilistic Movement Primitives **12**
3.1 Mixture of Linear Experts . . . 12
3.2 Training of Probabilistic Task Dependent Movement Primtives . . . 13
3.3 Toy-Task-Example . . . 14
3.4 Training-Example . . . 15
3.5 Test of Probabilistic Task Dependent Movement Primtives . . . 15

4 Product of Task Dependent Movement Primitives **17**
4.1 Concept . . . 17
4.2 Evaluation without Gating-Network . . . 18
4.3 Gating-Network . . . 18

5 Expert Learning of Mixture Product **20**
5.1 Model of the Mixture Product . . . 20
5.2 Training of Mixture Product . . . 20

6 Experiments **23**
6.1 Single TaskMP . . . 23
6.2 Combination of TaskMPs . . . 25
6.3 Expert-Learning . . . 27

7 Conclusion and Future Work **29**

A Appendix: **31**

1 Introduction

In real world scenarios, it is necessary to solve many tasks simultaneously, e.g. reaching a point and rotating the hand to the same time or throwing a ball with certain direction and range.
Relearn every task-combination is impractical due to its diversity and complexity. A more promising approach to handle this problem is to combine several skills or movements to solve a combination of tasks. Hence, less elemental movements are needed to solve more complex problems.
An elemental movement is a basic movement representation from which complex movements and skills can be built up. Similar to bricks of a building, the overall behaviour is constructed by combining and sequencing these basic movement modules.
The overall idea for this type of skill combination is to build up a modular control architecture which selects, combines and processes elemental movements from a provided library, to solve the task in hand. The advantage of such a control architecture is the re-usability of previously learned primitives. A new task can be solved by a combination of given movement primitives, instead of re-learning a new primitive (according to the observed task). This simplifies the generation of complex behaviours, due to a reduced amount of needed modules and parameters.

An example for the described movement construction could be playing table tennis by a robot. During such a game many different elemental skills are required. Shooting the ball on a certain place of the table, or with a specific force, receive the ball and shoot it back to the opponent player or serving with a certain spin could be examples of elemental behaviours. The successful execution of the game will depend on the selection, sequencing, adaptation and combination of these elemental movements by a control architecture. Selecting and sequencing primitives would result in the crucial step of playing table tennis, whereas adapting and combining the given primitives would adjust the desired behaviours and increase the performance of the player, to e.g. shooting the ball to a certain place of the plate with a specific power and a certain spin.

This skill compilation by a control architecture requires a representation of movements dependent on a task, which leads to the structure of movement primitives [1]. Similar to the described example we aim to combine several movement-representations and to continuously switch or blend between different movements. The adaptation of MPs to new situations, like speed-tuning or additional via points is also a desirable property.

Probabilistic Movement Primitives (ProMPs) [2] can cope up with most of these requirements. The probabilistic approach permits the learning of movement primitives, which can depend on a specific task (Task-Dependent Movement Primitives), similarly to other approaches like dynamical movement primitives [3]. The probabilistic approach enables to capture the variability of human movement demonstrations, which can be used to represent a optimal behaviour in a stochastic system as described in Paraschos et al. [2]. With ProMPs it is possible to combine and blend between several primitives, as well as to modulate each primitive to reach a desired state. This adaptation can be implemented by conditioning the probability distribution of a movement on an additional parameter like a new final position or an additional via-position or velocity. The combination is performed by a product of several probability distributions which extracts the common probability areas to form a resulting distribution. A Gaussian approach ensures an analytical solution for this kind of elemental combinations.

This work concentrates on the development of Task-Dependent Probabilistic Movement-Primitives (TaskMPs), their combination and specifically the modeling and learning of TaskMPs to achieve a complex behaviour. We show how independent trained primitives can be combined to solve a task. We evaluate different schemes for modelling and learning a combination of independently trained primitives. We propose an approach which solves the combination of several sub-tasks by using already trained primitives and as a result a small number of task dependent primitives would be needed to generate complex behaviours. Our approach is effectively reducing the number of parameters to learn and generalize new tasks by reusing prior information encoded in the trained primitives.

2 Preliminaries and related Works

According to the focus on movement primitives and their combination, a basic overview about the biological background, the different forms of movement primitives and an example of MP combination is given in this section.

2.1 Movement Primitives

Movement Primitives represent a single movement as a closed modul (i.e. by a group of specified parameters). This representation could be the trajectory of joint-angles or the position, velocity, acceleration of a robot end-effector. The way how these movements are represented and stored can be different and characterizes the specific form of a movement primitive (MP).
MPs provide the required modularity, which is needed to build up the previously described control architecture. There are also neuro-scientific insights, which confirm the idea of movement primitives used in a modular control architecture. The following chapter considers these biological insights.

2.1.1 Biological Background

The neuro-scientific results of Bizzi et al. [4] or Avella et al. [5]–[7] indicates motor control principles, which are based on the combination of different modules. A single module represents a discrete activation pattern for a certain muscle and is located within the spinal cord [4]. The time-varying activation and combination of these patterns causes a specific movement or behaviour like walking, jumping or swimming (see Figure 1).
This combination of many pre-defined patterns, reduces the number of parameters which need to be adapted according to a certain situation. These variables are the pattern selection, start-point, as well as the activation-amplitude [4].
Figure 1 shows the time dependent activation (EMG pattern) of thirteen muscles for different behaviours of a frog. The muscle synergies are the re-used patterns, from which the desired muscle-activations are built of. The considered behaviours are jumping, walking and swimming. As can be seen from Figure 1, the behaviours uses different as well as similar synergies to form the activation patterns. The results show a sharing of certain patterns across the different behaviours, which reduces the complexity of movement control.

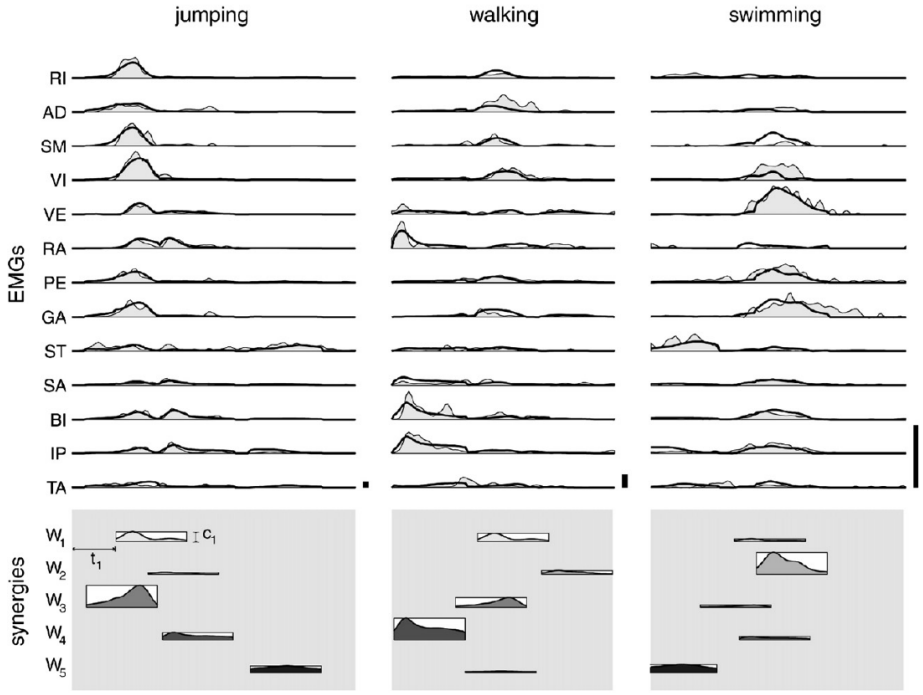

Figure 1: EMG signals and muscle synergies for a jumping, walking and swimming frog [4]. The different movements, are composed of recurrent EMG patterns and synergies, which control the different muscle contractions. This concept reduces the number of dimensions of the control-problem. The different synergies mirror the re-used activation patterns.

According to these results, vertebrates seems to use some kind of modularity for limb control in form of adapted and combined muscle activation patterns [4]. This reinforces the idea of a modular control architecture and gives a biological background to this kind of systems.

2.1.2 Forms of Movement Primitives and Policy Representations

In the field of robotics a movement primitive results in a policy representation. A policy π maps given parameters (e.g. joint-angles) to specific actions (e.g. motor commands) of a robot. Policy representations can be grouped into time-dependent and time-independent representations [1]. The main-difference relies in the complexity of the apply-able policy structures. Time-independent policies uses the same structure for all time-points, whereas the structure of time-dependent policies may differ according to different time-points. This flexibility enables the usage of representations with lower complexity. Time-independent representations are linear policies and radial basis functions. Splines and Dynamical Movement Primitives are examples of time-dependent policies.

The first approach of **linear policies** [1] uses a linear relationship between the parameters-set $\boldsymbol{\theta}$ and a corresponding vector of basis-functions $\boldsymbol{\phi}(x)$ to $\pi_\theta(x) = \boldsymbol{\theta}^\top \boldsymbol{\phi}(x)$, where x is the current state (e.g. joint-angles). The parameter vector $\boldsymbol{\theta}$ is the only trained variable.
Radial basis function networks [1] are similar to the idea of linear policies, except the amount of training-parameters. Again the policy contains a linear dependency between weights $\boldsymbol{w}^\top$ and the basis-functions (Eq. 1).

$$\pi_\theta(x) = \boldsymbol{w}^\top \boldsymbol{\phi}(x), \qquad \boldsymbol{\phi}(x) = \exp\left(-\frac{1}{2}(x-\mu_i)^\top D_i (x-\mu_i)\right), \qquad \boldsymbol{D}_i = \mathrm{diag}(d_i). \tag{1}$$

The training parameters contains $\boldsymbol{w}$ as well as the basis-function variables μ_i and d_i (beside $w^\top$). The larger amount of training parameters makes it more difficult to train an appropriate RBF-Network. Both methods, linear policy and radial basis function networks are time-independent approaches.
A time-dependent version are **splines** [8]. A trajectory is represented by a polynomial $q(t,w) = \sum_i w_i t^i$. The trajectory can be followed by a separate controller, which leads to the principle of trajectory based representations. The parameters of the primitive directly (and only) defines the observed trajectory. No motor-commands are provided by a policy. An external controler is needed to execute the given trajectory.
Another time-dependent approach are Dynamical Movement Primitives (DMP).

Dynamical Movement Primitives are based on the combination of a linear spring damper system and a non-linear forcing term f (Eq. 2, 3) [1], [3].

$$\ddot{y} = \tau^2 \alpha_y (\beta_y (g-y)\dot{y}/\tau) + \tau^2 f(z), \qquad \dot{z} = -\tau \alpha_z z, \tag{2}$$

$$f(z) = \frac{\sum_{i=1}^K \phi_i(z) w_i}{\sum_{i=1}^K \phi_i(z)} z, \qquad \phi_i(z) = \exp\left(-\frac{1}{2\sigma_i^2}(z-c_i)^2\right). \tag{3}$$

The parameters α_y and β_y define the spring damper system. The term $\ddot{y}$ represents the desired acceleration e.g. of a joint. The position y and velocity $\dot{y}$ can be extracted by integrating $\ddot{y}$. The goal parameter g is determining the final position of the trajectory. The time-constant τ defines in combination with the phase variable $\dot{z}$ (Eq. 2) the execution velocity of the trajectory.
The forcing function f incorporates the desired behaviour of the movement primitive (Eq. 3). It contains the basis-functions $\phi_i(z)$, the shape-parameter w_i and the phase-variable z. To ensure stability, f is disappearing for $t \to \infty$, due to the time-dependent decrease of z.
To learn a DMP, a given demonstration of the desired movement ($\ddot{y}$, $\dot{y}$, y) is used to calculate the corresponding forcing function values f_t by Eq. 3. Given f_t, the shape parameters w_i can be calculated by linear regression and is determining the progress of $\ddot{y}$.

Dynamical Movement Primitives are common in robotics, due to their favourable properties like a small training-effort, scaling of execution speed or the possibility to couple several degrees of freedom. According to its relevance, several extensions of DMPs exists, like incorporating specific velocity-points for a table-tennis task [9], coupling the DMP-Learning to an external variable [10] or enable obstacle avoidance [11].

2.1.3 Combination of MPs

The term "combination" can include different forms of movement processing, like combining movements in a sequential way or solving a problem by combining several primitives in a simultaneously way. For example the work of [12] includes aspects of both ways by building a control architecture, which chooses and adapts simple pre-defined primitives (discrete/rhythmic) to model an overall behaviour like crawling.

An example of sequential MP combinations provides the work of Daniel et. al [13]. The goal was to sequence several movement primitives and to improve the parameters of these primitives at the same time by one algorithm.

The used learning-method is a modification of the Hierarchical Relative Entropy Policy Search (HiREPS), which itself stems from the Episodic Relative Entropy Policy Search algorithm (REPS).
The applied method uses the policy structure from HiREPS - containing a Gaussian gating $\pi(o|s)$ and linear Gaussian parameter policies $\pi(w|s,o)$. Such an approach allows choosing one movement primitive from many by

$$\pi(w|s) = \sum_o \pi(o|s)\pi(w|s,o). \tag{4}$$

Due to this structure, the overall policy $\pi(w|s)$ contains several primitive options, which can be selected. The primitive parameters are represented by w, o defines the corresponding primitive selection and s the current state or task. To incorporate the possibility of primitive sequencing, the learning process is described as a time-indexed Markov Decision Process, with a discrete number of time-steps per episode and state transitions s', corresponding probability models and certain transition constraints. These constraints connect the different policies to each other and allow for the improvement of the global reward of a task (taking several sequenced MP into account).
For the specific implementation and evaluation of the described approach, dynamical movement primitives have been used.
This evaluation, resulting in a comparison of the applied method with the episodic HiREPS - which didn't contain any movement primitive sequencing - showed improvements in training-speed and the separation of the complex problem into smaller sub-problems.

Another approach to sequence movement primitives is given in Neumann et al. [14]. This work presents policy search methods, which provide the selection, sequencing and adaptation of elemental movement primitives to solve complex tasks. The approach focused on the processing of probabilistic movement primitives, which are detailed described in Section 2.3. The learning method is based on different policy levels and on a Markov Decision Process, similar to the previous approach.

A sequential combination or a blending between primitives makes sense if a main behaviour can be decomposed into a sequence of sub-tasks. According to the current (sub-)task, the main behaviour can be executed by activating the best given primitive for the current sub-task.
A simultaneous combination of primitives is useful if a new task appears, which can't be solved by the given primitives alone but by a combination. This approach enables to solve more complex tasks by less primitive parameters.

An example of simultaneous combinations of movement primitives is the approach of [12], which combines simple primitives to generate a new behaviour. Also the modular muscle activation of the nervous system uses simultaneous combinations of signal patterns to solve an overall task like jumping or swimming [4].

2.2 Task Dependency and Decomposition

As already mentioned this work focuses on the combination of task dependent probabilistic movement primitives. Such a task dependency was analysed by different works like the previously described velocity point incorporation of [9]. Sentis and Khatib [15] worked on the implementation of global tasks and their decomposition into sub-tasks for an interactive robot control. Todorov [16] worked on the composition of task optimal control laws by different primitives. The approach is influenced by the idea of task dependency and the advantages of a modular primitive structure. Ude et al. [17] analysed the possibilities of generalizing the solution of movement primitives according to a specific task and the current state of the robot and its environment. The developed methods have been evaluated on the example of a 3D vision system of a robot. Also the approach of Kupcsik et al. [18] tries to incorporate a task dependency and generalization into a hierarchical policy. A model based policy is used to represent lower and higher policy levels. This method splits the control work into robot control for a given task and generalization over or between tasks.

2.3 Probabilistic Movement Primitives

While DMPs are commonly used in robotics, they also come with severe disadvantages like the usage of only one movement demonstration, which excludes the incorporation of demonstration variance. This section describes a new formulation of MPs - the probabilistic movement primitives, that have been presented in [2].

The core-idea of Probabilistic Movement Primitives (ProMPs), is the representation trajectories by a probability distribution p(movement|parameters) $\rightarrow p(\tau|\theta)$ [2]. A Gaussian distribution is used to model $p(\tau|\theta)$. According to such a normal distribution, a movement (e.g. of joints) is specified by its mean and variance over the time. These means and variances are obtained by learning from several demonstrations of a desired behaviour and represents the main characteristics of a movement (see Figure 2). Especially the variance-progress, incorporates additional informations, like

importance of time-points. Figure 2 shows a changing progress of the time-dependent variance. At two points of the trajectory-representation the variance is small. Points with such a small variance are important to reach, due to most demonstration have shown this behaviour. Areas with a larger variance mirror a more flexible progress of the trajectory. This time-dependent variance provides the basis for an optimal controller, which can be used to execute a trained ProMP [2]. This controller matches exactly mean and variance of the given trajectory distribution.

All the following descriptions are based on the work of Paraschos et al. [2].

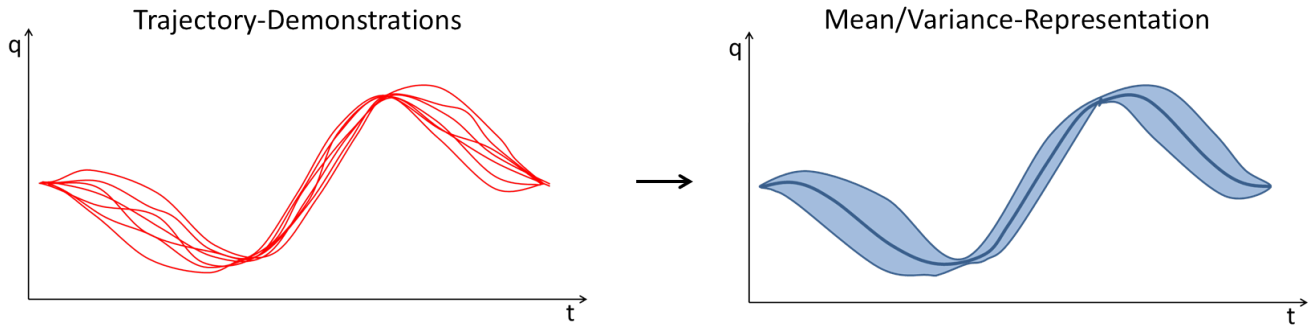

Figure 2: Idea of probabilistic movement primitives: Representation of several movement demonstrations (left trajectories) by a probability distribution that stores a time dependent mean and variance over the joint positions (right curve). The variance progress vary over the time. Points with a small variance have a higher importance than points with a larger variance. A small variance means that the most demonstrations have passed a certain point. Due to this, it is important for the task to reach this point.

2.3.1 Principle Form

A movement results in several demonstrations in form of trajectories. These trajectories mirror the time-dependent joint-angle-progress and are approximated by a combination of basis-functions ϕ_t and weight-vectors $\boldsymbol{w}$,

$$\boldsymbol{y}_t = \begin{bmatrix} q_t \\ \dot{q}_t \end{bmatrix} = \boldsymbol{\Phi}_t^\top \cdot \boldsymbol{w} + e_y, \qquad e_y \sim N(0, \Sigma_\theta). \tag{5}$$

The variable $\boldsymbol{y}_t$ expresses the joint-angle-progress for a certain time-point t defined by the joint-angles q_t and the corresponding angle-velocities $\dot{q}_t$. The basis-matrix $\boldsymbol{\Phi}_t = \begin{bmatrix} \phi_t & \dot{\phi}_t \end{bmatrix}$ contains the basis-functions for q_t and $\dot{q}_t$. A single trajectory-representation decomposes to $q_t = \phi_t w$ and $\dot{q}_t = \dot{\phi}_t w$. The weights $\boldsymbol{w}$ define the shape of the trajectories. The error of representing a trajectory is defined by the normal-distribution e_y , see Eq. 5.
The weight-vector w can be computed by linear ridge regression, i.e.

$$\boldsymbol{w} = (\boldsymbol{\phi}^\top \boldsymbol{\phi} + \lambda \boldsymbol{I})^{-1} \boldsymbol{\phi}_t^\top \boldsymbol{y}, \tag{6}$$

due to their linear dependency on the basis-functions. The matrix $\boldsymbol{\Phi}$ contains the basis-functions for all time points t. The basis-functions ϕ_t enable a smooth trajectory approximation (in combination with $\boldsymbol{w}$). They are defined by

$$\boldsymbol{\Phi}_t = \begin{bmatrix} \phi_t & \dot{\phi}_t \end{bmatrix}, \qquad \phi_t = \phi(z_t), \qquad \dot{\phi}_t = \phi(z_t)' \dot{z}_t, \tag{7}$$

$$\phi(z_t) = \frac{b_i(z)}{\sum_j b_j(z)}. \tag{8}$$

The variable z_t replaces the time-index t. The phase z_t is an arbitrary monotonic increasing function of t. This variable enables a speed-regulation of a specific movement, by changing the function-parameters of z_t. The single basis-functions are normalized by their sum, such that all basis functions always sum to one, see Eq. 8.
Different basis-functions are required for different kinds of movements. Stroke-based movements can be modelled by Gaussian activation functions, see Eq. 9. Rhythmic movements by Von Mises activation functions see Eq. 10, where c_i represents the centres and h the widths of the basis-functions.

$$b_i(z) = \exp\left(\frac{-(z_t - c_i)^2}{2h}\right) \tag{9}$$

$$b_i(z) = \exp\left(\frac{\cos(2\pi(z_t - c_i)}{h}\right) \tag{10}$$

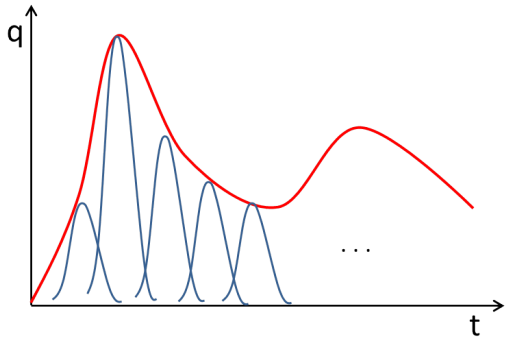

Figure 3: Trajectory representation by basis-functions and a weight-vector. An arbitrary trajectory can be approximated by a few basis functions, which are scaled by the corresponding weight-vectors.

The Trajectory Representation is visualized by Figure 3. The basis-functions fit a pre-specified number of data-points and the weights $\boldsymbol{w}$ define the high of these basis-functions.
The distribution of the joint position and velocity $\boldsymbol{y}_t$ at time point t can be written as

$$p(\boldsymbol{y}_t|\boldsymbol{w}) = N\left(\boldsymbol{y}_t|\boldsymbol{\Phi}_t\boldsymbol{w}, \boldsymbol{\Sigma}_y\right). \tag{11}$$

The part $\boldsymbol{\Phi}_t\boldsymbol{w}$ contains the trajectory-representation of the joint-angle/velocities at time-point t and defines the mean of the Gaussian distribution. The variance $\boldsymbol{\Sigma}_y$ of the error e_y defines the variance.
The joint-angle-values are assumed to be independent and identical distributed (i.i.d.). The product of Gaussian probability distributions of (i.i.d.) variables results in the distribution of a trajectory τ:

$$p(\boldsymbol{y}|\boldsymbol{w}) = p(\tau|\boldsymbol{w}) = \prod_t N\left(\boldsymbol{y}_t|\boldsymbol{\Phi}_t\boldsymbol{w}, \boldsymbol{\Sigma}_y.\right) \tag{12}$$

To get an probability distribution over several trajectories, we represent a distribution over $\boldsymbol{w}$. One weight-vector $\boldsymbol{w}$ represents a single trajectory. We consider Gaussian distributions, i.e.

$$p(\boldsymbol{w}|\boldsymbol{\theta}) = N\left(\boldsymbol{w}|\boldsymbol{\mu}_w, \boldsymbol{\Sigma}_w\right), \tag{13}$$

$$\boldsymbol{\theta} = [\boldsymbol{\mu}_w, \boldsymbol{\Sigma}_w]. \tag{14}$$

The parameter vector $\boldsymbol{\theta}$ contains the mean $\boldsymbol{\mu}_w$ and variance $\boldsymbol{\Sigma}_w$ of the trajectory-weights w of all given demonstrations of a single elemental movement. This distribution is incorporated by defining the joint-probability of $\boldsymbol{y}$ and $\boldsymbol{w}$. The trajectory-weights can be marginalized out by taking the integral over w. This marginalization leads to an integral of a product of two normal distributions, which can be expressed as

$$p(\boldsymbol{y}|\boldsymbol{\theta}) = \int p(\boldsymbol{y}, \boldsymbol{w}|\boldsymbol{\theta})\mathrm{d}\boldsymbol{w} = \int p(\boldsymbol{y}|\boldsymbol{w})p(\boldsymbol{w}|\boldsymbol{\theta})\mathrm{d}\boldsymbol{w}, \tag{15}$$

$$p(\boldsymbol{y}_t|\boldsymbol{\theta}) = \int N\left(\boldsymbol{y}_t|\boldsymbol{\Phi}_t^\top\boldsymbol{w}, \boldsymbol{\Sigma}_\theta\right) N\left(\boldsymbol{w}|\boldsymbol{\mu}_w, \boldsymbol{\Sigma}_w\right)\mathrm{d}\boldsymbol{w} \tag{16}$$

$$= N\left(\boldsymbol{y}_t|\boldsymbol{\Phi}_t^\top\boldsymbol{\mu}_w, \boldsymbol{\Phi}_t^\top\boldsymbol{\Sigma}_w\boldsymbol{\Phi}_t + \boldsymbol{\Sigma}_\theta\right). \tag{17}$$

From Eq. 17 we see that the probability distribution of joint-angles for a certain time-point is expressed by a normal-distribution with time-dependent mean $\boldsymbol{\mu}_t = \boldsymbol{\Phi}_t^\top\boldsymbol{\mu}_w$ and variance $\boldsymbol{\Sigma}_t = \boldsymbol{\Phi}_t^\top\boldsymbol{\Sigma}_w\boldsymbol{\Phi}_t + \boldsymbol{\Sigma}_\theta$.

The parameters $\boldsymbol{\theta} = [\boldsymbol{\mu}_w, \boldsymbol{\Sigma}_w]$ are the remaining unknowns and can be learned by maximum likelihood estimation. Figure 4 gives an overall view on the complete learning process. We start with the representation of source-data (joint-angles) by weight-vectors and basis-functions (through linear ridge regression). The probability distributions are composed of the two components $p(\boldsymbol{y}|\boldsymbol{w})$ and $p(\boldsymbol{w}|\boldsymbol{\theta})$. As previously discussed, $p(\boldsymbol{y}|\boldsymbol{w})$ is composed of the trajectory-representation $\boldsymbol{\Phi}_t^\top\boldsymbol{w}$ and the demonstration-noise $e_y \sim N(0, \boldsymbol{\Sigma}_\theta)$. In combination with $p(\boldsymbol{w}|\boldsymbol{\theta})$, the target distribution $p(\boldsymbol{y}|\boldsymbol{\theta})$ is constructed. After learning the parameters $\boldsymbol{\theta}$, mean and variance are defined for each time point of the movement.

2.3.2 New Operators of ProMPs

The probabilistic MP approach enables us to apply new operators on movement representations. These are the modulation, combination and blending between primitives [2]. The **modulation** of probabilistic movement primitives involves the implementation of a new via-point or final-position, by applying the bayes-rule with $p(\boldsymbol{w}|\boldsymbol{x}_{\text{new}}) \propto p(\boldsymbol{x}_{\text{new}}|\boldsymbol{w}) \cdot p(\boldsymbol{w})$.

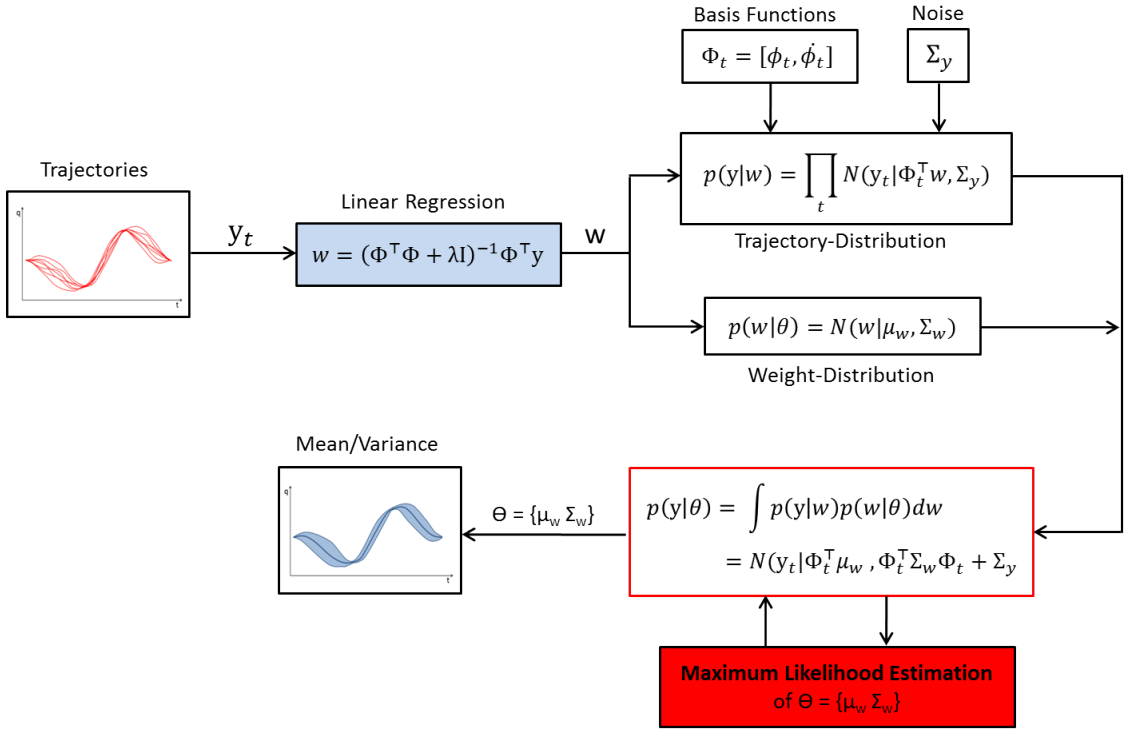

Figure 4: Learning Process of Probabilistic Movement Primitives. The given demonstrations of joint-angles/velocities y are represented by a combination of weight vector and basis functions $\Phi_t^\top w$. One movement demonstration is characterized by one weight-vector w. Due to this parametrization, two normal distributions are modelled. One distribution $p(y|w)$ for y given w and another one $p(w|\theta)$ for w given the model-parameter θ of the probabilistic movement primitives. Both distributions can be combined to $p(y|\theta)$ by marginalizing out w. The resulting distribution is the basis for the maximum likelihood estimation of $\theta = \{\mu_w, \Sigma_w\}$, which represents mean and variance of the given joint-angle demonstrations.

Combination and Blending

Apart from modulation, blending and especially the combination of ProMPs is the main-point for the presented work. A combination of probabilistic movement primitives provides more modularity and flexibility for generating new movement-solutions. The combination of ProMPs enables the solving of more complex tasks by reusing existing primitives, which are already trained.

The continuous blending between several ProMPs completes these capability. A smooth switching between existing primitives avoids the retraining of new ProMPs. It provides the basis for selecting the best primitive for the current time-step or situation.

Due to the Gaussian form, the combination of ProMPs can be done by multiplying both primitives [2]. The multiplication of both distributions extract the areas with high probability. The new distribution

$$p_{\text{new}}(\tau) \propto \prod_i p_i(\boldsymbol{y}) \tag{18}$$

is build up by the product of i pre-trained ProMPs. Several multiplied Gaussian distributions can be transformed into a resulting distribution with new mean and variance. Figure 5-left illustrates this combination. The co-activation of the red and blue primitive, extracts the common regions of mean and variance of both MPs $p_i(\boldsymbol{y}_t)$ and build up a new distribution $p_{\text{new}}(\boldsymbol{y}_t)$. This results in a simultaneous activation and extraction of all primitive parts with high probability during the execution.

Using the Eq. 18 the operator blending can be applied by incorporating a time-dependent activation function α_t^i. The new distribution, resulting from blending between several primitives, can be generated by

$$p_{new}(\tau) \propto \prod_t \prod_i p_i(\boldsymbol{y}_t)^{\alpha_t^i}. \tag{19}$$

The level of activation of a single primitive can be controlled by adjusting α_t^i for every time step. An continuous blending between several primitives can be done by using smoothly progressing activation-functions for every primitive, which

sum up to one for every time-step, see Figure 5-right. Starting with the red primitive a smooth blending to the blue ProMP is enforced by the progress of both activation-functions.
A constant factor of one results in the co-activation of all MPs through the complete movement (figure 5 - left).
The mean and variance of the new (blended) distribution $p^{\text{new}}(\boldsymbol{y}_t)$ (in case of Gaussian distributions) can be calculated analytically by

$$p^{\text{new}}(\boldsymbol{y}_t) = N\left(\boldsymbol{y}_t | \boldsymbol{\mu}_t^{\text{new}}, \boldsymbol{\Sigma}_t^{\text{new}}\right), \qquad p_i(\boldsymbol{y}_t) = N\left(\boldsymbol{y}_t | \boldsymbol{\mu}_t^i, \boldsymbol{\Sigma}_t^i\right), \tag{20}$$

$$\boldsymbol{\mu}_t^{\text{new}} = \left(\boldsymbol{\Sigma}_t^{\text{new}}\right)^{-1} \left(\sum_i \left(\boldsymbol{\Sigma}_t^i / \alpha_t^i\right)^{-1} \boldsymbol{\mu}_t^i\right), \qquad \boldsymbol{\Sigma}_t^{\text{new}} = \left(\sum_i \left(\boldsymbol{\Sigma}_t^i / \alpha_t^i\right)^{-1}\right)^{-1}. \tag{21}$$

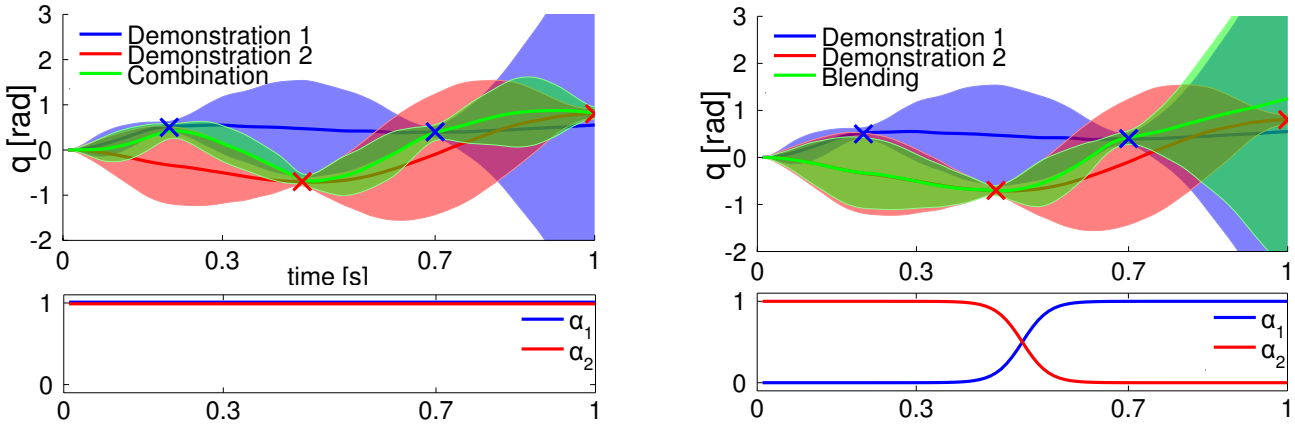

Figure 5: (**Left**) Illustration of the combination of two probabilistic movement primitives [2]. The product of both primitive demonstrations activates the common areas of high probability. The result is a new ProMP distribution, which incorporates the critical time points with low variance of both primitives. (**Right**) This figure demonstrates the blending between both demonstrations by adjusting the time-dependent activation factors α_1 and α_2 [2]. This results in a continuously switching between both primitives.

3 Task-Dependent Probabilistic Movement Primitives

According to the desired control architecture (see Introduction), we need to be able to adapt the primitives to different tasks. This task can be everything that helps to define elemental movements. For example reaching a certain position and orientation with a robots end-effector, the velocity of a robot-link or the force, which a robot applies with his hands on a tool. Reaching several via-points with different links could be considered as a task, as well as reaching different points at different time-steps. Jumping to a certain point or throwing a ball into a specific direction are other examples. A Task mirrors a desired elemental behaviour of a robot.
To define such a behaviour, we consider a continues set of tasks (e.g. different demonstrations of reaching position-coordinates or velocity values). The characterization of these tasks is done by a multidimensional task-variable, which stores parameters like the reached position or velocity (with the end-effector). If a task and the corresponding task-variable is specified, we want to find a movement representation (movement primitive) that can be used for a multitude of tasks.

The following chapter describes the derivation of task dependent probabilistic movement primitives (TaskMP) on the example of a robot arm with several links.
To explain the concept and derivation of TaskMPs, two certain tasks will be considered. These are reaching a certain position and taking a specific orientation with the robots end-effector. In this case, the reached position and orientation are building the corresponding task-variables.
The idea is to model the relationship between the task variable $\boldsymbol{x}$ and the movement representation of a primitive. For example, we want to capture the relationship between the desired end-effector position and the parameters of the ProMP representation. The MP-parameters in turn remap the joint-angles of the robot according to the applied end-effector position.

The trajectories of the desired robot-joint-angles are represented by a basis-function weight-vector combination $q_t = \phi_t^\top w$ (see Chapter 2), were the basis functions ϕ_t are Gaussian.

In the first approach of deriving probabilistic movement primitives, no certain tasks were considered. Several trajectories of joint-angles (and velocities) were modelled as probability distributions $p(\boldsymbol{w}|\boldsymbol{\mu}_w, \boldsymbol{\Sigma}_w)$ over the weight vector $\boldsymbol{w}$, which is modelled as Gaussian distribution. We want to model a distribution $p(\boldsymbol{w}|\boldsymbol{x})$ over $\boldsymbol{w}$ that depends on the task variable $\boldsymbol{x}$.
This desired task dependency is incorporated by a task dependent mean $\boldsymbol{\mu}_w(x)$. In the most simple case a linear relationship between $\boldsymbol{w}$ and task-variable $\boldsymbol{x}$ is established by modelling the mean as $\boldsymbol{\mu}_w(x) = \boldsymbol{\beta} \cdot \boldsymbol{x}$, where $\boldsymbol{\beta}$ are the linear regression-parameters. The corresponding distribution is,

$$p(\boldsymbol{w}|\boldsymbol{\mu}_w(x), \boldsymbol{\Sigma}_w) = p(\boldsymbol{w}|\boldsymbol{\beta}\boldsymbol{x}, \boldsymbol{\Sigma}_w). \tag{22}$$

This results in a linear mapping between the task-space $\boldsymbol{x}$ and the joint-space $\boldsymbol{w}$. However, the dependency between $\boldsymbol{x}$ and $\boldsymbol{w}$ is non-linear for the most tasks. A linear model typically works only for a small area of the task variable $\boldsymbol{x}$, but is not appropriate for a larger area.
To solve this problem, several linear models can be combined to cover a larger area of $\boldsymbol{x}$. Such a combination of linear models is typically represented as a mixture of linear experts [19], [20].

3.1 Mixture of Linear Experts

A mixture of experts is a probabilistic model, which fits a distribution of data-points by combining several model-components (called experts) [19], [20]. Such a mixture is applied, if one single model do not fit the complete data-set, but a combination of several models does.
A simple example is a mixture of Gaussian distributions (see Figure 6). We can represent arbitrary data distributions by using a mixture of simple Gaussian distributions.
It is possible to apply this approach to model the desired non-linearity between task and joint space [21]. The task dependent probability distribution $p(\boldsymbol{w}|\boldsymbol{x})$ is modelled by the sum of k experts $p_k(\boldsymbol{w}|\boldsymbol{x})$, which are gated by the gating-network $p(\boldsymbol{k}|\boldsymbol{x})$ (see Figure 7 - left), as follows

$$p(\boldsymbol{w}|\boldsymbol{x}) = \sum_k p(k|\boldsymbol{x}) p_k(\boldsymbol{w}|\boldsymbol{x}), \qquad p_k(\boldsymbol{w}|\boldsymbol{x}) = N(\boldsymbol{w}|\boldsymbol{\beta_k}\boldsymbol{x}, \boldsymbol{\Sigma}_k^\top). \tag{23}$$

The experts are linear Gaussian distributions. The gating-coefficients are controlling the influence of each individual expert to the desired distribution $p(\boldsymbol{w}|\boldsymbol{x})$. The coefficients need to be dependent on the task-variable $\boldsymbol{x}$, as different primitives should be activated for different task-areas.

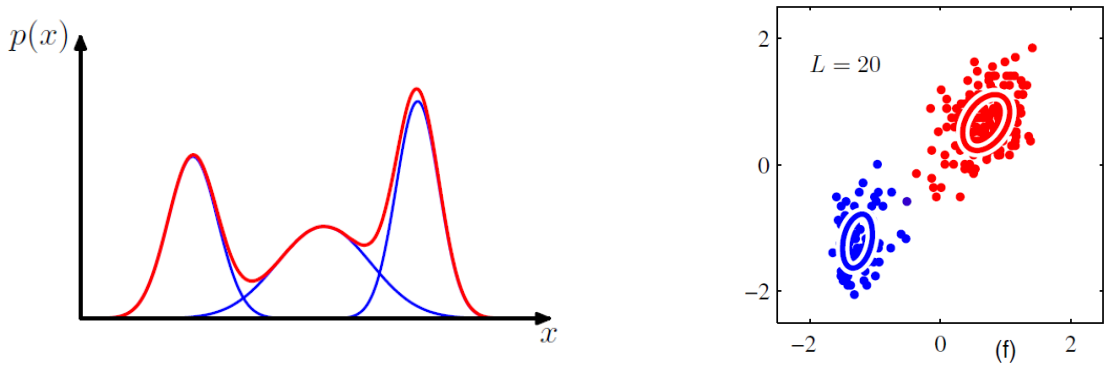

Figure 6: Approximation of arbitrary distributions by several Gaussian. **(Left)** One dimensional example of distribution fitting with three normal distributions [19] (p. 111). **(Right)** Two dimensional data point cloud example. Both clusters are represented by two (two-dimensional) Gaussian distributions [19] (p. 437).

The equations to generate the gating coefficients (Eq. 24) can be considered as an own Gaussian mixture model $p(\boldsymbol{x}) = \sum_k p(k) \cdot p(\boldsymbol{x}|k)$, similar to Figure 6. Other implementations exists, using softmax functions [20] or logistic sigmoid functions [22] for the gating.
The difference of the gating-mixture to $p(\boldsymbol{w}|\boldsymbol{x})$ is, that we learn the joint distribution over the task-variable $\boldsymbol{x}$ and the mixture component k. To model the gating $p(k|\boldsymbol{x})$, we condition this distribution on the task-variable $\boldsymbol{x}$ as follows

$$p(k|\boldsymbol{x}) = \frac{p(k)p(\boldsymbol{x}|k)}{\sum_j p(j)p(\boldsymbol{x}|j)}, \qquad p(\boldsymbol{x}|k) = N(\boldsymbol{x}|\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k). \tag{24}$$

The probability distribution $p(k|\boldsymbol{x})$ represents the desired gating-coefficients for the expert k given the task variable $\boldsymbol{x}$. The gating $p(k|\boldsymbol{x})$ is modelled by the Gaussian gating-experts $p(\boldsymbol{x}|k)$ and the priors $p(k)$. Due to this Gaussian form, the gating model is defined by the mean vectors $\boldsymbol{\mu}_k$, the variance matrices $\boldsymbol{\Sigma}_k$ and the priors $p(k)$.

Figure 7 -right illustrates a simple example for a mixture of linear experts. The trajectory of joint-angles q over the y-coordinates of an end-effector (represented by $\boldsymbol{x}$) is fitted by several linear models (experts). Every expert is activated by a Gaussian distribution (part of the gating-network) which is responsible for a certain area of the task-space (in this case the y-coordinates).
The activation of all experts according to its corresponding task-space-area results in a smooth fitting of the target trajectory.

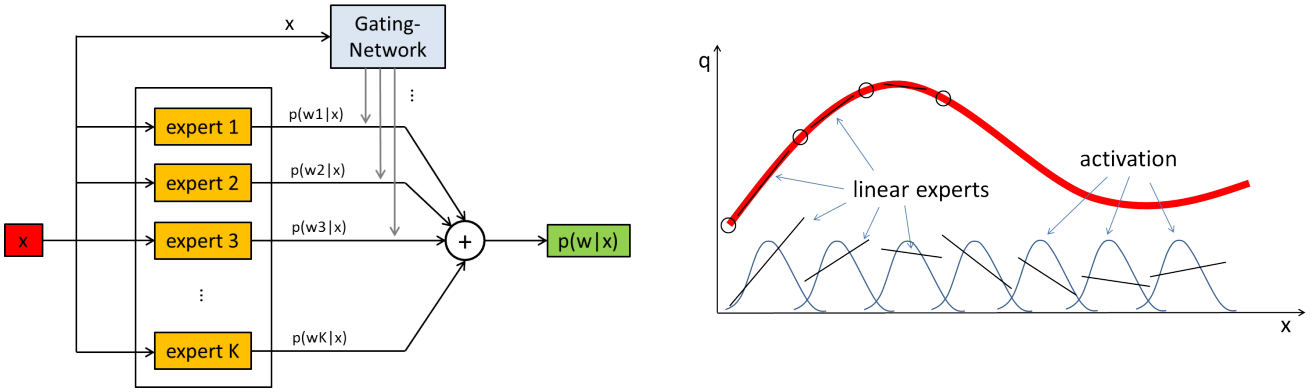

Figure 7: **(Left)** Illustration of a mixture of experts. The different expert distributions $p(\boldsymbol{w}_k|\boldsymbol{x})$ are selected and weighted by the gating network $p(k|\boldsymbol{x})$, according to the task $\boldsymbol{x}$. The weighted sum of all experts models the resulting probability distribution $p(\boldsymbol{w}|\boldsymbol{x})$. **(Right)** Example of linear expert fitting. The different experts of the mixture, model different parts of the trajectory. The combination of Gaussian activation-functions (gating) and linear experts, results in a smooth approximation of the target function (trajectory).

3.2 Training of Probabilistic Task Dependent Movement Primtives

The Training of a task-dependent probabilistic movement primitive can be done with the expectation maximization algorithm (EM-Algorithm). The EM-Algorithm trains the gating and expert parameters of the TaskMP mixture model.

The needed training data includes demonstrations of the task-variable $\boldsymbol{x}$ and the corresponding joint-trajectories $\boldsymbol{w}$. After the learning-process, the TaskMP model is able to map unknown task-variables to an appropriate movement solution, i.e. the corresponding $\boldsymbol{w}$.

Expectation Maximization Algorithm

The Expectation Maximization Algorithm is estimating the distribution of hidden variables [19]. In the case of TaskMPs a hidden variable is the assignment of the expert k to a data-point n. The evaluation of the expert assignments and the training of the mixture-parameters cannot be done by once. Two separate calculation steps are necessary. The expectation step (E-Step) and the maximization step (M-Step).
The **E-Step** contains the described computation of the distribution over the hidden variables given the current model parameters and data, as follows

$$p(k|x,w) = \frac{p(k|\boldsymbol{x}) \cdot p(\boldsymbol{w}|k,\boldsymbol{x})}{\sum_j p(k|\boldsymbol{x}) \cdot p(\boldsymbol{w}|k,\boldsymbol{x})}, \qquad \gamma_{nk} = p(k|\boldsymbol{x},\boldsymbol{w}). \tag{25}$$

The used model parameter are fixed during this step. The calculated distribution $p(k|x,w)$ evaluates the responsibilities γ_{nk} of every mixture component for all data points [21]. These responsibilities represent the probability that mixture component k created the considered data-point $[\boldsymbol{x},\boldsymbol{w}]$.
The **M-Step** includes the optimization of the model-parameters given the distribution over the hidden variables [21]. This results in a maximization of the Q-function, i.e.

$$\ln(p(\boldsymbol{X}|\boldsymbol{\theta})) = Q(\boldsymbol{\theta},\boldsymbol{\theta}^{old}) = \sum_{n=1}^{N}\sum_{k=1}^{K}\gamma_{nk} \cdot (\ln(p(k|\boldsymbol{x}_n)) + \ln(p(\boldsymbol{w}_n|k,\boldsymbol{x}_n))), \tag{26}$$

which evaluates the log-likelihood of the complete data-set. The maximization can be done analytically, if the single mixture components are linear Gaussian distributions. This is the case for the described task dependent movement primitives. Setting the gradient of the Q-function to zero, results in the new (optimized) mixture model parameters. This includes the new mean-vectors $\boldsymbol{\mu}_k$, variance-matrices $\boldsymbol{\Sigma}_k^g$ and priors $p(k)$ of the gating-network, as follows

$$\mu_k = \frac{1}{N_k}\sum_{n=1}^{N}\gamma_{nk} \cdot \boldsymbol{x}_n, \qquad \boldsymbol{\Sigma}_k^g = \frac{1}{N_k}\sum_{n=1}^{N}\gamma_{nk} \cdot (\boldsymbol{x}_n - \boldsymbol{\mu}_k)(\boldsymbol{x}_n - \boldsymbol{\mu}_k)^\top, \qquad p(k) = \frac{N_k}{\sum_{k=1}^{K} N_k}, \tag{27}$$

as well as the new regression-parameters $\boldsymbol{\beta}_k$ and variance-matrices Σ_k^c of the experts. The modified task-variable $\boldsymbol{x}_b$ contains the bias for the linear regression parameter $\boldsymbol{\beta}_k$. The normalization factor N_k represents the number of data-points assigned to the mixture component k.

$$\boldsymbol{\beta}_k = (\boldsymbol{X}^\top \boldsymbol{R}_k \boldsymbol{X} + I \cdot A)^{-1} \boldsymbol{X}^\top \boldsymbol{R}_k \boldsymbol{y} \qquad \boldsymbol{R}_k = \begin{pmatrix} \gamma_{1k} & 0 & 0 \\ 0 & \dots & 0 \\ 0 & 0 & \gamma_{nk} \end{pmatrix}, \tag{28}$$

$$\boldsymbol{\Sigma}_k^c = \frac{1}{N_k}\sum_{n=1}^{N}\gamma_{nk}(\boldsymbol{w} - \boldsymbol{x}_b\boldsymbol{\beta}_k)(\boldsymbol{w} - \boldsymbol{x}_b\boldsymbol{\beta}_k)^\top, \qquad N_k = \sum_{n=1}^{N}\gamma_{nk}. \tag{29}$$

These equations contain the responsibilities γ_{nk}, which were calculated in the E-Step before. The maximization would not have been possible in closed form without calculating the responsibilities.

The EM-Algorithm is very sensitive to the initialization of the initial parameters (either γ_{nk} or the model parameters), which are used to calculate the first step. For the TaskMPs, the initialization is done by the k-means algorithm [19]. This clustering method assigns every data-point to one expert, which can be used as the first responsibilities. With this pre-defined responsibilities γ_{nk}, the following M-Step can be executed.

The E-step and M-step are repeated until the log-likelihood converges. The complete data log-likelihood can be calculated by evaluating the Q-function see Eq. 26.
For detailed descriptions of the EM-Algorithm and mixture models see [19].

3.3 Toy-Task-Example

To explain the training and testing of a task-dependent probabilistic movement primitive, we introduce a toy-task setup. The simulation of a robot-arm with seven links is used to generate training and test data for the considered task. This task includes reaching a certain via-position and orientation with the end-effector. The robot starts with no excursion. All joint-angles are zero. The movement is only two dimensional in the xy-plane simulated. After reaching the desired via-position and orientation, the robot arm drives back to its initial configuration (joint-angles are zero). The via-position and via-orientation represents the task-variable.

3.4 Training-Example

To be able to train a mixture model several steps are necessary. Figure 8 describes the process for the defined toy task with the seven link robot.
The first step includes the generation of random via points (position and orientation) within a predefined Cartesian task-space. Using these via points, an optimal controller calculates the best trajectory (according to a defined reward) for reaching one via point with the robots end-effector and go back to its start position.
The resulting joint-angles of the robot arm - caused by following this optimal trajectory - will be extracted and transformed into a weight-vector basis-function representation (see Section 2.3.1). The result of these steps are the task variable $\boldsymbol{x}$ in form of the generated via-points and the weight-vectors of the joint-angle trajectories.

This data is used for the training of the mixture model, which maps the provided task variable $\boldsymbol{x}$ to the weight-vectors $\boldsymbol{w}$. The training can be performed by the previously described expectation maximization algorithm. The resulting model parameters for gating and experts define the desired probabilistic task-dependent movement primitive.

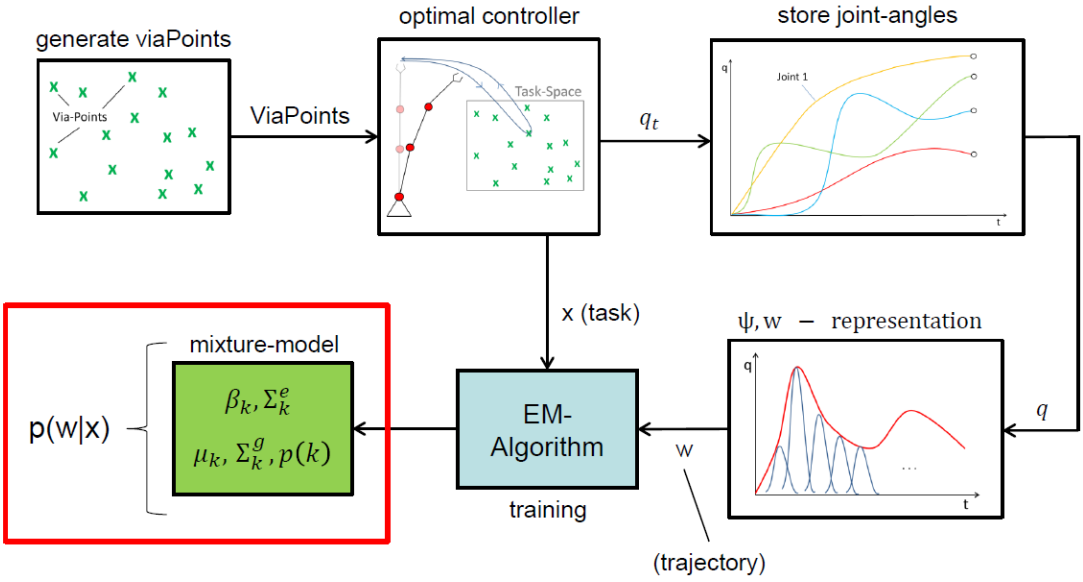

Figure 8: Training-Cycle of Task Dependent Movement Primitives: The first step is the generation of training data by an optimal controller for a multiple link robot arm. The controller calculates the optimal trajectory for reaching a random via-point. The joint angles - which correspond to the reached via-points - can be approximated by a combination of basis-function and weight-vector. The real via-points (task-variables) $\boldsymbol{x}$ and the corresponding joint-angle weight-vectors $\boldsymbol{w}$, represent the training-set. The learning process is executed by an EM-Algorithm, which estimates the mixture parameters of the task-dependent probabilistic movement primitives $p(\boldsymbol{w}|\boldsymbol{x})$.

3.5 Test of Probabilistic Task Dependent Movement Primtives

The necessary steps to test a trained task dependent movement primitive with new points is illustrated in Figure 9.
We use different data-sets for training and testing. The optimal expert for a considered test-point $\boldsymbol{x}_{\text{test}}$ is evaluated by the gating network $p(k|\boldsymbol{x}_{\text{test}})$, in form of selecting the expert with the highest gating, i.e.

$$\boldsymbol{x}_{\text{test}} \rightarrow p(k|\boldsymbol{x}_{\text{test}}) \rightarrow k_{\text{best}}. \tag{30}$$

The weight-vector of joint-angles, which correspond to the given test-point are extracted by calculating the expectation of the expert-probability distribution, as follows

$$p_{k_{\text{best}}}(\boldsymbol{w}_{\text{new}}|\boldsymbol{x}_{\text{test}}) = N(\boldsymbol{w}_{\text{new}}|\boldsymbol{\beta}_{k_{\text{best}}} \cdot \boldsymbol{x}_{\text{test}}), \tag{31}$$

$$E[p_{k_{best}}(\boldsymbol{w}_{\text{new}}|\boldsymbol{x}_{\text{test}})] = \boldsymbol{\beta}_{k_{\text{best}}} \cdot \boldsymbol{x}_{\text{test}} =: \boldsymbol{w}_{\text{new}}. \tag{32}$$

The selection of a single expert instead of averaging over all experts, avoids the add up of multiple solutions. The new joint-weight-vector $\boldsymbol{w}_{\text{new}}$ is equated to this calculated expectation, which equals to the mean $\boldsymbol{\beta}_{k_{\text{best}}}\boldsymbol{x}_{\text{test}}$ of the corresponding expert normal distribution.

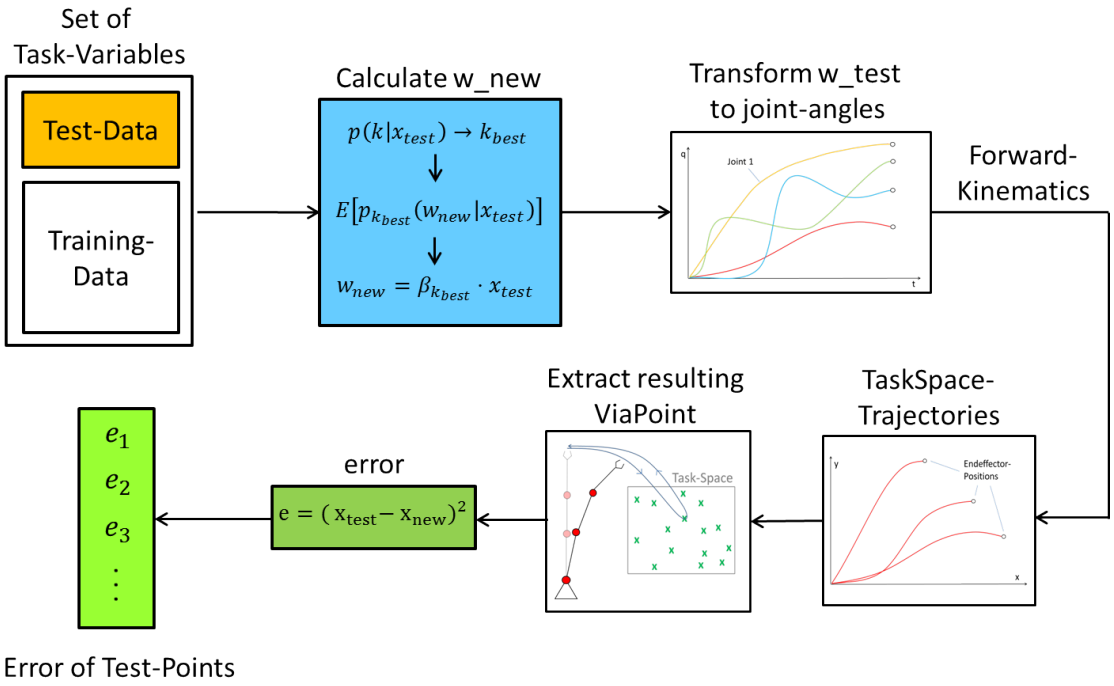

Figure 9: Test-Cycle of Task Dependent Movement Primitives: Given an individual test-set with test-points $\boldsymbol{x}_{\text{test}}$, the corresponding gating-coefficients of the mixture model are evaluated. The Expectation of the expert with the highest corresponding gating-coefficient is calculated. This expectation calculation results in the mean extraction of the chosen expert ($\boldsymbol{w}_{\text{new}} = \boldsymbol{\beta}_{k_{\text{best}}} \boldsymbol{x}_{\text{test}}$), due to its Gaussian form. The resulting weight-vectors $\boldsymbol{w}_{\text{new}}$ are re-transformed to joint-angles by the corresponding basis-functions. The resulting via-points are extracted from the task-space-trajectories, which are derived from the joint-angles by forward kinematics. The committed test-points and resulting task-variables $\boldsymbol{x}_{\text{new}}$ are compared by evaluating the mean square error.

This resulting weight-vector $\boldsymbol{w}_{\text{new}}$ has to be transformed into joint-angles (by applying the basis-function weight-vector representation). To control the test points with the result from the mixture ($\boldsymbol{w}_{\text{new}}$), the calculated joint-angles will be transformed into the task-space trajectories by applying forward kinematics. The via points, which correspond to the test-points are extracted by these trajectories.
To compare both points, the mean square error e is calculated (Eq. 33). This can be provided for every point of the test-set to get a corresponding test-set-error.

$$e = (\boldsymbol{x}_{\text{test}} - \boldsymbol{x}_{\text{new}})^2 \tag{33}$$

4 Product of Task Dependent Movement Primitives

The idea to represent a TaskMP by a linear Gaussian mixture model is sufficient for simple and elemental tasks, but two disadvantages appear. First we have to re-learn the TaskMPs for every new task separately. Second more complex behaviours will need more mixture components to provide a successful mapping. A more promising approach is to combine elemental TaskMPs to solve new and possibly more complex tasks with less model parameters.
To build such a combination, the product of two TaskMPs (similar to the ProMP combination) and hence the product of two mixture models is used.

4.1 Concept

The product of Task Dependent Movement Primitives results in a naive combination of two pre-trained TaskMPs, similar to the ProMPs of Section 2.3.1. The desired task is decomposed in two sub-task, e.g. reaching position (task 1) and orientation (task 2) by an end-effector. This task-splitting corresponds to a manually designed decomposition. The two described subtasks result in two disjunct task variables $\boldsymbol{x}_1$ and $\boldsymbol{x}_2$, which are used to train two TaskMPs independently of each other, i.e.

$$\text{task 1: } \boldsymbol{x_1} \rightarrow p_1(\boldsymbol{w}|\boldsymbol{x}_1), \tag{34}$$
$$\text{task 2: } \boldsymbol{x_2} \rightarrow p_1(\boldsymbol{w}|\boldsymbol{x}_2). \tag{35}$$

The trained TaskMPs can be combined to model a resulting movement primitive. This new (combined) TaskMP maps the two sub task variables $\boldsymbol{x}_1$ and $\boldsymbol{x}_2$, to a resulting joint-weight-vector $\boldsymbol{w}_{\text{new}}$, which allows us to solve a combination of both tasks with one movement.

Every TaskMP model contains K mixture components. The combination of two TaskMPs (Eq. 36, 37) results in a combination of $K \cdot K$ components. Due to this, the new movement primitive contains a larger number of K^2 components.

$$p(\boldsymbol{w}|\boldsymbol{x}) \propto p_1(\boldsymbol{w}|\boldsymbol{x_1}) \cdot p_2(\boldsymbol{w}|\boldsymbol{x_2}) \propto \sum_{i=1}^{K} p_{1i}(i|\boldsymbol{x}_1) p_{1i}(\boldsymbol{w}|\boldsymbol{x}_1) \cdot \sum_{j=1}^{K} p_{2j}(j|\boldsymbol{x}_2) p_{2j}(\boldsymbol{w}|\boldsymbol{x}_2) \tag{36}$$

$$\begin{aligned} \propto & p_{11}(1|\boldsymbol{x}_1) p_{21}(1|\boldsymbol{x}_2) \cdot p_{11}(\boldsymbol{w}|\boldsymbol{x}_1) p_{21}(\boldsymbol{w}|\boldsymbol{x}_2) + \\ & p_{11}(1|\boldsymbol{x}_1) p_{22}(1|\boldsymbol{x}_2) \cdot p_{11}(\boldsymbol{w}|\boldsymbol{x}_1) p_{22}(\boldsymbol{w}|\boldsymbol{x}_2) + \\ & p_{11}(1|\boldsymbol{x}_1) p_{23}(1|\boldsymbol{x}_2) \cdot p_{11}(\boldsymbol{w}|\boldsymbol{x}_1) p_{23}(\boldsymbol{w}|\boldsymbol{x}_2) + \ldots \end{aligned} \tag{37}$$

The combination of the different experts of every TaskMP can be implemented by using Gaussian and matrix transformation rules [23]. According to these rules, the product of two Gaussian-models can be expressed as a resulting normalized Gaussian-distribution, i.e.

$$N(\boldsymbol{x}|\boldsymbol{\mu}_1, \Sigma_1) \cdot N(\boldsymbol{x}|\boldsymbol{\mu}_2, \Sigma_2) = N(\boldsymbol{x}|\boldsymbol{\mu}_{\text{new}}, \Sigma_{\text{new}}) \cdot Z, \tag{38}$$

$$\boldsymbol{\mu}_{\text{new}} = \boldsymbol{\Sigma}_2(\boldsymbol{\Sigma}_1 + \boldsymbol{\Sigma}_2)^{-1}\boldsymbol{\mu}_1 + \boldsymbol{\Sigma}_1(\boldsymbol{\Sigma}_1 + \boldsymbol{\Sigma}_2)^{-1}\boldsymbol{\mu}_2, \tag{39}$$
$$\boldsymbol{\Sigma}_{\text{new}} = \boldsymbol{\Sigma}_1(\boldsymbol{\Sigma}_1 + \boldsymbol{\Sigma}_2)^{-1}\boldsymbol{\Sigma}_2, \tag{40}$$
$$Z = N(\boldsymbol{\mu}_1|\boldsymbol{\mu}_2, \boldsymbol{\Sigma}_1 + \boldsymbol{\Sigma}_2). \tag{41}$$

The normalization factor Z is necessary to keep the resulting form as a probability distribution, which sums up to one. Using this transformation, every TaskMP component combination is transformed into a resulting mixture-component of the new movement primitive. However, one problem of this model is that the combination of the gating-coefficients have not been successful, due to their different training tasks. A combination of gating coefficients like

$$p_{11}(1|\boldsymbol{x}_1) \cdot p_{21}(1|\boldsymbol{x}_2) = p_1(1|\boldsymbol{x}) \cdot Z_{\text{gate}}, \tag{42}$$

results in a wrong selection of expert-combinations for a given main task $\boldsymbol{x} = [\boldsymbol{x}_1, \boldsymbol{x}_2]$. The coefficients are not trained to gate the combined mixture-components according to the main task $\boldsymbol{x}$.

4.2 Evaluation without Gating-Network

Figure 10 describes the process to evaluate the TaskMP-Combination without using a separate trained gating-network. The method evaluates every single combination of the components of two mixtures. Such a analysis is done, due to the previously described problems with the combination of the TaskMP gating coefficients. The technique makes it possible to analyse the best expert mapping of a TaskMP combination, independent of the gating coefficients.
Every product is transformed into a resulting normal distribution with new mean, variance and normalization-factor (see Section 4.1). The mean represents the trajectory weights $\boldsymbol{w}_{\text{new}}$ for the considered test-points $\boldsymbol{x}_1$ and $\boldsymbol{x}_2$. These weights can be transformed into joint-angles and task-space trajectories from which the corresponding achieved task $\boldsymbol{x}_{\text{new}}$ can be extracted (see Section 3.5). To compare the desired task $\boldsymbol{x}$ and the achieved task $\boldsymbol{x}_{\text{new}}$ the mean squared error of both tasks is calculated, similar to the test cycle of Section 3.5.
These steps are repeated for every expert product. The expert combination which produces the lowest error for a given test point $\boldsymbol{x}$ is chosen as the best evaluated expert combination.

The described method estimates the best possible results for the combination of two separately trained TaskMPs, due to the evaluation of every single expert combination. Because of the testing of all possible combinations, the method is impractical to use. Already a small number of mixture components highly increases the computational effort to evaluate the combinations. Nevertheless, the analysis gives an idea about the potential of a TaskMP combination and the differences to a gated version. A gated version would be a more practical approach which includes the training of a separate gating-network, which analyses and selects the different expert-combinations.

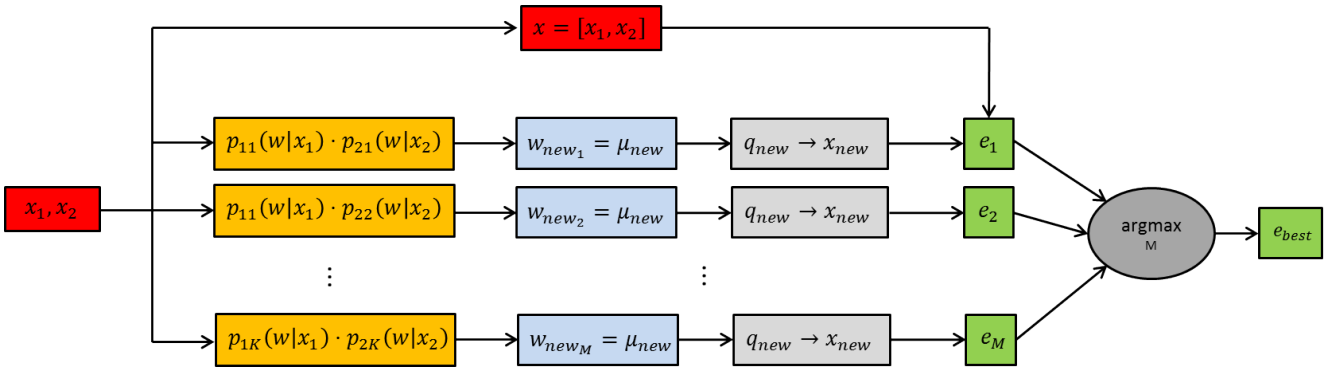

Figure 10: Evaluation of a product of two TaskMPs without gating-network: The product of two task dependent movement primitives results in a larger mixture model with $K \cdot K$ components. The resulting mixture component of every expert combination is generated. The mean of this resulting expert is defined as the new weight-vector. Similar to the test-cycle of a TaskMP (see 3.5), the weight-vectors are re-transformed to joint-angles q_{new} and the task-space trajectories into resulting tasks $\boldsymbol{x}_{\text{new}}$. The expert combination which produces the smallest mean squared error is chosen as the selected model component.

4.3 Gating-Network

As we have seen that the gating does not generalize well when combining two mixture of expert distributions, we first want to relearn the gating $p(k_{ij}|\boldsymbol{x})$ for each combination of mixture components. The EM-Algorithm, which can train such a gating network is similar to the previous one - used for the TaskMP-Learning.
The **E-Step** contains the responsibility calculation of every expert-combination, i.e.

$$p(k_{ij}|x_n, w_n) = \gamma_{ij}^n = \frac{p(k_{ij}|x_n) \cdot p(w_n|k_{ij}, x_n)}{\sum_{k_{ij}} p(k_{ij}|x_n) \cdot p(w_n|k_{ij}, x_n)}, \tag{43}$$

$$p(w_n|k_{ij}, x_n) = \frac{N(w_n|x_1^n\beta_i, \Sigma_{1i}) \cdot N(w_n|x_2^n\beta_j, \Sigma_{2j})}{N(x_1^n\beta_i|x_2^n\beta_j, \Sigma_{1i} + \Sigma_{2j})}. \tag{44}$$

The formulation differs to the TaskMP E-step in the additional expert distribution, the necessary normalization of the expert-combinations (see 4.1) and the larger amount of gating-coefficients $K_{ij} = K_i \cdot K_j$ (to cover combinations of expert i and expert j).

The normalization is needed to build a resulting probability distribution which sums up to one, according to the combination of Gaussian experts (see Section 4.1).

The **M-Step** involves only the maximization of the gating parameters (Eq. 45, 46). The expert-parameters are kept fixed.
The evaluation of the log-likelihood is identical to the TaskMP-Version.

$$\mu_{k_{ij}} = \frac{1}{N_{k_{ij}}} \sum_{n=1}^{N} \gamma_{nk_{ij}} \cdot x_n, \qquad \Sigma^g_{k_{ij}} = \frac{1}{N_{k_{ij}}} \sum_{n=1}^{N} \gamma_{nk_{ij}} \cdot (x_n - \mu_{k_{ij}})(x_n - \mu_{k_{ij}})^\top, \tag{45}$$

$$p(k_{ij}) = \frac{N_{k_{ij}}}{\sum_{k=1}^{K} N_{k_{ij}}}, \qquad N_{k_{ij}} = \sum_{n=1}^{N} \gamma_{nk_{ij}}. \tag{46}$$

5 Expert Learning of Mixture Product

In order to further improve the performance of the product of the mixture of experts (Section 4.1), we train the single experts from both mixture components to perform well when they are used for the combination.
Learning a product of two mixtures shares similarities to the product of two TaskMPs approach. Again the model contains two multiplied mixtures which are gated by a gating-network. The goal is to train the mixture and gating parameters together by a single process. This approach was introduced by Hinton [24]. In contrast to Hinton, we used Gaussian distribution to model the gating and experts and as result, no contrastive divergence function is needed for the training.
The composition of the task is different to the naive TaskMP-product. Since the previous approach used a pre-designed structure of the task, the main task was manually decomposed into the sub-task-variables $\boldsymbol{x_1}$ and $\boldsymbol{x_2}$. The training of individual TaskMPs have been performed on these pre-designed sub-tasks.
Our approach, should learn this task-decomposition. The experts of the new model are using both the main task $\boldsymbol{x}$ instead of the sub-tasks $\boldsymbol{x_1}$ and $\boldsymbol{x_2}$. The model parameters (especially the regression parameters $\boldsymbol{\beta}$) of the mixture product are controlling the influence of every dimension of $\boldsymbol{x}$ on the mapping process. The adaptation of these model parameters during the learning process should enforce a decomposition of the task-variable $\boldsymbol{x}$ into sub-tasks. Is this the case, the different experts of the mixtures have a different influence on the parts of $\boldsymbol{x}$, i.e. the different sub-task-variables. As a result, the task-decomposition has been learned.

5.1 Model of the Mixture Product

The model for learning the task decomposition is given by Eq.(47 - 50). The model $p(\boldsymbol{w}|\boldsymbol{x})$ maps the task-variables $\boldsymbol{x}$ to the corresponding joint-weights $\boldsymbol{w}$ and is defined by the product over all data points. The model $p(\boldsymbol{w}_n|\boldsymbol{x}_n)$ which maps a single task-variable $\boldsymbol{x}_n$ is a mixture model, similar to the TaskMP version, i.e.

$$p(\boldsymbol{w}_n|\boldsymbol{x}_n) = \sum_i \sum_j p(k_{ij}|\boldsymbol{x}_n) \cdot p(\boldsymbol{w}_n|k_{ij}, \boldsymbol{x}_n). \tag{47}$$

The expert model $p(\boldsymbol{w}_n|k_{ij}, \boldsymbol{x}_n)$ is the product of two different linear Gaussian experts, which are normalized by Z. Similar to the naive TaskMP combination of Section 4.1, the normalization is necessary to obtain a probability distribution (which sums up to one). The experts are fully defined by their parameters, the means $\boldsymbol{\mu}_{1i}$, $\boldsymbol{\mu}_{2j}$ and the co-variances $\boldsymbol{\Sigma}_{1i}$, $\boldsymbol{\Sigma}_{2j}$.
Although the model sums over the two expert-sets i and j, there is only one gating network, denoted by $p(k_{ij}|\boldsymbol{x}_n)$. It gates the expert-products and contains $k_{ij} = I \cdot J$ components (I and J are the number of components for the corresponding expert-sets).

$$p(\boldsymbol{w}_n|k_{ij}, \boldsymbol{x}_n) = \frac{p_{1i}(\boldsymbol{w}_n|\boldsymbol{x}_n) \cdot p_{2j}(\boldsymbol{w}_n|\boldsymbol{x}_n)}{Z} = \frac{N(\boldsymbol{w}_n|\boldsymbol{\mu}_{1i}(\boldsymbol{x}, \boldsymbol{\beta}_i), \boldsymbol{\Sigma}_{1i}) \cdot N(\boldsymbol{w}_n|\boldsymbol{\mu}_{2j}(\boldsymbol{x}, \boldsymbol{\beta}_j), \boldsymbol{\Sigma}_{2j})}{N(\boldsymbol{\mu}_{1i}|\boldsymbol{\mu}_{2j}, \boldsymbol{\Sigma}_{1i} + \boldsymbol{\Sigma}_{2j})}, \tag{48}$$

$$\text{with} \quad \boldsymbol{\mu}_{1i}(\boldsymbol{x}_n, \boldsymbol{\beta}_i) = \boldsymbol{x}_n \cdot \boldsymbol{\beta}_i, \quad \boldsymbol{\mu}_{2j}(\boldsymbol{x}_n, \boldsymbol{\beta}_j) = \boldsymbol{x}_n \cdot \boldsymbol{\beta}_j, \tag{49}$$

$$\text{and} \quad p(k_{ij}|\boldsymbol{x}_n) = \frac{p(k_{ij}) \cdot p(\boldsymbol{x}_n|k_{ij})}{\sum_{k_{ij}} p(k_{ij}) \cdot p(\boldsymbol{x}_n|k_{ij})}. \tag{50}$$

5.2 Training of Mixture Product

We again use the expectation maximization algorithm to train this mixture model, as described in Section 3.2. Due to the normalization term Z of Eq. (48), the training process is more complicated. It is not possible to obtain a closed form solution by deriving the corresponding Q-function. The E- and M-Step of the expectation maximization algorithm are defined as follows.

E-Step:

The E-Step calculates the responsibilities of the mixture components for every training-example (the task-variable $\boldsymbol{x}_n$). The equations for calculating $\boldsymbol{\gamma}_{ij}^n$ are identical to the responsibility calculation of the separate gating network, trained for a TaskMP product, as we proposed in Section 4.3, Eq. (43).

M-Step:

During the M-Step the Q-Functions will be maximized. The function is defined by the sum of responsibilities multiplied with the cluster likelihood, i.e.

$$p(\boldsymbol{w}_n, k_{ij}|\boldsymbol{x}_n) = p(k_{ij}|\boldsymbol{x}_n) \cdot p(\boldsymbol{w}_n|k_{ij}, \boldsymbol{x}_n). \tag{51}$$

The responsibilities $\boldsymbol{\gamma}_{ij}$ are pre-calculated by the E-Step and fixed during the maximization. The development of the Q-function is as follows

$$\begin{aligned} Q(\boldsymbol{\theta}, \boldsymbol{\theta}^{old}) &= \sum_{k_{ij}} p(k_{ij}|\boldsymbol{x}, \boldsymbol{w}_n, \boldsymbol{\theta}^{old}) \cdot \ln p(\boldsymbol{w}, k_{ij}|\boldsymbol{x}, \boldsymbol{\theta}) \\ &= \sum_{k_{ij}} \boldsymbol{\gamma}_{ij} \cdot \ln p(\boldsymbol{w}, k_{ij}|\boldsymbol{x}, \boldsymbol{\theta}) \\ &= \sum_{n=1}^{N} \sum_{k_{ij}} \boldsymbol{\gamma}_{ij}^n \ln p(k_{ij}|\boldsymbol{x}_n) + \sum_{n=1}^{N} \sum_{k_{ij}} \boldsymbol{\gamma}_{ij}^n \ln p(\boldsymbol{w}|k_{ij}, \boldsymbol{x}_n) \qquad (52) \\ &\rightarrow = \text{Gating-Cost-Function} + \text{Expert-Cost-Function} = l_1 + l_2, \qquad (53) \end{aligned}$$

which results in two different cost-functions Eq. (52 - 53), one for the gating network l_1 and one for the expert-model l_2. The gating network part can be maximized by the known Maximum Likelihood results form Section 4.3 .
The expert-part derives as follows

$$\begin{aligned} l_2 &= \sum_{n=1}^{N} \sum_{k_{ij}} \boldsymbol{\gamma}_{ij}^n \ln p(\boldsymbol{w}|k_{ij}, \boldsymbol{x}_n) \\ &= \sum_{n=1}^{N} \sum_{k_{ij}} \boldsymbol{\gamma}_{ij}^n \ln \left(\frac{N(\boldsymbol{w}_n|\boldsymbol{\mu}_{1i}(\boldsymbol{x}, \boldsymbol{\beta}_i), \boldsymbol{\Sigma}_{1i}) \cdot N(\boldsymbol{w}_n|\boldsymbol{\mu}_{2j}(\boldsymbol{x}, \boldsymbol{\beta}_j), \boldsymbol{\Sigma}_{2j})}{N(\boldsymbol{\mu}_{1i}|\boldsymbol{\mu}_{2j}, \boldsymbol{\Sigma}_{1i} + \boldsymbol{\Sigma}_{2j})} \right) \\ &= \sum_i \sum_j \sum_{n=1}^{N} \boldsymbol{\gamma}_{ij}^n \frac{1}{2} [-p \ln(2\pi) - \ln(\det(\boldsymbol{\Sigma}_{1i})) - (\boldsymbol{w}_n - \boldsymbol{\mu}_{1i}(x))^\top \boldsymbol{\Sigma}_{1i}^{-1} (\boldsymbol{w}_n - \boldsymbol{\mu}_{1i}(x)) \\ &\qquad - p \ln(2\pi) - \ln(\det(\boldsymbol{\Sigma}_{2j})) - (\boldsymbol{w}_n - \boldsymbol{\mu}_{2j}(x))^\top \boldsymbol{\Sigma}_{2j}^{-1} (\boldsymbol{w}_n - \boldsymbol{\mu}_{2j}(x)) \\ &\qquad + p \ln(2\pi) + \ln(\det(\boldsymbol{\Sigma}_{1i} + \boldsymbol{\Sigma}_{2j})) + (\boldsymbol{\mu}_{1i}(x) - \boldsymbol{\mu}_{2j}(x))^\top (\boldsymbol{\Sigma}_{1i} + \boldsymbol{\Sigma}_{2j})^{-1} (\boldsymbol{\mu}_{1i}(x) - \boldsymbol{\mu}_{2j}(x))]. \qquad (54) \end{aligned}$$

This cost-function can't be solved analytically, due to the normalization term $N(\boldsymbol{\mu}_{1i}|\boldsymbol{\mu}_{2j}, \boldsymbol{\Sigma}_{1i} + \boldsymbol{\Sigma}_{2j})$. The normalization part of the expert model incorporates additional versions of the means $\boldsymbol{\mu}_{1i}$, $\boldsymbol{\mu}_{2j}$ and the variances $\boldsymbol{\Sigma}_{1i}$, $\boldsymbol{\Sigma}_{2j}$, which prevent an explicit solution of l_2 from Eq.(54).

An optimization method must be used to get the model parameters which maximizes the corresponding cost-function. Instead of using only the general cost-function equation, a gradient descent approach speeds up the optimization process. To provide the necessary gradient for the optimization, the gradient of all relevant model parameters must be calculated. We illustrate our approach for one model parameter. The gradient of the mean $\boldsymbol{\mu}_{1i} = \boldsymbol{x}_n \cdot \boldsymbol{\beta}_{1i}$ can be derived as follows

$$\begin{aligned} \frac{\partial l_2}{\partial \boldsymbol{\mu}_{1i}} &= \sum_j \sum_{n=1}^{N} \frac{\partial}{\partial \boldsymbol{\mu}_{1i}} [\frac{1}{2} \boldsymbol{\gamma}_{ij}^n [-(\boldsymbol{w}_n - \boldsymbol{\mu}_{1i}(x))^\top \boldsymbol{\Sigma}_{1i}^{-1} (\boldsymbol{w}_n - \boldsymbol{\mu}_{1i}(x)) \\ &\qquad + (\boldsymbol{\mu}_{1i}(x) - \boldsymbol{\mu}_{2j}(x))^\top (\boldsymbol{\Sigma}_{1i} + \boldsymbol{\Sigma}_{2j})^{-1} (\boldsymbol{\mu}_{1i}(x) - \boldsymbol{\mu}_{2j}(x))]] \\ &= \sum_j \sum_{n=1}^{N} \boldsymbol{\gamma}_{ij}^n [\boldsymbol{\Sigma}_{1i}^{-1} \boldsymbol{w}_n + ((\boldsymbol{\Sigma}_{1i} + \boldsymbol{\Sigma}_{2j})^{-1} - \boldsymbol{\Sigma}_{1i}^{-1}) \boldsymbol{\mu}_{1i} - (\boldsymbol{\Sigma}_{1i} + \boldsymbol{\Sigma}_{2j})^{-1} \boldsymbol{\mu}_{2j}] = 0. \qquad (55) \end{aligned}$$

The calculation of the gradients of the covariances are more complex, due to the use of cholesky decomposition. The variances Σ_{1i} and Σ_{2j} are represented as follows,

$$\boldsymbol{\Sigma}_{1i} = \boldsymbol{A}_{1i}^\top \boldsymbol{A}_{1i} + \exp(a_{1i}) \cdot \boldsymbol{I}, \qquad \boldsymbol{\Sigma}_{2j} = \boldsymbol{A}_{2j}^\top \boldsymbol{A}_{2j} + \exp(a_{2j}) \cdot \boldsymbol{I}. \tag{56}$$

The product of triangular matrices $\boldsymbol{A}_{1i}$ and $\boldsymbol{A}_{2j}$ in combination with the additional factors $\exp(a_{1i})$ and $\exp(a_{2j})$, ensures symmetric and positive definite co-variance matrices $\boldsymbol{\Sigma}_{1i}$ and $\boldsymbol{\Sigma}_{2j}$ during the optimization process. A direct optimization of the co-variance entries would involve the danger of breaking this necessary constraints. According to this structure,

the gradient of the sub-terms $\boldsymbol{A}_{1i}$, $\boldsymbol{A}_{2j}$, a_{1i} and a_{2j} must be calculated to provide the complete gradient of l_2 for the optimization. Detailed explanations about the gradient calculations can be found in the Appendix.

The optimization calculation is implemented by the interior-point algorithm of the matlab function "fmincon". The calculated gradient equations are provided. An constraint optimization is necessary to keep the triangle structure of $\boldsymbol{A}_{1i}$ and $\boldsymbol{A}_{2j}$. A unconstrained optimization would change all parameters of $\boldsymbol{A}_{1i}$ and $\boldsymbol{A}_{2j}$, although one half of the triangle matrices must stay zero.

The optimization process is computational very expensive. The learning process with a small number of mixture components and training data, needs already a large amount of computational time and iterations to converge.

Figure 11-left shows the Q-function values of the expert part over several iterations. To limit the computational effort only five iterations have been used for a single maximization step. This iteration limit is compensated by a larger number of iterations of the EM-Algorithm. The results show continuously increasing Q-function evaluations. The expert parameters seems to improve.

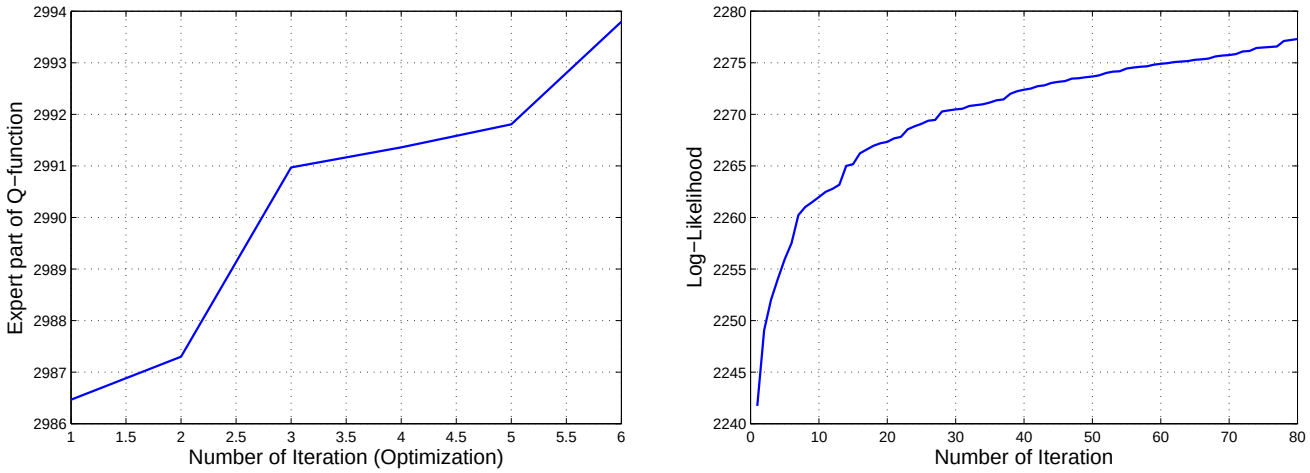

Figure 11: (**left**) Evaluations of the expert part of the Q-function (l_2) during the optimization. Only five iterations are used for maximizing the expert parameters, to keep the computational effort within scope. The reduced number of optimization iterations is compensated by the larger number of EM-Steps. The Q-function evaluations are continuously maximized. This indicates an improvement of the expert parameters. (**right**) Progress of the complete data log-likelihood of an expert learning approach with two components per expert-set. The likelihood progresses is continuously increasing. This indicates a successful learning process.

This is indicated in Figure 11-right, by the Q-function. The general Q-function represents the complete data-log-likelihood and is also used for the evaluation of the training-process. The training process can be stopped if the log likelihood values have converged. The likelihood of Figure 11 is continuously monotonic increasing, which indicates a successful learning process. Two mixture components have been used per expert-set.

Similar to a single TaskMP, the learning process is very sensitive to the initialization. We used a combination of independently trained TaskMPs and a separate learned gating network to initialize the model parameter for the expert learning process.

6 Experiments

We evaluate our proposed methods in a variety of different tasks, including reaching a certain position or orientation with the end-effector. The form and execution of the task has a high influence on the error-behaviour of a single mixture and the combined TaskMPs. Two highly correlated variables which cover only a small area of the task-space, decreases the level of difficulty for the learning process. Due to this observation, the error difference between a single TaskMP and a TaskMP-Combination might be small. A task with two as much as possible uncorrelated variables with large variance, renders the learning process more difficult and increases the error difference between a single TaskMP and a TaskMP combination.

Reaching random points located on a vertical line in the task-space (constant x-coordinate) and taking different orientations with the end-effector of a 20 link robot arm simulator (see Figure 12-left), results in a complicated task. The robot links have a length of one meter. Only a two dimensional example was considered, i.e. the robot can move only in the x,y plane. Reaching certain y-positions with the end-effector and link 11 represents the main task (Task1). According to this, the y-coordinate of the end-effector and the 11th link have been defined as the main-task-variable of these experiments. Plotting the dependency between both variables (see Figure 12-right) shows a low correlation between y_1 and y_2 with a high parameter variance.
For the following evaluations, 500 training points and 100 test points have been used. The error calculations have been averaged over five trials.

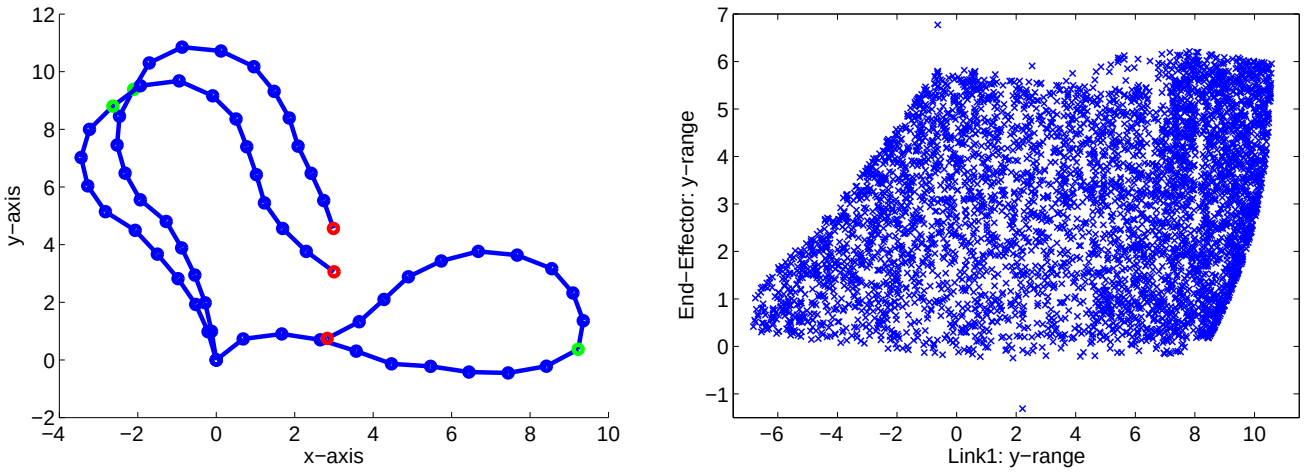

Figure 12: (**left**) Examples of robot configuration for the task reaching certain y-coordinates with the end-effector (red) and link 11 for a constant x-coordinate. The y-coordinates of link 11 (marked by the green joint) and the end-effector building the desired task-variable. (**right**) Dependency between y-coordinates (of reached points in task space) of end-effector and the considered link 11. The correlation is low, whereas the variance over both axis is high. Such a task set-up renders learning a single TaskMP difficult as many mixture components are needed. Hence, the combination of MPs has the potential to considerably improve the performance of the tasks.

6.1 Single TaskMP

The first part of the analysis considers the movement representation of the developed task dependent movement primitives. Hence, the mapping of the previously described task to the joint-angles with a single TaskMP is evaluated by calculating the mean square error over an increasing number of mixture components (see Figure 13). The error is calculated by the difference between the given task-variable (y-coordinates of link 11 and end-effector) and the mapped corresponding task-point. A separate test data-set as well as the training-data have been used to calculate the error. Figure 13-left shows a general plot of the error-progress, whereas Figure 13-right represents a magnified view. This graph starts with a mixture component number of 15 and also contains error bars showing the normalized error. The error is decreasing with an increasing number of experts. The training-error converges to 0.36 and the test-error to 0.39.

Figure 14 shows the results of a TaskMP with a different task (Task2), to enable a better understanding of the mapping quality. The position and orientation of the end-effector of an seven link robot arm at a certain via-point have been used as task-variables. The training and test error have been calculated for an increasing number of mixture components. The error is decreasing with the number of components. The training error converges to a value of $3.21 \cdot 10^{-5}$ while the

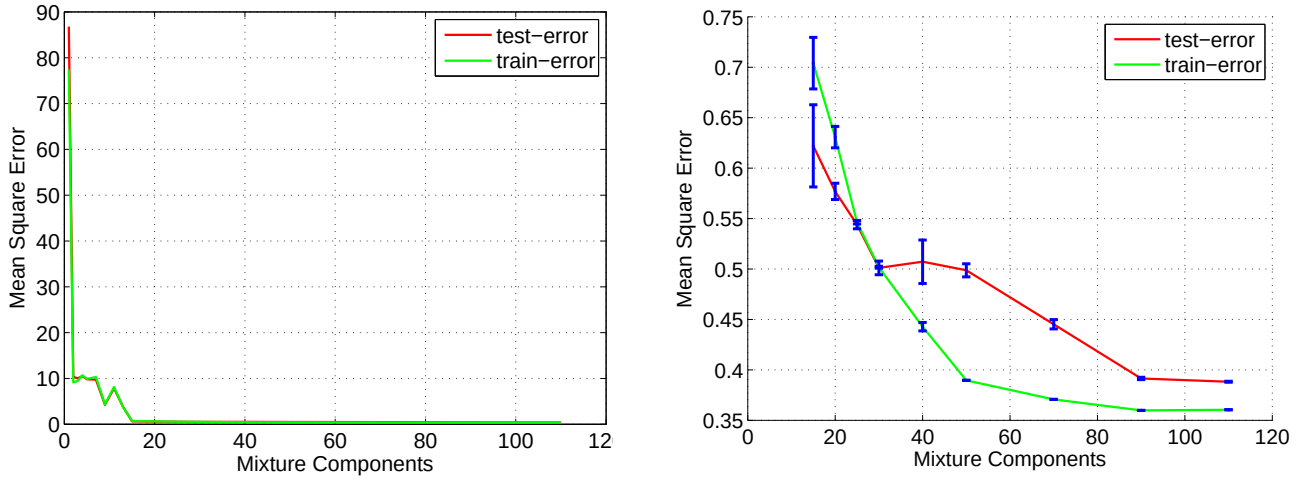

Figure 13: (**left**) Mean square error between test-points and re-constructed reaching-points over an increasing number of mixture components for the task, reaching y-coordinates with end-effector for constant x-coordinates. The error of the single TaskMP is calculated by the used training data and separate test-points. The error over both data-sets decreases significantly until 15 mixture-components. (**right**) Magnified plot of the left figure with error-bars. The plot starts with 15 mixture components. The error of the test-set decreases until 0.39 similar to the error of the training-set, which converges to 0.36.

test error has a minimum of $2.78 \cdot 10^{-4}$ at fifty mixture components. More components result in a larger error. This highly indicates over-fitting, since the test error increases whereas the training error is still decreasing. The amount of training samples (500 points) seems to be too small for more than 50 mixture components. A successful training with more mixture components would need more training data. The error and variance along all component points is small in compare with the previous task.

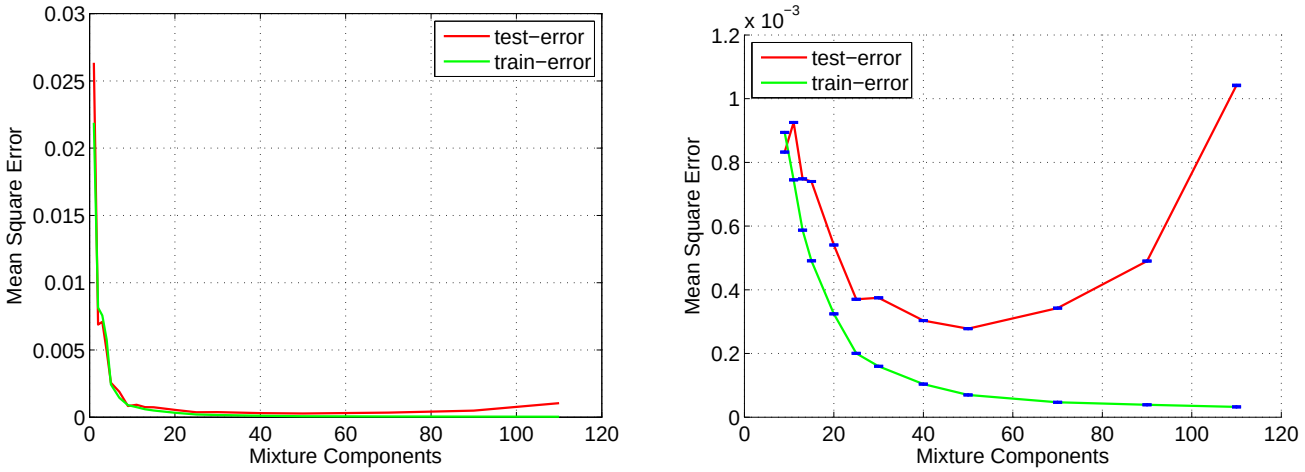

Figure 14: (**left**) Mean square error between the desired task variables (Task2) and the actual achieved task variables. The x- and y coordinates, as well as the orientation define the task-variable. The error is computed by the used training-data (green line) and separate test-data (red line). The error for both data-sets decreases rapidly until ten mixture components. (**right**) Magnified Plot of previously considered error-progress with error bars. Test- and training-set error are shown from 10 to 110 (maximum used) mixture components. The error of the test data is decreasing to an error of $2.78 \cdot 10^{-4}$ until 50 mixture components. The error increases with more components, which expresses over fitting. More training-data are needed to provide a appropriate data-fitting. The error of the training-data converges to $3.21 \cdot 10^{-5}$ over all used mixture components. According to this results, a small number of mixture components is enough to reach a sufficient mapping.

The comparison of Task1 (reach y-coordinates) and Task2 (reach position and orientation), shows the high influence of task-difficulty on the error-progress. The error of Task1 is higher than for Task2. A larger amount of mixture com-

ponents is necessary to observe a converging error on Task1. Due to this results, a single mixture model seems to be inefficient for tasks with a higher level of difficulty (like for Task1), according to the larger amount of needed mixture components. For Task2, already a small number of mixture components (10 experts) is sufficient to accurately learn the task. Therefore, we could not see an advantage in the combination of the ProMPs, as the reduction of the number of mixture components is not too significant. From these results, we conclude that Task1 is better suited to show the advantages of a combination of ProMPs. From now on we will only use Task1.

6.2 Combination of TaskMPs

After analysing the error behaviour of a single task dependent movement primitive, a combination of two TaskMPs is considered. We first evaluate the gain from the combination of independent trained TaskMPs. Subsequently we will look into optimizing the TaskMPs directly for the combination.
We evaluate a combination of two independently trained TaskMPs on Task1, in terms of the mean squared error over an increasing number of mixture components and we present the results in Figure 15. The number of components is related to the number of experts used for a single TaskMP where both TaskMPs containing an identical number of mixture components. For the combination of two TaskMPs, we require twice the data amount for the same number of components of a single TaskMP. For example, three components per TaskMP results in a general data amount of six components.

Three error-evaluations have been done and the results are presented in Figure 15-left. The first evaluation (red-line), results from a separately trained gating-network, which chooses the expert-combinations of the multiplied TaskMPs according to the combined main-task (the single TaskMPs have been trained on the sub-tasks y-position of link 11 and y-position of end-effector). For the second evaluation (blue line), a Nearest Neighbour classifier (K-NN) have been used to choose the described experts. A set of previously evaluated best-expert-combinations was used as references for this nearest neighbour approach. All expert combinations have been evaluated on a considered test-point. The combination that produces the smallest error was chosen as the best evaluated expert-combination, as described in Section 4.2. The green-line represents the error of these best evaluated expert-combinations.
The TaskMP-combination which is gated by a trained gating-network has a larger error compared to the KNN-gated version and the best evaluated expert-combinations. The KNN version is close to the error of the best-expert-combination.
Figure 15-right shows the different errors for Gaussian gated and best expert evaluated versions, calculated by a test-set and the used training-set. Test and training error are close to each other and decrease with more mixture components. No over fitting is observable.

According to these results, the gating by separately trained Gaussian mixture model seems to be insufficient for the selection of the best-expert combination for the considered task. Figure 16 shows the decision surfaces of the gating-network (right) and the best evaluated experts (left) according to the given task-data of link and end-effector position (y1 and y2). The mapping of the gating-network between the expert-combinations and the given task variables produces a tile-structure for the decision boundaries (uniformly distributed). For the presented results five experts are used.
The decision-surface of the optimal expert-combinations (which combination is the best for the given task y1/y2), results in a more complex structure with seven expert-combinations. Some expert combinations cover multiple zones of the task-space. The shape and the size of the occupied areas differ significantly, with the related used expert-combinations (non-uniformly distributed). A comparison of both decision-surfaces shows that only a few areas of the gating-network version coincides with the surface of the best-experts.
As a result, the gating-network approach can not cope up with the classification problem of selecting the best expert-combination. This is the main reason for the large error difference between gated and optimal expert-combinations. An appropriate gating-method would lead to solutions which are close to the optimal expert evaluation.

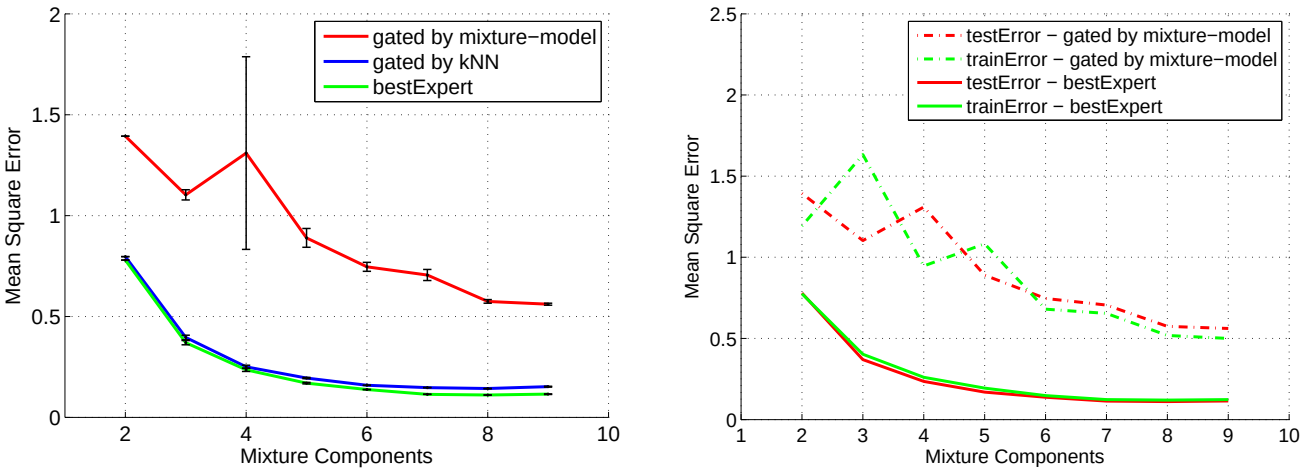

Figure 15: (**left**) Mean square error of a TaskMP product over an increasing number of mixture components per mixture model. The error is computed over different test points where the TaskMPs have been trained independently. One for reaching the desired y-coordinate with the end-effector and the other one for reaching the desired y-coordinate with link 11. Three versions of this mixture combination have been evaluated: One with a separately trained Gaussian gating network (red). Another combination is gated by a KNN nearest neighbour classifier (blue), which references to a set of best evaluated expert combinations. The last version represents the error of the best expert combination (green). All errors are decreasing with a higher number of mixture components. The error of the version with Gaussian gating is significantly higher than the error of the KNN gated version and the best expert combination. The last two are close to each other over a high range of mixture components. This difference between the version with Gaussian gating and the KNN gated version indicates the classification problem that we faced. A Gaussian mixture model do not cope up with the observed gating problem. (**right**) Comparison of mean square error between the test-data and the training-data for the Gaussian gated version and the best evaluated expert combinations. Training and test error are close for both versions and decrease with more mixture components. No over-fitting is observable.

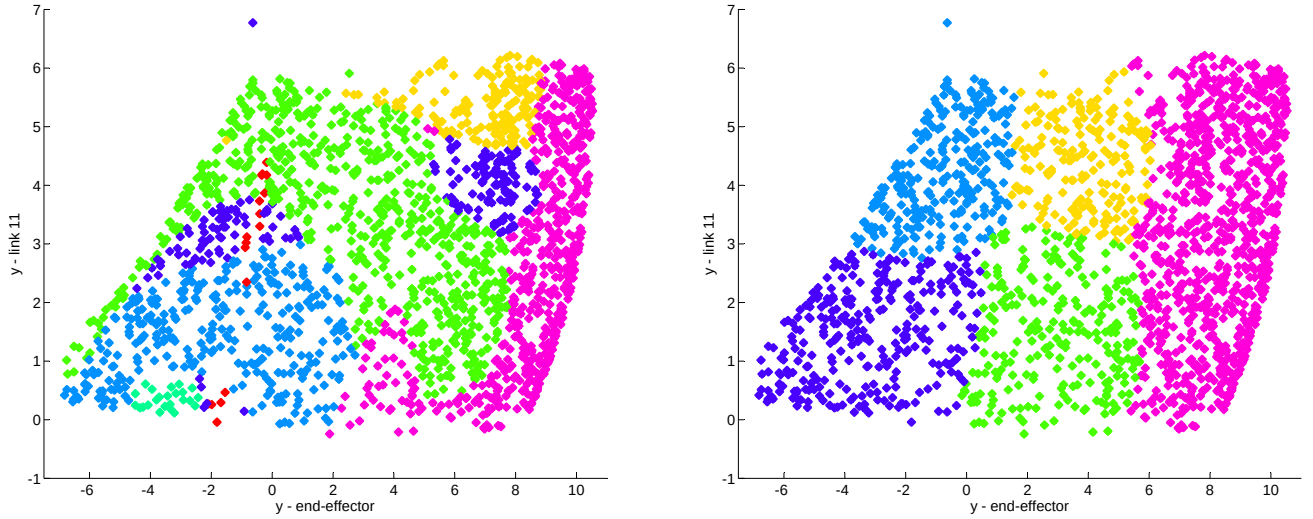

Figure 16: (**left**) Decision surface of the best evaluated expert-combination for a considered task-variable x containing the y coordinates of end-effector and link 11. The different colours correspond to the index of the expert-combinations. (**right**) Decision surface of the expert-combinations, which are gated by a separate trained Gaussian gating network. The decision surface of the gated version do not perform as well as the best evaluated version. The structure of the surface of the best expert-combination is much more complex than the gating-surface. The component areas of the gated surface do not agree with the areas of the best-expert surface. A possible explanation of the result would be that the Gaussian gating network is not appropriate for the given gating problem.

Comparison to single TaskMP

To analyse the quality of a TaskMP combination, we compare the described evaluations with a single mixture. The error of the KNN gated experts and optimal evaluated experts (Figure 15) converges to a range of ca. 0.11 to 0.15, with nine mixture-components for both TaskMPs. This error is considerably smaller than the best error observed on the previously described single TaskMP with a range of 0.39 for 110 components, despite less mixture-components have been used. The error of the single TaskMPs results also from the gated experts, which makes a direct comparison with a combination of two TaskMPs difficult. Nevertheless, the small error of the best evaluated expert-combination of two TaskMPs and the KNN version is a strong indication about the potential of such a combination approach. It remains to proof if a more effective gating approach, which selects the best expert-combinations, would be more effective than a single TaskMP for a corresponding "difficult" task.

6.3 Expert-Learning

In the previous section, a naive combination of two TaskMPs was analysed. Now we evaluate whether we can further improve these TaskMP combinations. Figure 17 illustrates the mean squared error of the learned TaskMP combination (Section 5). Independent trained TaskMPs and a separate learned Gaussian gating network have been used to initialize the learning process of the model. Different learning-conditions have been used for this approach than for the previous ones. Only 100 training points were used. The number of trajectory weights have been reduced to ten parameters by an principal component analysis, to decrease the computational expensive optimization process of the expert-learning.
Figure 17-left shows the error of the best evaluated expert combinations. The green line illustrates the error progress of the naive TaskMP combination, used for the initialization of the expert learning. The red line shows the error for the expert-learning approach (see Section 5). For both models the best expert combinations were used to assign the error described in Section 4.2. The error of the learned experts is lower than the error of the naive combination, except for the last data-point.
Figure 17-right shows the corresponding error for the Gaussian gated versions. The learned model is for four mixture components worse than the initialized model. The remaining data-points are either better or nearly unchanged. Both errors are increasing for three and four mixture components, which is a similar behaviour to Figure 15. The naive TaskMP combination has a high variance in this mixture component area. The gated error is also significantly higher than the error of the best expert combinations.

The small difference between the error of the naive TaskMP combination and the learned model, leads to the assumption that the naive TaskMP approach is already close to a, at least local, optimal solution. The problems with the Gaussian gating approach seems to be unaffected by the learning approach, due to the high error difference between gated model and the version with the best evaluated expert combinations.

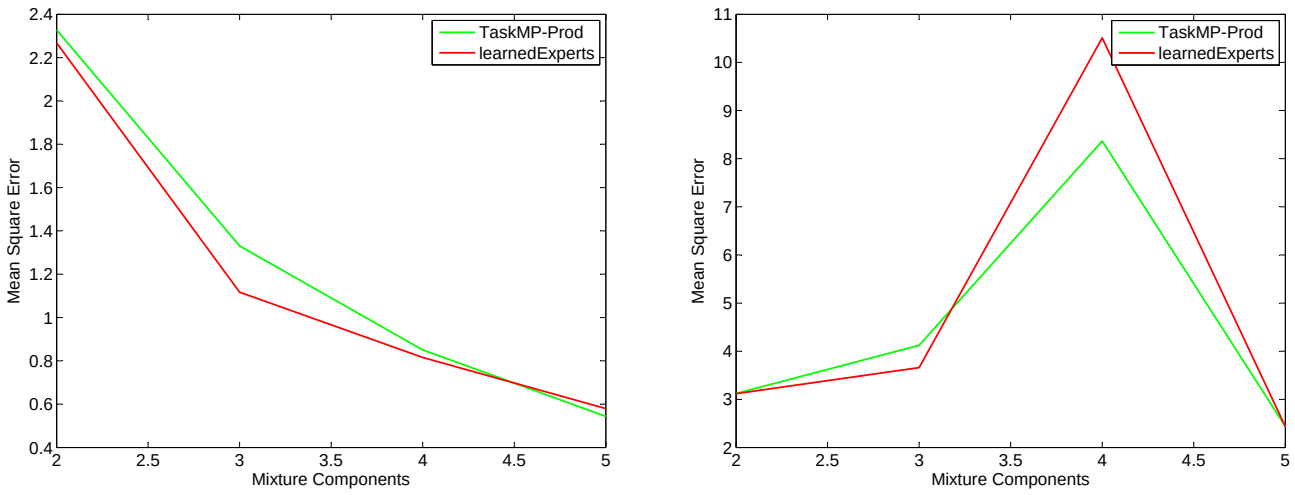

Figure 17: (**left**) Mean squared error of learned experts (red) and a naive TaskMP combination with separate learned gating network (green). The TaskMP combination and the trained gating have been used to initialize the expert learning process. The best expert combination have been evaluated and used for the error calculation. Both errors decrease with the amount of mixture components. Except for the last data-point, the structure learned error is smaller then the naive TaskMP combination. (**right**) Mean squared error of structure learning which is gated by a Gaussian gating network. The error is increasing and decreasing for different number of mixture components.

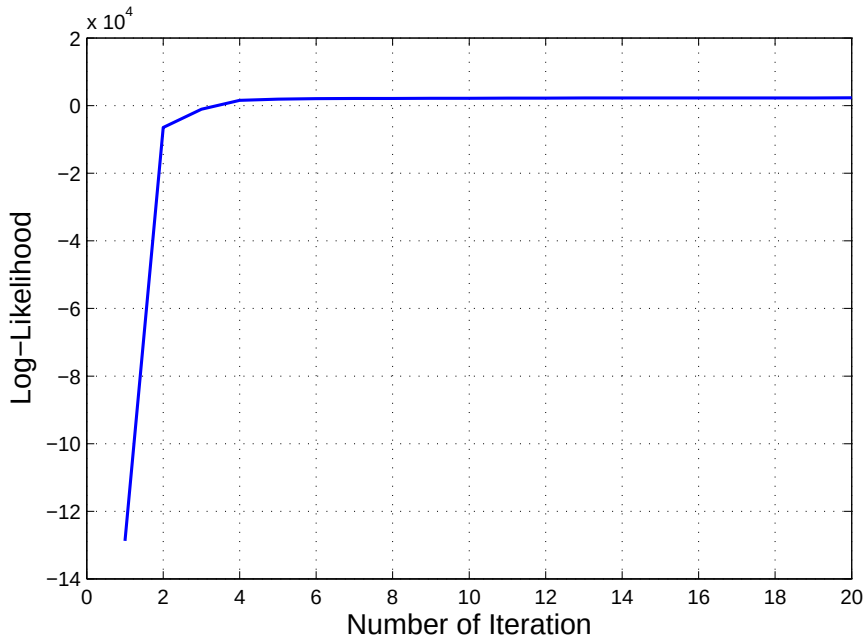

Figure 18: Log-likelihood progress of expert learning process over the number of iterations. The likelihood is converging after approximately five iterations of the EM-Algorithm.

Figure 18 shows the complete data log-likelihood for the expert learning process. The likelihood converges after five iterations of the EM-Algorithm. The optimization process during the M-Step involved five iteration steps. Three mixture components were used per expert-set. The continuously increasing log-likelihood indicates a successful learning process by the EM-Algorithm.

7 Conclusion and Future Work

During this work, we focused on the learning and combination of task dependent probabilistic movement primitives. We could successfully represent elemental tasks like reaching a certain position with an end-effector of a robot arm by these TaskMPs.
The implementation of the task-dependent primitives have been done by linear Gaussian mixture models. The evaluation of the described primitives has shown that the mapping quality of task dependent probabilistic movement primitives can be different according to the level of difficulty of the considered task. Complicated tasks, with a low correlation between the task-variables and a high variance, converge to a larger error and need more mixture components to be represented.

A part of this thesis, was the analysis of a naive combination of two TaskMPs. These TaskMPs were trained independently for two different sub-tasks. The idea was to solve a global task, composed from these sub-tasks by a combination of the trained primitives.
The analysis of this TaskMP combination has shown significant differences between the Gaussian gated version and the best evaluated expert combination. The error of the gated version is significantly higher than the error of the best experts. The used gating network does not cope up with the observed classification problem. This is underpinned by the error of a nearest neighbour classifier, which is close to the optimal error. Therefore a gating method which provides a better handling of the classification problem could lower the error to the area of best-expert combinations.
The error behaviour of these best expert combinations gives an idea about the potential of a TaskMP combination. The mapping quality of a pre-evaluated expert combination shows a lower error for less mixture components than for a single TaskMP. Hence less mixture components are needed to be involved and it requires less necessary training points. The two proposed versions are not directly comparable, due to the fact that the single TaskMPs are still gated by a Gaussian mixture model. Nevertheless, the results indicate that independently trained TaskMPs are successfully combinable to solve an overall task.

In the last part of the thesis, we implemented the expert learning of a TaskMP combination as a product of mixtures. The goal was to improve a combination of two TaskMPs and to learn a task decomposition. The model structure is identical to the TaskMP-combination previously described, except that the overall task $\boldsymbol{x}$ was used for the expert pairs (intead of the sub-tasks $\boldsymbol{x_1}$ and $\boldsymbol{x_2}$). The idea is to train all model parameters (gating and experts) by once. The model parameters of a TaskMP combination and a separate learned gating network (to select these expert-combinations) was used to initialize the expert learning process.
The expert learning of the TaskMP combination resulted in a small improvement of the initialized TaskMP combination. According to this small enhancement, the naive TaskMP combination seems to be close to a locally optimal solution. The learning of the task-decomposition did not succeed, since the results of the expert learning approach are close to the initialization with the TaskMP combination.

Future work must focus on the problems with the gating network to improve the performance of the TaskMP combination. Gating methods which can be considered are softmax approaches [20] or logistic sigmoid functions [22].
To improve the understanding of the generalisability of the described approaches, future work should include more complex tasks like a table tennis simulation or a walking robot.
Several modifications and extensions of the described TaskMP-Combinations are possible. The learning of single task dependent movement primitive as well as the structure learning of a mixture product is supervised. A promising idea would be to improve the delivered solutions of the primitives by reinforcement learning techniques.
Another interesting question is how to apply several experts to the combination task, instead of using the product of only two TaskMPs or experts for the expert learning.
The incorporation of a time-dependent activation function, which enables blending between different primitives and the development of an appropriate learning method which calculates these activation coefficients, can be based on this work.

References

[1] M. P. Deisenroth, G. Neumann, and J. Peters, "A survey on policy search for robotics," in *Foundations and Trends in Robotics*, 2013, pp. 11,12.

[2] A. Paraschos, C. Daniel, J. Peters, and G Neumann, "Probabilistic movement primitives," in *Advances in Neural Information Processing Systems (NIPS), Cambridge, MA: MIT Press.*, 2013.

[3] A. J. Ijspeert, J. Nakanishi, H. Hoffmann, P. Pastor, and S. Schaal, "Dynamical movement primitives: learning attractor models for motor behaviors," *Neural computation*, vol. 25, no. 2, pp. 328–373, 2013.

[4] E Bizzi, V. Cheung, A d'Avella, P Saltiel, and M. Tresch, "Combining modules for movement," *Brain Research Reviews*, vol. 57, no. 1, pp. 125–133, 2008.

[5] A. d Avella and M. C. Tresch, "Modularity in the motor system: decomposition of muscle patterns as combinations of time-varying synergies," *Advances in neural information processing systems*, vol. 1, pp. 141–148, 2002.

[6] A. d'Avella and E. Bizzi, "Shared and specific muscle synergies in natural motor behaviors," *Proceedings of the National Academy of Sciences of the United States of America*, vol. 102, no. 8, pp. 3076–3081, 2005.

[7] A. d'Avella, A. Portone, L. Fernandez, and F. Lacquaniti, "Control of fast-reaching movements by muscle synergy combinations," *The Journal of neuroscience*, vol. 26, no. 30, pp. 7791–7810, 2006.

[8] L. Sciavicco and L. Villani, *Robotics: modelling, planning and control*. Springer, 2009.

[9] J. Kober and J. Peters, "Movement templates for learning of hitting and batting," in *Learning Motor Skills*, Springer, 2014, pp. 69–82.

[10] J. Kober, B. Mohler, and J. Peters, "Learning perceptual coupling for motor primitives," in *Intelligent Robots and Systems, 2008. IROS 2008. IEEE/RSJ International Conference on*, IEEE, 2008, pp. 834–839.

[11] D.-H. Park, H. Hoffmann, P. Pastor, and S. Schaal, "Movement reproduction and obstacle avoidance with dynamic movement primitives and potential fields," in *Humanoid Robots, 2008. Humanoids 2008. 8th IEEE-RAS International Conference on*, IEEE, 2008, pp. 91–98.

[12] S. Degallier, L. Righetti, S. Gay, and A. Ijspeert, "Toward simple control for complex, autonomous robotic applications: combining discrete and rhythmic motor primitives," *Autonomous Robots*, vol. 31, no. 2-3, pp. 1–34, 2011.

[13] C. Daniel, G. Neumann, O. Kroemer, and J. Peters, "Learning sequential motor tasks," in *Proceedings of 2013 IEEE International Conference on Robotics and Automation (ICRA)*, 2013, pp. 1–7.

[14] G. Neumann, C. Daniel, A. Paraschos, A. Kupcsik, and J. Peters, "Learning modular policies for robotics," *Frontiers in Computational Neuroscience*, vol. 8, pp. 1–13, 2014.

[15] L. Sentis and O. Khatib, "Task-oriented control of humanoid robots through prioritization," in *IEEE International Conference on Humanoid Robots*, 2004.

[16] E. Todorov, "Compositionality of optimal control laws," in *Advances in Neural Information Processing Systems*, 2009, pp. 1856–1864.

[17] A. Ude, A. Gams, T. Asfour, and J. Morimoto, "Task-specific generalization of discrete and periodic dynamic movement primitives," *Robotics, IEEE Transactions on*, vol. 26, no. 5, pp. 800–815, 2010.

[18] A. G. Kupcsik, M. P. Deisenroth, J. Peters, and G. Neumann, "Data-efficient generalization of robot skills with contextual policy search.," in *AAAI*, 2013.

[19] C. M. Bishop, *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Secaucus, NJ, USA: Springer-Verlag New York, Inc., 2006, pp. 111, 423–455, 667–670.

[20] M. I. Jordan and R. A. Jacobs, "Hierarchical mixtures of experts and the em algorithm," *Neural computation*, vol. 6, no. 2, pp. 181–214, 1994.

[21] D. S. Schaal, "Lectureslides - the em-algorithm for supervised learning," in *Lecture: Neural Computation with Artificial Neural Network*, University of Southern California, 2008.

[22] C. M. Bishop and M. Svenskn, "Bayesian hierarchical mixtures of experts," in *Proceedings of the Nineteenth conference on Uncertainty in Artificial Intelligence*, Morgan Kaufmann Publishers Inc., 2002, pp. 57–64.

[23] K. B. Petersen and M. S. Pedersen, "The matrix cookbook," *Technical University of Denmark*, pp. 7–13,18,41, 2008.

[24] G. E. Hinton, "Training products of experts by minimizing contrastive divergence," *Neural computation*, vol. 14, no. 8, pp. 1771–1800, 2002.

A Appendix:

Detailed overview to expectation and maximization step of the expert learning of a mixture product (Section 5).
The **E-step** contains the calculations of the responsibility matrix γ_{ij}^n to evaluate the responsibilities of the different expert-combinations to the data-points, i.e.

$$p(k_{ij}|\boldsymbol{x}_n, \boldsymbol{w}_n) = \gamma_{ij}^n = \frac{p(k_{ij}|\boldsymbol{x}_n) \cdot p(\boldsymbol{w}_n|k_{ij}, \boldsymbol{x}_n)}{\sum_{k_{ij}} p(k_{ij}|\boldsymbol{x}_n) \cdot p(\boldsymbol{w}_n|k_{ij}, \boldsymbol{x}_n)}.$$

The **M-Step** results from the derivation of the Q-function. This function represents the complete data-log-likelihood and is constructed as follows

$$\begin{aligned}
Q(\boldsymbol{\theta}, \boldsymbol{\theta}^{old}) &= \sum_{k_{ij}} p(k_{ij}|\boldsymbol{x}, \boldsymbol{w}_n, \boldsymbol{\theta}^{old}) \cdot \ln p(\boldsymbol{w}, k_{ij}|\boldsymbol{x}, \boldsymbol{\theta}) \\
&= \sum_{k_{ij}} \gamma_{ij} \cdot \ln p(\boldsymbol{w}, k_{ij}|\boldsymbol{x}, \boldsymbol{\theta}) \\
&= \sum_{k_{ij}} \gamma_{ij} \cdot \ln[\prod_{n=1}^{N} p(k_{ij}|\boldsymbol{x}_n) \cdot p(\boldsymbol{w}|k_{ij}, \boldsymbol{x}_n)] \\
&= \sum_{n=1}^{N} \sum_{k_{ij}} \gamma_{ij}^n \left(\ln p(k_{ij}|\boldsymbol{x}_n) + \ln p(\boldsymbol{w}|k_{ij}, \boldsymbol{x}_n)\right) \\
&= \sum_{n=1}^{N} \sum_{k_{ij}} \gamma_{ij}^n \ln p(k_{ij}|\boldsymbol{x}_n) + \sum_{n=1}^{N} \sum_{k_{ij}} \gamma_{ij}^n \ln p(\boldsymbol{w}|k_{ij}, \boldsymbol{x}_n) \\
&\rightarrow = \text{Cost-Function_1} + \text{Cost-Function_2}.
\end{aligned}$$

Cost-function 1 enables the maximization of the gating-network. This can be done by the M-Step equations of section 4.3.

The second cost-function of the expert part enables the maximization of these model parameters. A detailed formulation of $L2$ is given by

$$\begin{aligned}
L2 &= \sum_{n=1}^{N} \sum_{k_{ij}} \gamma_{ij}^n \ln p(\boldsymbol{w}|k_{ij}, \boldsymbol{x}_n) \\
&= \sum_{n=1}^{N} \sum_{k_{ij}} \gamma_{ij}^n \ln \left(\frac{N(\boldsymbol{w}_n|\boldsymbol{\mu}_{1i}(\boldsymbol{x}, \boldsymbol{\beta}_i), \boldsymbol{\Sigma}_{1i}) \cdot N(\boldsymbol{w}_n|\boldsymbol{\mu}_{2j}(\boldsymbol{x}, \boldsymbol{\beta}_j), \boldsymbol{\Sigma}_{2j})}{N(\boldsymbol{\mu}_{1i}|\boldsymbol{\mu}_{2j}, \boldsymbol{\Sigma}_{1i} + \boldsymbol{\Sigma}_{2j})} \right) \\
&= \sum_{n=1}^{N} \sum_{k_{ij}} \gamma_{ij}^n \left[\ln N(\boldsymbol{w}_n|\boldsymbol{\mu}_{1i}(\boldsymbol{x}, \boldsymbol{\beta}_i), \boldsymbol{\Sigma}_{1i}) + \ln N(\boldsymbol{w}_n|\boldsymbol{\mu}_{2j}(\boldsymbol{x}, \boldsymbol{\beta}_j), \boldsymbol{\Sigma}_{2j}) - \ln N(\boldsymbol{\mu}_{1i}|\boldsymbol{\mu}_{2j}, \boldsymbol{\Sigma}_{1i} + \boldsymbol{\Sigma}_{2j}) \right] \\
&= \sum_{n=1}^{N} \sum_{k_{ij}} \gamma_{ij}^n \frac{1}{2} [-p \ln(2\pi) - \ln(\det(\boldsymbol{\Sigma}_{1i})) - (\boldsymbol{w}_n - \boldsymbol{\mu}_{1i}(x))^\top \boldsymbol{\Sigma}_{1i}^{-1} (\boldsymbol{w}_n - \boldsymbol{\mu}_{1i}(x)) \\
&\qquad - p \ln(2\pi) - \ln(\det(\boldsymbol{\Sigma}_{2j})) - (\boldsymbol{w}_n - \boldsymbol{\mu}_{2j}(x))^\top \boldsymbol{\Sigma}_{2j}^{-1} (\boldsymbol{w}_n - \boldsymbol{\mu}_{2j}(x)) \\
&\qquad + p \ln(2\pi) + \ln(\det(\boldsymbol{\Sigma}_{1i} + \boldsymbol{\Sigma}_{2j})) + (\boldsymbol{\mu}_{1i}(x) - \boldsymbol{\mu}_{2j}(x))^\top (\boldsymbol{\Sigma}_{1i} + \boldsymbol{\Sigma}_{2j})^{-1} (\boldsymbol{\mu}_{1i}(x) - \boldsymbol{\mu}_{2j}(x))] \\
&= \sum_{i} \sum_{j} \sum_{n=1}^{N} \gamma_{ij}^n \frac{1}{2} [-p \ln(2\pi) - \ln(\det(\boldsymbol{\Sigma}_{1i})) - (\boldsymbol{w}_n - \boldsymbol{\mu}_{1i}(x))^\top \boldsymbol{\Sigma}_{1i}^{-1} (\boldsymbol{w}_n - \boldsymbol{\mu}_{1i}(x)) \\
&\qquad - p \ln(2\pi) - \ln(\det(\boldsymbol{\Sigma}_{2j})) - (\boldsymbol{w}_n - \boldsymbol{\mu}_{2j}(x))^\top \boldsymbol{\Sigma}_{2j}^{-1} (\boldsymbol{w}_n - \boldsymbol{\mu}_{2j}(x)) \\
&\qquad + p \ln(2\pi) + \ln(\det(\boldsymbol{\Sigma}_{1i} + \boldsymbol{\Sigma}_{2j})) + (\boldsymbol{\mu}_{1i}(x) - \boldsymbol{\mu}_{2j}(x))^\top (\boldsymbol{\Sigma}_{1i} + \boldsymbol{\Sigma}_{2j})^{-1} (\boldsymbol{\mu}_{1i}(x) - \boldsymbol{\mu}_{2j}(x))].
\end{aligned}$$

The maximization of cost-function 2 has no closed form solution, due to the normalization part $N(\boldsymbol{\mu}_{1i}|\boldsymbol{\mu}_{2j}, \boldsymbol{\Sigma}_{1i} + \boldsymbol{\Sigma}_{2j})$. An optimization method must be used to calculate the model-parameters. To ensure a faster process, the gradient of L2 must be provided.

To simplify the understanding of the gradient calculation, a simplified example of L2 with three mixture components is given as follows:

$$
\begin{aligned}
\rightarrow f &= \sum_{n=1}^{N} \gamma_{ij}^{n} \frac{1}{2}[-p\ln(2\pi) - \ln(\det(\Sigma_{1i})) - (\boldsymbol{w}_n - \boldsymbol{\mu}_{1i}(x))^{\top} \ldots \\
L2 &= \sum_i \sum_j f(\mu_{1i}, \Sigma_{1i}, \boldsymbol{\mu}_{2j}, \Sigma_{2j}) \\
&= f(\boldsymbol{\mu}_{11}, \Sigma_{11}, \boldsymbol{\mu}_{21}, \Sigma_{21}) + f(\boldsymbol{\mu}_{11}, \Sigma_{11}, \boldsymbol{\mu}_{22}, \Sigma_{22}) + f(\boldsymbol{\mu}_{11}, \Sigma_{11}, \boldsymbol{\mu}_{23}, \Sigma_{23}) \\
&+ f(\boldsymbol{\mu}_{12}, \Sigma_{12}, \boldsymbol{\mu}_{21}, \Sigma_{21}) + f(\boldsymbol{\mu}_{12}, \Sigma_{12}, \boldsymbol{\mu}_{22}, \Sigma_{22}) + f(\boldsymbol{\mu}_{12}, \Sigma_{12}, \boldsymbol{\mu}_{23}, \Sigma_{23}) \\
&+ f(\boldsymbol{\mu}_{13}, \Sigma_{13}, \boldsymbol{\mu}_{21}, \Sigma_{21}) + f(\boldsymbol{\mu}_{13}, \Sigma_{13}, \boldsymbol{\mu}_{22}, \Sigma_{22}) + f(\boldsymbol{\mu}_{13}, \Sigma_{13}, \boldsymbol{\mu}_{23}, \Sigma_{23}).
\end{aligned}
$$

The first derivation is applied for the following parameters:

$$
\begin{aligned}
\boldsymbol{\mu}_{1i}(\boldsymbol{x}_n, \boldsymbol{\beta}_i) &= \boldsymbol{x}_n \cdot \boldsymbol{\beta}_i, \qquad & \Sigma_{1i}, \\
\boldsymbol{\mu}_{2j}(\boldsymbol{x}_n, \boldsymbol{\beta}_j) &= \boldsymbol{x}_n \cdot \boldsymbol{\beta}_j, \qquad & \Sigma_{2j}.
\end{aligned}
$$

The general derivation for the mean (of the example), results in the sum over the mixture components of the opponent expert:

$$
\begin{aligned}
\frac{\partial L_2}{\partial \boldsymbol{\mu}_{1i}} &= \sum_j \frac{\partial}{\partial \boldsymbol{\mu}_{1i}}[f(\boldsymbol{\mu}_{1i}, \Sigma_{1i}, \boldsymbol{\mu}_{2j}, \Sigma_{2j})], \\
\frac{\partial L_2}{\partial \boldsymbol{\mu}_{2j}} &= \sum_i \frac{\partial}{\partial \boldsymbol{\mu}_{2j}}[f(\boldsymbol{\mu}_{1i}, \Sigma_{1i}, \boldsymbol{\mu}_{2j}, \Sigma_{2j})].
\end{aligned}
$$

Example (μ_{11}):

$$
\begin{aligned}
\frac{\partial L_2}{\partial \boldsymbol{\mu}_{11}} &= \sum_j \frac{\partial}{\partial \boldsymbol{\mu}_{11}} |f(\boldsymbol{\mu}_{11}, \Sigma_{11}, \boldsymbol{\mu}_{2j}, \Sigma_{2j})| \\
\frac{\partial L_2}{\partial \boldsymbol{\mu}_{11}} &= \frac{\partial}{\partial \boldsymbol{\mu}_{11}} |f(\boldsymbol{\mu}_{11}, \Sigma_{11}, \boldsymbol{\mu}_{21}, \Sigma_{21}) + f(\boldsymbol{\mu}_{11}, \Sigma_{11}, \boldsymbol{\mu}_{22}, \Sigma_{22}) + f(\boldsymbol{\mu}_{11}, \Sigma_{11}, \boldsymbol{\mu}_{23}, \Sigma_{23})|
\end{aligned}
$$

The mean $\boldsymbol{\mu}_{1i}$ represents the product of the task $\boldsymbol{x}_n$ and the regression parameters $\boldsymbol{\beta}_{1i}$. The exact gradient calculation for $\boldsymbol{\mu}_{1i}$, derives as follows

$$
\begin{aligned}
\frac{\partial L_2}{\partial \boldsymbol{\mu}_{1i}} &= \sum_j \sum_{n=1}^{N} \frac{\partial}{\partial \boldsymbol{\mu}_{1i}} | \frac{1}{2} \gamma_{ij}^{n} [-(\boldsymbol{w}_n - \boldsymbol{\mu}_{1i}(x))^{\top} \Sigma_{1i}^{-1} (\boldsymbol{w}_n - \boldsymbol{\mu}_{1i}(x)) \\
&\qquad + (\boldsymbol{\mu}_{1i}(x) - \boldsymbol{\mu}_{2j}(x))^{\top} (\Sigma_{1i} + \Sigma_{2j})^{-1} (\boldsymbol{\mu}_{1i}(x) - \boldsymbol{\mu}_{2j}(x))] | \\
&= \sum_j \sum_{n=1}^{N} \frac{\gamma_{ij}^{n}}{2} [-(-2) \cdot (\Sigma_{1i}^{-1} (\boldsymbol{w}_n - \boldsymbol{\mu}_{1i}) + 2 \cdot (\Sigma_{1i} + \Sigma_{2j})^{-1} (\boldsymbol{\mu}_{1i} - \boldsymbol{\mu}_{2j})] \\
&= \sum_j \sum_{n=1}^{N} \gamma_{ij}^{n} [\Sigma_{1i}^{-1} \boldsymbol{w}_n - \Sigma_{1i}^{-1} \boldsymbol{\mu}_{1i} + (\Sigma_{1i} + \Sigma_{2j})^{-1} \boldsymbol{\mu}_{1i} - (\Sigma_{1i} + \Sigma_{2j})^{-1} \boldsymbol{\mu}_{2j}] \\
&= \sum_j \sum_{n=1}^{N} \gamma_{ij}^{n} [\Sigma_{1i}^{-1} \boldsymbol{w}_n + ((\Sigma_{1i} + \Sigma_{2j})^{-1} - \Sigma_{1i}^{-1}) \boldsymbol{\mu}_{1i} - (\Sigma_{1i} + \Sigma_{2j})^{-1} \boldsymbol{\mu}_{2j}] = 0,
\end{aligned}
$$

$$
\frac{\partial L_2}{\partial \boldsymbol{\mu}_{1i}^{n}(\boldsymbol{x}_n)} = \sum_j \gamma_{ij}^{n} [\Sigma_{1i}^{-1} \boldsymbol{w}_n + ((\Sigma_{1i} + \Sigma_{2j})^{-1} - \Sigma_{1i}^{-1}) \boldsymbol{mu}_{1i}^{n} - (\Sigma_{1i} + \Sigma_{2j})^{-1} \boldsymbol{\mu}_{2j}^{n}] = 0
$$

Parameter: $\boldsymbol{\mu}_{2j} = \boldsymbol{x}_n \cdot \boldsymbol{\beta}_{2j}$
The mean calculation of expert 2 corresponds to the mean-derivation of expert 1, as follows

$$\begin{aligned}
\frac{\partial L_2}{\partial \boldsymbol{\mu}_{2j}} &= \sum_i \sum_{n=1}^{N} \frac{\partial}{\partial \boldsymbol{\mu}_{2j}} | \frac{1}{2} \gamma_{ij}^n [-(\boldsymbol{w}_n - \boldsymbol{\mu}_{2j}(x))^\top \boldsymbol{\Sigma}_{2j}^{-1} (\boldsymbol{w}_n - \boldsymbol{\mu}_{2j}(x)) \\
&\qquad + (\boldsymbol{\mu}_{1i}(x) - \boldsymbol{\mu}_{2j}(x))^\top (\boldsymbol{\Sigma}_{1i} + \boldsymbol{\Sigma}_{2j})^{-1} (\boldsymbol{\mu}_{1i}(x) - \boldsymbol{\mu}_{2j}(x))] | \\
&= \sum_i \sum_{n=1}^{N} \frac{\gamma_{ij}^n}{2} [-(-2) \cdot (\boldsymbol{\Sigma}_{2j}^{-1} (\boldsymbol{w}_n - \boldsymbol{\mu}_{2j}) + (-2) \cdot (\boldsymbol{\Sigma}_{1i} + \boldsymbol{\Sigma}_{2j})^{-1} (\boldsymbol{\mu}_{1i} - \boldsymbol{\mu}_{2j})] \\
&= \sum_i \sum_{n=1}^{N} \gamma_{ij}^n [\boldsymbol{\Sigma}_{2j}^{-1} \boldsymbol{w}_n - \boldsymbol{\Sigma}_{2j}^{-1} \boldsymbol{\mu}_{2j} - (\boldsymbol{\Sigma}_{1i} + \boldsymbol{\Sigma}_{2j})^{-1} \boldsymbol{\mu}_{1i} + (\boldsymbol{\Sigma}_{1i} + \boldsymbol{\Sigma}_{2j})^{-1} \boldsymbol{\mu}_{2j}] \\
&= \sum_i \sum_{n=1}^{N} \gamma_{ij}^n [\boldsymbol{\Sigma}_{2j}^{-1} \boldsymbol{w}_n - (\boldsymbol{\Sigma}_{1i} + \boldsymbol{\Sigma}_{2j})^{-1} \boldsymbol{\mu}_{1i} + (\boldsymbol{\Sigma}_{2j}^{-1} - (\boldsymbol{\Sigma}_{1i} + \boldsymbol{\Sigma}_{2j})^{-1}) \boldsymbol{\mu}_{2j}] = 0
\end{aligned}$$

The gradient calculation of the co-variance matrices are more complicated, due to the choleski decomposed structure of the variance:

$$\begin{aligned}
\Sigma_{1i} &= \boldsymbol{A}_{1i}^\top \boldsymbol{A}_{1i} + \exp(a_{1i}) \cdot \boldsymbol{I}, \\
\boldsymbol{\Sigma}_{2j} &= \boldsymbol{A}_{2j}^\top \boldsymbol{A}_{2j} + \exp(a_{2j}) \cdot \boldsymbol{I}.
\end{aligned}$$

This structure is necessary to keep the covariance matrices symmetric and positive definite, during the optimization process of the M-Step.

To calculate the gradient of the parameters $\boldsymbol{A}_{1i}$, $\boldsymbol{A}_{2j}$, a_{1i} and a_{2j}, the (outer) derivation $\frac{\partial L_2}{\partial \boldsymbol{\Sigma}_{1i}}$ is needed, i.e.

Parameter: $\boldsymbol{\Sigma_{1i}}$

$$\begin{aligned}
\frac{\partial L_2}{\partial \boldsymbol{\Sigma}_{1i}} &= \sum_j \sum_{n=1}^{N} \frac{\partial}{\partial \boldsymbol{\Sigma}_{1i}} | \frac{1}{2} \gamma_{ij}^n [-\ln(\det(\boldsymbol{\Sigma}_{1i})) - (\boldsymbol{w}_n - \boldsymbol{\mu}_{1i}(x))^\top \boldsymbol{\Sigma}_{1i}^{-1} (\boldsymbol{w}_n - \boldsymbol{\mu}_{1i}(x)) \\
&\quad - \ln(\det(\boldsymbol{\Sigma}_{1i} + \boldsymbol{\Sigma}_{2j})) + (\boldsymbol{\mu}_{1i}(x) - \boldsymbol{\mu}_{2j}(x))^\top (\boldsymbol{\Sigma}_{1i} + \boldsymbol{\Sigma}_{2j})^{-1} (\boldsymbol{\mu}_{1i}(x) - \boldsymbol{\mu}_{2j}(x)) \\
&= \sum_j \sum_{n=1}^{N} \frac{\gamma_{ij}^n}{2} [-\boldsymbol{\Sigma}_{1i}^{-1} - \boldsymbol{\Sigma}_{1i}^{-1} \cdot (\boldsymbol{w}_n - \boldsymbol{\mu}_{1i})(\boldsymbol{w}_n - \boldsymbol{\mu}_{1i})^\top \cdot \boldsymbol{\Sigma}_{1i}^{-1} + [(\boldsymbol{\Sigma}_{1i} + \boldsymbol{\Sigma}_{2j})^{-1}]^\top \\
&\quad + (\boldsymbol{\Sigma}_{1i} + \boldsymbol{\Sigma}_{2j})^{-1} \cdot (\boldsymbol{\mu}_{1i} - \boldsymbol{\mu}_{2j})(\boldsymbol{\mu}_{1i} - \boldsymbol{\mu}_{2j})^\top \cdot (\boldsymbol{\Sigma}_{1i} + \boldsymbol{\Sigma}_{2j})^{-1}]
\end{aligned}$$

Sub-Derivatives ($\boldsymbol{\Sigma}_{1i}$):

$$\frac{\partial}{\partial \boldsymbol{\Sigma}_{1i}} | \ln(\det(\boldsymbol{\Sigma}_{1i})) | = (\boldsymbol{\Sigma}_{1i}^{-1})^\top = (\boldsymbol{\Sigma}_{1i}^\top)^{-1} = \boldsymbol{\Sigma}_{1i}^{-1}$$

$$\begin{aligned}
\frac{\partial}{\partial \boldsymbol{\Sigma}_{1i}} | (\boldsymbol{w}_n - \boldsymbol{\mu}_{1i})^\top \boldsymbol{\Sigma}_{1i}^{-1} (\boldsymbol{w}_n - \boldsymbol{\mu}_{1i}) | &= -(\boldsymbol{\Sigma}_{1i}^\top)^{-1} \cdot (\boldsymbol{w}_n - \boldsymbol{\mu}_{1i})(\boldsymbol{w}_n - \boldsymbol{\mu}_{1i})^\top \cdot (\boldsymbol{\Sigma}_{1i}^\top)^{-1} \\
&= -\boldsymbol{\Sigma}_{1i}^{-1} \cdot (\boldsymbol{w}_n - \boldsymbol{\mu}_{1i})(\boldsymbol{w}_n - \boldsymbol{\mu}_{1i})^\top \cdot \boldsymbol{\Sigma}_{1i}^{-1}
\end{aligned}$$

$$\begin{aligned}
\frac{\partial}{\partial \boldsymbol{\Sigma}_{1i}} | \ln(\det(\boldsymbol{\Sigma}_{1i} + \boldsymbol{\Sigma}_{2j})) | &= \frac{\partial \ln(\det(\boldsymbol{\Sigma}_{1i} + \boldsymbol{\Sigma}_{2j}))}{\partial \det(\boldsymbol{\Sigma}_{1i} + \boldsymbol{\Sigma}_{2j})} \cdot \frac{\partial \det(\boldsymbol{\Sigma}_{1i} + \boldsymbol{\Sigma}_{2j})}{\partial \boldsymbol{\Sigma}_{1i} + \boldsymbol{\Sigma}_{2j}} \cdot \frac{\partial (\boldsymbol{\Sigma}_{1i} + \boldsymbol{\Sigma}_{2j})}{\partial \boldsymbol{\Sigma}_{1i}} \\
&= \frac{1}{\det(\boldsymbol{\Sigma}_{1i} + \boldsymbol{\Sigma}_{2j})} \cdot \det(\boldsymbol{\Sigma}_{1i} + \boldsymbol{\Sigma}_{2j}) \cdot [(\boldsymbol{\Sigma}_{1i} + \boldsymbol{\Sigma}_{2j})^{-1}]^\top \cdot 1 \\
&= [(\boldsymbol{\Sigma}_{1i} + \boldsymbol{\Sigma}_{2j})^{-1}]^\top
\end{aligned}$$

$$\frac{\partial}{\partial \boldsymbol{\Sigma}_{1i}} [(\boldsymbol{\mu}_{1i} - \boldsymbol{\mu}_{2j})^\top (\boldsymbol{\Sigma}_{1i} + \boldsymbol{\Sigma}_{2j})^{-1} (\boldsymbol{\mu}_{1i} - \boldsymbol{\mu}_{2j})] = (\boldsymbol{\Sigma}_{1i} + \boldsymbol{\Sigma}_{2j})^{-1} (\boldsymbol{\mu}_{1i} - \boldsymbol{\mu}_{2j})(\boldsymbol{\mu}_{1i} - \boldsymbol{\mu}_{2j})^\top \cdot (\boldsymbol{\Sigma}_{1i} + \boldsymbol{\Sigma}_{2j})^{-1}$$

The derivation with respect to $\boldsymbol{\Sigma}_{2j}$ is similar, i.e.

$$\begin{aligned}\frac{\partial L_2}{\partial \boldsymbol{\Sigma}_{2j}} &= \sum_i \sum_{n=1}^{N} \frac{\partial}{\partial \boldsymbol{\Sigma}_{2j}} | \frac{1}{2} \gamma_{ij}^n [-\ln(\det(\boldsymbol{\Sigma}_{2j})) - (\boldsymbol{w}_n - \boldsymbol{\mu}_{2j}(x))^\top \boldsymbol{\Sigma}_{2j}^{-1} (\boldsymbol{w}_n - \boldsymbol{\mu}_{1i}(x)) \\ &\quad - \ln(\det(\boldsymbol{\Sigma}_{1i} + \boldsymbol{\Sigma}_{2j})) + (\boldsymbol{\mu}_{1i}(x) - \boldsymbol{\mu}_{2j}(x))^\top (\boldsymbol{\Sigma}_{1i} + \boldsymbol{\Sigma}_{2j})^{-1} (\boldsymbol{\mu}_{1i}(x) - \boldsymbol{\mu}_{2j}(x)) \\ &= \sum_i \sum_{n=1}^{N} \frac{\gamma_{ij}^n}{2} [-\boldsymbol{\Sigma}_{2j}^{-1} - \boldsymbol{\Sigma}_{2j}^{-1} \cdot (\boldsymbol{w}_n - \boldsymbol{\mu}_{2j})(\boldsymbol{w}_n - \boldsymbol{\mu}_{2j})^\top \cdot \boldsymbol{\Sigma}_{2j}^{-1} + [(\boldsymbol{\Sigma}_{1i} + \boldsymbol{\Sigma}_{2j})^{-1}]^\top \\ &\quad + (\boldsymbol{\Sigma}_{1i} + \boldsymbol{\Sigma}_{2j})^{-1} (\boldsymbol{w}_n - \boldsymbol{\mu}_{1i})(\boldsymbol{w}_n - \boldsymbol{\mu}_{1i})^\top \cdot (\boldsymbol{\Sigma}_{1i} + \boldsymbol{\Sigma}_{2j})^{-1}]\end{aligned}$$

The derivation of $\boldsymbol{A}_{1i}$ includes the derivation of $\boldsymbol{\Sigma}_{1i}$ by the chain rule as follows

$$\begin{aligned}\frac{\partial L_2}{\partial \boldsymbol{A}_{1i}} &= \sum_j \sum_{n=1}^{N} \frac{\partial}{\partial \boldsymbol{A}_{1i}} | \frac{1}{2} \gamma_{ij}^n [-\ln(\det(\boldsymbol{\Sigma}_{1i})) - (\boldsymbol{w}_n - \boldsymbol{\mu}_{1i}(x))^\top \boldsymbol{\Sigma}_{1i}^{-1} (\boldsymbol{w}_n - \boldsymbol{\mu}_{1i}(x)) \\ &\quad - \ln(\det(\boldsymbol{\Sigma}_{1i} + \boldsymbol{\Sigma}_{2j})) + (\boldsymbol{\mu}_{1i}(x) - \boldsymbol{\mu}_{2j}(x))^\top (\boldsymbol{\Sigma}_{1i} + \boldsymbol{\Sigma}_{2j})^{-1} (\boldsymbol{\mu}_{1i}(x) - \boldsymbol{\mu}_{2j}(x)) \\ &\quad ---------- \\ &= \frac{\partial \boldsymbol{\Sigma}_{1i}}{\partial \boldsymbol{A}_{1i}} \cdot \frac{\partial L_2}{\partial \boldsymbol{\Sigma}_{1i}} = \text{Inner-Derivative} \cdot \text{Outer-Derivative} \\ &\quad \rightarrow \frac{\partial \boldsymbol{\Sigma}_{1i}}{\partial \boldsymbol{A}_{1i}} = \frac{\partial \boldsymbol{A}_{1i}^\top \cdot 1 \cdot \boldsymbol{A}_{1i}}{\partial \boldsymbol{A}_{1i}} = (1 + 1^\top) \boldsymbol{A}_{1i} = 2\boldsymbol{A}_{1i} \\ &\quad ---------- \\ &= \sum_j \sum_{n=1}^{N} \frac{\gamma_{ij}^n}{2} \cdot 2\boldsymbol{A}_{1i} \cdot [-\boldsymbol{\Sigma}_{1i}^{-1} - \boldsymbol{\Sigma}_{1i}^{-1} \cdot (\boldsymbol{w}_n - \boldsymbol{\mu}_{1i})(\boldsymbol{w}_n - \boldsymbol{\mu}_{1i})^\top \cdot \boldsymbol{\Sigma}_{1i}^{-1} + [(\boldsymbol{\Sigma}_{1i} + \boldsymbol{\Sigma}_{2j})^{-1}]^\top \\ &\quad + (\boldsymbol{\Sigma}_{1i} + \boldsymbol{\Sigma}_{2j})^{-1} \cdot (\boldsymbol{\mu}_{1i} - \boldsymbol{\mu}_{2j})(\boldsymbol{\mu}_{1i} - \boldsymbol{\mu}_{2j})^\top \cdot (\boldsymbol{\Sigma}_{1i} + \boldsymbol{\Sigma}_{2j})^{-1}]\end{aligned}$$

The parameter a_{1i} is similar calculated, i.e.

$$\begin{aligned}\frac{\partial L_2}{\partial a_{1i}} &= \sum_{n=1}^{N} \frac{\partial}{\partial a_{1i}} | \frac{1}{2} \gamma_{ij}^n [-\ln(\det(\boldsymbol{\Sigma}_{1i})) - (\boldsymbol{w}_n - \boldsymbol{\mu}_{1i}(x))^\top \boldsymbol{\Sigma}_{1i}^{-1} (\boldsymbol{w}_n - \boldsymbol{\mu}_{1i}(x)) \\ &\quad - \ln(\det(\boldsymbol{\Sigma}_{1i} + \boldsymbol{\Sigma}_{2j})) + (\boldsymbol{\mu}_{1i}(x) - \boldsymbol{\mu}_{2j}(x))^\top (\boldsymbol{\Sigma}_{1i} + \boldsymbol{\Sigma}_{2j})^{-1} (\boldsymbol{\mu}_{1i}(x) - \boldsymbol{\mu}_{2j}(x)) \\ &\quad ---------- \\ &= \frac{\partial \exp(a_{1i})}{\partial a_{1i}} \cdot \frac{\partial \boldsymbol{\Sigma}_{1i}}{\partial \exp(a_{1i})} \cdot \frac{\partial L_2}{\partial \boldsymbol{\Sigma}_{1i}} = \text{Inner-Derivative} \cdot \text{Outer-Derivative} \\ &\quad \rightarrow \boldsymbol{\theta}_{1i} = [\boldsymbol{\Sigma}_{1i}(1,1), \boldsymbol{\Sigma}_{1i}(1,2), ...] \qquad \rightarrow \frac{\partial \boldsymbol{\theta}_{1i}}{\partial \exp(a_{1i})} = [100..., 01..., ...]^\top \\ &= \frac{\partial \exp(a_{1i})}{\partial a_{1i}} \cdot \frac{\partial \boldsymbol{\theta}_{1i}}{\partial \exp(a_{1i})} \cdot \frac{\partial L_2}{\partial \boldsymbol{\theta}_{1i}} = \exp(a_{1i}) \cdot \text{trace}\left(\frac{\partial L_2}{\partial \boldsymbol{\Sigma}_{1i}} \right) \\ &\quad ---------- \\ &= \sum_{n=1}^{N} \frac{\gamma_{ij}^n}{2} \cdot \exp(a_{1i}) \cdot \text{trace}(-\boldsymbol{\Sigma}_{1i}^{-1} - \boldsymbol{\Sigma}_{1i}^{-1} \cdot (\boldsymbol{w}_n - \boldsymbol{\mu}_{1i})(\boldsymbol{w}_n - \boldsymbol{\mu}_{1i})^\top \cdot \boldsymbol{\Sigma}_{1i}^{-1} + [(\boldsymbol{\Sigma}_{1i} + \boldsymbol{\Sigma}_{2j})^{-1}]^\top \\ &\quad + (\boldsymbol{\Sigma}_{1i} + \boldsymbol{\Sigma}_{2j})^{-1} \cdot (\boldsymbol{\mu}_{1i} - \boldsymbol{\mu}_{2j})(\boldsymbol{\mu}_{1i} - \boldsymbol{\mu}_{2j})^\top \cdot (\boldsymbol{\Sigma}_{1i} + \boldsymbol{\Sigma}_{2j})^{-1})\end{aligned}$$

To calculate the derivative of $\exp(a_{1i})$ we consider the covariance $\boldsymbol{\Sigma}_{1i}$ as a parameter vector $\boldsymbol{\theta}_{1i}$. The derivative of $\boldsymbol{\theta}_{1i}$ w.r.t. $\exp(a_{1i})$ corresponds to the identity matrix in vector form. The multiplication of this result with the original covariance matrix, represents the trace of $\boldsymbol{\Sigma}_{1i}$. The multiplication with the inner derivative of $\exp(a_{1i})$ w.r.t to a_{1i} represents the complete gradient.

This calculations can be applied for the parameters $\boldsymbol{A}_{2j}$ and $\exp(a_{2j})$ of the second expert:

Parameter: $\boldsymbol{A}_{2j} \quad \rightarrow \boldsymbol{\Sigma}_{2j} = \boldsymbol{A}_{2j}^{\top}\boldsymbol{A}_{2j} + \exp(a_{2j}) \cdot \boldsymbol{I}$

$$\begin{aligned}
\frac{\partial L_2}{\partial \boldsymbol{A}_{2j}} &= \sum_i \sum_{n=1}^{N} \frac{\partial}{\partial \boldsymbol{A}_{2j}} | \frac{1}{2} \boldsymbol{\gamma}_{ij}^n [-\ln(\det(\boldsymbol{\Sigma}_{2j})) - (\boldsymbol{w}_n - \boldsymbol{\mu}_{2j}(x))^{\top} \boldsymbol{\Sigma}_{2j}^{-1} (\boldsymbol{w}_n - \boldsymbol{\mu}_{2j}(x)) \\
&\quad - \ln(\det(\boldsymbol{\Sigma}_{1i} + \boldsymbol{\Sigma}_{2j})) + (\boldsymbol{\mu}_{1i}(x) - \boldsymbol{\mu}_{2j}(x))^{\top} (\boldsymbol{\Sigma}_{1i} + \boldsymbol{\Sigma}_{2j})^{-1} (\boldsymbol{\mu}_{1i}(x) - \boldsymbol{\mu}_{2j}(x)) \\
&= \sum_i \sum_{n=1}^{N} \frac{\boldsymbol{\gamma}_{ij}^n}{2} \cdot 2\boldsymbol{A}_{2j} \cdot [-\boldsymbol{\Sigma}_{2j}^{-1} - \boldsymbol{\Sigma}_{2j}^{-1} \cdot (\boldsymbol{w}_n - \boldsymbol{\mu}_{2j})(\boldsymbol{w}_n - \boldsymbol{\mu}_{2j})^{\top} \cdot \boldsymbol{\Sigma}_{2j}^{-1} + [(\boldsymbol{\Sigma}_{1i} + \boldsymbol{\Sigma}_{2j})^{-1}]^{\top} \\
&\quad + (\boldsymbol{\Sigma}_{1i} + \boldsymbol{\Sigma}_{2j})^{-1} \cdot (\boldsymbol{\mu}_{1i} - \boldsymbol{\mu}_{2j})(\boldsymbol{\mu}_{1i} - \boldsymbol{\mu}_{2j})^{\top} \cdot (\boldsymbol{\Sigma}_{1i} + \boldsymbol{\Sigma}_{2j})^{-1}]
\end{aligned}$$

Parameter: $a_{2j} \quad \rightarrow \boldsymbol{\Sigma}_{2j} = \boldsymbol{A}_{2j}^{\top}\boldsymbol{A}_{2j} + \exp(a_{2j}) \cdot \boldsymbol{I}$

$$\begin{aligned}
\frac{\partial L_2}{\partial a_{2j}} &= \sum_{n=1}^{N} \frac{\partial}{\partial a_{2j}} | \frac{1}{2} \boldsymbol{\gamma}_{ij}^n [-\ln(\det(\boldsymbol{\Sigma}_{1i})) - (\boldsymbol{w}_n - \boldsymbol{\mu}_{2j}(x))^{\top} \boldsymbol{\Sigma}_{2j}^{-1} (\boldsymbol{w}_n - \boldsymbol{\mu}_{2j}(x)) \\
&\quad - \ln(\det(\boldsymbol{\Sigma}_{1i} + \boldsymbol{\Sigma}_{2j})) + (\boldsymbol{\mu}_{1i}(x) - \boldsymbol{\mu}_{2j}(x))^{\top} (\boldsymbol{\Sigma}_{1i} + \boldsymbol{\Sigma}_{2j})^{-1} (\boldsymbol{\mu}_{1i}(x) - \boldsymbol{\mu}_{2j}(x)) \\
&= \sum_{n=1}^{N} \frac{\boldsymbol{\gamma}_{ij}^n}{2} \cdot \exp(a_{2j}) \cdot \operatorname{trace}(-\boldsymbol{\Sigma}_{2j}^{-1} - \boldsymbol{\Sigma}_{2j}^{-1} \cdot (\boldsymbol{w}_n - \boldsymbol{\mu}_{2j})(\boldsymbol{w}_n - \boldsymbol{\mu}_{2j})^{\top} \cdot \boldsymbol{\Sigma}_{2j}^{-1} + [(\boldsymbol{\Sigma}_{1i} + \boldsymbol{\Sigma}_{2j})^{-1}]^{\top} \\
&\quad + (\boldsymbol{\Sigma}_{1i} + \boldsymbol{\Sigma}_{2j})^{-1} \cdot (\boldsymbol{\mu}_{1i} - \boldsymbol{\mu}_{2j})(\boldsymbol{\mu}_{1i} - \boldsymbol{\mu}_{2j})^{\top} \cdot (\boldsymbol{\Sigma}_{1i} + \boldsymbol{\Sigma}_{2j})^{-1})
\end{aligned}$$