Undergraduate Thesis

Parameterized and Dynamic Gaussian Mixture Models for Robot Manipulation

Jonas Etienne Fischer

Examiner: Dr. Tim Welschehold
Adviser: Jan Ole von Hartz

University of Freiburg
Faculty of Engineering
Department of Computer Science
Robot Learning Lab

May 03rd, 2024

Writing Period

05. 02. 2024 – 03. 05. 2024

Examiner

Dr. Tim Welschehold

Adviser

Jan Ole von Hartz

Declaration

I hereby declare that I am the sole author and composer of my thesis and that no other sources or learning aids, other than those listed, have been used. Furthermore, I declare that I have acknowledged the work of others by providing detailed references of said work.
I hereby also declare that my Thesis has not been prepared for another examination or assignment, either wholly or excerpts thereof.

Place, Date	Signature

Abstract

Gaussian Mixture Models (GMMs) are a prominent method for learning robot motions in a data efficient way. A variety of different possibilities exist for learning the parameters of a Gaussian Mixture Models from demonstrations and adapting it to changing environments a robot encounters. However, a rigorous evaluation of the different models is still lacking.

In this work, we compare Task-Parameterized Gaussian Mixture Models where the task-parameters take the form of different reference frames with a so-called Elastic Dynamical System approach. The Elastic Dynamical System adapts to new task-parameters by transforming the Gaussian Mixture Model via Laplacian Editing. The transformed Gaussian Mixture Model is then used to construct new robot motion with a Linear Parameter Varying Dynamical System policy.

Furthermore, we compare two different methods for learning the Task-Parameterized Gaussian Mixture Model. The first method utilizes the Expectation Maximization algorithm, which maximizes the log-likelihood of the parameters in each iteration step. The second method employs Directionality-aware Mixture Model that learns a Gaussian Mixture Model according to the variation in direction observed in the demonstrations.

This work showcases the strengths and weaknesses of the different models and fitting methods. We evaluate their performance by reconstructing the trajectories of our dataset and compare them using different metrics, such as the Akaike Information Criterion the Mean Absolute Error to evaluate their quality. Furthermore, we simulate

the robot's end-effector in novel and changing environments to assess the task success achieved by the models.

Zusammenfassung

Gaussian Mixture Models (GMMs) sind eine bekannte Methode, um Roboterbewegungen auf dateneffiziente Weise zu lernen. Es gibt eine Vielzahl verschiedener Möglichkeiten, die Parameter eines Gaussian Mixture Models aus Demonstrationen zu lernen und es an die sich ändernden Umgebungen anzupassen, in denen ein Roboter seine Arbeit vollrichtet. Ein ausführlicher Vergleich der verschiedenen Modelle gibt es jedoch noch nicht.

In dieser Arbeit vergleichen wir aufgabenparametrisierte (englisch: task-parameterized) Gaussian Mixture Modelle, bei denen die Aufgabenparameter (englisch: task-parameter) die Form verschiedener Referenzrahmen annehmen, mit einem sogenannten Elastic Dynamical System Ansatz. Das Elastic Dynamical System passt sich neuen Aufgabenparametern an, indem es das Gaussian Mixture Modell durch Laplacian Editing transformiert. Das transformierte Gaussian Mixture Model wird dann verwendet, um neue Bewegungen des Roboters mit einem Linear Parameter Varying Dynamical System Ansatz zu konstruieren.

Darüber hinaus vergleichen wir zwei verschiedene Methoden zum Lernen des aufgabenparametrisierten Gaussian Mixture Model. Die erste Methode verwendet den expectation maximization (Erwartungsmaximierung) Algorithmus, der die Logarithmische Wahrscheinlichkeit der Parameter in jedem Iterationsschritt maximiert. Die zweite Methode verwendet das Directionalaty-Aware Mixture Model, dass ein Gaussian Mixture Model entsprechend der in den Demonstrationen beobachteten Richtungsänderungen lernt.

In dieser Arbeit werden die Stärken und Schwächen der verschiedenen Modelle und

Lernmethoden beschrieben. Wir bewerten die Modelle anhand der Rekonstruktion der Trajektorien unseres Datensatzes und vergleichen sie anhand verschiedener Metriken, wie dem Akaike Information Criterion und dem Mean Absolute Error. Darüber hinaus simulieren wir den Endeffektor des Roboters in unbekannten und sich verändernden Umgebungen, um zu evaluieren wie gut die Modelle die Aufgaben erfüllen können.

Contents

List of Figures

List of Tables

List of Algorithms

1 Introduction

The field of robot learning and autonomous systems has seen rapid development in recent years. This led to rising numbers of autonomous robots in industrial applications, such as manufacturing and logistic, as well as in our daily lives. Unlike traditional robots that were programmed for repetitive tasks, they now must adapt their motion to new and changing environments . The critical need for robust adaptation to even slightly different task set-ups, is especially important in environments where humans and robots are working together to ensure safety [1].

The mapping between the state of the environment and the actions of a robot is called a policy. To define a policy for a robot is often very challenging, especially for more complex models the modern world demands. Therefore, more and more Machine Learning models are utilized to learn these policies.
A common approach learning robot policies is Learning from Demonstrations (LfD), where the state of the environment and the robot motion for a task are learned from a set of trajectories [2][3]. These demonstrations are usually provided by simulation of the task, human controlled robots, or could even be provided directly from human movements (imitation learning). The learned motion must be accurate enough to create robot motions that are able to fulfill the given task and be general enough to adapt the motion to new scenarios.
Gaussian Mixture Models (GMMs) are a very data efficient way to learn robot motion from demonstrations. The structure of GMMs is compatible with many robot learning methods. Furthermore, it allows to adapt to new applications in a variety of different

ways.

Task-Parameterized Gaussian Mixture Models (TPGMM) are a modification of standard GMMs that encode task-relevant parameters into their model and capture their relationship to the robot motion in various ways. The task-parameters could take the form of landmarks a robot has to pass or different object positions [4]. For a new set of task-parameters, TPGMMs can construct new motions based on their learned relationship to the tasks. In this thesis, we encode the task-parameters as reference frames [4].

Another approach to LfD is by using Dynamical Systems which learn motion policies from only a few demonstrations. An approach adapting Dynamical Systems to a changing environment is by embedding the task-parameters into a GMM, which will then be used to predict new motion by solving different optimization problems [5]. Transforming robot trajectory to pass through computed landmarks via Laplacian-Editing is one possibility to define such an optimization problem [6].

There exist different possibilities to generalize GMMs to different task set-ups. However, there is still a lack of rigorous comparisons between the methods. In this thesis, we conduct a comparative analysis between TPGMM and a Dynamical system approach, alongside two fitting methods for the TPGMM. In Section 3 we provide the background and mathematical foundations of these models and specify the considered TPGMM and Dynamical System. Moving to Section 4, we describe the set-up in which we compare our different models. Section 5 describes the policies of our different methods. The results of our experiments, along with a discussion of the advantages and limitations of every model, are presented at Section 6.

2 Related Work

The design of the model policy is an important part of robot learning applications and LfD in general, and can be done in many different ways. B. D. Argall et al. described many different approaches to designing motion policies for robot learning like imitation learning, where a robot learns from humans demonstrations, or active learning, which sees the robot as an active partner and lets the robot ask questions [2]. We chose a system model policy where we learn a model of the world dynamics from demonstrations and derive the policy using the learned model.

S. M. Khansari-Zadeh et al. introduced utilizing Gaussian Mixture Models to adapt to robot motions. In this work, we used Gaussian Mixture Models, to model the end-effector motion of a robot in different tasks.
Calinon demonstrated how to adapt GMMs to a changing task set-up through a Task-Parameterized Gaussian Mixture Model (TPGMM) [4]. This work represents task-parameters as reference frames. The model learns multiple Gaussian Mixture Models with respect to the task-parameter frames. To adapt to new environments, it transforms the GMMs of the task based on the position and orientation of the new task parameters to the world frame relative to the new positions and orientations of the task parameters [4]. Furthermore, the Gaussians of the different GMMs are joined by computing the Gaussian product to generate a GMM with lower variance in the surrounding areas of the task-parameters [4]. In this thesis, we expanded this approach by adding a time dimension to the end-effector trajectories of our demonstrations. Instead of learning multiple Gaussian Mixture Models with respect to the task-

parameters, we combine all transformed trajectories and learn one multidimensional GMM. We then create multiple marginals from this GMM for every task parameter frame, where each marginal contains the time dimension of the learned Gaussians and the dimensions of the end-effector positions according to their task-parameter frame. This enables us to generate new trajectories according to the given time step and task-parameters.

One of the main challenges is finding a good fit for Gaussian Mixture Models in the demonstrations. One approach we employ in this thesis, proposed by A.P. Dempster, is via an Expectation Maximization (EM) algorithm [7]. EM aims to find a suitable fit by maximizing the log-likelihood of the multivariate normal distribution of the GMM in every iteration step until convergence. We chose this model, as it is a simple yet effective algorithm to learn the parameters of a GMM, adaptable to a multitude of different demonstrations, regardless of their dimension [7].
Directionality-aware Mixture Models (DAMM) represent another approach to fit a Gaussian Mixture Model on robot motion, proposed by S. Sun et al. [8]. DAMM is a Markov chain Monte Carlo [9] technique that is employed to fit Gaussian Mixture Models according to the change in direction of the trajectories via a split/merge procedure [10] . The change in direction is computed by normalizing the velocity of our trajectory at every time step, and computing a direction vector. DAMM is the successor of PC-GMM [11]. Both methods fit a GMM on robot trajectories according to their directional change. In this thesis, we used DAMM in its original form and extend it to fit time-based TPGMM with an additional gripper dimension. Fitting the model according to the directional data of the demonstrations enables us to catch the complexity of the robot motion for different tasks.

The reconstruction of robot motion from GMMs can be viewed as a regression problem [12]. The work of F. Stulp and O. Sigaud gives a good overview of different regression algorithms [13]. We decided to use Gaussian Mixture Regression, which offers simple and fast computations of continuous robot movements from a GMM

[14]. In combination with the time-based TPGMM are we able to construct smooth trajectories for a new task environment on the fly during our simulations.
Another approach to reconstruct trajectories are Dynamical System-based motion policies [5]. However, most of the dynamical systems approaches lack the ability to adapt to novel environments [6]. T. Li and N. Figuero proposed an Elastic Dynamical System (Elastic-DS) approach that is based on a GMM and is able to adapt to new task set-ups [6]. Elastic-DS learns a GMM from Demonstrations and transforms the GMM via Laplacian Editing, according to new task-parameters. While T. Li and N. Figuero used PC-GMM to fit the GMM, we are using the successor DAMM.

3 Background

This chapter introduces the background of our employed models, the algorithms that are utilized for fitting the models and reconstructing the end-effector trajectories, and the mathematical underlying formulations.

3.1 Gaussian Mixture Models

In this chapter, we define Gaussian Mixture Models and describe how they can be initialized, fitted on demonstrations via the Expectation Maximization algorithm and the procedure of creating new end-effector trajectories.

3.1.1 Definition

A Gaussian Mixture Model (GMM) is a probabilistic model with parameters $\mu = \{\mu_i\}_{i=1}^K, \Sigma = \{\Sigma_i\}_{i=1}^K$ and $\pi = \{\pi_i\}_{i=1}^K$, which assumes all the data points are generated from a mixture of K Gaussian distributions. Each Gaussian represents a cluster of data points in the data. The density function of a GMM is the weighted sum of these Gaussian components, given as

$$p(x) = \sum_{i=1}^{K} \pi_i \mathcal{N}(\mu_i, \Sigma_i), \tag{1}$$

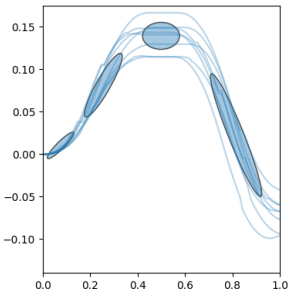


Figure 1: Gaussian mixture model with four Gaussians fitted on an end-effector trajectory.

where x is a D-dimensional continuous data point, π_i is the weight of the i-th Gaussian ($\sum_{i=1}^{K} \pi_i = 1$), $\mathcal{N}(\mu_i, \Sigma_i)$ the normal distribution of the i-th component with Gaussian mean μ_i, covariance matrix Σ_i and $i = 1, ..., K$.

$$p(\xi) \sim \sum_{i=1}^{K} \pi_i \mathcal{N}(\mu_i, \Sigma_i) \tag{2}$$

In the scope of this bachelor thesis, we employ GMM's to model the end-effector poses $\xi \in \mathbb{R}^D$ from demonstrations (Eq. 2), which can be seen in Figure 1.

3.1.2 Initialization of Gaussian Mixture Models

Because the Expectation Maximization algorithm only converges to a local optima, a good initialization is necessary to generate a good fit. There are various different methods to initialize GMM's with K components. Our dataset ξ comprises N trajectories all sampled to the same size L. We chose a time-based K-bins initialization to initialize the Gaussians to catch the temporal order of the data points of our robot trajectories [15]. Other algorithms, e.g., K-means, do not explicitly exhibit the temporal dependencies of the data.

We divide our demonstration ξ into K subsets ξ_t along the time axis, compute the

mean and covariance matrix of these subsets, and assign every weight the same value. Let $x_{ij} \in \xi$, where $i = 1, ..., N$ and $j = 1, ..., L$. The time based initialization defines the parameters of the GMM as follows:

$$\mu_t = \frac{1}{|\xi_t|} \sum_{x_t \in \xi_t} x_t, \tag{3a}$$

$$\Sigma_t = \frac{1}{|\xi_t|} \sum_{x_t \in \xi_t} (x_t - \mu_t)^T (x_t - \mu_t), \tag{3b}$$

$$\pi_t = \frac{1}{K}, \tag{3c}$$

where $\xi_t = \{x_{ij} | (t-1) * \lfloor \frac{L}{K} \rfloor \leq j < t * \lfloor \frac{L}{K} \rfloor, i \in \{1, ..., N\}, j \in \{1, ..., L\}$ and $x_{ij} \in \xi\}$, $t = 1, ..., K$. A time-based initialized GMM can be seen in the first picture of Figure 2.

3.1.3 Expectation Maximization Algorithm

Expectation Maximization (EM) is an iterative algorithm used to maximize the likelihood estimations of the parameters in statistical models, first proposed by Dempster et al. [7]. The EM algorithm for GMMs consists of two main steps, which are repeated until convergence [4]:

- Expectation-Step (E-step):

$$z_{t,i} = \frac{\pi_i N(x_t | \mu_i, \Sigma_i)}{\sum_{j=1}^{K} \pi_j N(x_i | \mu_j, \ \Sigma_j)} \tag{4}$$

- Maximization-Step (M-step):

$$\pi_i = \frac{\sum_{t=1}^{N} z_{ti}}{N} \tag{5}$$

$$\mu_i = \frac{\sum_{t=1}^{N} z_{t,i} x_t}{\sum_{t=1}^{N} z_{t,i}} \tag{6}$$

$$\Sigma_i = \frac{\sum_{t=1}^{N} zt, i(x_t - \mu_i)(x_t - \mu_i)^T}{\sum_{t=1}^{N} z_{t,i}}. \tag{7}$$

During the E-step we compute $z_{t,i}$, which represents the probability that the data point at time step t is assigned to the i-th Gaussian. In the maximization step the Gaussian components are updated according to their weighted assigned data points (weighted by the probability $z_{t,i}$). These two steps are repeated E times (called epochs) or until the likelihood change is smaller than a given threshold. Before applying the EM algorithm, the components of the GMM are initialized, as described in 3.1.2, Figure 2 visualizes a GMM during multiple epochs of the EM algorithm.

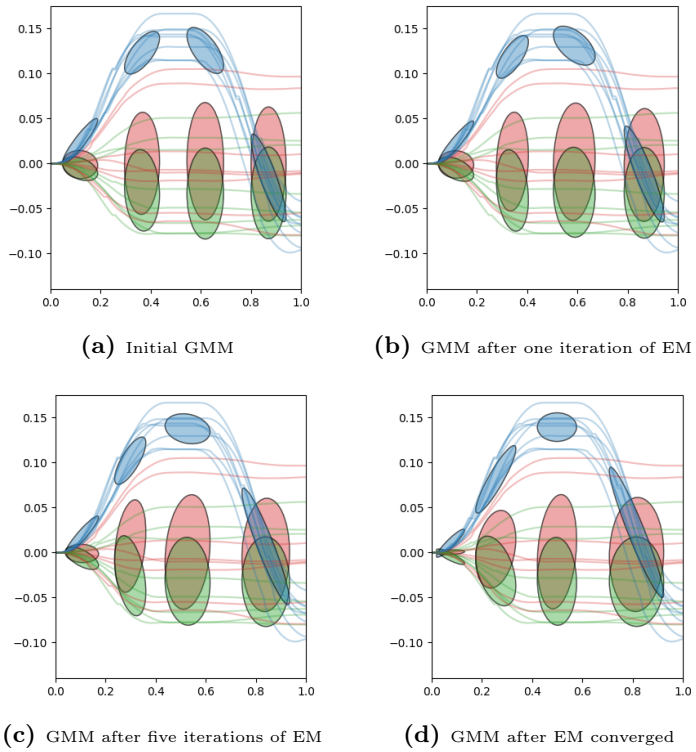
To prevent overfitting and singular covariance matrices during EM we introduce a regularization term in every M-step of an epoch. For each GMM we decided between three different regularization terms [4] :

- L1 regularization: $\Sigma_i = \Sigma_i + \lambda$

- Tikhonov regularization: $\Sigma_i = \Sigma_i + \mathbf{I} * \lambda$

- Scaled identity matrix regularization: $\Sigma_i = \lambda \frac{Tr\Sigma_i}{d} + (1 - \lambda)\Sigma_i$,

where $\frac{Tr\Sigma_i}{d}$ is the average of the diagonal entries of Σ_i[16], and $\mathbf{I}$ the identity matrix.



(a) Initial GMM

(b) GMM after one iteration of EM

(c) GMM after five iterations of EM

(d) GMM after EM converged

Figure 2: Expectation Maximization for a Gaussian Mixture Model. (a) shows the GMM after the time-based initialization, (b) after the first iteration, (c) after the fifth iteration and (d) after one-hundred iterations. During the em algorithm does the third Gaussian changes the most.

3.1.4 Model selection

To determine the number of Gaussians in the GMM, the regularization type and the regularization weight λ we employ a grid search. This approach evaluates the models of the grid by computing the Akaike Information Criterion (AIC) score of each different model configuration, it then chooses the model with the lowest score. Figure 3 illustrates the grid search. The AIC score is defined as,

$$AIC = 2K - 2log(L), \tag{8}$$

where K is the number of components and L the likelihood of the model on our data [17]. We choose AIC over the Bayesian Information Criterion (BIC) as the AIC score

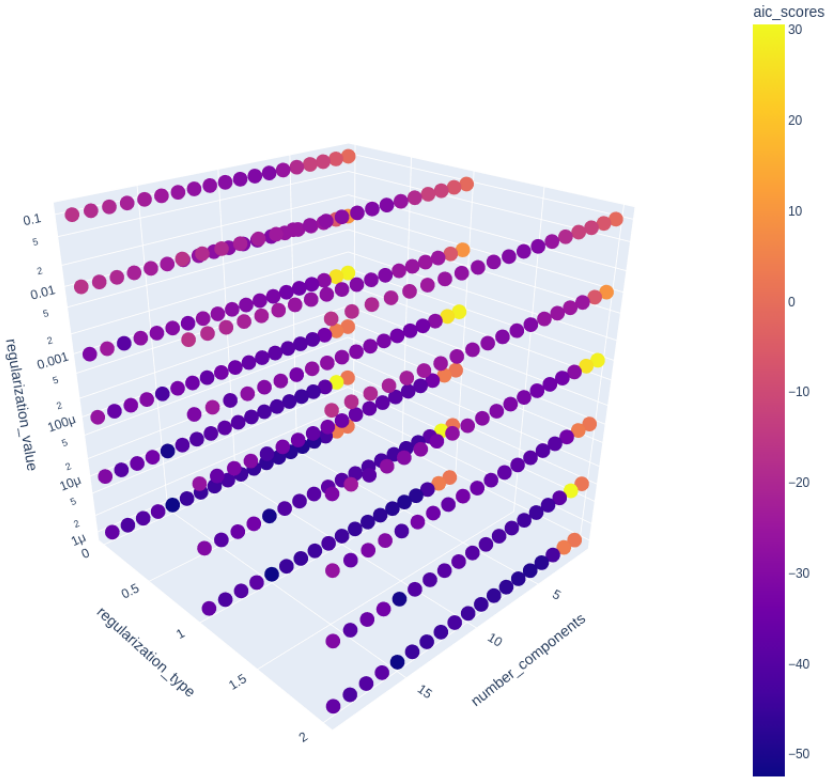


Figure 3: Grid Search for a GMM. Note that the z axis is logarithmic scaled. Regularization type 0:=L1 regularization, 1:=Tikhonov regularization, 2:=Scaled idendity matrix regularization.

works better with smaller data sets and does not punish more complex models as hard as the BIC score [17]. The AIC score balances the complexity and the fit of the model by punishing models with a higher number of Gaussians.

3.1.5 Gaussian Mixture Regression

Gaussian Mixture Regression is a regression method that enables us to generalize the motion of our end-effector during reconstruction by modeling the joint probability density function of the GMM. The rapid computation of the next data point during reconstruction is an advantage in robot manipulation, as the model can respond more

effectively to a changing environment [4].

Let ξ be our dataset of robot trajectories, containing N data points each with dimension D. We can divide ξ into two subsets $\xi^{\mathcal{O}}$, $\xi^{\mathcal{I}}$ containing N data points each, with their corresponding input dimension $\mathcal{I}$ and output dimension $\mathcal{O}$ (dimension we want to predict). In this thesis, we used GMR in a time-based manner by time indexing the data as $\xi = [T, \xi_{pos}^T]^T$, where T represents the time dimension ($\xi^{\mathcal{I}} = T$), and ξ_{pos} the end-effector pose of the robot ($\xi^{\mathcal{O}} = \xi_{pos}$). The GMM encodes ξ with the joint distribution as follows [4],

$$\mathcal{P}(\xi^{\mathcal{I}}, \xi^{\mathcal{O}}) \sim \sum_{i=1}^{K} \mathcal{N}(\mu_i, \Sigma_i). \tag{9}$$

We can further represent the end-effector pose $x \in \xi$ as a combination of two subvectors spanning for the input dimension $x^{\mathcal{I}} \in \xi^{\mathcal{I}}$ (the point in time of this data point) and a vector spanning for the output dimension $x^{\mathcal{O}} \in \xi^{\mathcal{O}}$ (end-effector pose at the current time-step). This gives us the following notation:

$$x = \begin{bmatrix} x^{\mathcal{I}} \\ x^{\mathcal{O}} \end{bmatrix}, \qquad \mu_i = \begin{bmatrix} \mu_i^{\mathcal{I}} \\ \mu_i^{\mathcal{O}} \end{bmatrix}, \qquad \Sigma_i = \begin{bmatrix} \Sigma_i^{\mathcal{I}} & \Sigma_i^{\mathcal{I}\mathcal{O}} \\ \Sigma_i^{\mathcal{O}\mathcal{I}} & \Sigma_i^{\mathcal{O}} \end{bmatrix}. \tag{10}$$

Given a new input vector $\tilde{x}^{\mathcal{I}}$ we can now compute the joint probability density distribution given $\tilde{x}^{\mathcal{I}}$ (Eq. 11-14) [4]. We then chose the mean of the computed conditional distribution as our output $\hat{x}^{\mathcal{O}}$.

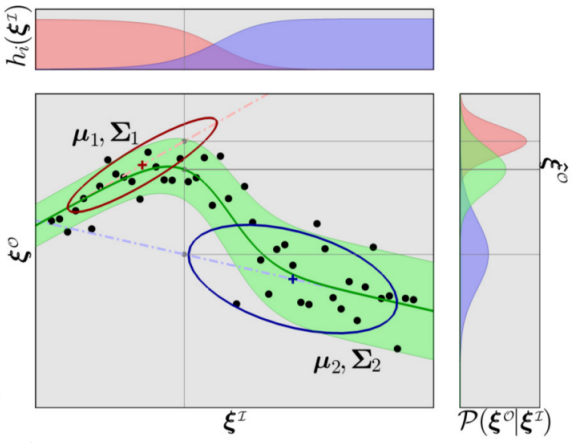


Figure 4: Illustration of the encoding $\mathcal{P}(\xi^{\mathcal{O}}, \xi^{\mathcal{I}})$ as GMM and the estimation $\mathcal{P}(\xi^{\mathcal{I}}|\xi^{\mathcal{O}})$ using GMR. On the right side of the image are the conditional probability distribution $\mathcal{P}(\xi^{\mathcal{O}}, \xi^{\mathcal{I}})$ of the two Gaussian's ,in red and blue, and their joint probability distribution in green. At the top is the probability h_i of each point $^{\mathcal{I}}$ being assigned to each Gaussian. Source: Adapted from [4].

$$\mathcal{P}(\hat{x}^{\mathcal{O}}|\hat{x}^{\mathcal{I}}) \sim \sum_{i=1}^{K} h_i(\hat{x}^{\mathcal{I}})\mathcal{N}(\hat{\mu}_i^{\mathcal{O}}(\hat{x}^{\mathcal{I}}), \hat{\Sigma}_i^{\mathcal{O}}) \tag{11}$$

$$\text{with } \hat{\mu}_i^{\mathcal{O}}(\hat{x}_i^{\mathcal{I}}) = \mu_i^{\mathcal{O}} + \Sigma_i^{\mathcal{O}\mathcal{I}}\Sigma_i^{\mathcal{I}-1}(\hat{x}_i^{\mathcal{I}} - \hat{\mu}_i^{\mathcal{O}}), \tag{12}$$

$$\hat{\Sigma}_i^{\mathcal{O}} = \Sigma_i^{\mathcal{O}} - \Sigma_i^{\mathcal{O}\mathcal{I}}\Sigma_i^{\mathcal{I}-1}\Sigma_i^{\mathcal{I}\mathcal{O}}, \tag{13}$$

$$\text{and } h_i(\hat{x}_t^{\mathcal{I}}) = \frac{\pi_i\mathcal{N}(\hat{x}_t^{\mathcal{I}}|\mu_i^{\mathcal{I}}, \Sigma_i^{\mathcal{I}}}{\sum_k^K \pi_k\mathcal{N}(\hat{x}_t^{\mathcal{I}}|\mu_k^{\mathcal{I}}, \Sigma_k^{\mathcal{I}})} \tag{14}$$

In this work, we utilize GMR to compute smooth trajectories. GMR allows us to compute the joint probability distribution $\mathcal{P}(\hat{x}_{pos}|t)$ of the end-effector pose $\hat{x}_{pos}$ given a time step $t \in T$. Figure 4 illustrates the computation of the conditional distribution via GMR.

3.2 Directionality-Aware Mixture Model

Directionality-aware Mixture Models (DAMM) were first proposed by Sunan Sun et al. [8]. DAMM processes data points $\xi \in \mathcal{R}^n$ containing velocity information and associates the velocity with a specific direction. DAMM is designed to capture the variation in direction of a nonlinear trajectory and adapt the corresponding Gaussians, which can be seen in Figure 5. DAMM was originally proposed as a Mixture Model but we used it as a fitting method for our Gaussian Mixture Models. In this chapter, it is explained how DAMM works and how it can be used to compute the Gaussians for an GMM.

3.2.1 Preconditions

The goal of DAMM is to fit the GMM in a way that the Gaussians are fitted along the direction of the trajectory. Therefore, if a Gaussian has a high variation of direction, then it should be split into two new Gaussians with a lower variation of direction. A data point ξ can be divided into two subvectors, the pose and velocity $\xi = (\xi^{pos}\ \dot{\xi})^T$. The velocity vector of each data point is at first normalized to obtain a unit-norm directional vector ξ^{dir} (Eq. 15) in the unit-sphere, a nonlinear Riemannian manifold [8]. M. do Carmo defines a Riemannian equivalent of a mean for directional data as the center of mass of the unit sphere (Eq. 16):

$$\xi^{dir} = \frac{\dot{\xi}}{\|\dot{\xi}\|} \in \mathbb{S}^{d-1} \subset \mathbb{R}^d, \tag{15}$$

$$\mu^{dir} = \underset{p \in \mathbb{S}^{d-1}}{\operatorname{argmin}} \sum_{i=1}^{N} d(\xi_i^{dir}, p)^2 = \underset{p \in \mathbb{S}^{d-1}}{\operatorname{argmin}} \sum_{i=1}^{N} ||\log_p(\xi_i^{dir})||_2^2, \tag{16}$$

where $d(x, y) = \arccos(x^T y)$ is the geodesic distance between $x, y \in \mathbb{S}^d$ [18]. The geodisc distance $d(x, y)$ is equal to the L2 norm of x being mapped to the tangent space defined by the point of tangency y via the logarithmic map $\log_y : \mathcal{M} \to T_x\mathbb{S}$,

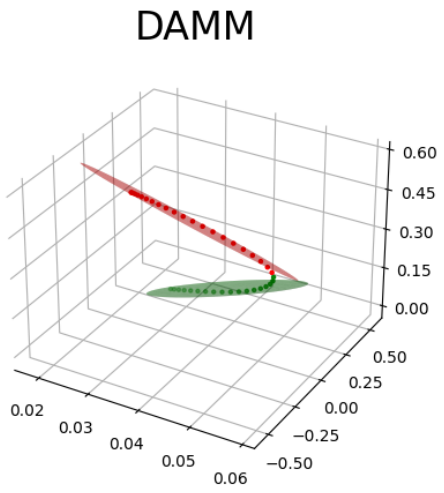


Figure 5: Directionality-aware Mixture Model with two Gaussians fitted on a single trajectory. The color of a point determines it affiliation z_i to a Gaussian computed with Eq. 20.

which explains Eq. 15 and 16 [18].

The variation of direction $(\sigma^2)^{dir} \in \mathcal{R}$ with respect to μ^{dir} is defined as follows

$$(\sigma^2)^{dir} = \frac{1}{N}\sum_{i=1}^{N} d(\xi_i^{dir}, \mu^{dir})^2 = \frac{1}{N}\sum_{i=1}^{N} ||\log_{\mu^{dir}}(\xi_i^{dir})||_2^2, \tag{17}$$

where $||\cdot||_2$ defines the L2 norm [8]. We define the state variable $\hat{\xi}$, the mean $\hat{\mu}$ and covariance matrix of DAMM as follows,

$$\hat{\xi} = \begin{bmatrix} \xi^{pos} \\ ||\log_{\mu^{dir}}(\xi^{dir})||_2 \end{bmatrix}, \qquad \hat{\mu} = \begin{bmatrix} \mu^{pos} \\ \mathbf{0} \end{bmatrix}, \qquad \hat{\Sigma} = \begin{bmatrix} \Sigma^{pos} & \mathbf{0} \\ \mathbf{0} & (\sigma^2)^{dir} \end{bmatrix}, \tag{18}$$

where $\mathbf{0}$ is the zero matrix.

3.2.2 Generative Model

Sunan Sun et al. define the generative model of DAMM as [8],

$$\pi \sim GEM(1, \alpha), \tag{19}$$

$$z_i \sim Cat(\pi_1, \pi_2, ...), \tag{20}$$

$$(\hat{\mu_k}, \hat{\Sigma}_k) \sim NIW(\Psi, \nu, \mu_0, \kappa), \tag{21}$$

$$\hat{\xi}_i | z_i = k \sim \mathcal{N}(\hat{\mu_k}, \hat{\Sigma}_k). \tag{22}$$

The weights π are sampled from a Griffiths Engen McCloskey (GEM) distribution following a stick process via a beta distribution given as [19],

$$\beta_k \sim Beta(1, \alpha), \tag{23}$$

$$\pi_k \sim \beta_k \prod_{l=1}^{k-1} (1 - \beta_l). \tag{24}$$

The assignment of data point z_i to a component is then sampled from a categorical distribution (Cat) defined by π. Sample the Gaussian mean $\hat{\mu}_k$ and covariance matrix $\hat{\Sigma}_k$, as defined in Eq. 21 by a conjugate prior Normal-Inverse-Wishart (NIW) distribution [20]. $(\sigma^2)^{dir}$ can be regulated by adapting the NIW scale matrix $\Psi \in \mathbb{S}_+^{d-1}$ and the degrees of freedom $\nu \in \mathcal{R}_+$ as the Gaussians are sampled from the NIW distribution and their parameters embody the prior belief of the distribution. Given that $\hat{\xi}_i$ is assigned to the k-th component, it is then sampled from the normal distribution of the k-th Gaussian.

Each variable can be sampled by an Instantiated Gibbs Sampler, given the generative model as follows [20],

$$(\pi_1, ..., \pi_K, \tilde{\pi}_K) \sim Cat(N_1, ..., N_K, \alpha), \tag{25}$$

$$(\hat{\mu_k}, \hat{\Sigma}_k) \sim NIW(\Psi_n, \nu_n, \mu_n, \kappa_n), \forall k \in \{1, ..., K\}, \tag{26}$$

$$z_i \sim \sum_{k=1}^{K} \pi_k \mathcal{N}(\hat{\xi}_i | \hat{\mu}_k, \hat{\Sigma}_k), \forall i \in \{1, ..., N\}, \tag{27}$$

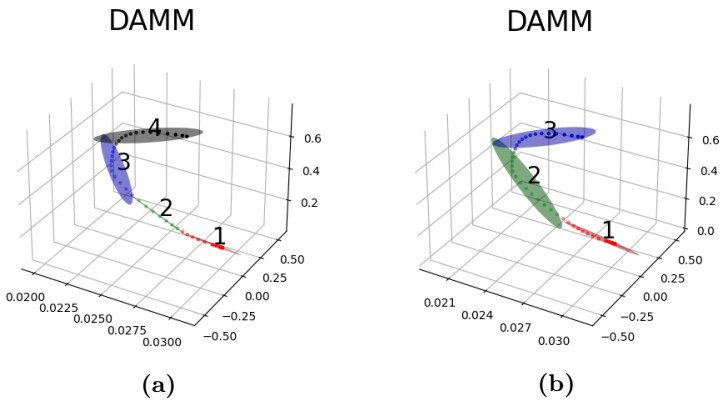
where $\tilde{\pi}_K$ is the sum of all empty weights given by the GEM distribution in Eq. 19 [8]. In Eq. 25 the weights are sampled by a categorical distribution given the number of data points N_k associated with the k-th component and the concentration factor from Eq. 19. The Gaussians are sampled by the NIW distribution in Eq. 26 and the assignment z_i of the i-th data point is then drawn from the density function of the multivariate Gaussian distribution and the weights π_k[8]. This procedure is called a scan of the IW Gibbs sampler.

3.2.3 Split-Merge Procedure

DAMM uses a Split-Merge Markov Chain Monte Carlo Procedure proposed by S. Jain and R. M. Neal to fit the Gaussians to the given trajectories and to compute the number of Gaussians [10]. $z = (z_1, ..., z_n)^T$ is the state vector containing all

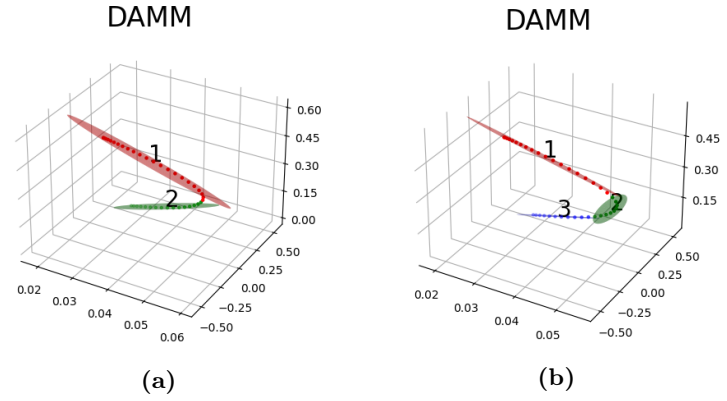
Algorithm 1 Instantiated-Weight Parallel Sampling [8]

```
Initialize T Number of iterations
Initialize t ← 0
for t = 0, ..., T do
    Select a Split/Merge proposal from z_t and define launch state z_t^l
    Compute candidate state z_t^* by multiple IW Gibbs sampler scans (Eq 16-18)
    Evaluate the acceptance probability a in Eq. 19
    if a > α then
        z_t ← z_t^*                                              ▷ Accept
    end if
    z_{t+1} ← z_t
end for
```



(a) **(b)**

Figure 6: Visualization of a merge of two components. (a) illustrates the GMM fitted with DAMM before the merge of component 2 and 3 and (b) shows the GMM after the merge.



(a) **(b)**

Figure 7: Visualization of a merge of two components. (a) illustrates the GMM fitted with DAMM before the split of component 2 and (b) shows the GMM after the split. The new Gaussians are 2 and 3.

assignments of each data point $\hat{\xi}_i \in \mathbb{R}^{d+1}$. If we are currently at state z of the Markov chain, we can propose a new state z^* by either splitting a Gaussian into two or merging two Gaussians into one. DAMM evaluates the new state via the Metropolis-Hasting acceptance probability,

$$a(z^*, z) = min\left[1, \frac{q(z|z^*)}{q(z*|z)} \frac{\pi(z^*)}{\pi(z)}\right], \tag{28}$$

where $\pi(z) = p(z|\hat{\xi})$ is the posteriori distribution from where we sample [21][8]. S. Jain and R. M. Neal showed, that it is unlikely for a random split or merged state z^* to be accepted [10]. Each split or merge proposal gives us a launch state z^l. This state is then multiple times scanned by the IW Gibbs sampler, defined in Eq. 25-27 [8]. After the final scan, we get the candidate state z^* which will then be accepted as the new state z if the Metropolis-Hasting acceptance probability is higher than a given threshold alpha. Algorithm 1 gives an overview of the described algorithm for the Split-Merge procedure in DAMM. The visualization of a split or merge of a Gaussian can be seen in Figures 6 and 7. If the variation of direction $(\sigma^2)^{dir}$ is high, then is $a(z^*, z)$ higher and the proposed state is more likely to be accepted [8].

Split Proposal

Assuming z^l is the state after the split of a Gaussian. Assign the data points corresponding to the original component randomly to the resulting new components, and then perform multiple scans of the IW Gibbs sampler by Eq. 25-27 [8] to reach the final state z^*. The IW Gibbs sampler scans only inside the new Gaussians. The probabilities in Eq. 28 are then defined as follows:

$$\frac{\pi(z^*)}{\pi(z)} = \frac{\prod_{z_i=z_1^*} \pi_1 \mathcal{N}(\hat{\xi}_{z_i}|\hat{\theta}_{z_1^*}) \prod_{z_i=z_2^*} \pi_2 \mathcal{N}(\hat{\xi}_{z_i}|\hat{\theta}_{z_2^*})}{\prod_{z_i=z_{12}^*} \mathcal{N}(\hat{\xi}_{z_i}|\hat{\theta}_{z_{12}^*})}, \tag{29}$$

$$\frac{q(z|z^*)}{q(z^*|z)} = \prod_{z_i=z_1^l} \prod_{z_i=z_2^l} \frac{\pi_{z_1^l} \mathcal{N}(\hat{\xi}_{z_i}|\hat{\theta}_{z_1^l}) + \pi_{z_2^l} \mathcal{N}(\hat{\xi}_{z_i}|\hat{\theta}_{z_2^l})}{\pi_{z_i} \mathcal{N}(\hat{\xi}_{z_i}|\hat{\theta}_{z_i})}, \tag{30}$$

where z_1^* and z_2^* are the proposed assignments of the two new components, z_{12}^* is the assignment of the old component after the final scan of the IW Gibbs sampler. z_1^l and z_2^l are the proposed assignments of the two new components and z_{12}^l is the assignment of the old component before the final scan of the IW Gibbs sampler [8].

Merge Proposal

The two components for a merge are chosen based on their Gaussian similarity and the Euclidean distance of their means. We reach the new state in the MCMC z^l after

multiple IW Gibbs scans are done. The terms of Eq. 28 computed as follows

$$\frac{\pi(z^m)}{\pi(z)} = \frac{\prod_{z_i=z_{12}^m} \mathcal{N}(\hat{\xi}_{z_i}|\hat{\theta}_{z_{12}^m})}{\prod_{z_i=z_1^m} \pi_{z_1^l} \mathcal{N}(\hat{\xi}_{z_i}|\hat{\theta}_{z_1^m}) \prod_{z_i=z_2^m} \pi_{z_2^l} \mathcal{N}(\hat{\xi}_{z_i}|\hat{\theta}_{z_2^m})}, \tag{31}$$

$$\frac{q(z|z^m)}{q(z^m|z)} = \prod_{z_i=z_1^l} \prod_{z_i=z_2^l} \frac{\pi_{z_i} \mathcal{N}(\hat{\xi}_{z_i}|\hat{\theta}_{z_i})}{\pi_{z_1^l} \mathcal{N}(\hat{\xi}_{z_i}|\hat{\theta}_{z_1^l}) + \pi_{z_2^l} \mathcal{N}(\hat{\xi}_{z_i}|\hat{\theta}_{z_2^l})}, \tag{32}$$

where z_1^l and z_2^l are the assignments of the components to be merged before the final scan, z_1^m and z_2^m are the original assignments of the components to be merged, and $z_{12}^m = z_1^m \cup z_2^m$ is the assignment of the new component, which is the union of the assignment of the merged Gaussians [8].

3.3 Task-Parameterized Gaussian Mixture Model

Task-parameterized Gaussian Mixture Models (TPGMM) aim to adapt their movements to new situations, given the parameters of their task. These task-parameters could be the object pose of the given tasks or landmarks the robot has to pass. We chose to use reference frames as our task-parameters [4]. In this thesis, we decided to use a time based TPGMM that uses multiple reference frames as task-parameters, as already described in 3.1.5.

3.3.1 Model definition

Let P be the number of frames and $\{b_i, A_i\}_{i=1}^P$ be the observer pose and rotation matrix of each frame with respect to the world frame. We define the state variable $\hat{\xi}$ as $\hat{\xi} = \begin{bmatrix} T & \hat{\xi}^1 & ... & \hat{\xi}^P \end{bmatrix}$, where $\hat{\xi}^i = (\xi - b_i)A_i^T$ are the trajectory data with respect to the i-th frame, ξ is the trajectory data of our data set in the world frame with length N, and $T = \{t_1, t_2, ..., t_N\}$ is the time dimension [4]. Our dataset is $D = \{\hat{\xi}_i\}_{i=1}^M$ where M is the number of trajectories.
To select and fit a model using TPGMM, a procedure equal to the ones described

in Section 3.1 can be followed. The Gaussian mean and covariance is defined as follows,

$$\hat{\mu} = \begin{bmatrix} \mu_T & \mu_1 & ... & \mu_P \end{bmatrix}, \qquad \hat{\Sigma} = \begin{bmatrix} \Sigma_{TT} & \Sigma_{T1} & \dots & \Sigma_{TP} \\ \Sigma_{1T} & \Sigma_{11} & \dots & \Sigma_{1P} \\ \vdots & \vdots & \ddots & \vdots \\ \Sigma_{PT} & \dots & \dots & \Sigma_{PP} \end{bmatrix}, \tag{33}$$

where μ_T, μ_i is the mean with respect to time dimension and i-th frame, Σ_{Ti} is the covariance matrix of the time dimension and the i-th frame. We define local marginals for every refernece frame as:

$$\hat{\mu}_i^l = \begin{bmatrix} \mu_T & \mu_i \end{bmatrix}, \qquad \hat{\Sigma}_i^l = \begin{bmatrix} \Sigma_{TT} & \Sigma_{Ti} \\ \Sigma_{iT} & \Sigma_{ii} \end{bmatrix}. \tag{34}$$

Therefore, we have a GMM for every task-parameter all with the same time dimension.

3.3.2 Adapt to new task parameters

Let $\{b_i^{new}, A_i^{new}\}$ be the observation and rotation matrix of the parameters in the new environment. First, the local marginals are transformed into the world frame (Eq. 35) to adapt to their new poses and orientations [4]. The transformed marginals are then joined into one GMM with K Gaussians, means $\hat{\mu}_k$ and covariance matrices $\hat{\Sigma}_k$ via the Gaussian product described in Eq. 36 [4], where $\hat{\Sigma}_{k,i}^w$ and $\hat{\mu}_l^w$ are the k-th mean and covariance matrix of the i-th transformed marginal.

$$\hat{\mu}_i^w = A_i^{new}\hat{\mu}_i^l + b_i^{new} \qquad \hat{\Sigma}_i^w = \hat{A}_i\hat{\Sigma}_i^l\hat{A}_i^T, \tag{35a}$$

$$\text{where } \hat{A}_i = \begin{bmatrix} 1 & \mathbf{0} \\ \mathbf{0} & A_i \end{bmatrix} \qquad \hat{b}_i = \begin{bmatrix} 0 & b_i \end{bmatrix} \tag{35b}$$

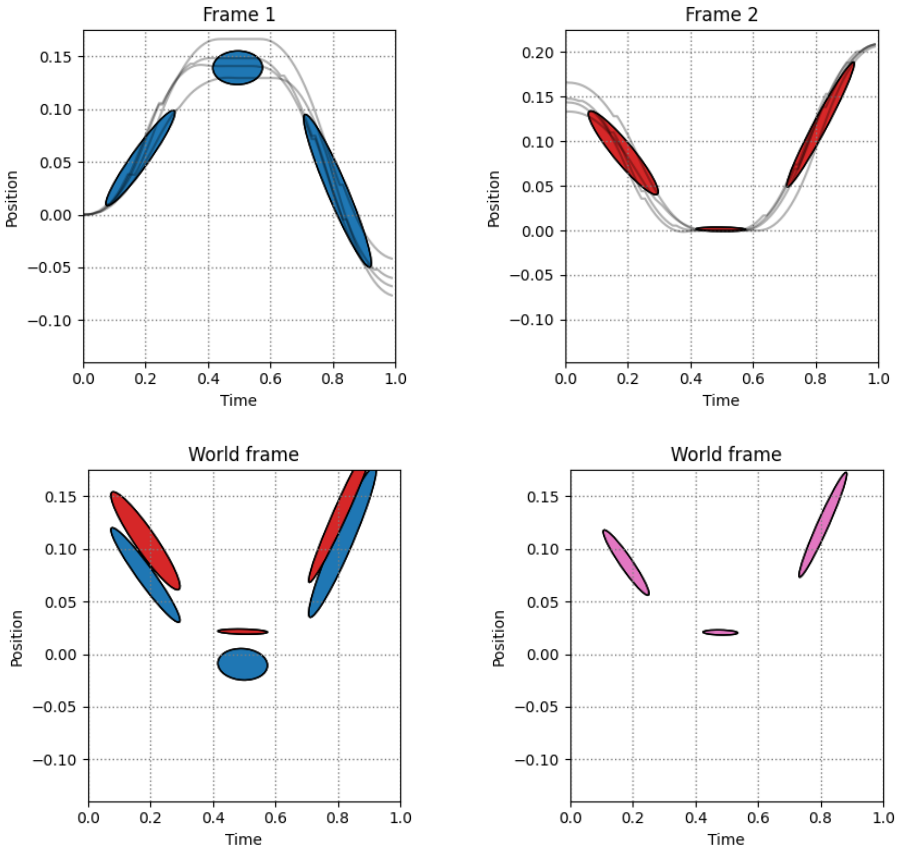


Figure 8: Visualization of a TPGMM procedure with two task-parameters. The two pictures at the top visualise the two marginals with respect to their task-parameter frame. The bottom left shows the transformed marginals in the world frame and the bottom right the joint GMM.

$$\hat{\mu}_k = \sum_{i=1}^{P} \left(\hat{\Sigma}_{k,i}^{w}\right)^{-1}, \qquad \hat{\Sigma}_k = \hat{\Sigma}_k^{w} \sum_{i=1}^{P} \left(\hat{\Sigma}_{k,i}^{w}\right)^{-1} \hat{\mu}_k^{w} \tag{36}$$

Figure 2 illustrates this procedure. By learning the marginals from the perspective of the task parameter frames and joining the marginals, the variance of the Gaussians, especially those close to the task parameters, is significantly reduced. This is the case, since the trajectories within the frames associated with the task parameters converge close to the task-parameters, leading to a more concentrated distribution of data points within the frame and the joint GMM.

3.4 Elastic Dynamical System

Elastic Dynamical System (Elastic-DS) motion policies are another approach, proposed by Li and Figueroa, to adapt GMMs to different task parameters [6]. This approach fits a GMM along the end-effector trajectory from one to another task parameter. For a new set of task parameters, the Elastic-DS approach transforms the Gaussians via Laplacian editing and constructs a new trajectory.

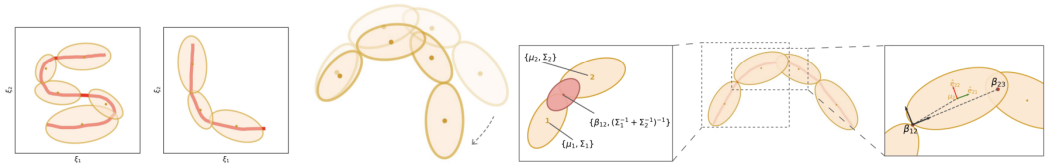
3.4.1 Preconditions

First, we fit a GMM on the given trajectories. The GMM should fit the trajectory via the change in direction, as illustrated in Figure 9. It is easier to later transform this GMM via Laplacian Editing if the GMM is fitted along the direction of the demonstrations [6]. While Li and Figueroa used a PC-GMM to fit a GMM on the data, are we using the successor DAMM as described in Section 3.2 [11][8].

The i-th task-parameter is described as a set of geometric constraint descriptors $O_i = \{o_{enter}, o_{exit}\} \in SE(3)^2$ of multiple trajectories, where o_{enter} and o_{exit} are the two endpoint poses of the corresponding trajectory and $SE(3)$ are the frames of the task-parameters.

The joints β_i are defined as the mean of the Gaussian product of the i-th and (i+1)-th Gaussian, β_0 is defined as the start, and β_n is the end of the trajectory . The Gaussian product of two Gaussians was defined by Eq. 36 and can be seen in Figure 9c in the right picture in red.

The Gaussian joints β can be transformed from Cartesian space into Laplacian coordinates ∇ as



(a) GMM from trajectories (b) GMM Chain (c) Gaussian *joints* illustration in 2D

Figure 9: Illustration of a GMM fittet along the directional change along the trajectoy data (a), the Gaussian chain transformation (b) and the Gaussian joints β. Source: Adapted from [6].

$$\nabla = L\beta, \tag{37}$$

$$L_{ij} = \begin{cases} 1 & \text{if } i = j \\ -\frac{w_{ij}}{\sum_{j \in \mathbf{N}_i} w_{ij}} & \text{if } j \in \mathbf{N}_i \\ 0 & \text{else} \end{cases}, \tag{38}$$

where $L \in \mathbb{R}^{nxn}$ is the Laplacian matrix, n the number of Gaussian joints, $\mathbf{N}_i$ is the set of neighbor points of β_i, and w_{ij} a weight [6][22].

3.4.2 Transforming Gaussian mixture models

Let β^O be the joints of the initial GMM. To transform the Gaussian Mixture Model according to a new task-parameter and the geometric descriptors O_i the initial joints are transformed into Laplacian coordinates, as described in the previous Section, and a least-square optimization problem is defined as,

$$\min_{\beta^{new}} ||L\beta^{new} - \nabla||_2^2 \; s.t. \begin{cases} T_{0,1}(\beta_0^{new}, \beta_1^{new}) = O_{start} \\ T_{n-1,n}(\beta_{n-1}^{new}, \beta_n^{new}) = O_{end} \end{cases}, \tag{39}$$

where L and ∇ are as defined in Section 3.4.1, O_{start} is o_{exit} of our current task-parameter and O_{end} is o_{entry} of our new task-parameters, or to put it simply, the start

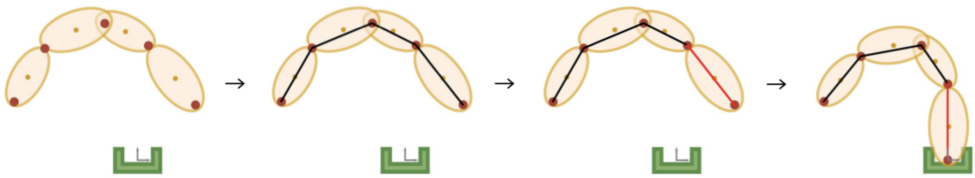


Figure 10: At the start we can see the initial joints β^0 of our GMM in red. Then is a piece-wise linear trajectory between the joints formed. The linear trajectory between the last joint is then aligned with the new task parameter (the green polygon) and the other joints β^{new} are calculated by Eq. 32. Note that the linear trajectories are just for understanding and are not computed in the transforming algorithm. Source: Adapted from [6].

and end transformations of the trajectory, we would like to construct. The constraints of the optimization problem align the first and the last two joints with the start and end transformations of our geometric descriptor. $T_{i,j}(\beta_i, \beta_j) = O_{start}$ represents the transformation from β_i to β_j. The solution to this optimization problem gives us the new joints β_{new} of the transformed GMM. We align the GMM according to the new joints, by transforming the i-th Gaussian similar to the transformation of $(\beta_i^0, \beta_{i+1}^0)$ to $(\beta_i^{new}, \beta_{i+1}^{new})$. Figure 10 illustrates the process of transforming a GMM to a new task parameter.

3.4.3 Create end-effector poses

To create the new trajectory for our end-effector pose, we solve another least-square optimization problem via a Linear Parameter Varying Dynamical System (LPV-DS) formulation [6]. First, the first joint β_0^{new} and last joint β_n^{new} are connected with a linear trajectory ζ. The trajectory is adjusted to pass through β^{new} with Laplacian editing,

$$\min_{\zeta} ||L_\zeta \beta^{new} - \nabla_\zeta||_2^2 \; s.t. \; \zeta_i = \beta_q^{new}, \tag{40}$$

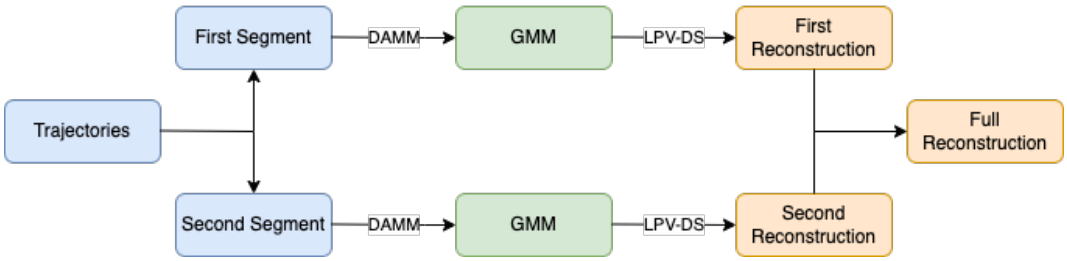


Figure 11: This flow chart visualizes the procedure of Elastic-DS with three geometric descriptors. First, every trajectory is divided into two segments at the position of the second geometric descriptor. Then a GMM is fitted for every segment. For a new set of geometric descriptor we construct a new trajectory for every segment via LPV-DS and combine them in the last step.

where $i \in \{1, .., p-1\}$, p is the number of data points in the trajectory, and ζ_j is the i-th point of the trajectory.

3.4.4 Expansion to multiple task-parameters

In tasks with multiple task-parameters the trajectory data where the end-effector reaches each task parameter is split [6]. For example, for the StackCube task, where an end-effector has to stack one cube onto another, we would segment the trajectory into two parts. The first segment would go from the start pose of the end-effector to the first cube, while the second segment would start at the first and end at the second cube. We are able to tailor the model for every segment by fitting one GMM for each segment.

For a new task environment, we transform each GMM, as described in Section 3.4.2, according to the new task parameters. We then create the trajectory for every segment and combine all trajectories into one. The flow chart in Figure 11 illustrates the procedure for two task-parameters.

4 Setup

In this Section, we define our problem we want to solve and describe the different tasks we used and the Maniskill2 environment in which we simulated our tasks [23].

4.1 Problem Definition

Understanding the performance of different models for a variety of tasks is crucial in robot learning. In this work, we evaluated two different policies and fitting methods for GMMs based on their ability to learn the end-effector motion from demonstrations and adapt it to new task-setups and task-parameters.
Let $D = \{\{\xi_{t,i}\}_{t=1}^{T}\}_{i=1}^{M}$ be our demonstration set of M robot trajectories, where $\xi_{t,i} \in \mathbb{R}^3$ represent the end-effector pose of the i-th trajectory at time step t in a three-dimensional Cartesian space. $\{\{A_{j,i}, b_{j,i}\}_{j=1}^{P}\}_{i=1}^{M}$ is the set of our task-parameters, where $A_{j,i}, b_{j,i}$ are the orientation and pose of the i-th task-parameter of the j-th frame and $\{\{g_{t,i}\}_{t=1}^{T}\}_{i=1}^{M}$ is the set of gripper states of the end-effector. In this work, we do not consider the change of orientation or pose of the different task-parameters since DAMM can not handle non Euclidean data and the TPGMM only with a Riemanschen extension. The goal of this work is to find the parameters $\{\pi_i, \mu_i, \Sigma_i\}_i^K$ of a Gaussian Mixture Model and predict the end-effector motion for a new set of task-parameters $\{A_j^{new}, b_j^{new}\}_{j=1}^{P}$ based on the GMM. We consider the start pose and orientation of the end-effector as task-parameter.

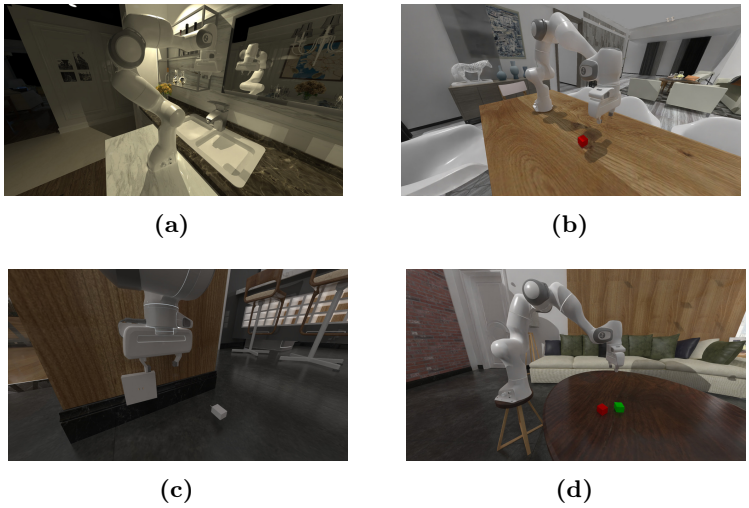
4.2 Environment

To evaluate our models in simulation, we use Maniskill2 , which offers multiple realistic robot manipulation tasks [23]. In order to properly evaluate our models, we use a set of diverse tasks:

1. TurnFaucet: The end-effector must turn a faucet
2. PickCube: The end-effector must pick a cube and lift it
3. PlugCharger: Pick up a charger and plug it into the receptacle
4. StackCube: Pick a cube and stack it onto another cube

The TurnFaucet and PickCube tasks are single object tasks. TurnFaucet is a simple task which offers a low variance because the faucet pose and orientation are the same for every task environment. Only the start pose of the end-effector changes respectively in different task set-ups. The PickCube task is a slightly more advanced task with a higher variance, because the transformation of the cube varies in every demonstration. Compared to other tasks, it has a small horizon.
PlugCharger and StackCube are multi-object tasks with a higher horizon. We have three task-parameters in both tasks. PlugCharger has the start pose of the end-effector, the to be picked charger and the receptacle pose. The end-effector start and the charger pose and orientation have a high variance throughout the demonstrations, while the receptacle is fixed in the world frame and therefore has a low variance. The most difficult task is StackCube. All task-parameters have different poses and orientations across all demonstrations. Therefore has this task the most variance of all tasks and the policies need to adapt to a multitude of changing parameters in new environments.

We evaluated our offline learned models on every task using different metrics. However, the TurnFaucet and PlugCharger are not suitable for evaluating the task success of



(a) **(b)**

(c) **(d)**

Figure 12: Illustration of the tasks we evaluated. (a) TurnFaucet, (b) PickCube, (c) PlugCharger, and (d) StackCube. Source adapted from [23].

our policies in an online simulation. In the scope of this thesis, we focused on the prediction of the end-effectors pose and not the orientation. In both tasks, orientation is crucial for the task success. If we only take the end-effector pose and not the task success into account in an offline prediction of the demonstrations, they are still suitable tasks to evaluate our models with different metrics.

4.3 Data Generation

We generated our demonstrations and the corresponding information about our task-parameters using the build in expert policies of the Maniskill2 benchmark [23]. For every task, we executed ten different simulations and fetched the information of the end-effector trajectory $\{\xi_t\}_{t=1}^{T_i}$, the corresponding task-parameters $\{A_j, b_j\}_{j=1}^{P}$ and the state of the end-effector gripper $\{g_i\}_{i=1}^{T}$ at every time step. After which, we sampled every trajectory to the same size T with a simple sample algorithm 2. We did this to make it easier to implement our models.

Algorithm 2 Sample trajectory

sample length T, trajectory ξ
sampled_trajectory = []
for t = 1,..., T **do**
    indices = $\lfloor$ t/len(ξ)$\rceil$
    sampled_trajectory.append(ξ[indices])
end for
return sampled_trajectory

5 Policies

In this Section, we describe the two policies we evaluate to solve the tasks. First, we explain how we prepared the given demonstrations to suit the given model to improve their performance. We then briefly describe the models and fitting methods we use, which were described in Section 3, to reconstruct our demonstrations in an offline phase. Afterwards, we describe the online phase, where we adapted our, in the offline phase, learned model to new simulations in Maniskill2 [23].

5.1 Elastic-DS Policy

5.1.1 Data preparation

To execute the Elastic-DS, we must define the geometric descriptors for every task-parameter. Since we have fetched the pose b_i of all task-parameters, we have to define the rotation of our geometric descriptors. Since the geometric descriptors are not the same as the transformations A_i, we compute the rotation of $o_{i,exit}$ as the angle between the x-axis of the rotation matrix A_i and the next task-parameter b_{i+1}. In the scope of the thesis, we set o_{exit} by hand, as T. Li and N. Figueroa [6].

5.1.2 Learning Phase

Let $D = \{\xi_{t,i}, \dot{\xi_{t,i}}\}_{i=1}^{M}$ be our segmented trajectories. For every segment, we learn a GMM utilizing the DAMM fitting method, as described in Section 3.3. We noticed

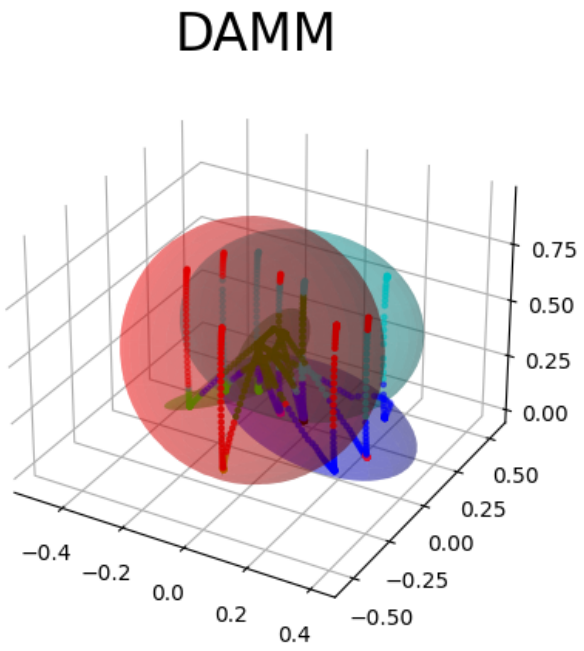


Figure 13: DAMM fitted on multi-pose demonstrations of the PickCube task. DAMM detects multiple changes in direction and different trajectories and therefore is not able to fit the GMM accordingly.

that DAMM was not able to fit a GMM on multiple trajectory data, which can be seen in Figure 13. T. Li and N. Figueroa proposed that Elastic-DS is capable of learning a policy from only one demonstration, which we adapted [6].

5.1.3 Offline Policy

To demonstrate the ability of Elastic-DS to reconstruct end-effector trajectories, we implement an offline policy. Given the geometric descriptors O_i of each task, we can create end-effector poses for every task as described in Section 3.4. We transform each GMM according to the new geometric descriptors (Section 3.4.2) and construct new end-effector trajectories given the transformed GMMs [6]. Afterwards, we combine every constructed trajectory into one trajectory (see Figure 9). Figure 14 shows two example reconstructions of the Elastic-DS offline policy.

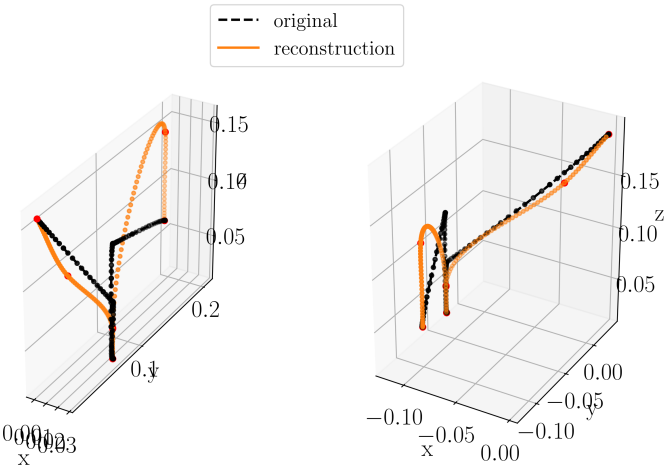


Figure 14: Elastic-DS offline reconstruction of the StackCube task. The black line represents the original trajectory, while the orange line represents the reconstruction. The red dots represent joints β of the transformed GMMs that the trajectory has to pass. The first reconstruction is not similar to the original trajectory since the reconstructed end-effector always approach the second cube from above and the original trajectory sometimes aprroaches the cube from the side.

5.1.4 Online Policy

To evaluate the ability of the Elastic-DS approach to fulfill tasks in novel environments, we use the Maniskill2 environment to simulate an end-effector in real-time according to our policy predictions.

At the start of every episode, we fetch the pose and rotation of every task-parameter. According to this information, we define our geometric descriptors, as described in 5.1.1. Given our geometric descriptors O_i, we transform the GMMs and pre-compute the end-effector trajectory for every segment, as described in Section 3.4.

We iterate over each trajectory, and compute the action for every step, and send it to the Maniskill2 environment. Since we do not model the gripper state in the Elastic-DS policy, we set the gripper state, according to the current task, by hand at the end of every trajectory segment. Figure 15 illustrates a flow chart of the Elastic-DS online policy.

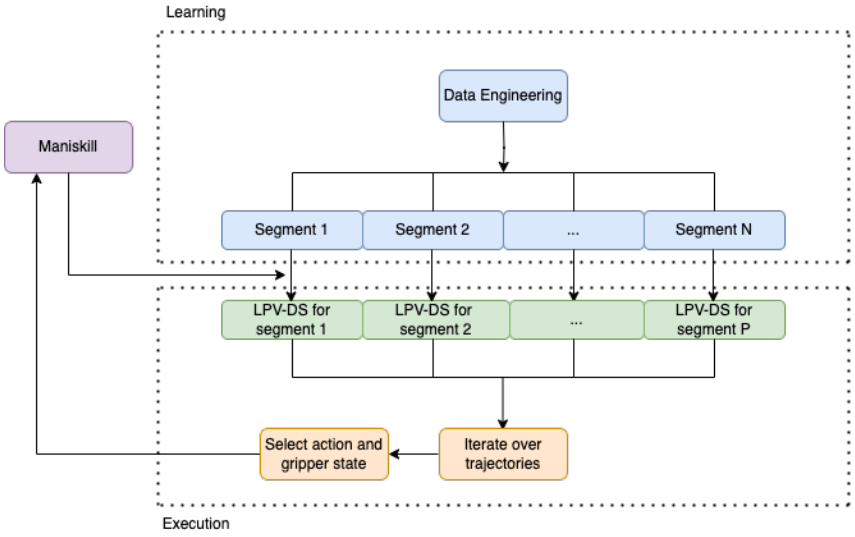


Figure 15: Flow chart of the Elastic-DS online policy.

5.2 Time-Based Task-Parameterized Gaussian Mixture Model Policy

This Section describes the time-based TPGMM policy. We will refer to the time-based TPGMM fitted via Expectation Maximization as EM-TPGMM and to TPGMM fitted via the Directionality-aware Mixture Model as DAMM-TPGMM.

5.2.1 Data Preparation

In this policy, we construct trajectories given each time step and model the gripper actions of our end-effector to enable our policy to fulfill the pick-and-place task. To enable the TPGMM to learn the relationship between end-effector motion, time, and gripper actions, we must include time and gripper dimensions into our demonstrations. For this purpose, we assign every end-effector pose the corresponding time-step and gripper state. Additionally, we transform every end-effector trajectory ξ_i as described in chapter 3.3.1 and receive new sets of demonstrations $\hat{D} = \{\{t, \hat{\xi}_{t,i}, g_{t,i}\}_{t=1}^{T}\}_{i=1}^{M}$.

For every dimension in $\hat{D}$, DAMM needs a corresponding velocity dimension. Therefore, we simply compute the velocity of the end-effector and the time dimension at time step t as the difference of the current time step and the next time step $t+1$ (Eq. 41) and normalize these velocities. Even though the gripper dimension changes every few time steps, we do not want the DAMM to include the gripper velocity in the split/merge procedure and therefore set it to zero at every time step. We do this since the change of direction at the grasp point would be very high, and it is more likely that a Gaussian is split at this time step. This would cause one Gaussian before and one after the grasp point, which would lead to a higher variance at the grasp point. Since the grasp point is the most important step to securing task success, we want a low variance at this point and therefore just ignore the velocity of the gripper dimension by assigning it to zero.

$$\dot{x}_t = x_t - x_{t+1} \tag{41}$$

We append these velocities and get our new demonstrations for the DAMM-TPGMM $\hat{D} = \{\{t, \hat{\xi}_{t,i}, g_{t,i}, \dot{t}, \dot{\xi}_{t,i}, \dot{g}_{t,i}\}_{t=1}^{T}\}_{i=1}^{M}$.

DAMM is sensitive to outliers. If only one trajectory has an outlier where the motion is different, DAMM has problems to split/merge the Gaussian in the area where the outlier occurs. The reason is that this outlier has different directions at one or multiple time steps and changes the variation of direction for the whole data set, see Figure 16c. The outliers must be removed from the dataset. In the scope of this thesis, this was done by hand, which was possible because of the small set of trajectories for every task. The difference between the DAMM-TPGMM on the dataset with and without outliers is illustrated in Figure 16.

Another issue is the temporal alignment of the trajectories. Which means that the motion of each trajectory is a little shifted in time. If the time difference between the motion is too large, DAMM has problems detecting a change in direction and split/merge the Gaussians appropriately. For example, if the end-effector picks a cube,

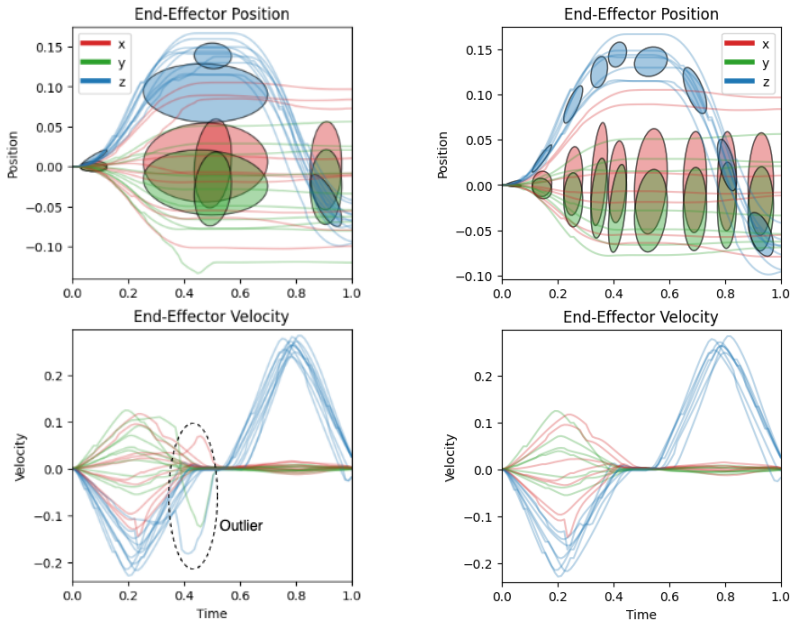


Figure 16: DAMM-TPGMM on (a) raw trajectory and (b) adjusted trajectory data of the PickCube task with respect of the end-effector frame.

there is a motion towards the cube that DAMM detects. At the time step t, where the end-effector changes the direction of the gripper, the velocity is zero and DAMM would detect a change in direction. However, if other demonstration trajectories remain in motion with higher velocities, the overall change in direction across all demonstrations is not that high. It could appear that DAMM does not detect the change in motion and does not fit the demonstrations accordingly. Von Hartz et al. proposed the idea of segmenting the demonstrations to lower the variance across the demonstrations close to the task-parameters [24]. We segment each trajectory according to the velocity. We split the trajectory at the time step, where the velocity is lower than a given threshold close to zero, and then sample them to the same size with our sample algorithm 2. The segmented data can be seen in Figure 17. In the demonstrations of our tasks, the end-effector velocity is only zero, when the end-effector approaches a task-parameter and picks/places it. Therefore ,the grasp points of our demonstrations are now aligned at the same time step. We split the

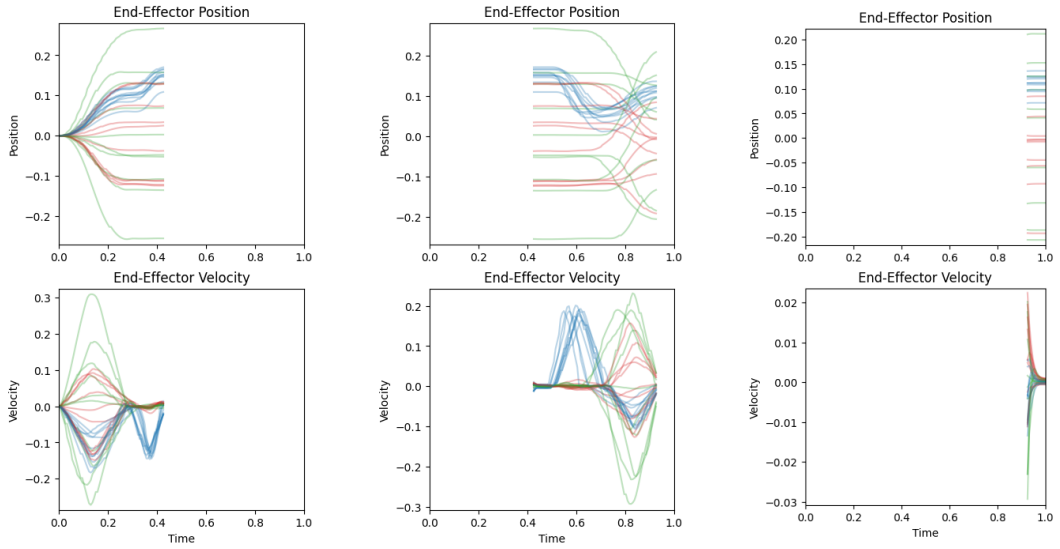


Figure 17: Segmental data of the StackCube demonstrations.

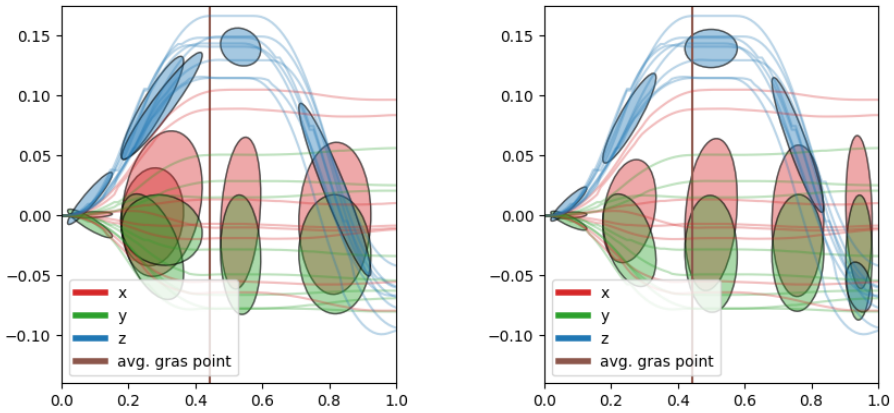
demonstrations for the EM-TPGMM the same way to achieve a lower variance across the demonstrations close to the task-parameters. This allows us to better model the end-effector pose and gripper dimension close to the task-parameters.

5.2.2 Learning Phase

EM-TPGMM

The first step in learning the EM-TPGMM was computing the optimal hyperparameters of the model. We use a grid search to compute the hyperparameters of our model, as described in Section 3.1.4. Due to the fact, that our EM-TPGMM only has a small number of hyperparameters, the grid search is not computationally expensive and returns hyperparameters close to the optimum.

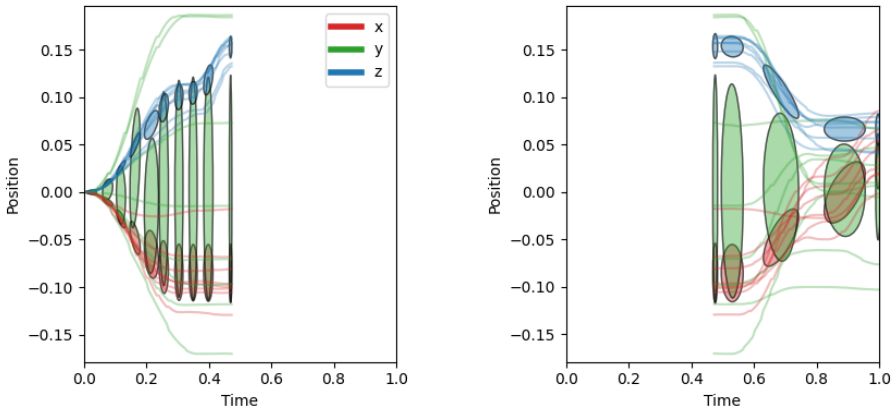
Given the hyperparameters, we first initialized a TPGMM, for each segment, time based (Section 3.1.2), before fitting our models to the segmented demonstrations via Expectation Maximization. EM encounters difficulties fitting the end-effector motion with the inclusion of the additional gripper dimension. The end-effector gripper is,



(a) **(b)**

Figure 18: Difference in the fit of a GMM fittet with EM when excluding the gripper dimension (a) and when including the gripper dimension (b) in the expectation step.

most of the time, either fully closed (-1) or open (1). Only in a short period of time, around the grasp point, is the value of the gripper dimension g_t in between -1 and 1. The issue with the EM algorithm lies in the E-step. Because there are only a small number of data points with $-1 < g_t < 1$ which leads to a lower assignment probability $z_{t,i}$ for these points. Because the Gaussians are updated according to their assignment probability $z_{t,i}$ in the M-step, this leads the EM to "push" the Gaussians further from these points to maximize the log-likelihood, as illustrated in Figure 18a. This leads to the issue, that the EM-TPGMM has a higher uncertainty close to the grasp point. The area close to the grasp points of our tasks is the most important, since they are crucial for the task success. E.g., it is disadvantageous for the PickCube task if we model the end-effector motion close to the cube imprecisely, we could miss the cube. To solve this, we adjust the E-step of the EM algorithm in a way that the gripper dimension is not taken into account for computing the assignment probability $z_{t,i}$, as follows:



(a) **(b)**

Figure 19: Segmented GMMs with first and last component fixed.

$$z_{t,i} = \frac{\tilde{\pi}_i N(\tilde{x}_t | \tilde{\mu}_i, \tilde{\Sigma}_i)}{\sum_{j=1}^{K} \tilde{\pi}_j N(\tilde{x}_i | \tilde{\mu}_j, \ \tilde{\Sigma}_j)}, \tag{42}$$

$$\text{where } \tilde{x}_t = (t, \hat{\xi}_t), \tag{43}$$

$$\tilde{\mu}_i = (\tilde{\mu}_{i,t}, \tilde{\mu}_{i,\hat{\xi}_t}), \tag{44}$$

$$\tilde{\Sigma}_i = (\tilde{\Sigma}_{i,t}, \tilde{\Sigma}_{i,\hat{\xi}_t}). \tag{45}$$

In the M-step, the Gaussians (including the gripper dimension) are updated according to the time and motion of end-effector (Eq. 5-7). The effect of the improvement can be seen in Figure 18.

To ensure smooth transitions between GMMs of each segment and achieve lower variance near the task-parameters, we fix the first and last component of each GMM. To fix the components, first we initialize the first and last Gaussian as the mean and covariance of the first and last two end-effector poses as follows:

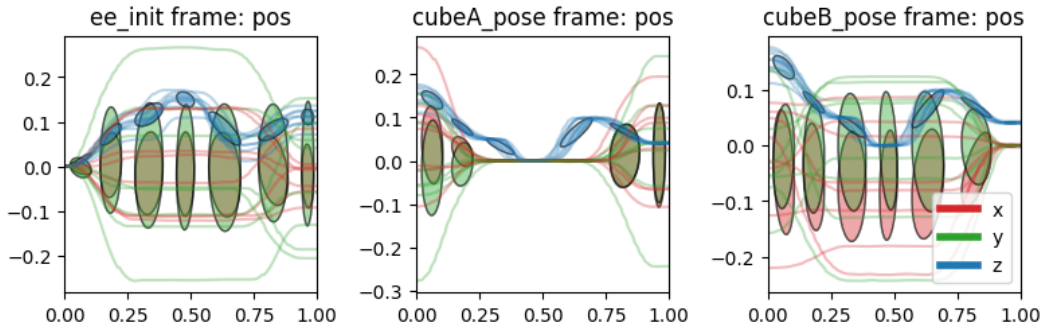


Figure 20: Visualization of the time and end-effector dimensions of the local marginals of EM-TPGMM fittet on the StackCube demonstrations.

$$\mu_1 = \frac{x_1 + x_2}{2}, \qquad \Sigma_1 = \frac{\sum_{i=1}^{2}(x_i - \mu_i)^T((x_i - \mu_i)}{2}, \qquad \pi_1 = \frac{2}{K}, \tag{46}$$

$$\mu_K = \frac{x_M + x_{M-1}}{2}, \qquad \Sigma_1 = \frac{\sum_{i=M-1}^{M}(x_i - \mu_i)^T((x_i - \mu_i)}{2}, \qquad \pi_1 = \frac{2}{K}, \tag{47}$$

where K is the number of components of the TPGMM. We adapt the M-step of the Expectation Maximization algorithm to not update the mean and covariance of the first and last component but the priors. Figure 19 illustrates an example of a segmented GMM with fixed Gaussians.

After fitting the EM-TPGMM we provide the local marginals with respect to the task-parameter frames, as described in Section 3.2.2, with the only difference being that every marginal includes the gripper dimension (see Figure 20).

DAMM-TPGMM

DAMM-TPGMM does not need a hyperparameter optimization since the algorithm determines the hyperparameters during the split/merge procedure, as explained in Section 3.3. We learn one DAMM-TPGMM for every segment of our demonstrations and concatenate the mean, weights, and covariance of the models, as described in Eq. 48-51 and illustrated in Figure 21. The problem that the EM-TPGM encounters with

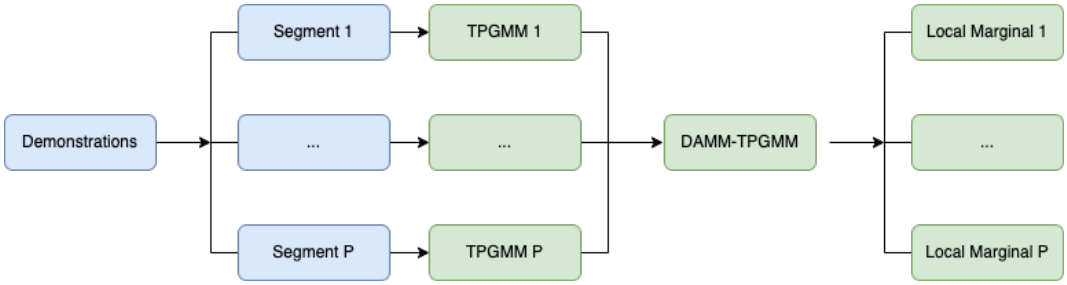


Figure 21: Flow-chart of the TPGMM learning phase.

the gripper dimension was already resolved for the DAMM-TPGMM in the previous Section by setting $\dot{g}_t = 0$.

$$Concat(\{\pi_i^1, \mu_i^1, \Sigma_i^1\}_{i=1}^{K}, \{\pi_i^1, \mu_i^1, \Sigma_i^1\}_{i=1}^{K'}) = \{\pi_i^{12}, \mu_i^{12}, \Sigma_i^{12}\}_{i=1}^{\hat{K}^{new}}, \tag{48}$$

$$\text{where } \pi_i^{12} = (\frac{\pi_1^1}{2}, ..., \frac{\pi_K^1}{2}, \frac{\pi_1^2}{2}, ..., \frac{\pi_{\hat{K}}^2}{2}), \tag{49}$$

$$\mu_i^{12} = \begin{bmatrix} \mu_1^1, ..., \mu_K^1 \\ \mu_1^2, ..., \mu_{\hat{K}}^2 \end{bmatrix}, \tag{50}$$

$$\Sigma_i^{12} = \begin{bmatrix} \Sigma_1^1, ..., \Sigma_K^1 \\ \Sigma_1^2, ..., \Sigma_{\hat{K}}^2 \end{bmatrix}, \tag{51}$$

After the DAMM-TPGMM is concatenated, we compute the local marginals, as we did for the EM-TPGMM.

5.2.3 Offline Policy

In our offline policy, we want to reconstruct the trajectories of our demonstrations, given their task-parameters, which we fetched from Maniskill2. Since the offline policy is the same for EM-TPGMM and DAMM-TPGMM we will refer to TPGMM in the ongoing Section.

Let $\{A_i, b_i\}_{i=1}^{P}$ be the task-parameters for the given task. Since we augment the time and gripper dimensions to our model, we define the pose and rotation of the

task-parameters so that these dimensions are not modulated [4]:

$$\hat{A}_i = \begin{bmatrix} 1 & \mathbf{0} & 0 \\ \mathbf{0} & A_i & \mathbf{0} \\ 0 & \mathbf{0} & 1 \end{bmatrix}, \qquad \hat{b}_i = \begin{bmatrix} 0 \\ b_i \\ 0 \end{bmatrix}. \tag{52}$$

We then compute the transformed marginals to adapt to the task-parameters, as defined by Eq. 35 and then join these marginals via the Gaussian product (Eq. 36). For every time step t of the original trajectory, we perform Gaussian Mixture Regression and predict the pose and gripper state of the end-effector at this time step. With this approach, we generate a reconstructed trajectory of the same length and with the same time steps. Example reconstructions can be seen in Figure 24.

5.2.4 Online Policy

To evaluate the performance of our models based on their task success, we simulate new task set-ups in the Maniskill2 environment using the offline learned models. We define the end-effector pose and gripper state as our action space.

At the start of every episode, we fetch the pose b_i and the rotation A_i of every task-parameter. We adapt the pose and rotation to the time and gripper dimension according to Eq. 52 and transform the local marginals, as described in Section 3.3.2. The marginals are joined via their Gaussian product (Eq. 36). We use the resulting GMM to predict the end-effector pose and gripper state in this episode via GMR.

Since we have a time-based model and do not want to predict the end-effector pose at every step of Maniskill2, we must manage the time steps by ourselves. We start our model by predicting the end-effector pose and gripper state at $t_{curr} = 0$ with GMR. We compute the pose and gripper delta for every step and send the action to Maniskill2, until the end-effector reaches our prediction. After we reach our prediction,

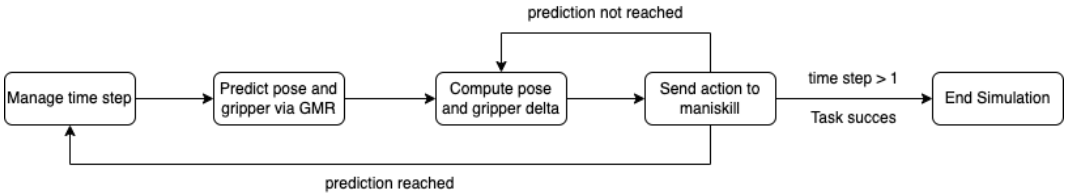


Figure 22: Automata that illustrates the prediction process of the TPGMM online policy.

we compute a new time step as

$$t_{curr} = t_{curr} + t_{scale}, \tag{53}$$

where t_{scale} represents the size of our time steps. We then predict the next end-effector pose and gripper state given the new time step via GMR. We repeat this as previously defined multiple times or until Maniskill2 ends the episode, due to task success. Figure 22 illustrates an automata that describes the prediction process of the TPGMM online policy, and Figure 23 gives an overview of the TPGMM online policy.

For a better comparison to the Elastic-DS policy, we additionally learn a policy that excludes the gripper dimension and implements the grasp point by hand for every task.

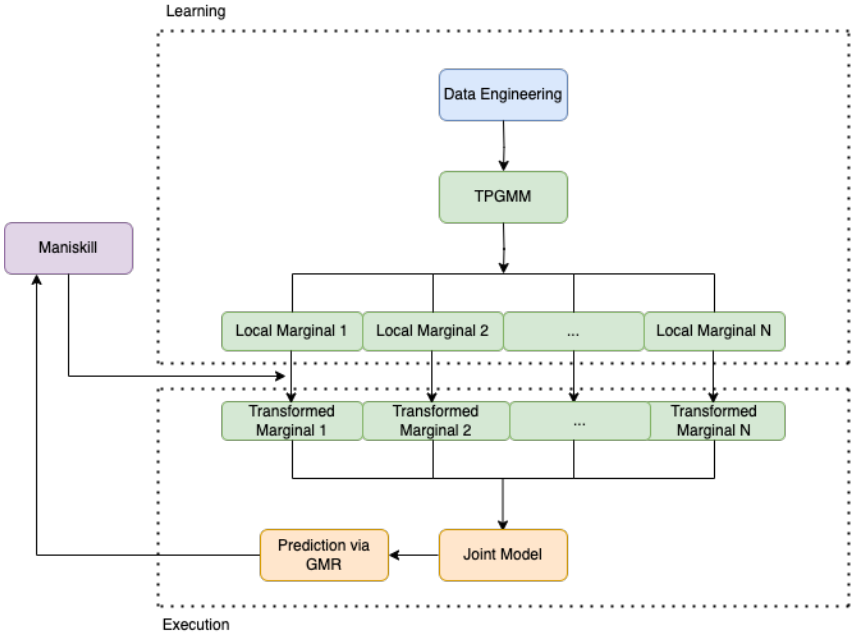


Figure 23: Flow chart of TPGMM online policy.

6 Experimental Results

6.1 Offline Experiments

In this Section, we evaluated our models ability to reconstruct the trajectories of our demonstrations. We use the Mean Absolute Error (MAE) and the Standard Derivation (STDV) over the complete trajectories and MAE and STDV three time steps before and after the grasp point (GMAE, GSTDV) as our metrics:

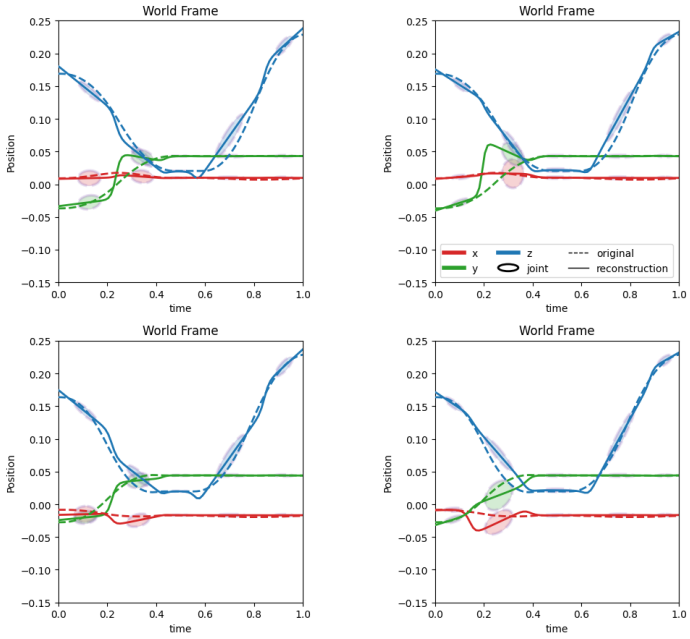
$$\text{MAE}(X, Y) = \frac{1}{|X|} \sum_{i=1}^{N} |x_i - y_i|, \tag{54}$$

$$\text{STDV}(X, Y) = \frac{1}{|X|} \sum_{i=1}^{N} (x_i - y_i)^2, \tag{55}$$

$$\text{GMAE}(X, Y) = \frac{1}{|U_3(t)|} \sum_{i \in U_3(t)} |x_i - y_i|, \tag{56}$$

$$\text{GSTDV}(X, Y) = \frac{1}{|U_3(t)|} \sum_{i \in U_3(t)} (x_i - y_i)^2, \tag{57}$$

where $U_\epsilon(x)) = \{x^{'} | |x^{'} - x| \leq \epsilon\}$ is the epsilon sphere and t is the grasp point. Table 1-4 shows the results of these metrics applied to our offline policies for the PickCube, TurnFaucet, PlugCharger, and StackCube tasks. The EM-TPGMM and DAMM-TPGMM show similar results throughout every task and metric. In general, the GMAE and GSTDV are smaller than the MAE and STDV. Since the covariance of the Gaussians close to the task-parameters is small in the respective task-parameter marginal. Therefore, the covariance of the joint Gaussian is also low [4]. Additionally,



(a) **(b)**

Figure 24: Visualization of two offline reconstructions of the PickCube demonstrations with (a) DAMM-TPGMM and (b) EM-TPGMM. We can see that both models have difficulties to reconstruct the trajectory around time 0.3. This is the case, as the end-effector moves in this period to the cube and its position has higher variance due to different cube positions. Since this area has a high variance in the end-effector and cube frame, the joint model also has a higher variance in this area, which leads to more inaccurate predictions.

we lower the variance in this area, for the EM-TPGMM, by fixing the Gaussians at the position of the task-parameters, as described in Section 5.1.2. Figure 24 illustrates examples of the reconstruction of the EM-TPGMM and DAMM-TPGMM.

However, the Elastic-DS has a far higher result for the MAE and STDV over all tasks. We can explain this behavior since the Elastic-DS approach adapts to new task-parameters by aligning the first and last joint with the geometric descriptors and then solving a least-square optimization problem (Eq. 39). This approach focuses on correctly fitting the start and end positions of the task-parameters but lacks the ability to reconstruct the motion between the task-parameters. Elastic-DS shows

PickCube				
Model	MAE	STDV	GMAE	GSTDV
EM-TPGMM	0.0058	**0.0002**	0.0015	0.000082
DAMM-TPGMM	**0.004**	0.0009	**0.0006**	0.00007
Elastic-DS	0.023	0.004	0.00088	**0.00001**

Table 1: Evaluation of the PickCube task. Optimal results are bold.

TurnFaucet				
Model	MAE	STDV	GMAE	GSTDV
EM-TPGMM	**0.0056**	**0.0013**	**0.0047**	0.0003
DAMM-TPGMM	0.006	0.0008	0.0061	0.0008
Elastic-DS	0.0099	0.0032	0.00495	**0.000276**

Table 2: Evaluation of the TurnFaucet task. Optimal results are bold.

PlugCharger				
Model	MAE	STDV	GMAE	GSTDV
EM-TPGMM	**0.005**	**0.0004**	**0.0021**	0.00008
DAMM-TPGMM	0.007	0.0009	0.0025	0.00010
Elastic-DS	0.017	0.008	0.004	**0.00005**

Table 3: Evaluation of the PlugCharger task. Optimal results are bold.

StackCube				
Model	MAE	STDV	GMAE	GSTDV
EM-TPGMM	0.0019	0.00037	0.0015	0.00009
DAMM-TPGMM	**0.0014**	**0.00033**	0.0026	0.00007
Elastic-DS	0.011	0.0074	**0.0012**	**0.00001**

Table 4: Evaluation of the StackCube task. Optimal results are bold.

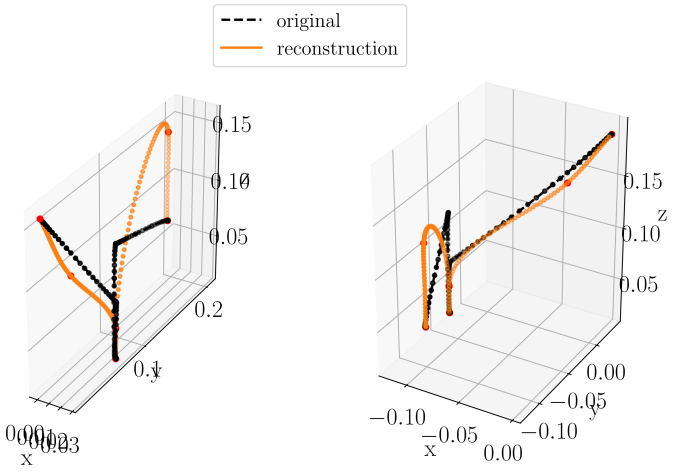
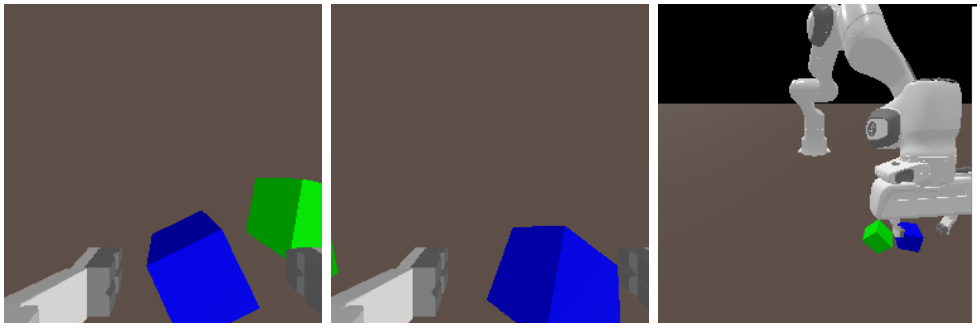


Figure 25: Elastic-DS offline reconstruction of the StackCube task. The black line represents the original trajectory, while the orange represents the reconstruction. The red dots represent joints β of the transformed GMMs. The cubes in the left reconstruction are to the right of the end-effector and to the left in the right reconstruction. In the demonstration, are the cubes to the left of the end-effector, which explains the worse reconstruction in the left plot.

difficulties transforming the GMMs accordingly to the motion in task-setups, where the task-parameters are placed in another direction with respect to the end-effector. Figure 25 shows a example of this behavior. However, as the Elastic-DS aligns the start and end position of the trajectory with the task-parameters, the GMAE scores are similar to the EM-TPGMM, and the GSTDV even better. Nevertheless, do we hypothesize that the worse reconstruction of the general motion of the end-effector could lead to problems in the simulations.

6.2 Online Experiments

To demonstrate the ability of the models to adapt to novel task set-ups, we simulate real-time scenarios of the StackCube and PickCube task. For a better comparison between TPGMM and Elastic-DS we simulate both tasks without modeling the gripper state. Table 5 shows the task success of every model over one-hundred



(a) (b) (c)

Figure 26: Failure cases of the StackCube task. (a) illustrates the case that the cubes are to0 close. All models fail to succed in this case, because end-effector is stuck at the second cube. This could be solved by integrating the rotation of the end-effector. (b) shows the case, where the prediction was too unprecise and the end-effector moved the cube it wants to pick. The end-effector is then not able to grip the cube or fails to stack the cube. This happens in all models. (c) shows a a failure case for the Elastic-DS model. Through the imprecise motion the end-effector moves the second cube before picking the first one.

episodes. All models achieve good results for the more simple PickCube task. While EM-TPGMM and the Elastic-DS succeed in every episode, has DAMM-TPGMM one error over a hundred episodes.

All models have more difficulties to execute the StackCube task. EM-TPGMM and DAMM-TPGMM show similar results and always fail, as their prediction was too imprecise. Elastic-DS shows worse results than both TPGMM approaches. The Elastic-DS has, in addition to some imprecise predictions, the problem that it sometimes moves the cubes before it reaches the end-point of the trajectory. This is caused by the imprecise movement modeling of the Elastic-DS approach, as already discussed in the previous Section. Examples of the failure cases are shown in Figure 26.

Additionally, we simulate one-hundred episodes of EM-TPGMM and DAMM-TPGMM that include the gripper state to demonstrate their ability to model the gripper state and end-effector pose. The models achieve the same result for the StackCube task as without including the gripper state (table 6). Even when additionally predicting

TaskSuccess		
Model	PickCube	StackCube
EM-TPGMM	100%	94%
DAMM-TPGMM	99%	95%
Elastic-DS	100%	87%

Table 5: Task success of every model without modeling the gripper state over one-hundred episodes.

TaskSuccess		
Model	PickCube	StackCube
EM-TPGMM	97.5%	94%
DAMM-TPGMM	95%	95%

Table 6: Task success of every EM-TPGMM and DAMM-TPGMM including the gripper dimension over one-hundred episodes.

the gripper state, the TPGMM models achieve a higher task success rate compared to the Elastic-DS approach that does not consider the gripper dimension. However, when including the gripper dimension, the task success of the PickCube task is for both models lower than without the gripper dimension, see table 6. Since we predict the poses the same as without the gripper dimension, as described in Section 5.2.2, the additional failure cases are caused by miss-predicting the gripper state. In the failure cases, the end-effector gripper closes too early and the gripper stick to the cube.

6.3 Discussion

The results show a superiority of the TPGMM over the Elastic-DS approach. In the offline experiments, both TPGMM models showed far better results for the MAE and STDV, and similar results for the GMAE and GSTDV metrics. Throughout the offline and online experiments, the Elastic-DS shows difficulties modeling the learned end-effector motion, especially for tasks with a higher variance like the StackCube task. Furthermore, the EM-TPGMM and DAMM-TPGMM have higher task-success

for the StackCube task, even when additionally modeling the gripper state.

Another advantage of the TPGMM approach, is that it can easily be adapted to higher dimensions, as shown in Section 5.2.2, by adding the gripper and time dimensions. Elastic-DS is, for now, only capable of modeling two or three dimensional Cartesian coordinates [6]. To produce motion outside the Cartesian space, the model would have to be adapted [6].

The DAMM fitting method showed similar result as the EM algorithm for TPGMM. DAMM needs more time to compute the Gaussians than the EM algorithm [8], but does not need a model selection due to the split/merge procedure. However, since the split/merge procedure of DAMM is a non-deterministic algorithm due to the random split/merge proposals, we do not get the same result every execution, even for the same parameters. Because of this, DAMM must be executed multiple times before generating a good fit, while EM generates the same result in every iteration. Overall, fitting the TPGMM using DAMM is slightly more time consuming.
Both DAMM and EM can be easily adapted to higher dimensions, as they show similar task success when modeling the gripper state and when not (table 6).

7 Conclusion

In this work, we compared a time-based Task-Paramterized Gaussian Mixture Model with the Elastic Dynamical System approach. We evaluated their performance to reconstruct end-effector poses on the basis of multiple metrics in an offline manner. To put the models into practice and demonstrate their ability to adapt to changing task set-ups, we simulated the PickCube and StackCube task in the Maniskill2 environment and evaluated the task success of our models. We found out that the Elastic Dynamical System struggles to create suitable end-effector motion in novel environments, which in some cases leads to task failure.

By segmenting the demonstrations and temporally aligning the grasp points (Von Hartz et al. [24]) and fitting a TPGMM, with a fixed first and last component for every segment, we proposed a method to lower the variance of our model close to our task-parameters. We demonstrate the ability of the Directionality-aware Mixture Model to be utilized for TPGMMs.

Subsequently, we demonstrate how to adapt time-based Gaussian Mixture Models to model the gripper state of our end-effector and how the Expectation Maximization and Directionality-aware Mixture Model can be adapted to fit additional dimensions. We observed that the Directionality-aware Mixture Model struggles to fit demonstrations in three-dimensional Cartesian space when the demonstrations involve large distances. At the end of the thesis, we discussed the advantages and limitations of our Models and different fitting methods.

7.1 Future work

In Section 6.1, we hypothesised, that the struggle of the Elastic-DS approach to fit the motion between the task-parameters could lead to a task failure in a real environment. Even if the lower task-success in the online simulation of the StackCube task supports this hypothesis, this could be evaluated with more different tasks to support our hypothesis.

T. Li and N. Figueroa [6] discussed how the Elastic Dynamical System could be improved to model the gripper state. With this extension, we could evaluate the online policies of the proposed models better.

Since this work only considers the end-effector pose, the models could be enhanced by modeling the rotation of the end-effector to achieve higher task-success in real-time simulations.

8 Acknowledgments

First and foremost, I would like to thank Jan Ole von Hartz and Dr. Tim Welschehold for the opportunity to write my bachelor thesis at the robot learning lab. They where always available for questions and their guidance has been instrumental in my progress.

Furthermore, I would like to thank all my friends that supported me during the time of my bachelor studies. Special thanks to Niklas Obländer and Malek Elwarshfani for proof reading and being great study partners throughout my whole bachelor journey.

Last but not least I want to thank my family, who have been there for me at every step of my journey. Without them I could never have made it where I am today.

Bibliography

[1] C. Kreutzer, "Südkorea: Verpackungsroboter ergreift Arbeiter und tötet ihn." SWR3. 9. November 2023, 12:39 Uhr.

[2] B. D. Argall, S. Chernova, M. Veloso, and B. Browning, "A survey of robot learning from demonstration," *Robotics and Autonomous Systems*, vol. 57, no. 5, pp. 469–483, 2009.

[3] J. Lee, "A survey of robot learning from demonstrations for human-robot collaboration," *CoRR*, vol. abs/1710.08789, 2017.

[4] S. Calinon, "A tutorial on task-parameterized movement learning and retrieval," *Intell. Serv. Robot.*, vol. 9, p. 1–29, jan 2016.

[5] A. Billard, S. Mirrazavi, and N. Figueroa, *Learning for Adaptive and Reactive Robot Control: A Dynamical Systems Approach*. MIT Press, 2022.

[6] T. Li and N. Figueroa, "Task generalization with stability guarantees via elastic dynamical system motion policies," 2023.

[7] A. P. Dempster, N. M. Laird, and D. B. Rubin, "Maximum likelihood from incomplete data via the EM algorithm," *Journal of the Royal Statistical Society: Series B*, vol. 39, pp. 1–38, 1977.

[8] S. Sun, H. Gao, T. Li, and N. Figueroa, "Damm: Directionality-aware mixture model parallel sampling for efficient dynamical system learning," 2023.

[9] N. Jespersen, "An introduction to markov chain monte carlo," *SSRN Electronic Journal*, 03 2010.

[10] S. Jain and R. M. Neal, "A split-merge markov chain monte carlo procedure for the dirichlet process mixture model," *Journal of Computational and Graphical Statistics*, vol. 13, no. 1, pp. 158–182, 2004.

[11] N. Figueroa and A. Billard, "A physically-consistent bayesian non-parametric mixture model for dynamical system learning," in *Proceedings of The 2nd Conference on Robot Learning* (A. Billard, A. Dragan, J. Peters, and J. Morimoto, eds.), vol. 87 of *Proceedings of Machine Learning Research*, pp. 927–946, PMLR, 29–31 Oct 2018.

[12] Z. Ghahramani and M. I. Jordan, "Supervised learning from incomplete data via an em approach," in *Advances in Neural Information Processing Systems (NIPS)* (J. D. Cowan, G. Tesauro, and J. Alspector, eds.), vol. 6, (San Francisco, CA, USA), pp. 120–127, Morgan Kaufmann Publishers, Inc., 1994.

[13] F. Stulp and O. Sigaud, "Many regression algorithms, one unified model - a review," *Neural Networks*, vol. 69, pp. 60–79, September 2015.

[14] S. Calinon, F. D'halluin, E. L. Sauser, D. G. Caldwell, and A. G. Billard, "Learning and reproduction of gestures by imitation," *IEEE Robotics & Automation Magazine*, vol. 17, no. 2, pp. 44–54, 2010.

[15] H. Hamdan, "Mixture model clustering of binned uncertain data: the classification approach," in *2006 2nd International Conference on Information Communication Technologies*, vol. 1, pp. 1645–1650, 2006.

[16] R. Sampaio, J. Dias Garcia, M. Poggi, and T. Vidal, "Regularization and global optimization in model-based clustering," 02 2023.

[17] H. Akaike, *Information Theory and an Extension of the Maximum Likelihood Principle*, pp. 199–213. New York, NY: Springer New York, 1998.

[18] M. do Carmo, *Riemannian Geometry*. Mathematics (Boston, Mass.), Birkhäuser, 1992.

[19] J. Sethuraman, "A constructive definition of dirichlet priors," *Statistica Sinica*, vol. 4, no. 2, pp. 639–650, 1994.

[20] K. Murphy, "Conjugate bayesian analysis of the gaussian distribution," 11 2007.

[21] N. Metropolis, A. W. Rosenbluth, N. Rosenbluth, A. H. Teller, and E. Teller, "Equation of state calculations by fast computing machines," *Resonance*, vol. 3, 1953.

[22] T. Nierhoff, S. Hirche, and Y. Nakamura, "Spatial adaption of robot trajectories based on laplacian trajectory editing," *Autonomous Robots*, vol. 40, no. 1, pp. 159–173, 2016.

[23] J. Gu, F. Xiang, X. Li, Z. Ling, X. Liu, T. Mu, Y. Tang, S. Tao, X. Wei, Y. Yao, X. Yuan, P. Xie, Z. Huang, R. Chen, and H. Su, "Maniskill2: A unified benchmark for generalizable manipulation skills," in *International Conference on Learning Representations*, 2023.

[24] A. V. Jan Ole von Hartz, Tim Welschehold and J. Boedecker, "Tapas-gmm: Learning generalizable und reusable manipulation policies from few complex task demonstrations." University of Freiburg, Robot Learning Lab. Currently being prepared for submision. Author contact at rl.uni-freiburg.de/people/hartz, 2024.