Hierarchical Contextualization of Movement Primitives

Hierarchische Kontextualisierung von Bewegung Primitive

Master thesis by Zhiyuan Gao

Date of submission: May 23, 2023

1. Review: Prof. Dr. Christoph Hoog Antink
2. Review: Prof. Dr. Jan Peters
3. Review: Prof. Dr. Georgia Chalvatzaki
4. Review: M.Sc. Kay Hansel

Darmstadt

TECHNISCHE UNIVERSITÄT DARMSTADT

Erklärung zur Abschlussarbeit gemäß § 22 Abs. 7 und § 23 Abs. 7 APB der TU Darmstadt

Hiermit versichere ich, Zhiyuan Gao, die vorliegende Masterarbeit 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.

Mir ist bekannt, dass im Fall eines Plagiats (§ 38 Abs. 2 APB) ein Täuschungsversuch vorliegt, der dazu führt, dass die Arbeit mit 5,0 bewertet und damit ein Prüfungsversuch verbraucht wird. Abschlussarbeiten dürfen nur einmal wiederholt werden.

Bei der abgegebenen Thesis stimmen die schriftliche und die zur Archivierung eingereichte elektronische Fassung gemäß § 23 Abs. 7 APB überein.

Bei einer Thesis des Fachbereichs Architektur entspricht die eingereichte elektronische Fassung dem vorgestellten Modell und den vorgelegten Plänen.

Darmstadt, 23. Mai 2023

Z. Gao

Abstract

Learning from Demonstration (LfD) is a widely used approach in robot learning. However, applying the learned motions from expert data to different environments still poses challenges. This is because the robot's working space is often more complex than the demonstrated environment, especially in cases where there is interaction between the robot and humans. Therefore, it is crucial to adjust the learned movement based on the environment. This thesis proposes a hierarchical approach that combines reinforcement learning with the existing framework of Probabilistic Movement Primitives (ProMP), enabling that the robot maintains a similar movement pattern to the demonstration while avoiding obstacles. Specifically, this method regards the position and time of predicted via points as the policy representation of the upper layer, and generates a trajectory similar to the demonstrations in the lower layer. Different from previous work, this paper proposes a more flexible time modulation method. The method is evaluated on the LASA handwriting dataset. The results show that the algorithm is able to avoid obstacles that do not exist in the demonstration and maintain a high similarity.

Zusammenfassung

Learning from Demonstration ist ein weit verbreiteter Ansatz im Bereich des Lernens von Robot. Die Anwendung der erlernten Bewegungen aus Expertendaten auf unterschiedliche Umgebungen stellt jedoch nach wie vor Herausforderungen dar. Dies liegt daran, dass der Arbeitsraum des Roboters oft komplexer ist als die demonstrierte Umgebung, insbesondere in Fällen, in denen es zu Interaktionen zwischen dem Roboter und Menschen kommt. Daher ist es entscheidend, die erlernte Bewegung basierend auf der Umgebung anzupassen. Diese Arbeit schlägt einen hierarchischen Ansatz vor, der das verstärkte Lernen mit dem bestehenden Framework von ProMP kombiniert und sicherstellt, dass der Roboter ein ähnliches Bewegungsmuster wie die Demonstration beibehält, während er Hindernissen ausweicht. Konkret betrachtet diese Methode die Position und Zeit der vorhergesagten Via-Punkte als die Darstellung der Richtlinie auf der oberen Ebene und erzeugt in der unteren Ebene eine Trajektorie, die den Demonstrationen ähnlich ist. Im Unterschied zu früheren Arbeiten wird in diesem Paper eine flexiblere Zeitmodulationsmethode vorgeschlagen. Die Methode wird anhand des LASA-Handschrift-Datensatzes evaluiert. Die Ergebnisse zeigen, dass der Algorithmus in der Lage ist, Hindernissen auszuweichen, die in der Demonstration nicht vorhanden sind, und eine hohe Ähnlichkeit beizubehalten.

Contents

Figures and Tables

List of Figures

List of Tables

Abbreviations

List of Abbreviations

Notation	Description
DMP	Dynamic Movement Primitives
DTW	Dynamic Time Warping
FID	Fréchet Inception Distance
HVMP	Hierarchical Via-Movement Primitives
MDP	Markov Decision Process
MP	Movement Primitives
ProMP	Probabilistic Movement Primitives
ReLU	Rectified Linear Unit
RL	Reinforcement Learning

SAC	Soft Actor Critic
SMDP	Semi Markov Decision Process
VMP	Via-Point Movement Primitives

1. Introduction

In recent years, imitation learning has been investigated in the field of robotics as an effective and intuitive method to dealing with the complex dynamical systems and task scenarios [36, 17, 37]. The core idea of imitation learning is to enable the robot to learn the desired policy directly from the demonstration without manually engineer it. As a sub-domain of imitation learning, movement primitives aim to extract the underlying patterns from expert demonstrations. These primitives capture the essential characteristics of the expert's behavior and can be used to generalize and reproduce similar actions in new situations. There are two most commonly used frameworks of movement primitives, Dynamic Movement Primitives (DMP) [17] and ProMP [37]. DMP assumes that the trajectory is generated by a second-order linear system with a nonlinear term. This nonlinear term can be learning by using regression models [18, 46] . ProMP is a probabilistic method, which abandons the concept of dynamic system, but directly regards the trajectory itself as a regression model[39, 38, 37]. However, whether using DMP or ProMP , the learned primitives need to be adapted to the context, because the workspace of the robot is often not exactly the same as the environment of the demonstration. For example, the robot will encounter obstacles that do not exist in the demonstration's environment. There have been some studies to deal with this problem by optimizing nominal primitives[9, 21, 27], but most of them cannot be optimized online due to the high computational cost. And they only changed the distribution of trajectories, never adjusted the speed of trajectories execution.

Reinforcement learning [52, 30, 2], on the other hand, allows robots to learn from their own experience and feedback, without relying on human demonstrations. Reinforcement learning has been shown to be effective in a wide range of robotic applications, from manipulation[11][10] and grasping [45] to locomotion [25] and navigation [26]. In these works, the robot can learn a policy which adapts to the environment without the high computational cost of online optimization. But on the one hand, reinforcement learning requires exploration to obtain a large amount of interactive data, resulting in low data utilization efficiency. On the other hand, reinforcement learning is highly dependent on the setting of the reward function, and the design of the reward function is often difficult. The expert demonstration can provide a prior for the agent, so it is intuitive and natural to combine reinforcement learning with imitation learning.

In this thesis, we propose a hierarchical approach that combines the advantages of combined imitation learning and reinforcement learning with hierarchical decision-making and probabilistic inference. Specifically, on the higher level we describe the problem as a Markov Decision Process (MDP), and we use the Soft Actor Critic (SAC)[12] algorithm to generate via point points according to changes in the context to ensure safety requirements, such as obstacle avoidance. On the lower level, we use movement primitives to generate the trajectory similar to the pattern of the demonstration from the via point given by the higher layer. To evaluate the performance of our method, we implemented a navigation task of 2d particles using the LASA data set. The results show that our method can avoid obstacles while maintaining the trajectory pattern.

Contribution Our contributions are: (i) Developing a hierarchical robot motion generation framework that adapt movement primitives using via points; (ii) Our method allows the robot to change not only the path but also the speed to avoid obstacles.

1.1. Related Work

Adapting Movement Primitives (MP) to new environments and task constraints is a challenging and some previous research has explored the topic of MP-based trajectory adaptation. In this section, we will introduce these related works.

Some solutions to specific motion adaptation problems are provided in the original ProMP paper [39], but these methods require a lot of manual tuning and lack flexibility. Koert et al. [21] used the kl divergence as the constraint to optimize the demonstrations trajectories offline. The optimized trajectories are able to adapt to the context and to meet the similarity requirements. Then they obtained the new ProMP from the optimized trajectory. Colome et al. [7] proposed Demonstration-Free Contextual ProMP, which can get the nominal ProMP only based on a set of via points given by users without the need for a complete demonstrations trajectory. The new mean of ProMP , which adapted to context were also obtained by the offline optimization. Different from [21], [7] used the Mahalanobis distance as the similarity metric and the variance of new ProMP were not considered. Frank et al. [9] made a major extension to the primitive adaptation framework. The algorithm can optimize both the mean and covariance matrix of ProMP . In addition to avoiding obstacles, they also set new adaptation techniques, such as constraints on temporarily unbound waypoints and dual-arm avoidance. There are also some related studies on the temporal modulation of primitives. For example, Koutras et al. [22] proposed a method to adapt to the new goal position and velocity by scaling the time of DMP and proved the stability of their method.

1.2. Outline

This thesis is organized as follows. Chapter 2 presents the basic concepts. It explains different variants of movement primitives and introduces the basic concepts of reinforcement learning such as markov decision process and the state of the art algorithm like Soft Actor-Critic. Chapter 3 formulates the problems to solve and proposes our hierarchical approach. Chapter 4 shows the results of the experiments on the LASA-Datasets and evaluates different versions of our methods. Chapter 5 summarizes the the proposed framework and gives an outlook for future research.

2. Preliminaries and Related Works

On the lower level of our hierarchical approach, the trajectories are generated by movement primitives using via point modulation. Specifically, we tried the ProMP and VMP in this thesis. The difference between them lies in the dealing with of the via point which is far away from the demonstration. We will introduce them in detail in 2.1. On the higher level of our method, we use reinforcement learning to get the via point according to the environment. In 2.3.1, we will start from the basic concepts of Reinforcement Learning (RL), such as MDP and Semi Markov Decision Process (SMDP), then introduce some state of the art reinforcement learning algorithms and compare their advantages and disadvantages, and why we choose SAC. Since our purpose is to generate a trajectory as similar as possible to the demonstration, we will introduce common similarity metrics in 2.2 including Dynamic Time Warping (DTW), KL divergence, Mahanolobis distance and Fréchet Inception Distance (FID) score, and explain our choices.

2.1. Movement Primitives

Movement Primitives refers to a concept used in the field of robotics. It refers to basic, repeatable units of movement that can be combined and adjusted to perform more complex movements. Movement primitives can be seen as a modular motion representation capable of decomposing complex actions into simpler components. In this section, we will introduce three MP frameworks, DMP , ProMP and VMP.

2.1.1. Dynamic Movement Primitives

DMP [46, 47] is a motion generation framework based on nonlinear system theory. They decompose complex movements into a series of basic, repeatable motion primitives, allowing robots to learn and execute tasks more efficiently. The core of DMP is a linear spring damper system and a non-linear forcing term f

$$\ddot{y} = \tau^2 \alpha_y (\beta_y (g - y) \frac{\dot{y}}{\tau} + \tau^2 f(z), \tag{2.1}$$

$$\dot{z} = -\tau \alpha_z z, \tag{2.2}$$

where $\ddot{y}$, $\dot{y}$ and y are denoted as the desired acceleration, velocity and position respectively. The parameters α_y and β_y can be regarded as the parameters of a PD-controller, which ensure the stability of the system when they are correctly selected. g is the goal of the final position of the trajectory. By changing g, a single DMP can generate motions which adapt to the new goal. τ is a time-constant defines the execution velocity of the trajectory in combination with the phase variable z. The forcing function f

$$f(z) = \frac{\sum_{i=1}^{K} \psi_i(z) w_i}{\sum_{i=1}^{K} \psi_i(z)}, \tag{2.3}$$

determines the shape of the trajectory. It contains a set of basis functions

$$\psi_i(z) = \exp\left(-\frac{1}{2\sigma_i^2}(z - c_i)^2\right), \tag{2.4}$$

where $i = 1 \ldots K$, w_i is the shape-parameter and z is the phase-variable. With the right choice of α and β, the convergence of the underlying dynamic system to a unique attractor point at $y = g$, $z = 0$ is ensured, thus ensuring the stability of the system[46, 18]. When given a demonstration trajectory, forcing term can be easily obtained by linear regression.

2.1.2. Probabilistic Movement Primitives

Although DMP has many benefits, such as stability, and the ability to represent stroke- and rhythm-based motion, they also have some limitations, such as they cannot represent the behavior of stochastic systems, and DMP cannot directly adapt to intermediate via points without extra methods. A probabilistic extension of the original DMP was proposed in [1] that could include via-points, but it requires specifying also the velocity and acceleration for a specific via-point that may be considered a disadvantage. These issues have been fixed in a probabilistic framework called ProMP [37].

The core idea of ProMP is to represent the trajectories $\boldsymbol{\tau} = \boldsymbol{y}_{1:T}$ with a probability distribution $p(\boldsymbol{\tau})$. In order to achieve this, trajectories are approximated by a combination of basis-functions $\boldsymbol{\Phi}_t$ and weight-vectors $\boldsymbol{w}$

$$\boldsymbol{y}_t = \begin{bmatrix} q_t \\ \dot{q}_t \end{bmatrix} = \boldsymbol{\Phi}_t \boldsymbol{w} + \boldsymbol{\epsilon}_y, \tag{2.5}$$

where $\boldsymbol{\epsilon}_y \sim \mathcal{N}(\boldsymbol{0}, \boldsymbol{\Sigma}_y)$ is the Gaussian noise, q_t and $\dot{q}_t$ denote the position and velocity in joint-space, or task-space for a certain time t, and $\boldsymbol{\Phi}_t$ is the basis-matrix contains the basis-functions for q_t and $\dot{q}_t$. $\boldsymbol{\epsilon}_y$ expresses the error of the trajectory representation, which conforms to a standard normal distribution. The $\boldsymbol{w}$ vector represents the shape of the trajectory, and it can be computed by linear ridge regression

$$\boldsymbol{w} = (\boldsymbol{\Phi}_t^{\boldsymbol{T}} \boldsymbol{\Phi}_t + \lambda \boldsymbol{I})^{-1} \boldsymbol{\Phi}_t^{\boldsymbol{T}} \boldsymbol{y}, \tag{2.6}$$

where λ is a regularization parameter of linear ridge regression, used to prevent overfitting, and $\boldsymbol{I}$ represents the identity matrix.

The marginal distribution of the joint position and velocity $\boldsymbol{y}_t$ at time point t is a Gaussian distribution

$$p(\boldsymbol{y}_t|\boldsymbol{w}) = \mathcal{N}(\boldsymbol{y}_t|\boldsymbol{\Phi}_t \boldsymbol{w}, \boldsymbol{\Sigma}_y) \tag{2.7}$$

In order to capture the variance of the trajectories, $\boldsymbol{w}$ is assumed to be a probability distribution $p(\boldsymbol{w}; \boldsymbol{\theta})$. In most cases, we consider it also as a Gaussian distribution

$$p(\boldsymbol{w}; \boldsymbol{\theta}) = \mathcal{N}(\boldsymbol{w}|\boldsymbol{\mu}_{\boldsymbol{w}}, \boldsymbol{\Sigma}_{\boldsymbol{w}}) \tag{2.8}$$

For multi-dimensional systems, the basis function matrix is represented by a block-diagonal matrix

$$\boldsymbol{\Psi}_t = \begin{bmatrix} \boldsymbol{\Phi}_t & \cdots & \mathbf{0} \\ \vdots & \ddots & \vdots \\ \mathbf{0} & \cdots & \boldsymbol{\Phi}_t \end{bmatrix}. \tag{2.9}$$

By marginalizing out the weight vector $\boldsymbol{w}$, the marginal distribution $p(\boldsymbol{y};\boldsymbol{\theta})$ for time t can now be computed in a closed form

$$\begin{aligned} p(\boldsymbol{y}_t;\boldsymbol{\theta}) &= \int \mathcal{N}(\boldsymbol{y}_t|\boldsymbol{\Psi}_t^{\boldsymbol{T}}\boldsymbol{w}, \boldsymbol{\Sigma}_y)\mathcal{N}(\boldsymbol{w}|\boldsymbol{\mu}_{\boldsymbol{w}}, \boldsymbol{\Sigma}_{\boldsymbol{w}})d\boldsymbol{w} \\ &= \mathcal{N}(\boldsymbol{y}_t|\boldsymbol{\Psi}_t^{\boldsymbol{T}}\boldsymbol{\mu}_{\boldsymbol{w}}, \boldsymbol{\Psi}_t^{\boldsymbol{T}}\boldsymbol{\Sigma}_{\boldsymbol{w}}\boldsymbol{\Psi}_t + \boldsymbol{\Sigma}_y). \end{aligned} \tag{2.10}$$

To adjust the speed of the movement, a phase variable z is introduced. z can be any monotonically increasing function of time t, usually $z_0 = 0, z_T = 1$, where T is the time of end. By changing the phase function, we can execute the trajectory at any speed we want. Correspondingly, the basis functions will also become functions of z instead of t:

$$\psi_t = \psi(z_t), \tag{2.11}$$

$$\dot{\psi}_t = \dot{\psi}(z_t)\dot{z}_t, \tag{2.12}$$

$$\psi(z_t) = \frac{b_i(z)}{\sum_{j=1}^{n} b_j(z)}, \tag{2.13}$$

where $b_i(z)$ is Gaussian basis functions. For stroke-based movements, we use

$$b_i(z) = \exp\left(-\frac{(z_t - c_i)^2}{2h}\right) \tag{2.14}$$

as basis function, where $\boldsymbol{h}$ defines the width of the basis and c_i the center for the i-th basis function.

In addition, via-points modulation is a very important property of ProMP. When given an observation $\boldsymbol{x}_t^* = \{\boldsymbol{y}_t^*, \boldsymbol{\Sigma}_y^*\}$, we can calculate the conditional weights using Bayes theorem

$$p(\boldsymbol{w}|\boldsymbol{x}_t^*) \propto \mathcal{N}(\boldsymbol{y}_t^*|\boldsymbol{\Psi}_t\boldsymbol{w}, \boldsymbol{\Sigma}_y^*)p(\boldsymbol{w}). \tag{2.15}$$

This observation can be either a point or part of a trajectory. $\boldsymbol{\Sigma}_y^*$ describes the accuracy of the desired observation. To ensure the trajectory to pass through a certain point, the diagonal elements of $\boldsymbol{\Sigma}_y^*$ should be set to a very small value, close to 0. Since we assume that the estimates are Gaussian distributed, $p(\boldsymbol{w}|\boldsymbol{x}_t^*)$ is also Gaussian, and the mean as well as variance can be obtained from

$$\boldsymbol{\mu}_{\boldsymbol{w}}^{[\text{new}]} = \boldsymbol{\mu}_{\boldsymbol{w}} + \boldsymbol{L}(\boldsymbol{y}_t^* - \boldsymbol{\Psi}_t^{\boldsymbol{T}}\boldsymbol{\mu}_{\boldsymbol{w}}), \tag{2.16}$$

$$\boldsymbol{\Sigma}_{\boldsymbol{w}}^{[\text{new}]} = \boldsymbol{\Sigma}_{\boldsymbol{w}} - \boldsymbol{L}\boldsymbol{\Psi}_t^{\boldsymbol{T}}\boldsymbol{\Sigma}_{\boldsymbol{w}}, \tag{2.17}$$

with

$$\boldsymbol{L} = \boldsymbol{\Sigma}_{\boldsymbol{w}}\boldsymbol{\Psi}_t(\boldsymbol{\Sigma}_y^* + \boldsymbol{\Psi}_t^{\boldsymbol{T}}\boldsymbol{\Sigma}_{\boldsymbol{w}}\boldsymbol{\Psi}_t)^{-1}. \tag{2.18}$$

Figure 2.1 and Figure 2.2 show an example of via point modulation of ProMP .

Theoretically we can use this property to obtain the conditioned primitives under any observations, but if the via-points are too far away from the demonstrations, the confidence of the result could be very low, and the generated trajectory is likely to collapse. For this situation, Via-points Movement Primitives provides a better solution.

2.1.3. Via-points Movement Primitives

Via-points Movement Primitives (VMPs) [53] devides a trajectory into two parts, an elementary trajectory $\boldsymbol{h}$ and a shape modulation $\boldsymbol{f}$.

$$\boldsymbol{y}_t = \boldsymbol{h_t} + \boldsymbol{f}_t$$

The elementary trajectory $\boldsymbol{h}$ connects the start and the goal of the motion and forms the basic structure for VMP , similar to the goal-directed damped spring system of DMP.

The shape modulation $\boldsymbol{f}$ is exactly the same as ProMP , which is assumed to follow a Gaussian distribution

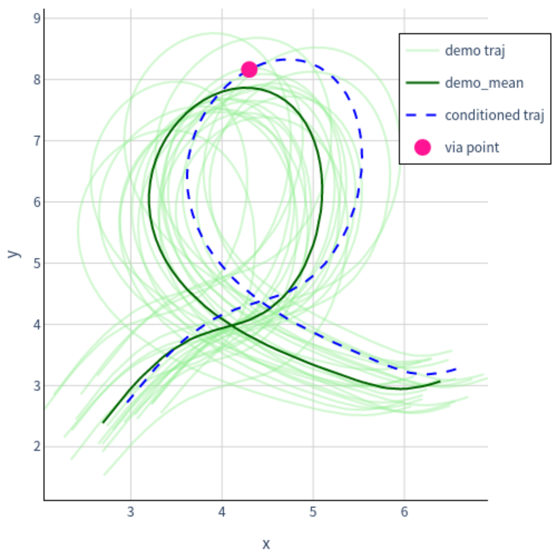


Figure 2.1.: An example of via point modulation of ProMP . The light green trajectories are demonstrations, and the dark green trajectory is the mean of the demonstrations. The blue dashed line is the conditioned trajectory passing through the given red via point.

$$\boldsymbol{f}_t = \boldsymbol{\Phi}_t \boldsymbol{w} + \boldsymbol{\epsilon}_f, \tag{2.19}$$

with $\boldsymbol{\epsilon}_f \sim \mathcal{N}(\mathbf{0}, \boldsymbol{\Sigma}_f)$.

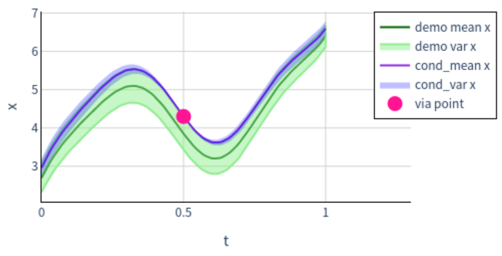
Similar as ProMP , the marginal distribution of the trajectory in VMP is also Gaussian

$$\boldsymbol{y}(\boldsymbol{z}) \sim \mathcal{N}(h(z) + \boldsymbol{\Psi}(z)^{\boldsymbol{T}} \boldsymbol{\mu}_{\boldsymbol{w}}, \boldsymbol{\Psi}(z)^{\boldsymbol{T}} \boldsymbol{\Sigma}_{\boldsymbol{w}} \boldsymbol{\Psi}(z) + \boldsymbol{\Sigma}_f). \tag{2.20}$$

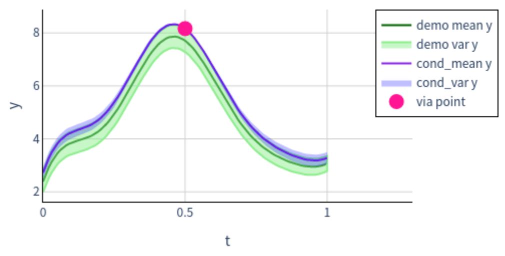
The elementary part $\boldsymbol{h}$ can be obtained through the starting point and end point of the mean of the demonstration, which is a deterministic trajectory. Then we subtract $\boldsymbol{h}$ from the demonstration $\boldsymbol{y}$ to get $\boldsymbol{f}$. $\boldsymbol{f}$, like demonstration, is also a distribution of trajectories. Learning VMP is to learn the prior probability distribution of the parameter vector $\boldsymbol{w}$.

For a given via point, we can also divide it into two parts

$$\boldsymbol{y_{via}} = \boldsymbol{h_{via}} + \boldsymbol{f_{via}} \tag{2.21}$$



(a) Distribution of dimension x



(b) Distribution of dimension x

Figure 2.2.: (a) and (b) show the change in trajectory distribution after via point modulation of ProMP on the two degrees of freedom x and y respectively. The light green area is the distribution of the demo, and the blue area is the conditioned distribution.

Since $\boldsymbol{f}$ has all the properties of ProMP , it is possible to use via point modulation on $\boldsymbol{f}$. The mean as well as the variance of conditional weights can be obtained from

$$\boldsymbol{\mu}_{\boldsymbol{w}}^{[\text{new}]} = \boldsymbol{\mu}_{\boldsymbol{w}} + \boldsymbol{L}(\boldsymbol{y}_{via} - \boldsymbol{h}(z_{via}) - \boldsymbol{\Psi}(z_{via})^{\boldsymbol{T}}\boldsymbol{\mu}_{\boldsymbol{w}}) \tag{2.22}$$

$$\boldsymbol{\Sigma}_{\boldsymbol{w}}^{[\text{new}]} = \boldsymbol{\Sigma}_{\boldsymbol{w}} - \boldsymbol{L}\boldsymbol{\Psi}(z_{via})^{\boldsymbol{T}}\boldsymbol{\Sigma}_{\boldsymbol{w}}, \tag{2.23}$$

with

$$\boldsymbol{L} = \boldsymbol{\Sigma}_{\boldsymbol{w}}\boldsymbol{\Psi}(z_{via})(\boldsymbol{\Sigma}_y^* + \boldsymbol{\Psi}(z_{via})^{\boldsymbol{T}}\boldsymbol{\Sigma}_{\boldsymbol{w}}\boldsymbol{\Psi}(z_{via}))^{-1}. \tag{2.24}$$

In addition to this, VMP also offers another possibility perform via point modulation, that is, to modulate $\boldsymbol{h}$. Regarding how to modulate the $\boldsymbol{h}$, several different methods are provided in [53]. In this thesis, we chose one of the methods, which considers $\boldsymbol{h}$ as a linear trajectory with a constant velocity, because this method has a close relationship with DMP . Since a linear trajectory with a constant speed that goes through start $(1, y_0)$ and goal $(0, g)$ has the formula $h(z) = (y_0 - g)z + g$. Adding $\boldsymbol{h}$ and $\boldsymbol{f}$ gives us the expression of resulted trajectory y for the phase z

$$\boldsymbol{y}(z) = (\boldsymbol{y}_0 - \boldsymbol{g})z + \boldsymbol{g} + \boldsymbol{f}(z). \tag{2.25}$$

By modulating the basic structure $\boldsymbol{h}$, via-points modulation can also be achieved. We can get the new $\boldsymbol{h}$ adapted to the via point by interpolation. As an example, when there is only one intermediate via point, $\boldsymbol{h}$ becomes a piecewise function. From three constraints

$$\boldsymbol{h}(0) = \boldsymbol{g}, \tag{2.26}$$

$$\boldsymbol{h}(1) = \boldsymbol{y}_0, \tag{2.27}$$

$$\boldsymbol{h}(z_{via}) = \boldsymbol{y}_{via} - \boldsymbol{f}(z_{via}), \tag{2.28}$$

we can derive the new $\boldsymbol{h}$

$$\boldsymbol{h}(z) = \begin{cases} -\dfrac{(\boldsymbol{h}_{via} - \boldsymbol{y}_0)z}{1 - z_{via}} + \boldsymbol{y}_0 + \dfrac{\boldsymbol{h}_{via} - \boldsymbol{y}_0}{1 - z_{via}} \\ -\dfrac{(\boldsymbol{g} - \boldsymbol{h}_{via})z}{z_{via}} + \boldsymbol{g}, \end{cases} \tag{2.29}$$

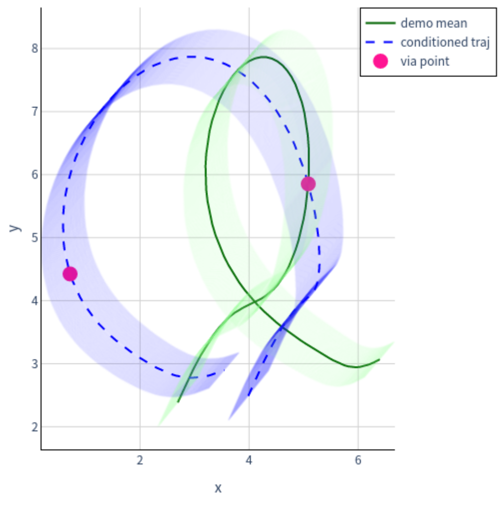
where $\boldsymbol{h}_{via} = \boldsymbol{y}_{via} - \boldsymbol{f}_{via}$. In the same way, $\boldsymbol{h}$ with multiple intermediate via points can also be obtained.

VMP supposes that the task-specific trajectories, which are only near the demonstrations, follow a Gaussian distribution. In other words, for a via point, VMP modulate $\boldsymbol{f}$ only when the via point near the demonstrations. When the via point is outside the distribution of demonstrations, VMP will uses $\boldsymbol{h}$ modulation. Therefore, before via point modulation, we have to first check whether the given via point is near the demonstrations.

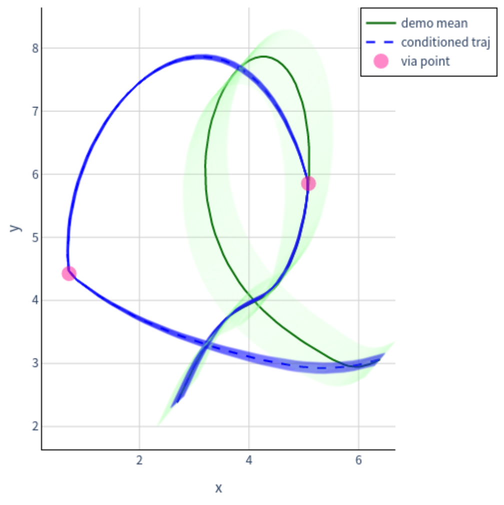
In [53], this is achieved by calculating the probability density of the desired via-point given the current learned prior probability of $\boldsymbol{w}$

$$p(\boldsymbol{y}_{via}|\boldsymbol{\mu}_w, \boldsymbol{\Sigma}_{\boldsymbol{w}}) = \mathcal{N}(\boldsymbol{y}_{via}; \boldsymbol{h}(z_{via}) + \boldsymbol{\Psi}(z_{via})^{\boldsymbol{T}}\boldsymbol{\mu}_{\boldsymbol{w}}, \boldsymbol{\Psi}(z_{via})^{\boldsymbol{T}}\boldsymbol{\Sigma}_{\boldsymbol{w}}\boldsymbol{\Psi}(z_{via}) + \boldsymbol{\Sigma}_{via}), \tag{2.30}$$

where $(z_{via}, \boldsymbol{y}_{via}, \boldsymbol{\Sigma}_{via})$ define the position, phase and variance of the via point respectively. When $p(\boldsymbol{y}_{via}|\boldsymbol{\mu}_w, \boldsymbol{\Sigma}_{\boldsymbol{w}})$ is higher than a given threshold η, it means via point is near the demonstrations. Conversely, it is the opposite. For the case of multiple intermediate via points, we modulate $\boldsymbol{f}$ when one of the via points is outside the demonstration.



(a) Condtiond trajectory of ProMP



(b) Condtiond trajectory of ProMP

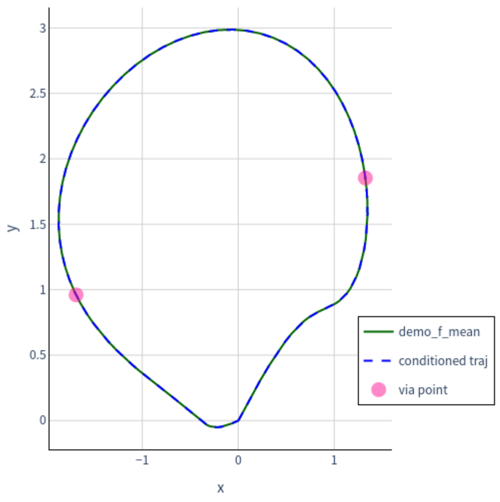
Figure 2.3.: When there is a via point out of statistic, the conditioned trajectory by ProMP is collapsed, while the conditioned trajectory by VMP still keeps the shape of demonstration

2.2. Similarity Metrics

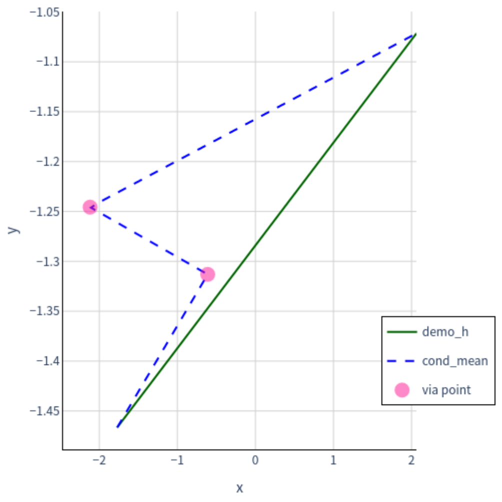
In the lower level of our hierarchical framework, we use MP to generate a trajectory as similar as possible to the demonstration. So it is necessary to measure the similarity between the adapted trajectory and the demonstration. However, similarity is an abstract concept, and its definition is not unique, so it is difficult to quantify it with a single formula. Several similarity metrics such as KL divergence, Mahalanobis distance, DTW, and FID score have been used in previous related researches, and we will introduce and compare these metrics in this section. [21, 7, 1],

KL Divergence The KL divergence [19] is defined based on the Shannon entropy [51] in information theory. Shannon entropy of a variable X is given by

$$H(X) = -\sum_i p(x_i)\log_2(p(x_i)), \tag{2.31}$$



(a) Conditiond trajectory of $\boldsymbol{f}$



(b) Conditiond trajectory of $\boldsymbol{h}$

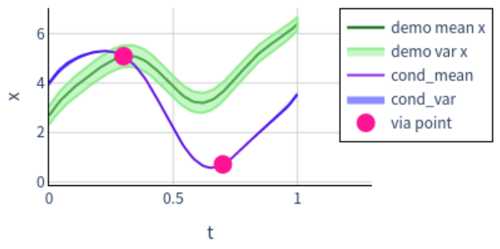
Figure 2.4.: For the far away via point, VMP uses DMP modulation to directly translate the $\boldsymbol{h}$ part, while the $\boldsymbol{f}$ part only reduces the variance

with $p(x_i)$ representing the probability of the variable $X = x_i$. Shannon entropy reflects the relationship between the information size of a piece of information and its uncertainty. For two probability distributions, P and Q, the relative entropy of P with respect to Q is called KL divergence of P from Q. For discrete samples, KL divergence of P from Q is calculated by

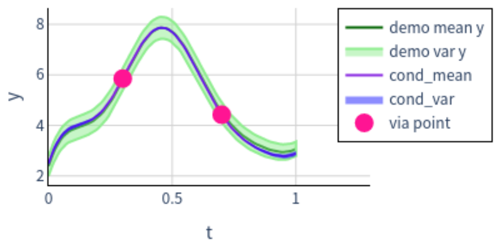
$$D_{KL}(P||Q) = \sum_i P(i) log \frac{P(i)}{Q(i)}. \tag{2.32}$$

KL divergence can be regarded as amount of information lost when P is used to approximate Q. In [21], KL divergence is used as a constraint for trajectory optimization, so that the optimized trajectory is as close as possible to the original distribution of the demonstrations.

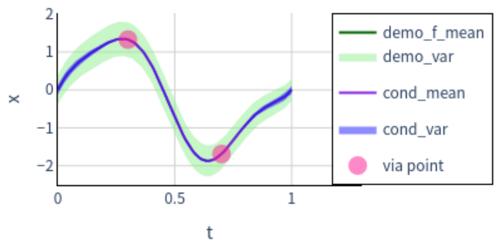
Mahalanobis Distance Mahalanobis distance [31] is a measure of the distance between a point and a distribution. It takes into account the covariance structure of the data, which allows it to handle correlations between variables. Given a probability distribution Q, with



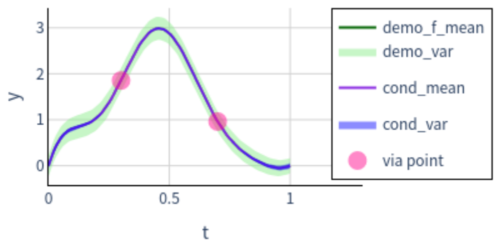
(a) Conditioned distribution of dimension x from ProMP



(b) Conditioned distribution of dimension y from ProMP



(c) Conditioned distribution of dimension x from VMP



(d) Conditioned distribution of dimension y from ProMP

Figure 2.5.: The conditional distribution from VMP is near the mean of the demonstration, while the distribution of x dimension from ProMP is completely different from the demonstration. Only the VMP $\boldsymbol{f}$ part distribution is showed , which is obtained by subtracting the deterministic $\boldsymbol{h}$ part from the demonstration.

mean $\boldsymbol{\mu}$ and positive-definite covariance matrix $\boldsymbol{S}$, the Mahalanobis distance of a point $\boldsymbol{x}$ from Q is:

$$d_M(\boldsymbol{x}, Q) = \sqrt{(\boldsymbol{x}-\boldsymbol{\mu})^T \boldsymbol{S}^{-1}(\boldsymbol{x}-\boldsymbol{\mu})} \tag{2.33}$$

[7] used Mahalanobis Distance as the similarity metric between the adapted trajectory and the nominal primitives.

DTW DTW [34] is a measure of similarity between two sequences that may vary in time or speed. DTW aligns the two sequences by warping one sequence in time to match the other, and then calculates the distance between the aligned sequences. The formula for DTW is:

$$DTW(X,Y) = min_{\gamma \in \Gamma} \sqrt{\sum_{i=1}^{n} \sum_{j=1}^{m} d(x_i, y_j)^2} \tag{2.34}$$

where X and Y are the two sequences being compared, and Γ is the set of all possible alignments between X and Y. In [1], DTW is used to estimate the current phase from the current position.

FID score FID score [14], or Frechet Inception Distance score, is a measure of the similarity between two sets of images. FID score is calculated by computing the Frechet distance between the feature representations of the two sets of images, using a pre-trained Inception network. The formula for FID score is:

$$FID(P,Q) = ||\mu_P - \mu_Q||^2 + Tr(\Sigma_P + \Sigma_Q - 2(\Sigma_P \Sigma_Q)^{1/2}) \tag{2.35}$$

where P and Q are the two sets of images being compared, and μ and Σ are the mean and covariance of the feature representations of each set.

In our work, conditioned trajectories are generated by via point modulation. In order to ensure that the trajectory passes through a given via point, the trajectory generated by via point modulation is usually much smaller than the variance of the demonstrations. So FID and KL divergence are not applicable in our case. DTW ignores the variance of the demonstrations, so it is not the best option either. In this thesis, we will use Mahalanobis distance as the metric of similarity, to measure the distance between the generated deterministic trajectory and the original distribution.

2.3. Reinforcement Learning

On the higher level of our method, we use RL to choose the next via point according to the environment. Reinforcement learning is a machine learning method designed to allow an agent to learn how to make decisions through interaction with the environment to obtain the highest cumulative reward [52]. The basic components of RL include an agent, an environment, and a reward signal. The agent is the subject of learning and decision-making, it takes actions based on the observed state of the environment, and receives feedback from the environment. The environment is the external world in which the agent lives, which responds to the agent's actions and provides feedback based on the agent's behavior and current state. The reward signal is the numerical feedback provided by the environment to the agent, which is used to evaluate the action results of the agent, with the purpose of guiding the agent to learn in the direction of obtaining greater cumulative rewards.

2.3.1. Markov Decision Process

A MDP is a classical mathematical framework used to model decision problems involving randomness[44]. It finds wide applications in fields such as artificial intelligence, operations research, and control theory. MDP provides a mathematical way to describe the relationships among states, actions, and rewards, facilitating the formulation of optimal decision policies. An MDP can be defined by a quintuple (S, A, P, R, γ), where S represents the state space, which is the set of possible states the system can be in. A represents the action space, which is the set of different actions available for the decision-maker. $P(s'|s, a)$ represents the state transition probability function, which defines the probability of transitioning from state s to state s' given action a. This function captures the stochasticity in the environment. $R(s, a, s')$ represents the reward function, which provides the immediate reward obtained when transitioning from state s to state s' after taking action a. Rewards can be positive, negative, or zero and are used to measure the system's objectives. γ is the discount factor, which balances the importance of immediate and future rewards. γ typically takes values in the range [0, 1]. As γ approaches 1, future rewards have greater influence on decision-making; as γ approaches 0, more weight is given to immediate rewards.

The objective in MDP is to find a policy π that, given the current state, selects the optimal action to maximize long-term cumulative rewards [52], which is defined by

$$J^{\pi} = \mathbb{E}\left[\sum_{t=0}^{N} \gamma^t R(s_t, a_t) \mid s_0 \sim \mu_0(s), a_t \sim \pi(a \mid s_t), s_{t+1} \sim P(s \mid s_t, a_t)\right]. \tag{2.36}$$

The optimal policy can be represented through value functions, where $V(s)$ denotes the maximum expected cumulative reward starting from state s, and $Q(s, a)$ represents the maximum expected cumulative reward starting from state s and taking action a.

2.3.2. Semi-Markov Decision Process

SMDP is a natural extension of the traditional MDP that takes into account the variable duration of actions [4]. In an SMDP, the state transitions are not only dependent on the current state and action but also on the time elapsed since the last action. Formally, an SMDP can be defined as a tuple (S, A, P, r, H) where S is the state space, A is the action space, $P : S \times A \times D \times S \to [0, 1]$ is the transition probability function, $r : S \times A \times D \to \mathbb{R}$ is the reward function, and $H : S \to \mathbb{N}$ is the holding time distribution function that determines the duration of each action.

The transition probability function P now takes into account the duration of an action, which is represented by a duration variable $d \in D$. Specifically, $P(s, a, d, s')$ represents the probability of transitioning from state s to state s' when taking action a for duration d. The reward function r is also defined over the state-action-duration triples. Finally, the holding time distribution function H specifies the probability distribution over the possible durations of an action. The goal of an SMDP is to find a policy $\pi : S \to A$ that maximizes the expected discounted reward over an infinite horizon. This can be done using various extensions of the traditional dynamic programming algorithms, such as the Semi-Markov Decision Process Value Iteration (SMDP-VI) algorithm or the Semi-Markov Decision Process Q-learning (SMDP-Q) algorithm.

When compared to standard MDP, SMDP has two main differences: (i) Uncertainty in action duration: In SMDP, each action has an uncertain duration, while in standard MDP, the duration of each action is fixed. This means that the time required to complete an action may vary due to different initial states or other factors in SMDP. (ii) Uncertainty in reward function: In SMDP, the reward associated with each action is usually calculated at the end of the action. This means that the reward function may also be time-dependent, and rewards may differ due to the duration of the action. In standard MDP, the reward for each action is calculated at the beginning of the action.

In summary, compared to standard MDP, SMDP is more suitable for problems that require consideration of action duration and time-dependent rewards.

2.3.3. Policy Gradient

Policy Gradient is a method used to solve RL problems, where an agent learns the optimal policy by interacting with the environment. Unlike value-based methods, policy gradient directly learns the policy function instead of the value function. This allows policy gradient methods to handle complex situations such as continuous action spaces and stochastic policies.

In policy gradient methods, we define a parameterized policy function $\pi(a|s;\theta)$ that gives the probability of selecting action a in state s. Here, θ is the parameter of the policy function. The objective is to find the optimal policy by maximizing the cumulative reward. To achieve this, we use the policy gradient theorem to compute gradients for updating the parameters.

Firstly, we define the performance measure of the policy, typically the expected cumulative reward. Let's assume that the agent's state at time step t is S_t, the action taken is A_t, and the cumulative reward is R_t. Then, the performance measure of the policy can be represented as:

$$J(\theta) = \mathbb{E}_\pi \left[\sum_{t=0}^{\infty} \gamma^t R_t \right] \tag{2.37}$$

Here, γ is the discount factor used to balance the importance of immediate and future rewards.

Our goal is to maximize this performance measure by adjusting the parameter θ to improve the policy function. To compute the policy gradient, we introduce a crucial theorem called the policy gradient theorem. According to this theorem, the policy gradient can be expressed as:

$$\nabla_\theta J(\theta) = \mathbb{E}_\pi \left[\sum_{t=0}^{\infty} \nabla_\theta \log \pi(A_t|S_t;\theta) Q^\pi(S_t, A_t) \right] \tag{2.38}$$

Here, $Q^\pi(S_t, A_t)$ is the value function of the state-action pair (S_t, A_t), representing the expected cumulative reward when taking action A_t in state S_t and following policy π. The policy gradient theorem tells us that the policy gradient is given by the product of the cumulative reward and the sum of the logarithm of action probabilities and value functions.

Based on the policy gradient theorem, we can update the policy parameters θ using gradient ascent. By estimating the expected value of the policy gradient using Monte Carlo methods or value-based methods, we can update the parameters as follows:

$$\theta_{t+1} = \theta_t + \alpha \nabla_\theta J(\theta_t) \tag{2.39}$$

Here, α is the learning rate that controls the step size of parameter updates.

One important advantage of policy gradient methods is their ability to handle continuous action spaces since the policy function can directly parameterize the probability distribution of continuous actions. Additionally, policy gradient methods can learn stochastic policies, which is useful for certain tasks.

In summary, policy gradient is a RL method that maximizes the cumulative reward by directly learning the gradients of a parameterized policy function. By using the policy gradient theorem, we can compute the gradient and update the policy parameters using gradient ascent to improve the agent's policy.

2.3.4. Actor Critic

The Actor-Critic method is a RL approach that combines policy evaluation and policy improvement. It consists of two key components: an Actor and a Critic. The Actor uses a parameterized policy function $\pi(a|s;\theta)$ that represents the probability of selecting action a given state s. The Critic employs a value function $V(s;w)$ to evaluate the value of states. The goal of the Actor-Critic method is to maximize the long-term cumulative reward. The performance measure is defined as the expected cumulative reward:

$$J(\theta) = \mathbb{E}_\pi \left[\sum_{t=0}^{\infty} \gamma^t R_t \right] \tag{2.40}$$

where γ is the discount factor that balances the importance of immediate and future rewards.

The key idea of the Actor-Critic method is to use the estimated value function to compute the policy gradient, enabling simultaneous learning of the policy and the value function. According to the policy gradient theorem, the policy gradient of the Actor can be expressed as:

$$\nabla_\theta J(\theta) = \mathbb{E}_\pi \left[\sum_{t=0}^{\infty} \nabla_\theta \log \pi(A_t|S_t;\theta) Q^\pi(S_t, A_t) \right] \tag{2.41}$$

Here, $Q^{\pi}(S_t, A_t)$ denotes the value function of the state-action pair (S_t, A_t), representing the expected cumulative reward when taking action A_t in state S_t and following the policy π.

By updating the parameters of the Actor (θ) and the Critic (w) using the policy gradient and value function estimation, both the policy and the value function can be optimized. The specific update rules can vary depending on the method and algorithm used.

In summary, the Actor-Critic method learns the optimal policy by combining the policy and the value function. The Actor updates its parameters using the policy gradient, while the Critic evaluates the value of actions and provides feedback. By simultaneously learning the policy and the value function, the Actor-Critic method achieves good performance in RL problems.

Several SOTA algorithms based on the AC framework have achieved remarkable results in various tasks. Asynchronous Advantage Actor-Critic (A3C) is an advanced algorithm that builds upon the Actor-Critic (AC) framework [32]. It introduces asynchronous training, allowing multiple agents to interact concurrently with their own copies of the environment. These agents collect experiences, which are then used to update the shared policy (actor) and value function (critic) networks. A3C achieves high sample efficiency and scalability by leveraging parallelism, enabling faster learning and improved exploration in complex tasks. Trust Region Policy Optimization (TRPO) is another notable AC-based algorithm [49]. It focuses on stable policy updates by constraining the policy changes within a specified region defined by the Kullback-Leibler (KL) divergence. By carefully managing the policy updates, TRPO ensures consistent improvements while maintaining stability. It is known for its robust convergence properties, making it suitable for tasks with high-dimensional continuous action spaces. Proximal Policy Optimization (PPO) is an approximate policy gradient algorithm based on the AC framework[48]. PPO improves stability and sample efficiency by employing a "Proximal Policy Optimization" objective function. This objective function limits the magnitude of policy updates, preventing drastic policy changes. PPO strikes a balance between exploration and exploitation, making it effective in handling high-dimensional continuous action spaces. Soft Actor-Critic (SAC) is an AC algorithm that combines policy gradient updates with the maximum entropy optimization objective[12]. SAC seeks to maximize the expected reward while simultaneously maximizing entropy, promoting exploration. This balance between exploitation and exploration allows SAC to excel in continuous action spaces. It is known for its sample efficiency, stability, and robust performance in a variety of RL tasks.

2.3.5. Soft Actor-Critic

SAC (Soft Actor-Critic) [12] is a RL-based control algorithm that can be used to solve continuous control problems. It is a variant of the actor-critic algorithm that is designed to improve the stability and sample efficiency of learning in continuous action spaces. The key idea behind SAC is to use a soft value function that estimates the expected return of the current policy, instead of a hard value function that estimates the optimal return.

The soft value function is defined as:

$$V^{\pi}(s) = \mathbb{E}_{a \sim \pi(\cdot|s)} \left[Q^{\pi}(s,a) - \alpha \log \pi(a|s) \right] \tag{2.42}$$

where s is the current state, a is the action taken by the policy π, $Q^{\pi}(s,a)$ is the action-value function under policy π, and α is a temperature parameter that controls the trade-off between the policy entropy and the expected return.

The soft Q-function is defined as:

$$Q^{\pi}(s,a) = \mathbb{E}s' \sim p(\cdot|s,a) \left[r(s,a) + \gamma \mathbb{E}a' \sim \pi(\cdot|s') \left[V^{\pi}(s') \right] \right] \tag{2.43}$$

where s' is the next state, a' is the next action, $p(s'|s,a)$ is the transition probability, $r(s,a)$ is the reward function, and γ is the discount factor.

The policy is updated by maximizing the soft Q-function with respect to the policy parameters θ:

$$\nabla_{\theta} J(\theta) = \mathbb{E}s \sim \mathcal{D} \left[\nabla \theta \log \pi_{\theta}(a|s) \left(Q^{\pi}(s,a) - V^{\pi}(s) \right) \right] \tag{2.44}$$

where $\mathcal{D}$ is the replay buffer that stores the transitions experienced by the agent.

An important feature of the SAC algorithm is that it is an off policy algorithm. On-policy and off-policy are two types of RL algorithms based on how they handle the exploration-exploitation trade-off.

On-policy algorithms learn a policy that is directly updated by the agent's experience in the environment. In other words, the agent collects data using its current policy and then updates that policy based on the collected data. On-policy algorithms tend to be more sample-efficient than off-policy algorithms because they use the most recent data, but they can also be less effective at exploring the environment because they are constrained by their current policy.

Off-policy algorithms, on the other hand, learn a policy that is updated based on experience that was collected using a different policy. This allows the agent to explore more widely and potentially learn a better policy. However, off-policy algorithms can be less sample-efficient because they use data that may be outdated or unrepresentative of the current policy.

SAC is an off-policy algorithm because it learns a policy that is updated based on data collected by a different policy, such as a random policy or a previous version of the learned policy. The advantage of using an off-policy algorithm like SAC is that it can learn from a wider range of experiences and potentially converge to a better policy than on-policy algorithms. Moreover, SAC uses a replay buffer to store past experiences, allowing for more efficient use of the collected data.

In addition, SAC employs several techniques to improve the stability and sample efficiency of learning in continuous action spaces, such as using a soft value function and entropy regularization. These techniques help to prevent overfitting and encourage exploration, making SAC a powerful algorithm for solving complex continuous control problems.

Overall, SAC is a powerful algorithm for solving continuous control problems, and it has achieved state-of-the-art performance in a wide range of domains. So SAC is very suitable for our task requirements, we choose to use it as the RL algorithm in our framework.

3. Hierarchical Via-Movement Primitives

Due to the difference between the environment of the demonstration and the actual working environment of the robot, the actions learned directly from the demo cannot be directly used on the robot. We developed a hierarchical approach for adapting primitives to environments. In the higher layer, it can generate via points according to the environment, and use these via points to generate trajectories in the lower layer. In 3.1, the basic structure of framework will be presented, and the details of the reward function used in the higher level will be introduced in 3.2.

3.1. Structure of Framework

As we mentioned in the previous chapter, neither ProMP nor DMP can be used directly for trajectory planning problems. Because the actual environment and the demonstration environment are likely to be different, for example, there may be obstacles in the robot workspace that are not in the demonstration environment. An advantage of MP is that it can generate a trajectory with a similar shape to the demonstration through a given via point. And ProMP allows to modulate the trajectory at any speed. Based on these advantages and disadvantages, we propose a hierarchical approach, which not only allows the robot to avoid obstacles, but also allows the trajectory of the robot to be as close as possible to the demonstration.

The proposed algorithm is divided into two levels as figure 3.1. First let's focus on the high level. At the high level, we describe the problem as a SMDP or MDP. At the beginning of each episode, the obstacle randomly appears at an initial position with a random velocity, and its velocity will remain constant throughout the episode. The robot also has a random initial starting point, and gets an action at each step, which guides it to move to a certain position at a certain time point in the way of demonstration. The episode ends when the time reaches the maximum or the robot collides with the obstacle.

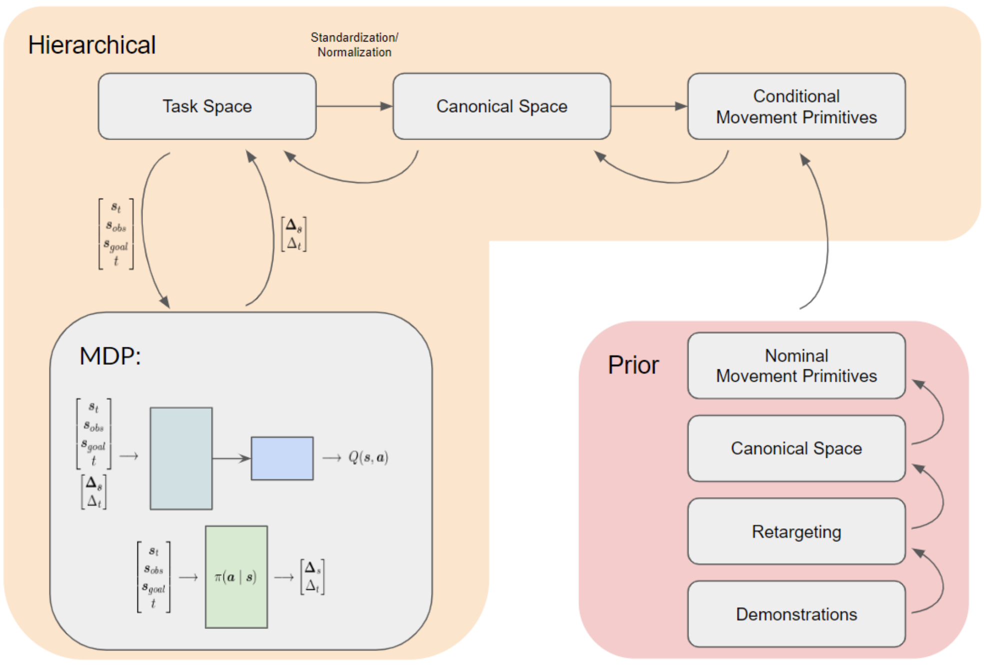


Figure 3.1.: Structure of Hierarchical Via-Movement Primitives.As the upper-level decision-maker, the MDP/ SMDP provides actions based on the current environment. These actions correspond to the position and timing of via points in the task space. The lower-level MP receives the via points from the upper level and generates conditional MP, which represents the distribution of trajectories to be executed by the robot in the task space. The lower-level MP is learned from demonstrations, and if the demonstrations are based on human motion data, they need to be retargeted to match the robot's demonstrations based on the robot's joint limit. From these retargeted demonstrations, we can obtain the nominal MP, which serves as prior knowledge or prior information

The states of the SMDP contain the current position of the robot p_{robot}, the current time t_{curr}, and the information of the environment, such as the position of the obstacle p_{obs} and velocity of the obstacle v_{obs}. The action includes the shift of position and time of next via point, respectively δ_p and δ_t. In other words, the action will not change during this period of time δ_t, which is the difference from a normal MDP. The low level can be regarded as a trajectory generator. At the low level, first we standardize the data set into a distribution with a mean of 0 and a variance of 1, to prevent the influence of different scales of the data set. This standardized data space is called canonical space. Then we compute the MP from the standardized demonstrations. Finally we use the via point given by the high level policy to generate the conditioned trajectory between the current time t_{curr} and next time $t_{curr} + \delta_t$. This trajectory will be passed to the high level to calculate the rewards obtained by in current step.

In addition to the aforementioned basic settings, we also allow modulation of the execution time of trajectories. To avoid ambiguity, in our approach, we refer to the timing of the demonstrations as "phase" and represent it as z. Phase represents a relative time that can be normalized between 0 and 1, regardless of the duration of the collected demonstrations. This normalization is possible because we know the starting and ending time points. The actual execution time of the trajectory is referred to as "time" and denoted as t, starting from 0 and extending to positive infinity. We control the speed of trajectory execution by mapping time to phase using a time-to-phase mapping, which is referred to as a phase function. In our approach, this is an optional setting. If we do not use a phase function, the phase and time will always be equal. For convenience, we refer to the case where the lower-level MP does not use a phase function as "MP with a linear phase." The details regarding the phase function we used will be discussed in the next section.

3.2. Reward Function

In this section we will detail our reward function. The reward function consists of different parts as follows

$$r = k_{sim} \cdot r_{sim} + k_{col} \cdot r_{col} + k_{goal} \cdot r_{goal} + k_{speed} \cdot r_{speed}, \tag{3.1}$$

with r_{sim}, r_{col}, r_{goal},r_{speed} corresponding to collision reward, similarity reward, goal reward and speed reward respectively and k_{sim}, k_{col}, k_{goal},k_{speed} are the coefficients of the corresponding part of reward function. The r_{sim} is a similarity reward as we desire that the behavior of the MP doesn't change. Before calculating it, we have to determine whether the via point is inside the demonstration, resulting in two cases

$$r_{sim} = \begin{cases} r_{sim_f}, & \text{when } d_M(\boldsymbol{y}_{via}, \boldsymbol{y}_{nominal}(t_{via})) \leq T, \\ r_{sim_h}, & \text{when } d_M(\boldsymbol{y}_{via}, \boldsymbol{y}_{nominal}(t_{via})) > T. \end{cases} \tag{3.2}$$

where $d_M(\boldsymbol{y}_{via}, \boldsymbol{y}_{nominal}(t_{via}))$ is the Mahalanobis distance between the via point $\boldsymbol{y}_{via}$ and distribution of the demonstration $\boldsymbol{y}_{nominal}$ at the corresponding time t_{via} and T is a threshold. The formulas of r_{sim_h} and r_{sim_f} are related to the choice of MP. When we use VMP on the lower level, we have

$$r_{sim_f} = (\boldsymbol{f}(t_{via}) - (\boldsymbol{y}_{via} - \boldsymbol{h}_{current}(t_{via})))^T \boldsymbol{S}^{-1} (\boldsymbol{f}(t_{via}) - (\boldsymbol{y}_{via} - \boldsymbol{h}_{current}(t_{via}))), \tag{3.3}$$

$$r_{sim_h} = (\boldsymbol{h}_{nominal}(t_{via}) - (\boldsymbol{y}_{via} - \boldsymbol{f}(t_{via})))^T \boldsymbol{I} (\boldsymbol{h}_{nominal}(t_{via}) - (\boldsymbol{y}_{via} - \boldsymbol{f}(t_{via}))), \tag{3.4}$$

where $\boldsymbol{h}_{nominal}$ is the elementary function of nominal VMP ,$\boldsymbol{h}_{current}$ is interpolated elementary function with the current via points, $\boldsymbol{f}$ is the shape function of learned VMP, $\boldsymbol{S}$ is covariance matrix of $\boldsymbol{f}$, and $\boldsymbol{I}$ is the identity matrix. When the via point is within the statistics, we use Mahalanobis distance as the metric of similarity. When the via point is outside of the statistics, VMP modulates the deterministic elementary function $\boldsymbol{h}$, and variance has no effect on the generated trajectories. So at this case we only calculate the Euclidean distance. When we use ProMP on the lower level, trajectories are no longer divided into $\boldsymbol{f}$ and $\boldsymbol{h}$, so r_{sim_h} and r_{sim_f} are calculated as follows

$$r_{sim_f} = (\boldsymbol{y}_{nominal}(t_{via}) - \boldsymbol{y}_{via})^T \boldsymbol{S}^{-1} (\boldsymbol{y}_{nominal}(t_{via}) - \boldsymbol{y}_{via}), \tag{3.5}$$

$$r_{sim_h} = (\boldsymbol{y}_{nominal}(t_{via}) - \boldsymbol{y}_{via})^T \boldsymbol{I} (\boldsymbol{y}_{nominal}(t_{via}) - \boldsymbol{y}_{via}), \tag{3.6}$$

where $\boldsymbol{y}_{nominal}$ is the trajectory of the nominal ProMP and $\boldsymbol{S}$ is covariance matrix of $\boldsymbol{y}$. Since we hope that the via point is not too far away from the demonstration, when the via point is within the distribution of the demonstration, the agent gets a positive reward, otherwise it gets a negative reward. So whether we use ProMP or VMP, r_{sim_f} is scaled to $(0, 1)$ and r_{sim_h} is scaled to $(-1, 0)$.

The collision penalty r_{col} is simply a constant penalty as follow,

$$r_{col} = \begin{cases} -1, & \text{when collided;} \\ 0, & \text{when not collided.} \end{cases} \tag{3.7}$$

Since collision is the worst case, the corresponding coefficient k_{col} should be set to a larger value.

r_{goal} is the goal reward, which is the same as the formula for r_{sim},

$$r_{goal} = \begin{cases} -r_{sim}, & \text{when} \quad z > 0.99; \\ 0, & \text{else.} \end{cases} \tag{3.8}$$

Although the endpoint is indeed considered as a via point as well, it is found in the experiment that some data sets require additional end point penalties, and only the similarity penalty is not enough. Therefore, we require a goal reward. Unlike the similarity reward, the goal reward is only obtained when the robot completes an episode without any collisions, specifically when the phase reaches 1.

As we mentioned in the previous chapter, we use not only the via point modulation but also the time modulation. When the lower-level MP is allowed to modulate time, the velocity reward is set as

$$r_{speed} = -||\alpha - \alpha_s||, \tag{3.9}$$

where r_{speed} is the speed reward, which is used to limit the speed of trajectory execution. α is the parameter of the phase function. The speed of the trajectory can be expressed by the difference between executed α and the standard $-\alpha_s$. In order to understand this parameter α, we need to know what the phase function looks like. We defined three optional phase functions as follows

$$\psi_1(t, \alpha) = \min(\alpha t, 1), \tag{3.10}$$

$$\psi_2(t, \alpha) = \frac{1 - \exp(-\alpha t)}{1 + \exp(-\alpha t)}, \tag{3.11}$$

$$\psi_3(t, \alpha) = 1 - \exp(-\alpha t). \tag{3.12}$$

Figure 3.2 shows the change of phase over time with different α. In the upper left corner of figure 3.2, we can observe the ReLU phase function. It exhibits a linear relationship between time and phase, with a maximum value of 1. The advantage of using this phase function is that the rate of change in the phase, which is determined by the adjustable parameter α, remains constant until phase reaches the value of 1. Once it reaches 1, the rate of change becomes 0. However, one drawback of this phase function is the presence of a non-smooth point in the function curve. Sigmoid and Exponential phase function are smoother the ReLU, but they do not have a fixed increasing rate like ReLU. As shown in the figure, they increase slower as phase gets closer to 1.

In our setting, phase is the relative time used to measure the progress of the task. The phase is always between 0 and 1. When phase is 0, it represents the starting point of the demonstrations trajectory, and when phase is equal to 1, it is the end point of the demonstrations trajectory. The MP in the lower level always need the phase of the via point to obtain the conditioned path, and the upper layer MDP decides at what speed the path will be executed. Since the phase represents the progress of the task, it is necessary to keep the phase continuous in the MDP. Otherwise, the trajectories generated by adjacent steps will be discontinuous. We hope to use different execution speeds in different steps, that is to say, α is not fixed in MDP. Each phase function increases monotonically over time, but if the alpha changes, the continuity of the phase between the entire MDP cannot be guaranteed. In response to this, we made some small modifications to the phase function. An offset term t_o is added to the phase function. The t in the original phase function becomes $t - t_o$, so we can get the expression of offset time. The formula of modified ReLU as follows:

$$\psi_1(t, t_0, \alpha) = \min(\alpha(t - t_0), 1), \tag{3.13}$$

$$\Rightarrow \quad t_0 = t - \frac{\psi_1}{\alpha}. \tag{3.14}$$

This modification allows the robot to execute trajectories at any desired speed multiplier, ensuring continuity of the trajectory while reaching the next via point. However, one drawback is that the variation introduced by this modification may not be sufficiently smooth, potentially resulting in higher speeds when reaching the endpoint. Similarly, we can also obtain expressions for the modified phase functions for the other two cases,

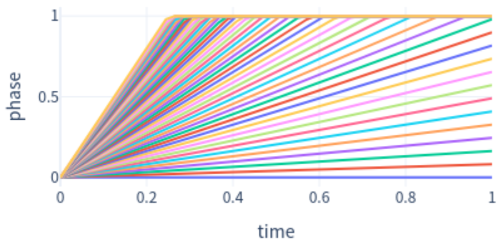
$$\psi_2(t, t_0, \alpha) = \frac{1 - \exp(-\alpha(t - t_0))}{1 + \exp(-\alpha(t - t_0))}, \tag{3.15}$$

$$\Rightarrow \quad t_0 = t + \frac{\log\left(\frac{2}{\psi_2 + 1} - 1\right)}{\alpha}, \tag{3.16}$$

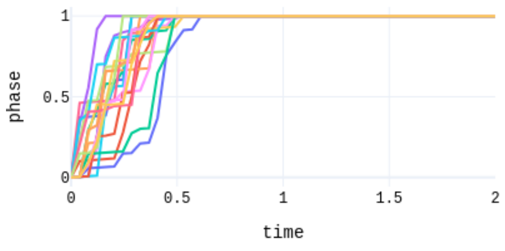
$$\psi_3(t, t_0, \alpha) = 1 - \exp(-\alpha(t - t_0)), \tag{3.17}$$

$$\Rightarrow \quad t_0 = t + \frac{\log(1 - \psi_3)}{\alpha}. \tag{3.18}$$

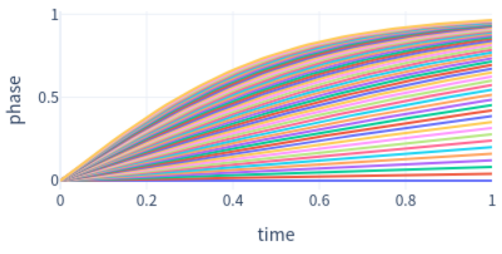
There are only subtle differences between the two phase functions. The growth rate of the exponential function is slightly faster than that of sigmoid in the early stage. Like the modified ReLU, these modified phase functions also ensure continuity of the phase, thus ensuring continuity of the trajectory. Additionally, they exhibit smoother variations compared to the modified ReLU, gradually slowing down to 0 velocity at the endpoint. However, one drawback is that they may generate excessively high initial velocities near the starting point. The offset time of each step will be saved in the observation value. In the next step, current time to next time can be mapped to current phase to next phase through the modified phase function. The modified phase function is shown in the right side of figure 3.2. In order to limit the α from changing too much, we set a standard α_s for each phase function. When $\alpha = \alpha_s$, the time to complete the task will be the same as the time of the demonstrations. Due to the exponential function, the phase functions rise very slowly when they are close to 1, so we use $\psi(t = 1) = 0.99$ to get the α_s. The standard α_s of the three phase functions are $\alpha_{s1} = 1$, $\alpha_{s2} = ln199$, $\alpha_{s3} = ln100$.



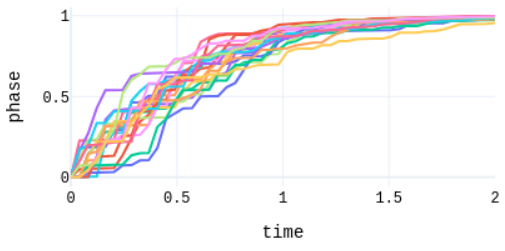
(a) ReLU



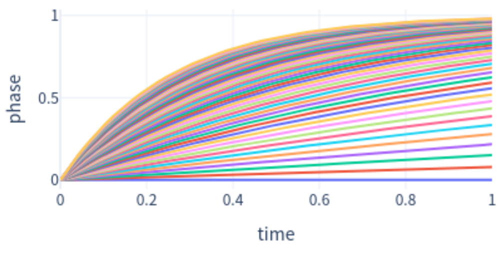
(b) modified ReLU



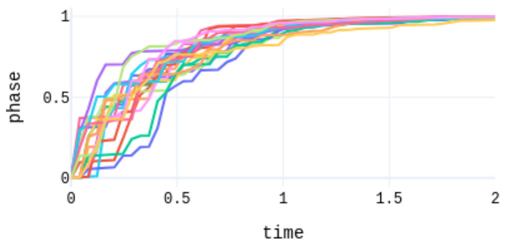
(c) Sigmoid



(d) modified Sigmoid



(e) Exponential



(f) modified Exponential

Figure 3.2.: Phase functions with different α

4. Experiments

In this chapter we will first introduce the specific settings of the experiment, and then present a part of the results of the four data sets. The experiments are designed to validate the effectiveness and performance of our approach, specifically focusing on assessing its efficacy in addressing the research objectives outlined in previous chapters. In our method, there are multiple combinations of phase functions and MP. Due to space limitations, we are unable to present all of them in the thesis. Only a partial set of results will be showcased in this chapter. These results serve to illustrate the capabilities and potential applications of our method, while also providing a basis for further discussion and analysis. In the section 4.1, we will provide details of our toy task. The section will compare the experimental results, with a particular focus on the differences between linear MP (where the phase is equal to time) and MP with the inclusion of a phase function.

4.1. Setup

We use 2D trajectories based on handwritten data as demonstrations. Figure 4.1 shows data sets we use. As shown in the 4.2, the largest red box is the boundary of the environment and the red rectangle in the middle with dashed line is the range of the initial position of the obstacle. At the beginning of each episode, obstacles will appear at a random position inside the smaller box the with a random speed. When the trajectory collides with the obstacle or with the the boundary of the environment, it will be punished for collision. We trained our model using the SAC algorithm, and the hyperparameters of the algorithm can be found in the appendix. After training for $500,000$ steps on each dataset, we conducted testing for 1000 steps using the same seed for random numbers.

4.2. Results

In this section, we will present some results on different data set. First, let us explain what each component in the image represents. In this section, images are represented by the following rules. Just as depicted in the figure 4.4, the larger red ellipses indicate obstacles. The dark green line represents the mean of the demonstration, and the light green area represents the distribution range of the demonstration. The starting point of the demonstration is a hollow circle, and the end point is a red star mark. The blue dotted line represents the mean value of the conditioned trajectory, and the blue area represents the distribution of the conditioned trajectory, which includes the actual implementation and the imaginary part. Two smaller pink circles represents the two via points of each step, that is, the current position and the next target position. To illustrate and compare the distinctions in via point modulation between ProMP and VMP, the entire conditional distribution is visually represented on the graph using a purple region. However, in practical execution, only the segment of the trajectory between the two via points is traversed. The deep red particles are used to represent the current position of the robot. The light blue ellipse is the 0.95 confidence interval of the nominal MP for the corresponding time of the via point. If the via point is outside of this light blue ellipse, it is considered to be out of statistics.

Then, we use the results of Lasa Bended Line data set to compare the differences between these methods. Figure 4.4, 4.5, 4.6 and 4.7 show the examples with same envrionment settings but using ProMP with linear phase, VMP with linear phase, ProMP with sigmoid phase and VMP wiith sigmoid phase respectively. Each case is represented by images taken at four selected time points. The obstacle emerges in the middle of the trajectory and gradually moves upward, and the initial position and velocity of the obstacle in four cases are same. The agent give two via points for each case, one in the middle and one at the end. We start with the example of ProMP with linear phase in figure 4.4. In this scenario, the conditional distribution generated by the agent's first via point, given at middle of the mean trajectory, closely overlaps with the distribution of the demonstration, as illustrated in the subplot located in the upper left corner . Due to the obstacle blocking the original path, the second via point is not set as the original endpoint but rather placed very close to it. However, since the variance of the demonstration data set near the endpoint is very small, even though this via point appears to be close to the endpoint, it is considered an out-of-statistic via point. Despite the robot successfully avoiding the obstacle, the conditional distribution of the trajectory no longer exhibits the pattern seen in the demonstration. And this is precisely the drawback of ProMP, as we discussed in the previous chapters. When we change the ProMP to VMP in the lower level, there will be a significant change in the outcome. As it is presented in 4.5, the agent selects a via point in the middle that is farther away from the demonstration to avoid colliding with the obstacle. Unlike ProMP, which updates the weights of primitives, VMP only modifies the elementary trajectory $\boldsymbol{h}$ and thereby maintains its shape. Moreover, the intermediate via point allows the agent to smoothly reach the endpoint in subsequent steps. Therefore, in this scenario, VMP showcases its advantage over ProMP. Since both of them employ a linear phase, the total time taken to complete the task is 1 for both. Next, let's take a look at the results with the inclusion of temporal modulation of ProMP. As depicted in Figure 4.6, the particle accelerates to bypass the obstacle and reaches the endpoint. The subplot in the upper right corner demonstrates that at $t = 0.3$, the particle has already traversed more than half of the trajectory. Compared to Figure 4.4, the generated trajectory closely resembles the shape of the demonstration. However, as it approaches the endpoint, its velocity gradually slows down. This is a characteristic of the sigmoid phase function, as we discussed earlier. Finally, let's take a look at the case of VMP with a sigmoid phase in figure 4.7. The agent selects a via point outside of the demonstration. Due to the small variance near the corresponding position in the nominal ProMP, the confidence ellipse representing the uncertainty region is too small to be visible on the graph. By modulating the elementary part, the trajectory maintains the shape of the demonstration. However, due to interpolation, the $\boldsymbol{h}$ component is not smooth, resulting in a non-smooth trajectory. Similar to Figure 4.4, the particle starts with a higher velocity and slows down towards the endpoint.

In these examples, the agent in the VMP-based lower-level framework provides out-of-statistics via points. However, it doesn't imply that this is always the case. The reason is that even when the via point is far from the demonstration, VMP can still generate promising trajectories, whereas ProMP's generated trajectories may collapse unexpectedly, as we mentioned in the earlier chapters. Therefore, compared to ProMP, when the agent operates in the VMP-based lower-level framework, it has more options for selecting via points. The agent will choose a via point based on the reward function's setting, aiming for the via point that accumulates the highest reward, rather than strictly choosing a via point outside the available data. Similar results are observed in other data set, and we won't repeat the discussion here. In summary, with our approach, the particle is able to consistently and successfully avoid obstacles nearly all the time, but the similarity of the generated trajectories varies. When the via point is far from the demonstration, VMP generates trajectories that are more similar to the demonstration compared to ProMP. With the introduction of the phase function, the agent has a greater range of actions, and both VMP and ProMP can generate trajectories that match the shape of the demonstration. VMP

Settings	ProMP linear	VMP linear	ProMP ReLU	VMP ReLU
r	0.13 ± 0.19	0.11 ± 0.21	1.33 ± 0.39	1.37 ± 0.38
r_{sim}	0.34 ± 0.54	0.34 ± 0.55	0.76 ± 0.38	0.7 ± 0.36
r_{speed}	0	0	-0.36 ± 0.05	-0.37 ± 0.06
r_{goal}	1.45	1.48	1.45	1.44
r_{col}	0	-0.005	0	0
Success rate	1	0.997	1	1
Number of via points	2	2	2.09 ± 0.51	2.03 ± 0.48

Table 4.1.: Test results for l-shape data set

exhibits relatively smaller changes in shape, but it may result in non-smooth trajectories. The charts labeled as 4.3 4.2 4.4 and 4.1 correspond to the test results of four different data sets. The corresponding learning curves for these results can be found in the appendix. It can be observed that when using MP with a phase function at the lower level, the average reward tends to be higher than it without phase function. Apart from the l-shaped dataset, VMP consistently outperforms ProMP in terms of average reward across all data sets. These results are consistent with the examples shown in the graphs. In all the displayed results, the number of via points is less than 3. This does not mean that our method can only generate a small number of via points. The coefficient of the velocity reward determines whether the agent prefers position modulation or time modulation. In our setup, the velocity reward is always negative, and more via points typically result in more frequent changes in velocity. Therefore, when the coefficient of the speed reward increases, an excessive number of via points incurs a higher penalty. The agent is more inclined to choose a longer-term via point to avoid the penalty associated with switching via points. Therefore, when the coefficient of the speed reward is relatively low, the agent tends to generate via points that are closer to its current position. As a result, the number of via points required to complete a trajectory increases. When the agent generates an excessive number of via points, it can lead to longer execution times to complete the trajectory. Due to space limitations, we will not present cases with a large number of via points here. But we can illustrate this point with an example of learning curves, which is shown in figure 4.3.

Settings	**ProMP linear**	**VMP linear**	**ProMP sig**	**VMP sig**
r	-0.7 ± 0.51	-0.53 ± 0.4	0.43 ± 0.51	0.8 ± 0.48
r_{sim}	0.34 ± 0.41	0.32 ± 0.41	0.3 ± 0.37	0.27 ± 0.52
r_{speed}	0	0	-0.1 ± 0.09	-0.12 ± 0.11
r_{goal}	0.35	0.34	0.06	1.38
r_{col}	0	0	-0.01 ± 0.1	-0.01 ± 0.1
Success rate	1	0	1	1
Number of via points	2 ± 0.03	2 ± 0.03	2 ± 0.03	1.93 ± 0.25

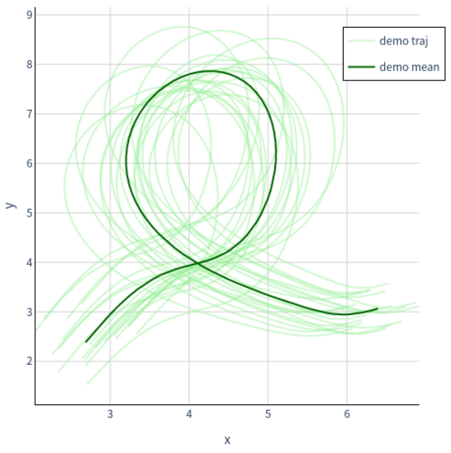
Table 4.2.: Test results for LASA Bended Line data

Settings	**ProMP linear**	**VMP linear**	**ProMP ReLU**	**VMP ReLU**
r	-0.75 ± 0.35	-0.61 ± 0.57	0.19 ± 0.48	0.24 ± 0.59
r_{sim}	0.073 ± 0.377	0.23 ± 0.47	0.23 ± 0.37	0.34 ± 0.42
r_{speed}	0	0	-0.45 ± 0.07	-0.33 ± 0.08
r_{goal}	-0.04	0.53	0.25	0.45
r_{col}	0	0	0	0
Success rate	1	1	1	1
Number of via points	2	2 ± 0.03	1.73 ± 0.13	1.6 ± 0.48

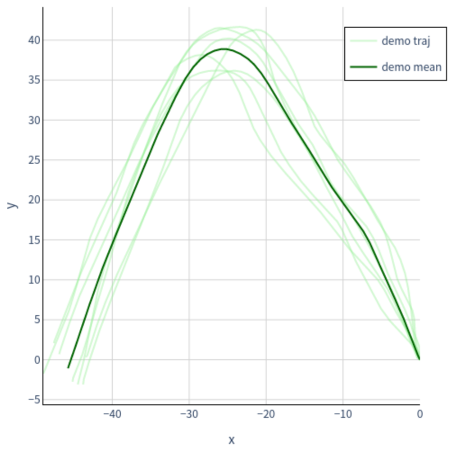
Table 4.3.: Test results for LASA Angle data set

Settings	**ProMP linear**	**VMP linear**	**ProMP exp**	**VMP exp**
r	-0.71 ± 0.97	-0.37 ± 0.6	0.21 ± 0.63	1.08 ± 0.80
r_{sim}	0.25 ± 0.42	0.12 ± 0.56	0.19 ± 0.36	0.35 ± 0.58
r_{speed}	0	0	-0.12 ± 0.09	-0.29 ± 0.24
r_{goal}	-0.08	1.39	0.19	0.92
r_{col}	-0.08 ± 0.47	-0.01 ± 0.1	-0.05 ± 0.38	0
Success rate	0.97	1	0.98	1
Number of via points	2.21 ± 0.46	2 ± 0.07	1.89 ± 0.25	1.66 ± 0.47

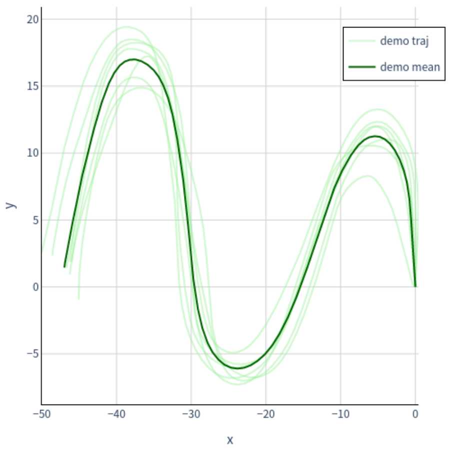
Table 4.4.: Test results for LASA Sine data set



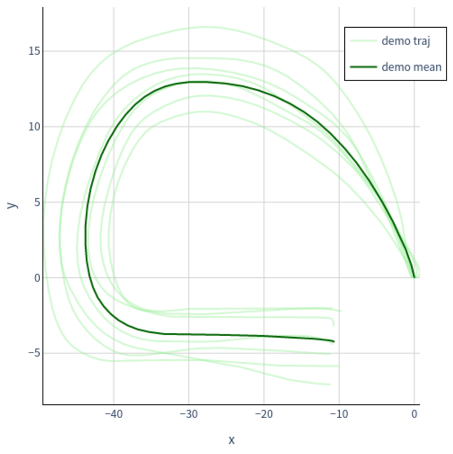
(a) I shape



(b) LASA angle



(c) LASA sine



(d) LASA bended line

Figure 4.1.: The handwritten data set used in the experiment. The first data set 4.1a was manually generated with more quantity and higher variance. The other three figures, 4.1b 4.1c and 4.1d are from the LASA dataset, with fewer quantity and smaller variance at the endpoints.

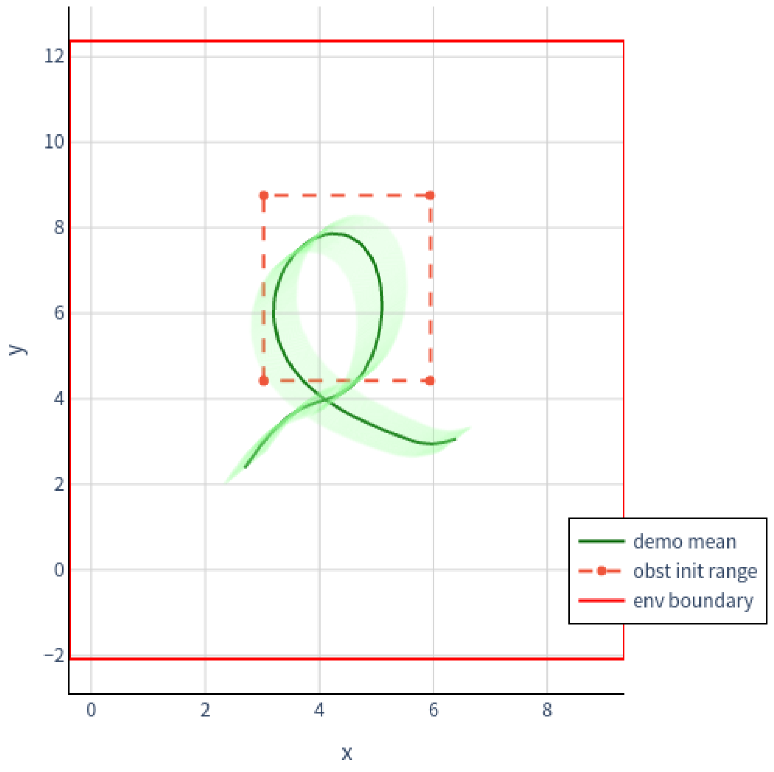


Figure 4.2.: Initial settings for the experiment on I-shaped data. Similar settings are used for other datasets.

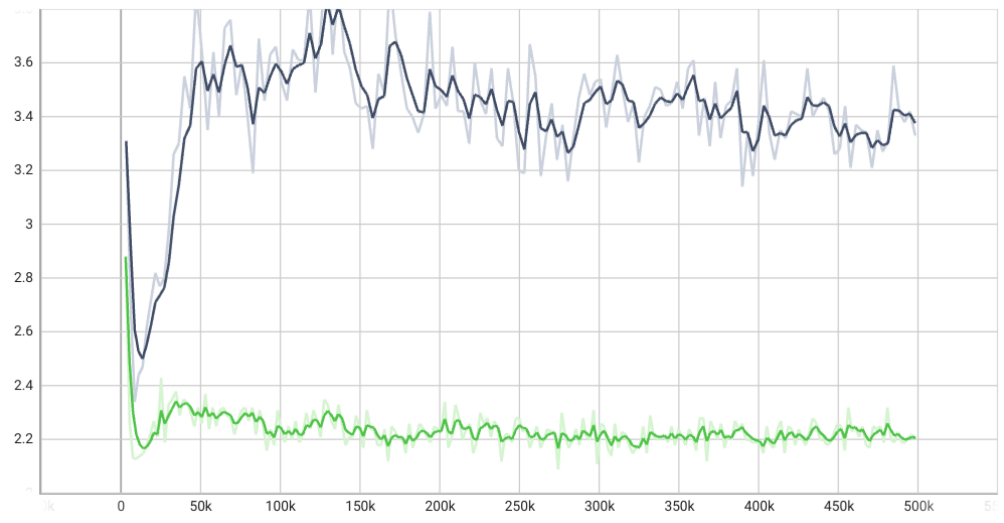
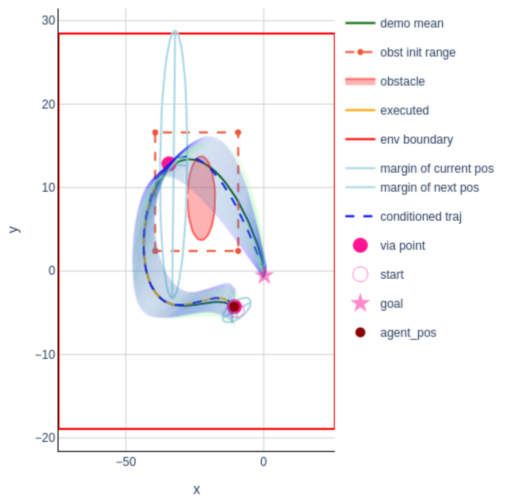
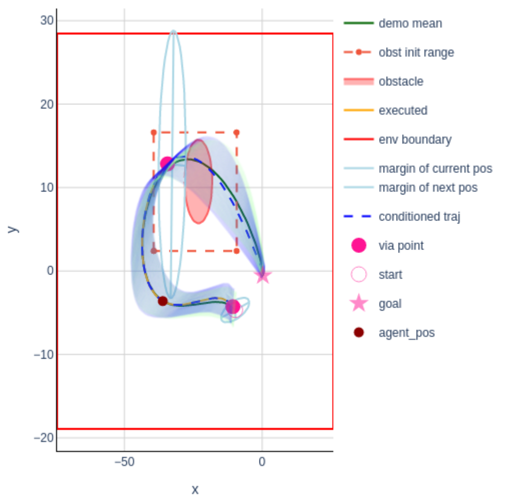


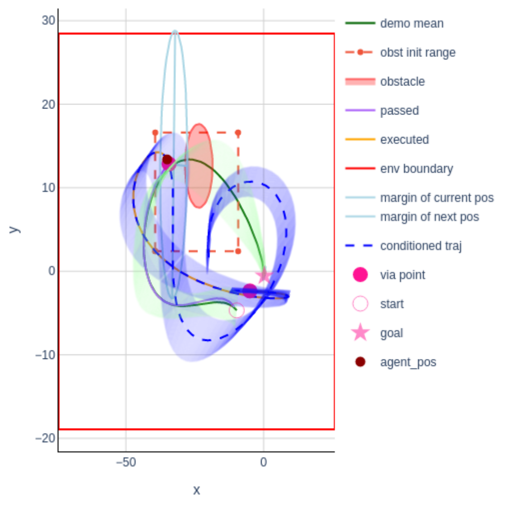
Figure 4.3.: Learning curves under different speed reward coefficients. Both experiments were conducted on the L-shape dataset using VMP with a ReLU phase function. The horizontal axis represents the number of training steps, while the vertical axis represents the number of via points in one episode. The black curve represents the learning process with a lower velocity penalty coefficient of 0.1, while the green curve represents the coefficient of 0.5. When the velocity penalty is relatively high, the agent tends to maintain its current movement speed until the end, resulting in a relatively smaller number of via points.



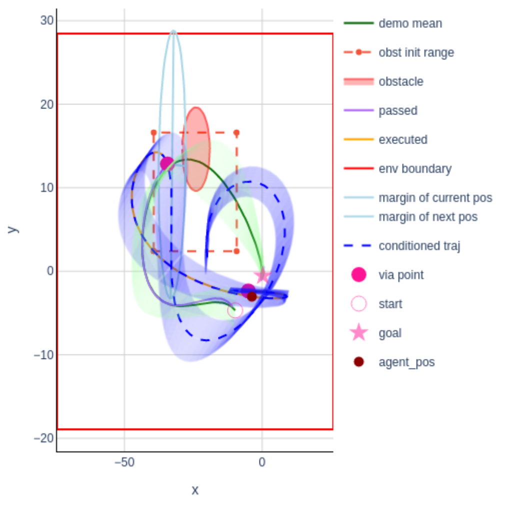
(a) $t = 0$



(b) $t = 0.3$

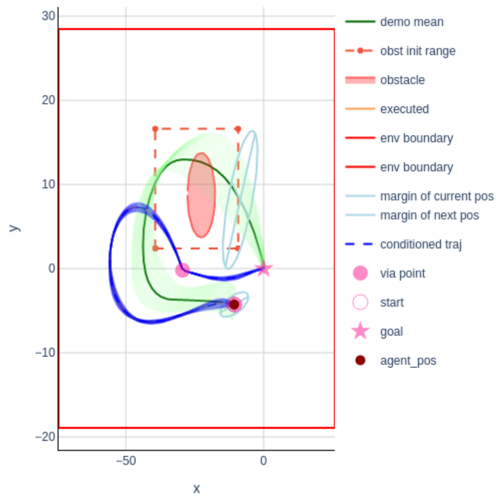


(c) $t = 0.6$

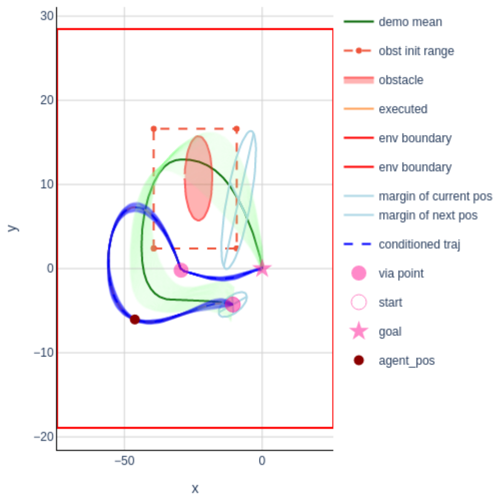


(d) $t = 0.9$

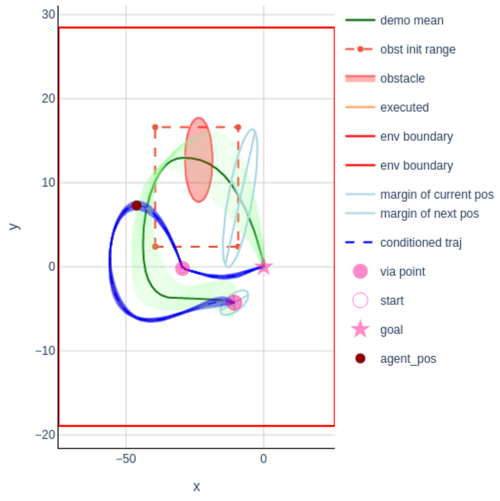
Figure 4.4.: Results of ProMP with linear phase on LASA Bended Line data. When the via point is close to the demonstration, it maintains the same shape, but when it is far away, the pattern changes.



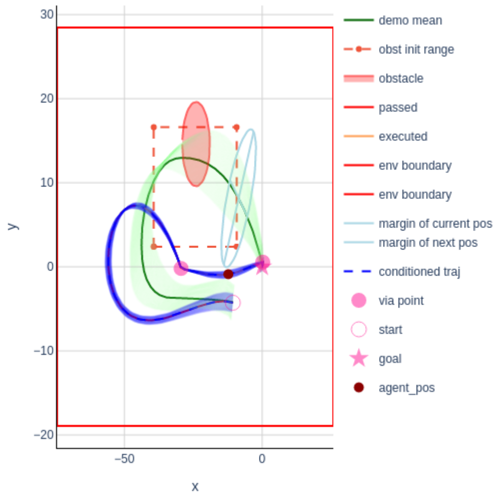
(a) $t = 0$



(b) $t = 0.3$

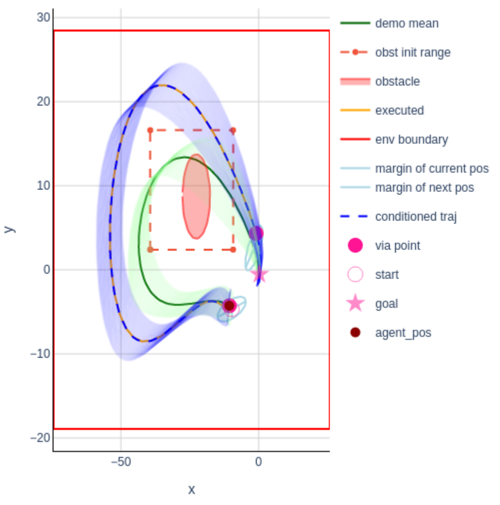


(c) $t = 0.6$

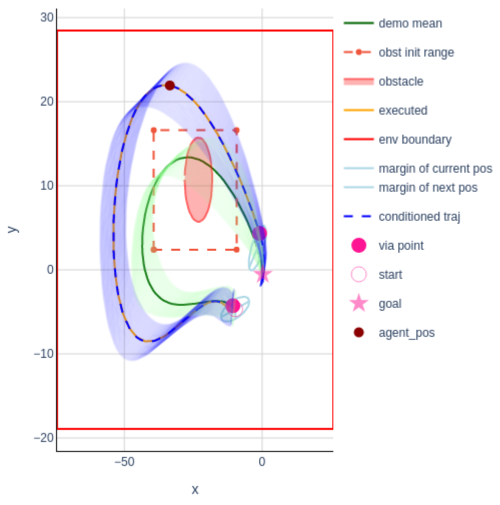


(d) $t = 0.9$

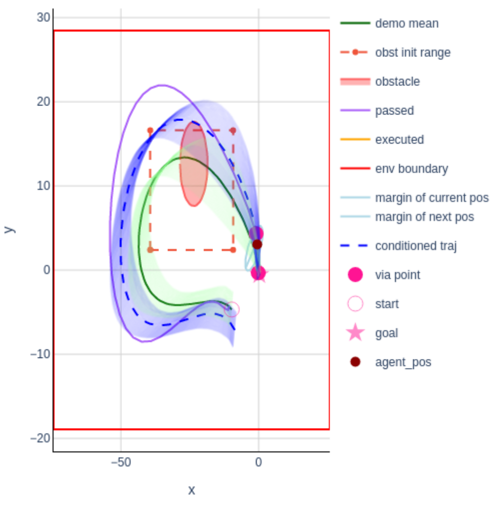
Figure 4.5.: Results of VMP with linear phase on LASA Bended Line data. VMP modifies the elementary trajectory h and thereby maintains the shape.



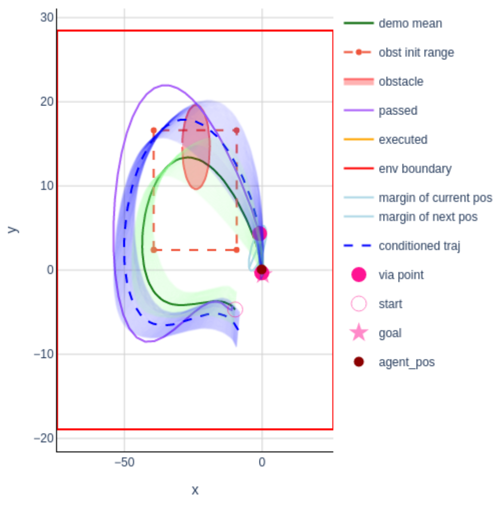
(a) $t = 0$



(b) $t = 0.3$

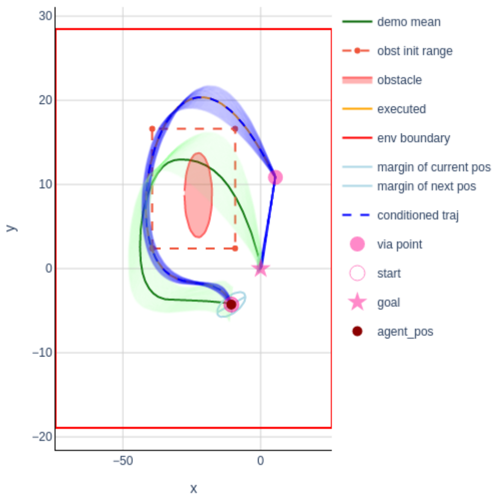


(c) $t = 0.6$

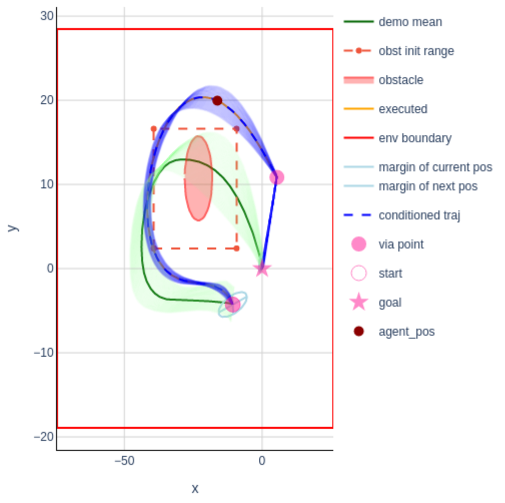


(d) $t = 0.9$

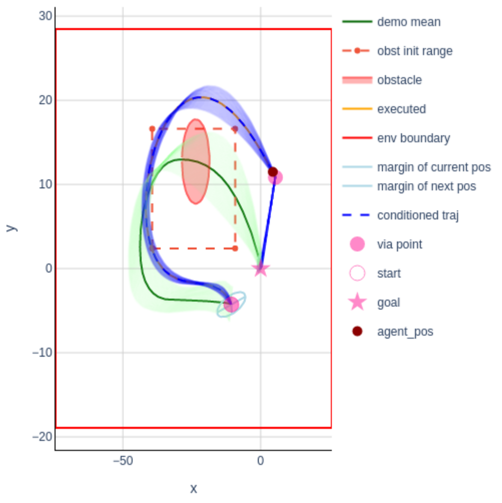
Figure 4.6.: Results of ProMP with sigmoid phase on LASA Bended Line data. At $t = 0.3$, the particle has already traversed more than half of the trajectory, and its velocity gradually slows down as it approaches the endpoint.



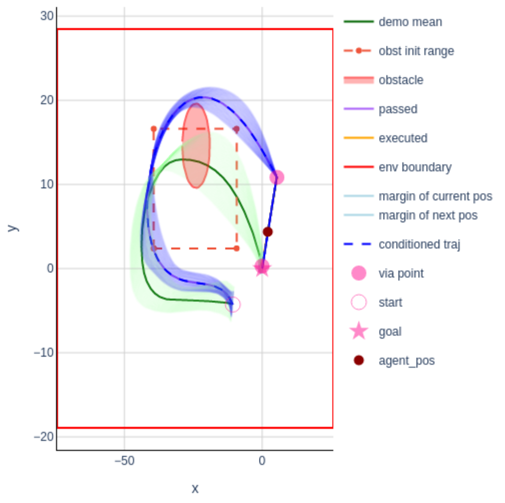
(a) $t = 0$



(b) $t = 0.3$

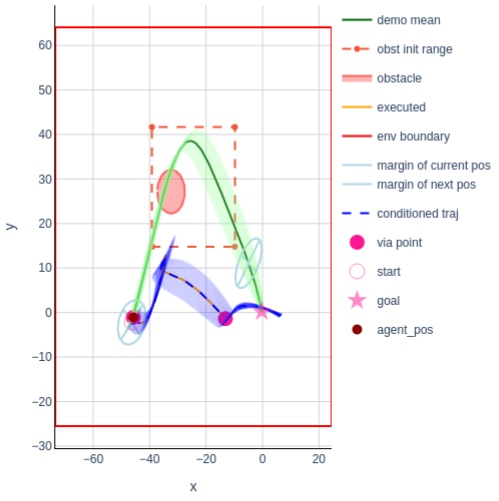


(c) $t = 0.6$

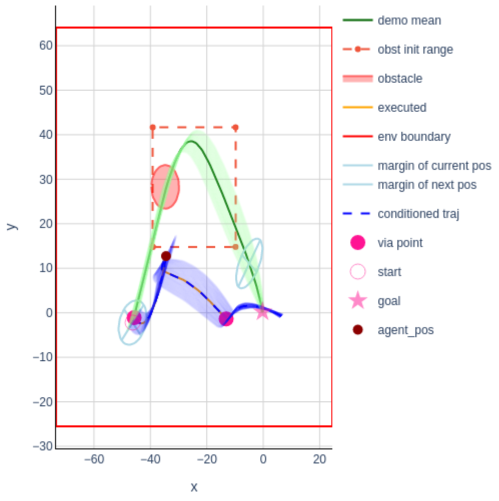


(d) $t = 0.9$

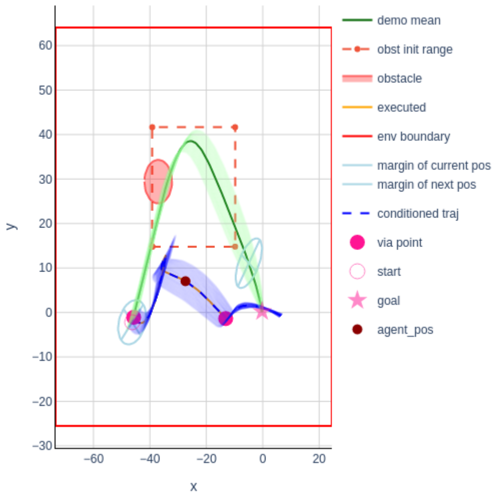
Figure 4.7.: Results of VMP with sigmoid phase on LASA Bended Line data.Interpolation results in a non-smooth trajectory



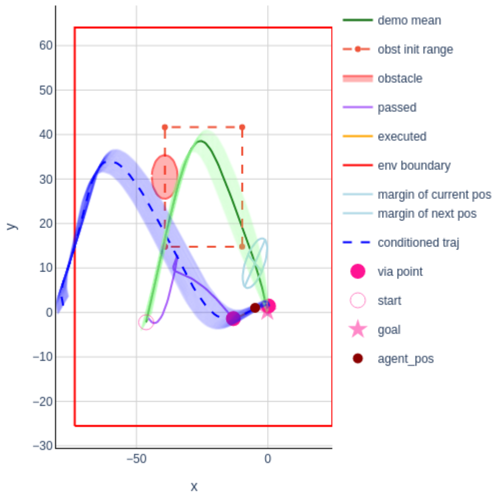
(a) $t = 0$



(b) $t = 0.3$

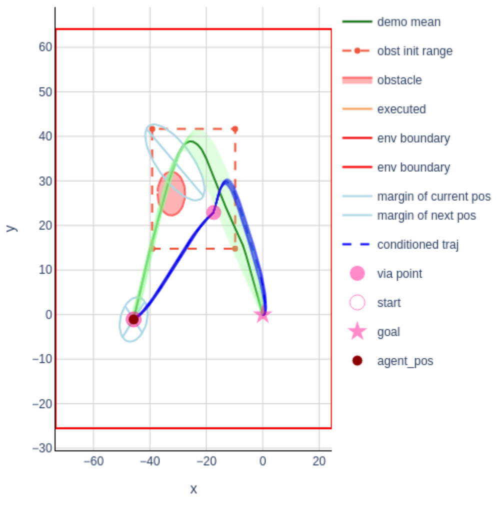


(c) $t = 0.6$

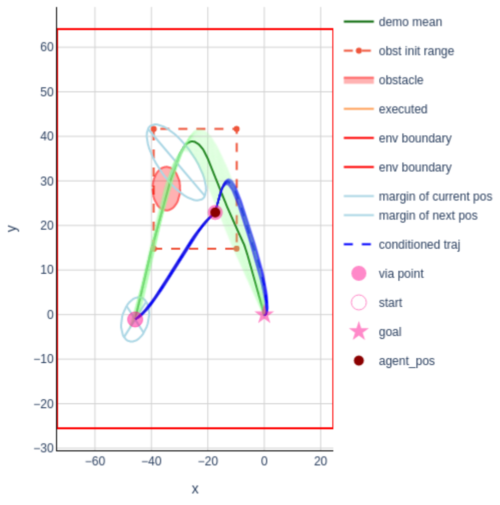


(d) $t = 0.9$

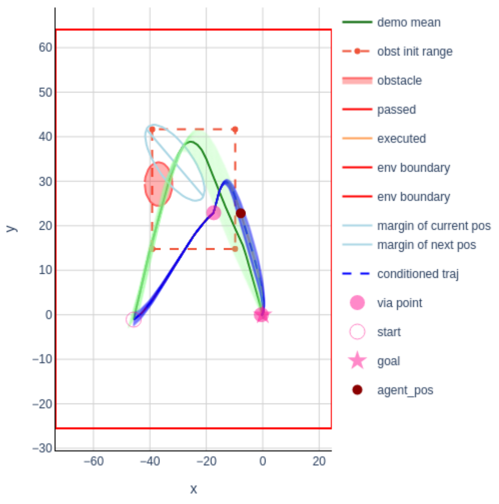
Figure 4.8.: Results of ProMP with linear phase on LASA Angle data



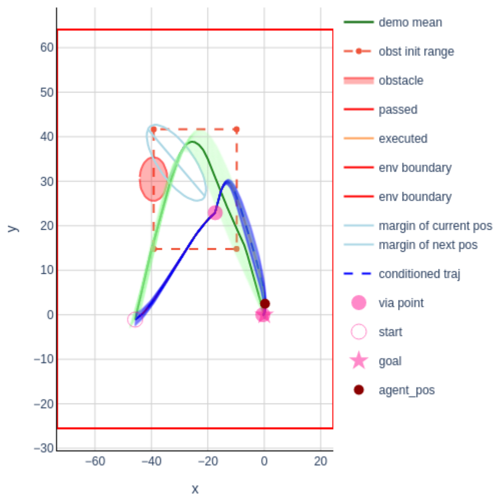
(a) $t = 0$



(b) $t = 0.3$

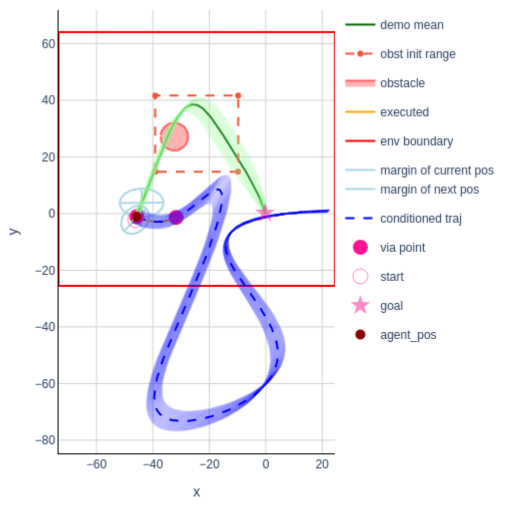


(c) $t = 0.6$

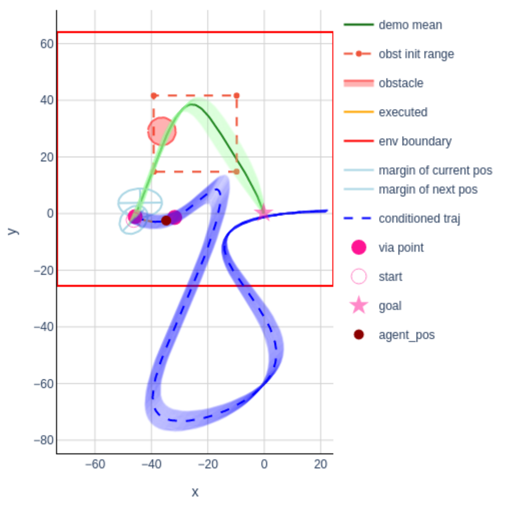


(d) $t = 0.9$

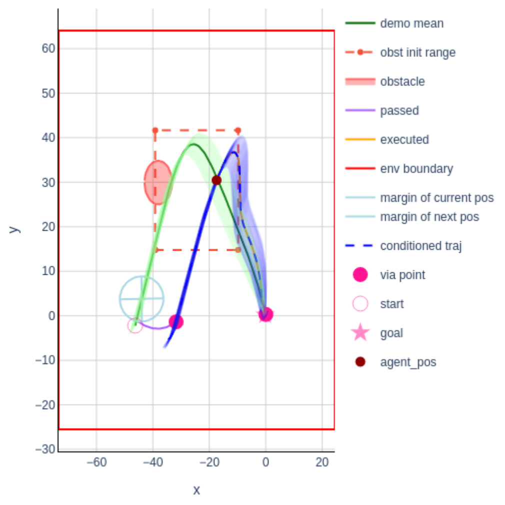
Figure 4.9.: Results of VMP with linear phase on LASA Angle data



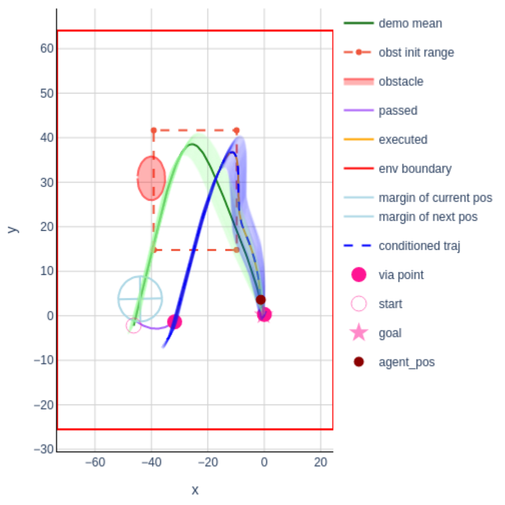
(a) $t = 0$



(b) $t = 0.5$

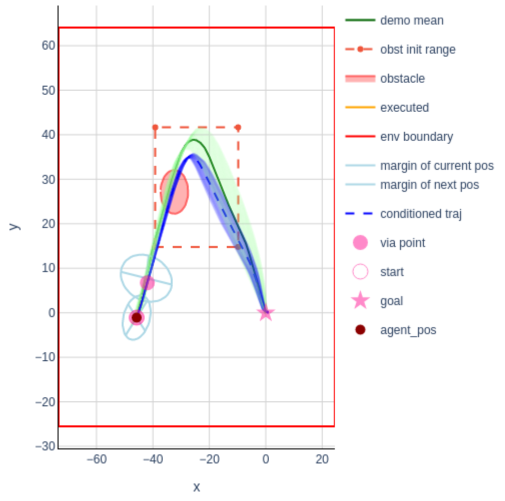


(c) $t = 0.75$

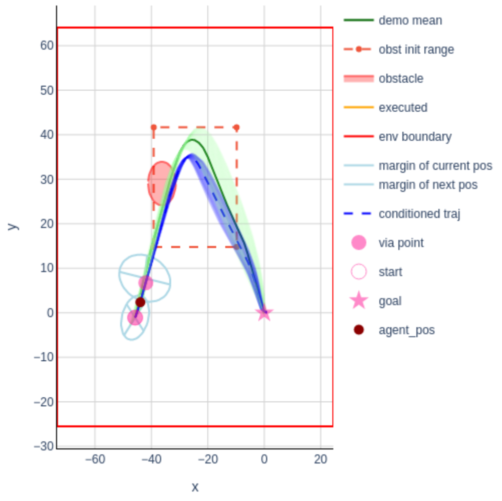


(d) $t = 1$

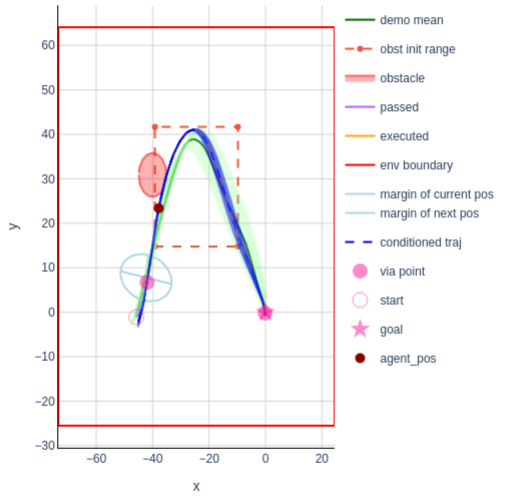
Figure 4.10.: Results of ProMP with ReLU phase on LASA Angle data



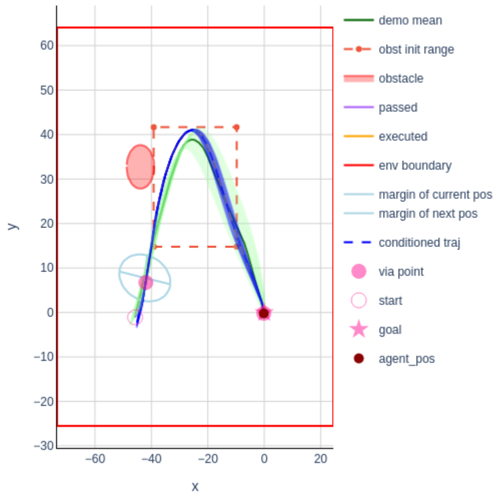
(a) $t = 0$



(b) $t = 0.5$

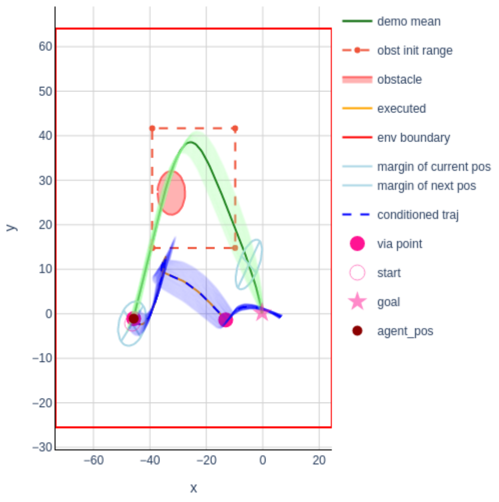


(c) $t = 0.75$

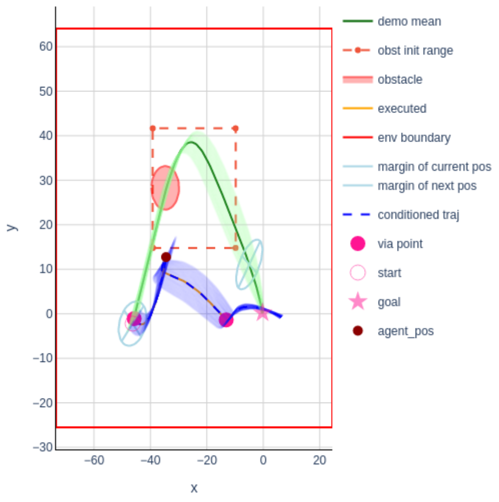


(d) $t = 1$

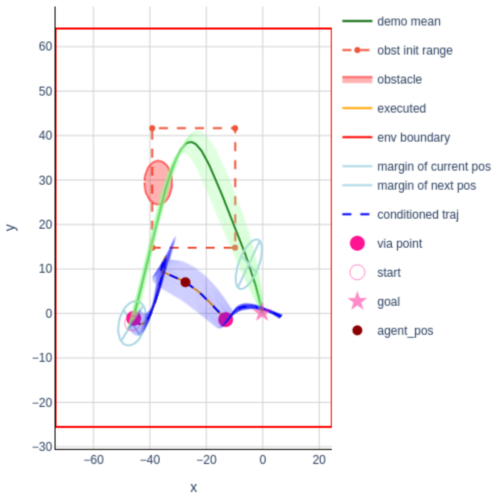
Figure 4.11.: Results of VMP with ReLU phase on LASA Angle data



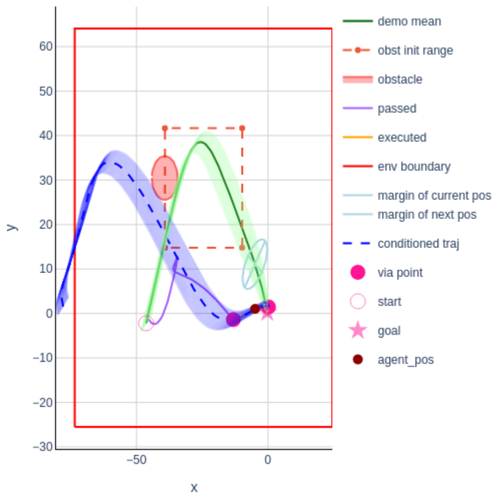
(a) $t = 0$



(b) $t = 0.5$

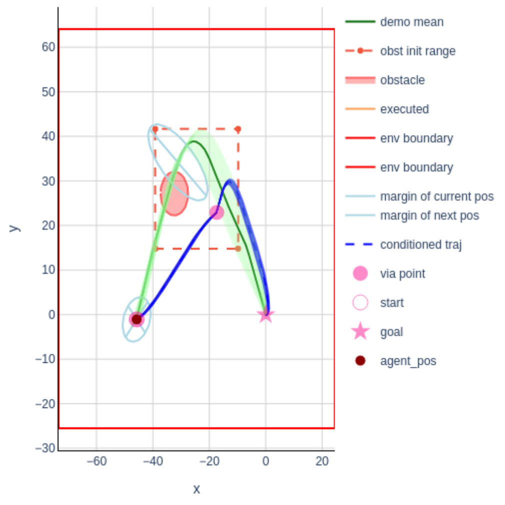


(c) $t = 0.75$

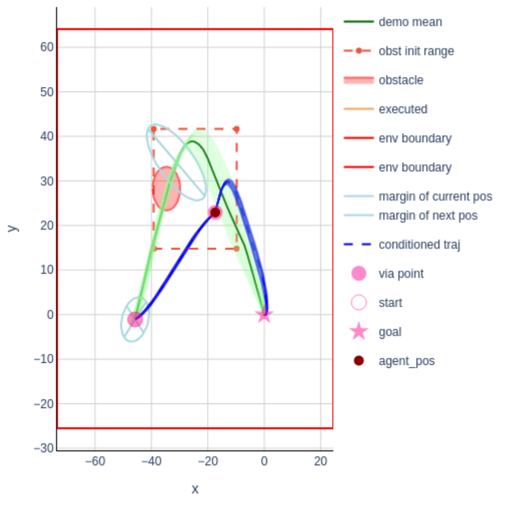


(d) $t = 1$

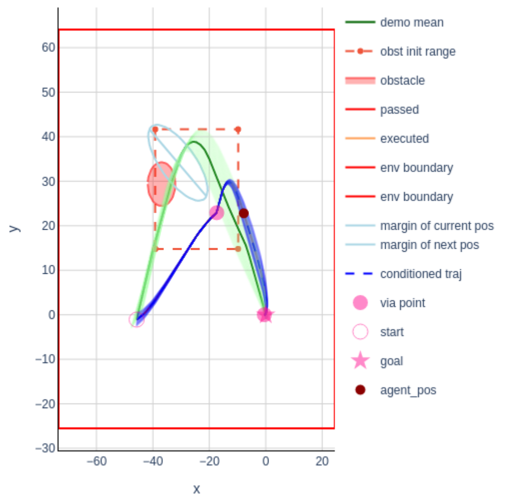
Figure 4.12.: Results of ProMP with linear phase on LASA Sine data



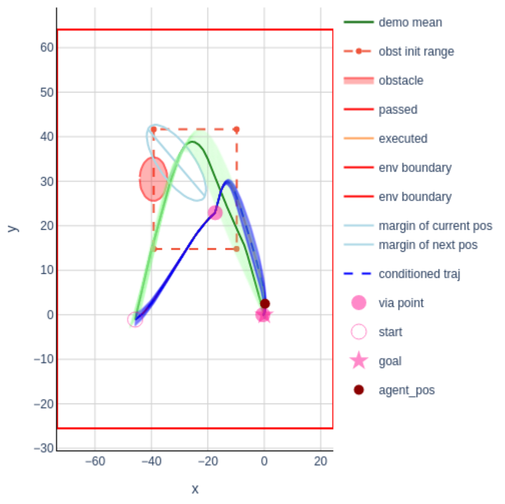
(a) $t = 0$



(b) $t = 0.5$

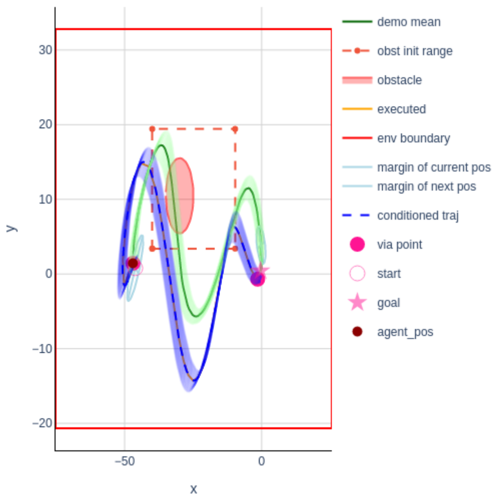


(c) $t = 0.75$

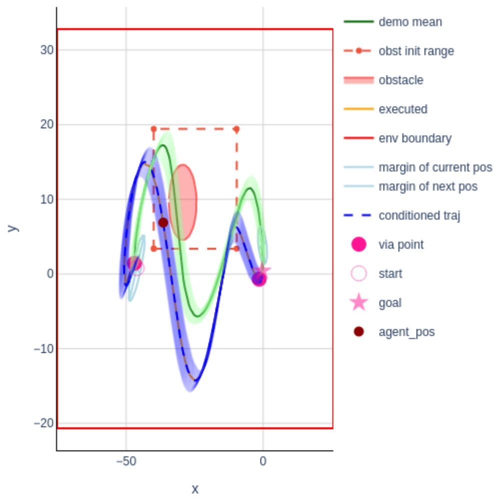


(d) $t = 1$

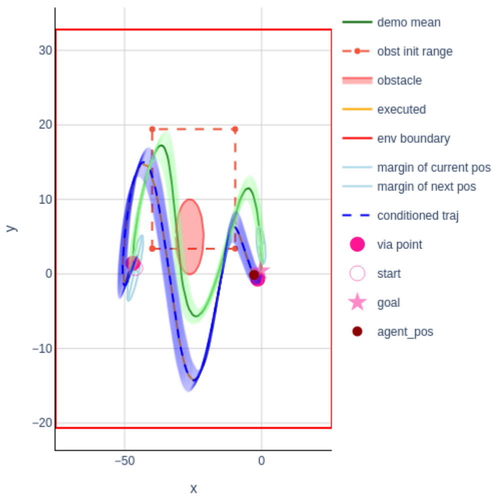
Figure 4.13.: Results of VMP with linear phase on LASA Sine data



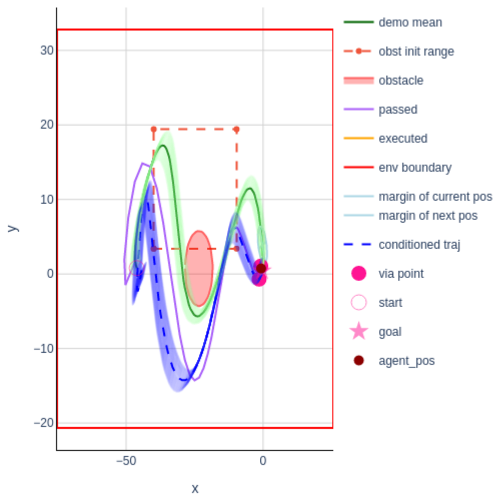
(a) $t = 0$



(b) $t = 0.5$

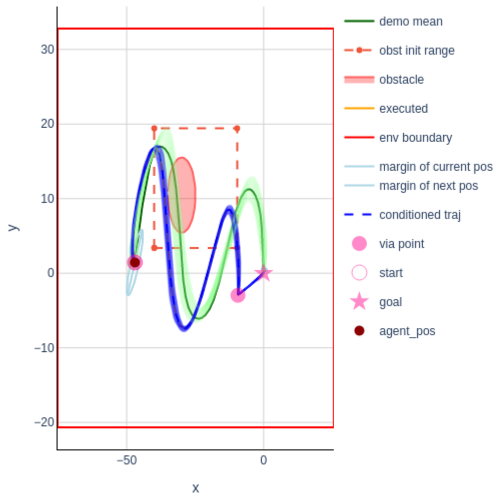


(c) $t = 0.75$

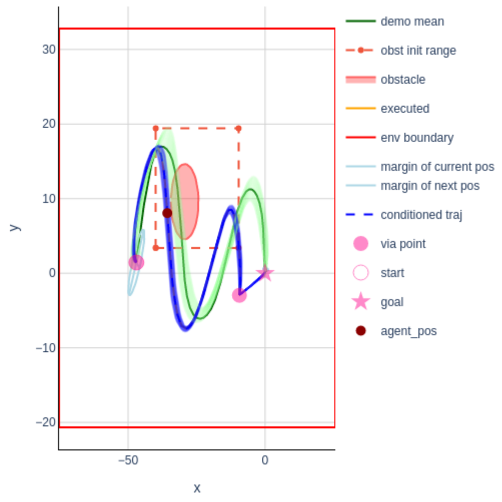


(d) $t = 1$

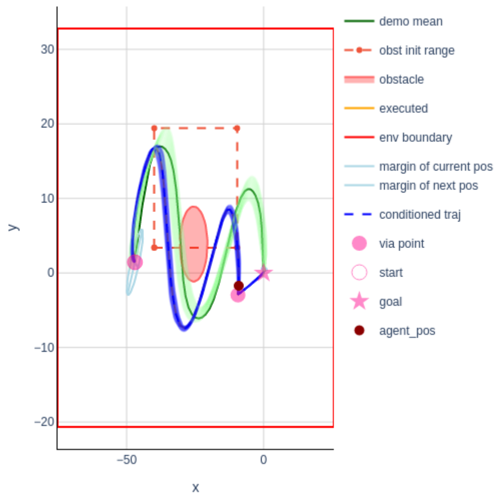
Figure 4.14.: Results of ProMP with exponential phase on LASA Sine data



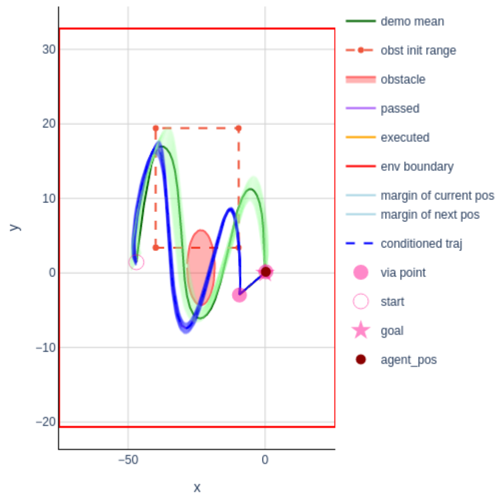
(a) $t = 0$



(b) $t = 0.5$

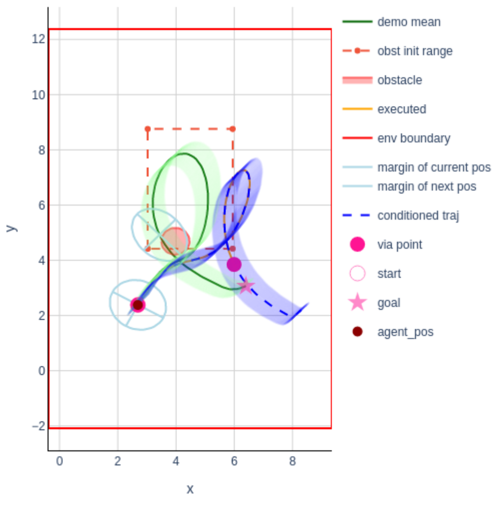


(c) $t = 0.75$

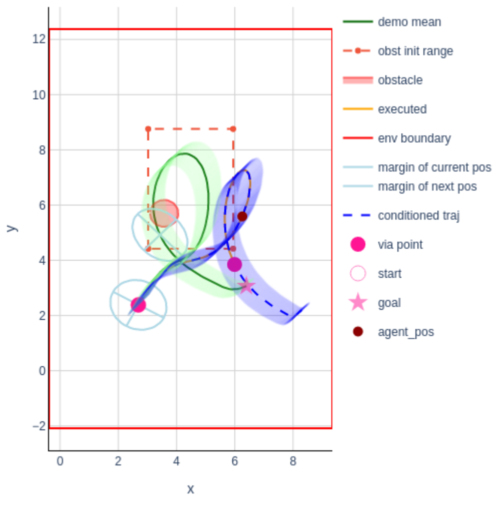


(d) $t = 1$

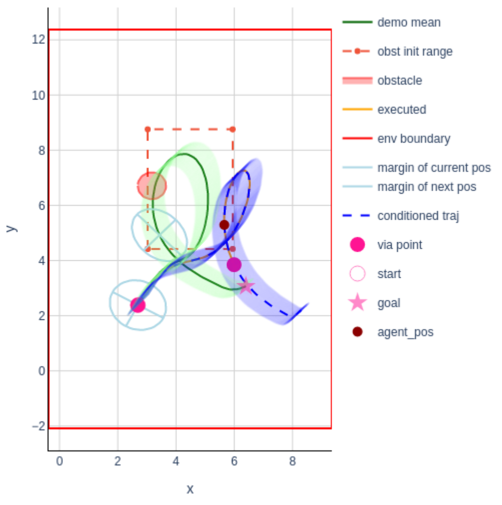
Figure 4.15.: Results of VMP with exponential phase on LASA Sine data



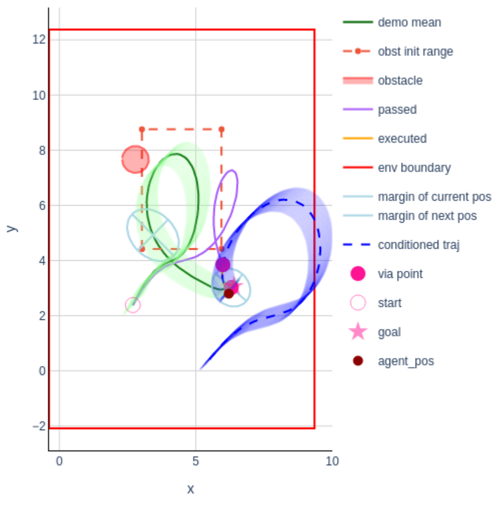
(a) $t = 0$



(b) $t = 0.5$

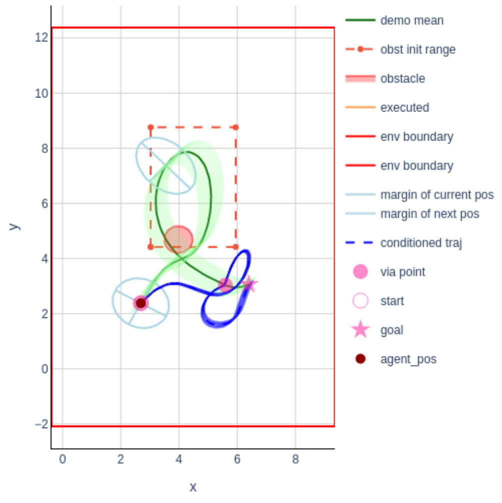


(c) $t = 0.75$

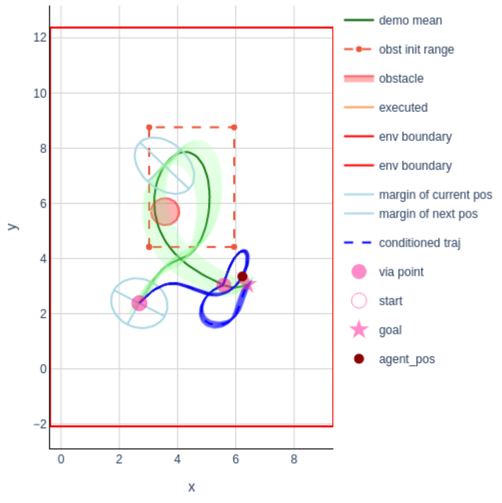


(d) $t = 1$

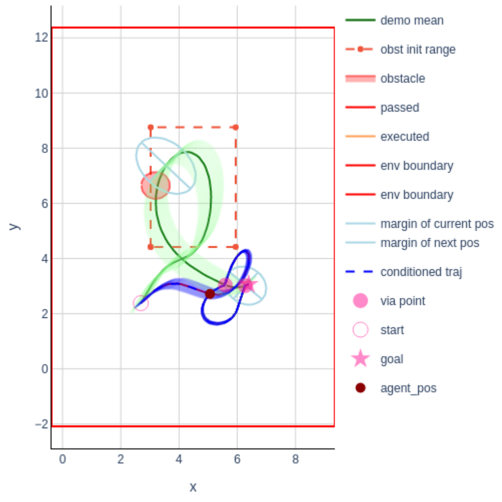
Figure 4.16.: Results of ProMP with linear phase on l-shaped data



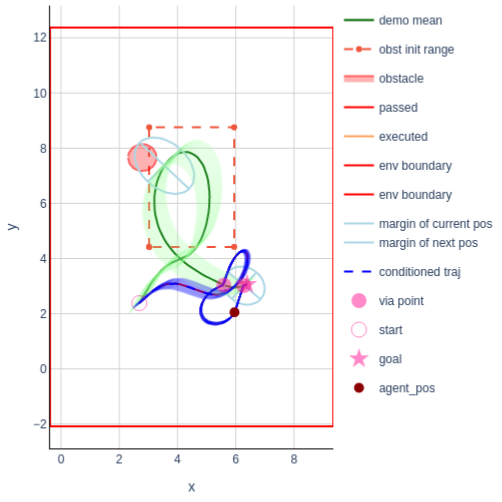
(a) $t = 0$



(b) $t = 0.5$

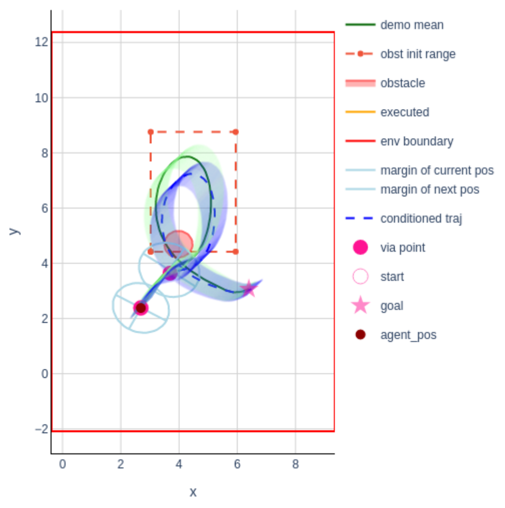


(c) $t = 0.75$

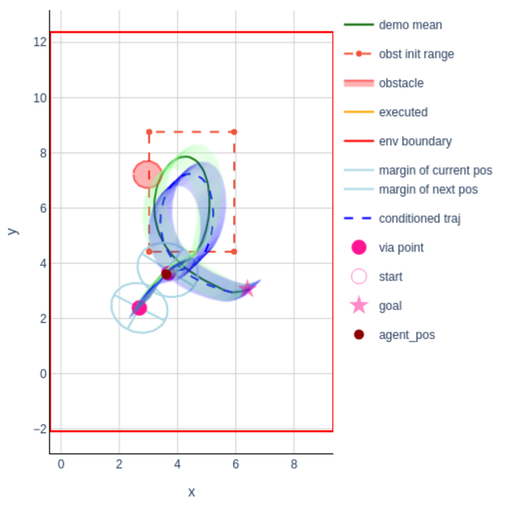


(d) $t = 1$

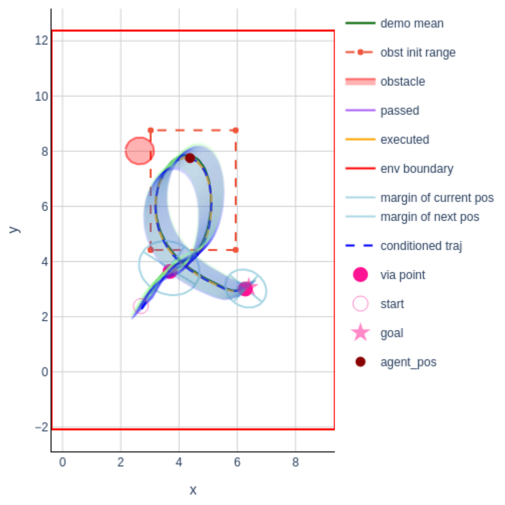
Figure 4.17.: Results of VMP with linear phase on l-shaped data



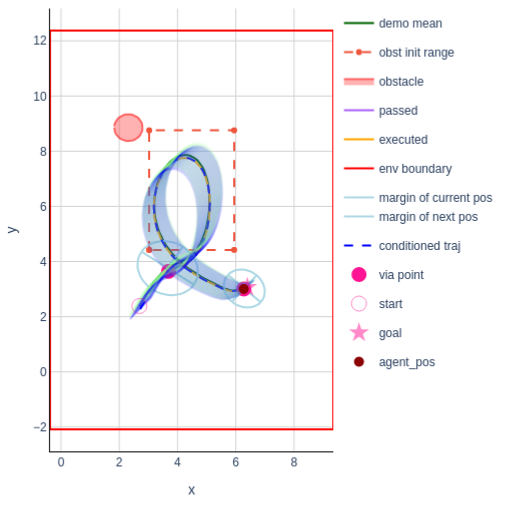
(a) $t = 0$



(b) $t = 0.5$

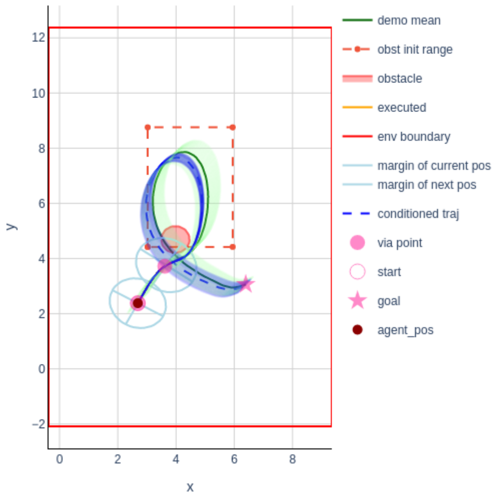


(c) $t = 0.75$

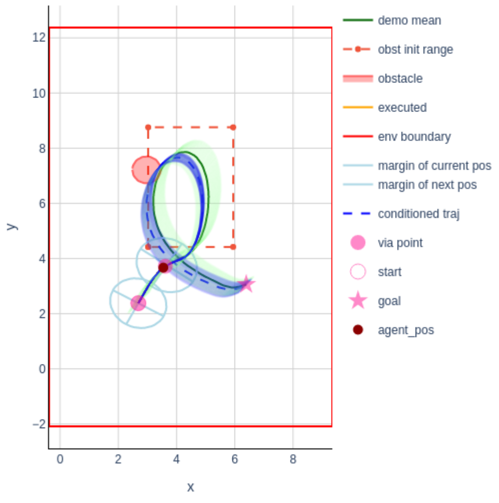


(d) $t = 1$

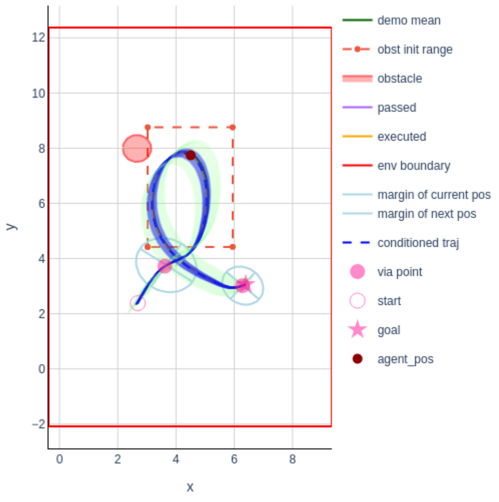
Figure 4.18.: Results of ProMP with ReLU phase on l-shaped data



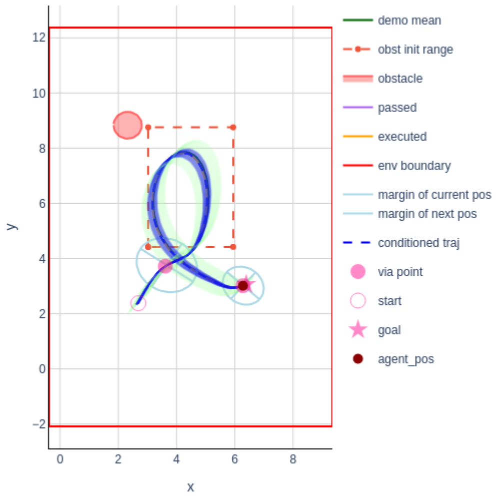
(a) $t = 0$



(b) $t = 0.5$



(c) $t = 0.75$



(d) $t = 1$

Figure 4.19.: Results of VMP with ReLU phase on l-shaped data

5. Discussion and Outlook

The incorporation of learning abilities from human demonstrations and their subsequent adaptation to various workspace is a vital requirement for the advancement of future robots. Although the ProMP framework allows the use of conditioning to generate trajectories that adapt to given task constraints, it does not directly provide a policy for how primitives should adjust as the environment changes. Some studies have tried adapting primitives using offline optimization, but they cannot be used online due to the high computational cost, and the potential of time modulation of primitives has not fully exploited. To overcome this issue, this thesis proposes a hierarchical method to use a high-level MDP that adapts the low level primitive based on contextual changes.

In Hierarchical Via-Movement Primitives (HVMP), the policy is represented by the mapping from context to the next via point. In the higher level, HVMP receives the information of the environment, such as the position and speed of obstacles, the current position and speed of the robot and phase, as the input, and then use the policy which is trained by the SAC algorithm to generate the position and time of the next passing point. The lower level uses the via point information given by the higher level's decision, to generate the trajectory between the current position and the next via point with the via point modulation of ProMP, and sends the generated trajectory collision and similarity information to the higher level as the foundation for next decision-making. In addition, we design three phase functions, which allow the robot to change the executed trajectory as well as the speed and time, and avoid obstacles by accelerating and decelerating. As far as we know, this has not been done in previous work.

In the experiment, We used the LASA handwritten data set as demonstration and tested the performance of the proposed algorithm in the environment of static obstacles and moving obstacles. The results show that our algorithm can make the robot maintain high similarity with the pattern of the demonstration while meeting the safety requirements. By changing the parameters of the corresponding part of the reward function, the agent can choose to trade off between position modulation and time modulation. Referring to the results of the experiments, further studies should focus on following objectives:

Test the effect of different confidence regions on the results In our work, we always use a fixed threshold when judging whether a via point is outside the statistic, that is, we assume that if the via point is far away from the demonstration when it is outside the 95% confidence interval of the demonstration. An ideal possibility is to obtain an optimal threshold based on the distribution of the demonstration, which may depend on the actual task requirements such as the maximal curvature of the trajectory.

Experiment with more complex obstacles In our experiments, we assume that the acceleration of the obstacle is 0 and we always have complete motion information of the obstacle. In the working environment of the actual robot, this setting is too ideal. So it is necessary to test our method with obstacles which have Brownian motion and different velocity levels. In this setting, collision prediction will be challenging, and using Kalman Filter to estimate the velocity of obstacles is an optional solution.

Test our algorithm on larger state space Due to time constraints, our method was not tested on real robots. In order to be able to apply the algorithm to real robots, some extensions would be significant. In the higher level, images can be used as the input and the information of obstacles can be obtained from the images. In the lower level, we can use human body motion data as demonstration, from which the humanoid robots can learn primitives for full-body locomotion. Before this, retargeting is essential because of the difference in the joint structure of humans and robots[8, 41].

Extended to interactive movement primitives In the future, it will be a common scenario for robots to assist humans in work, and robots will have a lot of interactions with humans in the workspace. There have been some studies on extending movement primitives to human-robot interaction scenarios in order to learn interactive actions[1, 28, 6]. In these works, primitives are obtained from the demonstration of both sides of the interaction. It will be interesting to implement our method with these frameworks.

Bibliography

[1] Heni Ben Amor et al. “Interaction primitives for human-robot cooperation tasks”. In: *2014 IEEE international conference on robotics and automation (ICRA)*. IEEE. 2014, pp. 2831–2837.

[2] Kai Arulkumaran et al. “Deep Reinforcement Learning: A Brief Survey”. In: *IEEE Signal Processing Magazine* 34.6 (2017), pp. 26–38. DOI: 10.1109/MSP.2017.2743240.

[3] Shikhar Bahl et al. “Neural dynamic policies for end-to-end sensorimotor learning”. In: *Advances in Neural Information Processing Systems* 33 (2020), pp. 5058–5069.

[4] Melike Baykal-Gürsoy. “Semi-Markov Decision Processes”. In: *Wiley Encyclopedia of Operations Research and Management Science* (2010).

[5] Christopher M Bishop and Nasser M Nasrabadi. *Pattern recognition and machine learning*. Vol. 4. 4. Springer, 2006.

[6] Joseph Campbell and Heni Ben Amor. “Bayesian interaction primitives: A slam approach to human-robot interaction”. In: *Conference on Robot Learning*. PMLR. 2017, pp. 379–387.

[7] Adrià Colomé and Carme Torras. “Demonstration-free contextualized probabilistic movement primitives, further enhanced with obstacle avoidance”. In: *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 2017, pp. 3190–3195. DOI: 10.1109/IROS.2017.8206151.

[8] Kourosh Darvish et al. “Whole-Body Geometric Retargeting for Humanoid Robots”. In: *2019 IEEE-RAS 19th International Conference on Humanoid Robots (Humanoids)*. 2019, pp. 679–686. DOI: 10.1109/Humanoids43949.2019.9035059.

[9] Felix Frank et al. “Constrained probabilistic movement primitives for robot trajectory adaptation”. In: *IEEE Transactions on Robotics* 38.4 (2021), pp. 2276–2294.

[10] Shixiang Gu et al. “Deep reinforcement learning for robotic manipulation”. In: *arXiv preprint arXiv:1610.00633* 1 (2016).

[11] Shixiang Gu et al. “Deep reinforcement learning for robotic manipulation with asynchronous off-policy updates”. In: *2017 IEEE International Conference on Robotics and Automation (ICRA)*. 2017, pp. 3389–3396. DOI: 10.1109/ICRA.2017.7989385.

[12] Tuomas Haarnoja et al. “Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor”. In: *International conference on machine learning*. PMLR. 2018, pp. 1861–1870.

[13] Kay Hansel et al. *Hierarchical Policy Blending as Inference for Reactive Robot Control*. 2022. DOI: 10.48550/ARXIV.2210.07890. URL: https://arxiv.org/abs/2210.07890.

[14] Martin Heusel et al. “Gans trained by a two time-scale update rule converge to a local nash equilibrium”. In: *Advances in neural information processing systems* 30 (2017).

[15] Yanlong Huang. “EKMP: Generalized imitation learning with adaptation, nonlinear hard constraints and obstacle avoidance”. In: *arXiv preprint arXiv:2103.00452* (2021).

[16] Yanlong Huang et al. “Kernelized movement primitives”. In: *The International Journal of Robotics Research* 38.7 (2019), pp. 833–852.

[17] Auke Ijspeert, Jun Nakanishi, and Stefan Schaal. “Learning attractor landscapes for learning motor primitives”. In: *Advances in neural information processing systems* 15 (2002).

[18] Auke Jan Ijspeert et al. “Dynamical movement primitives: learning attractor models for motor behaviors”. In: *Neural computation* 25.2 (2013), pp. 328–373.

[19] James M Joyce. “Kullback-leibler divergence”. In: *International encyclopedia of statistical science*. Springer, 2011, pp. 720–722.

[20] S Mohammad Khansari-Zadeh and Aude Billard. “Learning stable nonlinear dynamical systems with gaussian mixture models”. In: *IEEE Transactions on Robotics* 27.5 (2011), pp. 943–957.

[21] Dorothea Koert et al. “Demonstration based trajectory optimization for generalizable robot motions”. In: *2016 IEEE-RAS 16th International Conference on Humanoid Robots (Humanoids)*. IEEE. 2016, pp. 515–522.

[22] Leonidas Koutras and Zoe Doulgeri. “Dynamic Movement Primitives for moving goals with temporal scaling adaptation”. In: *2020 IEEE International Conference on Robotics and Automation (ICRA)*. 2020, pp. 144–150. DOI: 10.1109/ICRA40945.2020.9196765.

[23] An T. Le et al. *Hierarchical Policy Blending As Optimal Transport*. 2022. DOI: 10.48550/ARXIV.2212.01938. URL: https://arxiv.org/abs/2212.01938.

[24] Ge Li et al. “ProDMP: A Unified Perspective on Dynamic and Probabilistic Movement Primitives”. In: *IEEE Robotics and Automation Letters* (2023).

[25] Zhongyu Li et al. “Reinforcement Learning for Robust Parameterized Locomotion Control of Bipedal Robots”. In: *2021 IEEE International Conference on Robotics and Automation (ICRA)*. 2021, pp. 2811–2817. DOI: 10.1109/ICRA48506.2021.9560769.

[26] Lucia Liu et al. “Robot Navigation in Crowded Environments Using Deep Reinforcement Learning”. In: *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 2020, pp. 5671–5677. DOI: 10.1109/IROS45743.2020.9341540.

[27] Tobias Löw et al. “PROMPT: Probabilistic Motion Primitives based Trajectory Planning.” In: *Robotics: Science and Systems*. 2021.

[28] Guilherme Maeda et al. “Learning interaction for collaborative tasks with probabilistic movement primitives”. In: *2014 IEEE-RAS International Conference on Humanoid Robots*. IEEE. 2014, pp. 527–534.

[29] Guilherme J Maeda et al. “Probabilistic movement primitives for coordination of multiple human–robot collaborative tasks”. In: *Autonomous Robots* 41 (2017), pp. 593–612.

[30] Yutaka Matsuo et al. “Deep learning, reinforcement learning, and world models”. In: *Neural Networks* (2022).

[31] Goeffrey J McLachlan. “Mahalanobis distance”. In: *Resonance* 4.6 (1999), pp. 20–26.

[32] Volodymyr Mnih et al. “Asynchronous methods for deep reinforcement learning”. In: *International conference on machine learning*. PMLR. 2016, pp. 1928–1937.

[33] Janosch Moos et al. “Robust reinforcement learning: A review of foundations and recent advances”. In: *Machine Learning and Knowledge Extraction* 4.1 (2022), pp. 276–315.

[34] Meinard Müller. “Dynamic time warping”. In: *Information retrieval for music and motion* (2007), pp. 69–84.

[35] Kevin P Murphy. *Machine learning: a probabilistic perspective*. MIT press, 2012.

[36] Takayuki Osa et al. “An algorithmic perspective on imitation learning”. In: *Foundations and Trends® in Robotics* 7.1-2 (2018), pp. 1–179.

[37] Alexandros Paraschos, Gerhard Neumann, and Jan Peters. “A probabilistic approach to robot trajectory generation”. In: *2013 13th IEEE-RAS International Conference on Humanoid Robots (Humanoids)*. IEEE. 2013, pp. 477–483.

[38] Alexandros Paraschos et al. “Probabilistic movement primitives”. In: *Advances in neural information processing systems* 26 (2013).

[39] Alexandros Paraschos et al. “Using probabilistic movement primitives in robotics”. In: *Autonomous Robots* 42 (2018), pp. 529–551.

[40] Dae-Hyung Park et al. “Movement reproduction and obstacle avoidance with dynamic movement primitives and potential fields”. In: *Humanoids 2008-8th IEEE-RAS International Conference on Humanoid Robots*. IEEE. 2008, pp. 91–98.

[41] L. Penco et al. “Robust Real-Time Whole-Body Motion Retargeting from Human to Humanoid”. In: *2018 IEEE-RAS 18th International Conference on Humanoid Robots (Humanoids)*. 2018, pp. 425–432. DOI: 10.1109/HUMANOIDS.2018.8624943.

[42] Cristina Pinneri et al. “Sample-efficient Cross-Entropy Method for Real-time Planning”. In: *Proceedings of the 2020 Conference on Robot Learning*. Ed. by Jens Kober, Fabio Ramos, and Claire Tomlin. Vol. 155. Proceedings of Machine Learning Research. PMLR, 16–18 Nov 2021, pp. 1049–1065. URL: https://proceedings.mlr.press/v155/pinneri21a.html.

[43] Vignesh Prasad, Ruth Stock-Homburg, and Jan Peters. “Learning human-like hand reaching for human-robot handshaking”. In: *2021 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2021, pp. 3612–3618.

[44] Martin L Puterman. *Markov decision processes: discrete stochastic dynamic programming*. John Wiley & Sons, 2014.

[45] Deirdre Quillen et al. “Deep Reinforcement Learning for Vision-Based Robotic Grasping: A Simulated Comparative Evaluation of Off-Policy Methods”. In: *2018 IEEE International Conference on Robotics and Automation (ICRA)*. 2018, pp. 6284–6291. DOI: 10.1109/ICRA.2018.8461039.

[46] Stefan Schaal. “Dynamic movement primitives-a framework for motor control in humans and humanoid robotics”. In: *Adaptive motion of animals and machines* (2006), pp. 261–280.

[47] Stefan Schaal et al. “Control, planning, learning, and imitation with dynamic movement primitives”. In: *Workshop on Bilateral Paradigms on Humans and Humanoids: IEEE International Conference on Intelligent Robots and Systems (IROS 2003)*. 2003, pp. 1–21.

[48] John Schulman et al. “Proximal policy optimization algorithms”. In: *arXiv preprint arXiv:1707.06347* (2017).

[49] John Schulman et al. “Trust region policy optimization”. In: *International conference on machine learning*. PMLR. 2015, pp. 1889–1897.

[50] Muhammet Yunus Seker et al. “Conditional Neural Movement Primitives.” In: *Robotics: Science and Systems*. Vol. 10. 2019.

[51] Claude Elwood Shannon. “A mathematical theory of communication”. In: *ACM SIGMOBILE mobile computing and communications review* 5.1 (2001), pp. 3–55.

[52] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

[53] You Zhou, Jianfeng Gao, and Tamim Asfour. “Learning Via-Point Movement Primitives with Inter- and Extrapolation Capabilities”. In: *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 2019, pp. 4301–4308. DOI: 10.1109/IROS40897.2019.8968586.

A. Appendix

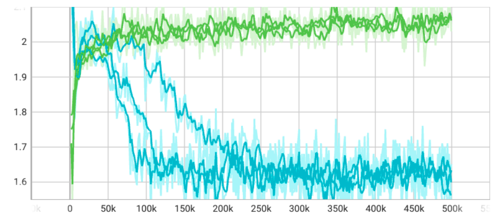
A.1. Hyperparameters of SAC

Table A.1.: Hyperparameters

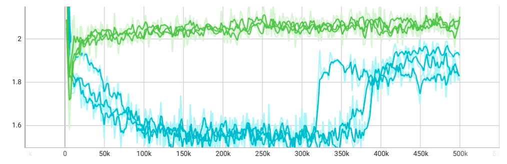
Hyperparameter	**Value**
learning rate	0.0003
buffer size	$1,000,000$
learning starts	100
batch size	256
ent coef	auto
gamma	0.99
tau	0.005
target entropy	auto
train freq	1
gradient steps	1
number of traning steps	$500,000$

The target entropy is set based on experimental observations and is the reciprocal of the number of the dimension of the action space. In our experiments, since the action space has a dimension of 4, the target entropy is set to 0.25. The entropy coefficient (ent coef) is automatically adjusted during training, and its initial value is set to 1.

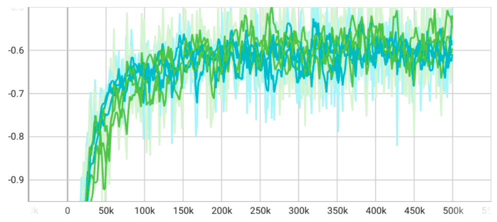
A.2. Learning Curve



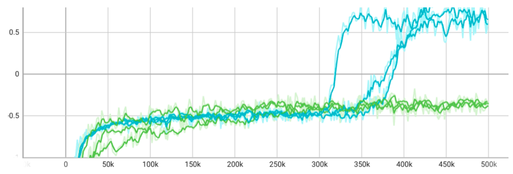
(a) Length of Episode of LASA Angle



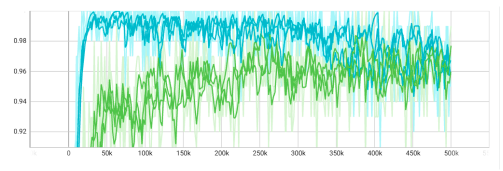
(b) Length of Episode of LASA Bended Line



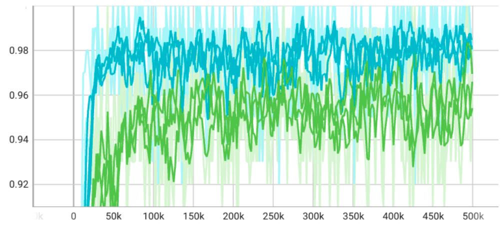
(c) Episodic reward of LASA Angle



(d) Episodic reward of LASA Bended Line

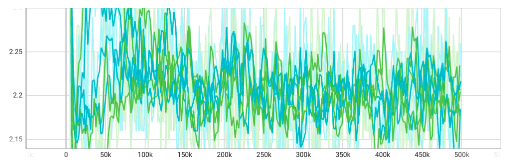


(e) Success rate of LASA Angle

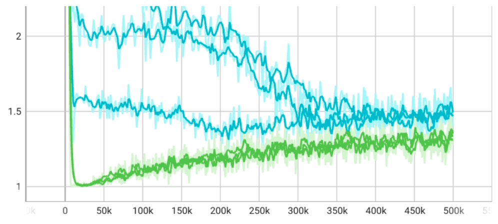


(f) Success rate of LASA Bended Line

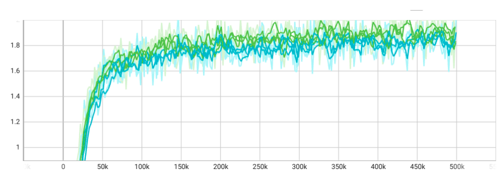
Figure A.1.: Learn curve of LASA Angle with ReLU phase function and Bendline data with sigmoid phase function. The blue curve in the graph represents the results of VMP, while the green curve represents the results of ProMP, each with three sets of random number seeds.



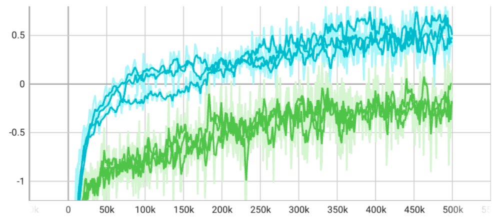
(a) Length of Episode of l-shape



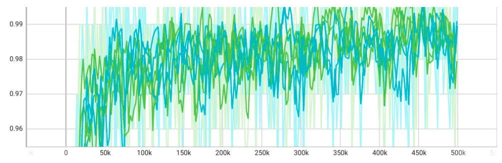
(b) Length of Episode of LASA Sine



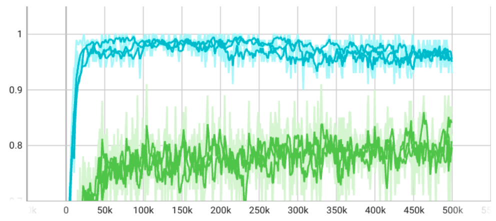
(c) Episodic reward of l-shape



(d) Episodic reward of LASA Sine



(e) Success rate of l-shape



(f) Success rate of LASA Sine

Figure A.2.: Learn curve of l-shape with ReLU phase function and LASA Sine data with exponential phase function. The blue curve in the graph represents the results of VMP, while the green curve represents the results of ProMP, each with three sets of random number seeds.