Collaborative Learning with Dirichlet Process Clustering for Rapid Online Adaptation in Robotics

Runjun Mao
August 12, 2023
Supervised by Antoine Cully

***Abstract*—Robots in deployment can face unforeseeable distortion in their motion due to unseen environment or hardware failures. Model-based online adaptation learns this distortion from the interactions with the current environment and corrects its motions accordingly. However, this can be very challenging as it is hard to select the most suitable model and its hyper-parameters for the current dynamics with little prior knowledge. If we could leverage the plentiful historical real-world interactions, we may build statistical models for the common types of dynamics faced by the robot, hence providing guidance for adaptations. This paper developed a method to enable fast online adaptation for the robot by grouping up historical interaction data into clusters. The historical data can be collected from several robots of the same design performing different tasks in different environments, hence allowing learning in a collaborative manner. The method takes an non-parametric approach of novelty by modelling the data distribution with infinite mixture of Gaussian Processes and conducting clustering using Dirichlet Processes. This achieves unbiased training while ensures high efficiency in terms of both data and computation during deployment. The experiment is conducted on a simulated robot to compare with the previous work for online adaptation. The method accurately groups up the collaboratively collected data without any knowledge of their sources, and it offers a significant improvement over the baseline in both the efficiency of learning the distortion and the capability of adaptation.**

I. Introduction

Locomotion controls of legged robots are typically too complex to be designed manually. In recent years, deep Reinforcement-Learning (deep RL) has made huge success in game playing [1], [2] and is regarded promising for application in robotics. However, training such well performing agents requires enormous amounts of interactions with the environment, meaning that this can only be done in simulation where we can model the interaction and collect the training data much faster than in real-time. This raises a problem for application in robotics since there is always a gap between simulation and reality, and the policy trained in simulation could suffer from this sim-to-real transfer. Another problem is that any unforeseen changes to the environment dynamics could lead to severely degraded performance. These deep RL trained agents require extra fine-tuning or even retaining to retain their capabilities. For example, the team of OpenAI Five [3] had to conduct three additional amendments in the architecture followed by extra trainings to incorporate small version updates in Dota 2 during the development of the agent. For game-playing agents, these drawbacks are tolerable as there is no sim-to-real problem, and the environment is overall stationary. For robotics however, sim-to-real must be considered, and the environment dynamics is typically non-stationary as the robot can be travelling between different terrain, carrying different payloads or suffering from damages in its hardware.

It would be better if we may learn the true dynamics from the interactions with the current dynamics and correct our policy accordingly. This is a model-based reinforcement learning (MBRL) solution that allows robots to learn policies with lesser interactions by learning a dynamics model and use it optimize the policy [4]–[6]. However, the amount of data required to learn a model typically scales exponentially with the dimensionality of the input space [7], hence it is not suitable for online adaptation. The promising approach to address this is to use a repertoire-based method, where we learn a repertoire of elementary policies (e.g., one policy for turning left, one for moving forward, etc.) using quality diversity optimisation [8] to fully cover the task space (e.g., the 2D displacement we want the robot to make). The repertoire-based method is a hierarchical control method that keeps a variety of elementary policies and treats each policy as an action. We then give each elementary policy a behaviour descriptor (BD) to quantify the way it acts. This enables us to skip the low-level states and actions (i.e., joint encoders and torques for all motors) and to work with the high-level behaviour space that has much smaller dimensionality. During the adaptation, we learn the distortion (called transformation model) in the behaviour space from real-world interactions. Thanks to the reduced dimensionality, the transformation model can be learned in real-time. Instead of optimising the policy with this learned model, we use a repertoire-based control that first predicts the outcomes for each elementary policy in the repertoire and then simply chooses the one with the predicted outcome that is most aligned with the current goal. This is made available for keeping a repertoire of policies rather than a single policy, so that the robot can adapt to the distortion by finding the corresponding policies that compensate the distortion (for example the robot aims to go forward and distortion is left, then a policy originally aiming right-front can be used for going forward instead).

The repertoire-based method has achieved very impressing results in task-solving [9] and online damage recovery [10]. But this method heavily relies on having a good model that is both data efficient and suitable for modelling the distortion. The most commonly used model is Gaussian Process (GP) [11]. The prior mean function and the kernel are the most

important factors of GP. If no prior knowledge is given about the dynamics, these have to be selected manually according to experience. If the distortion is very large (like broken legs) or complex, the GPs can be very inefficient or even misleading. If we have data of real-world interactions across several different environments, we can build a GP for each of the situations and then teach the robot to identify the one that mostly explains its current situation. However, such real-world data are expensive to collect and could lead to covariate shift (the collected data distribution is different from that during deployment). A promising solution that is to leverage the historical data of real-world interactions collected during deployment. After performing each task, the interaction data can be uploaded to an archive system where they will be analysed. Thus, the data doesn't have to be collected on purpose, and the acquired data distribution is completely unbiased. This also allows the robots to learn to adapt in a collaborative manner. If we have multiple robots performing different tasks in different environments, we can quickly get an overview of the common types of dynamics and build the adaptation strategy accordingly. If a robot encounters a new environment, with the interaction data uploaded, all the robots will be able to quickly adapt to this environment. Finally, considering the scalability of such system, we can group the data into clusters based on their dynamics and assign a GP (with the corresponding mean and kernel) to each cluster. This prevents the adaptation strategy from becoming computationally expensive after we have collected data from thousands of tasks.

There are many challenges needs to be addressed to enable this collaborative learning. First, traditional clustering algorithms like k-nearest neighbours [12] and k-means [13] need to specify the number of clusters. While in our case, this number is not even fixed as we might observe more clusters as we collect more data. Second, we know that each task corresponds to a single environment, and hence we are essentially clustering the tasks instead of clustering the data points. However, since some tasks are tougher than others, different tasks generate different amount of data. As a result, we will be clustering data of inconsistent sizes. Third, since there is a large repertoire of policies, only a small subset of them will be executed while performing each task. Hence, it is very likely that the data collected from two tasks have no overlapping policies, making it hard to determine whether they have similar dynamics and whether they should be grouped together. Moreover, after we have found the clusters, the data in each cluster can never cover the entire repertoire. Hence it is difficult to determine the prior mean for the policies that are not present in this cluster. This paper aims to solve all the above problems by taking an non-parametric clustering method and using a uniquely designed prior mean function.

II. Related Works

A. Quality-Diversity Optimization

The repertoire-based method uses quality diversity (QD) optimization to generate a repertoire of diversified and high-quality locomotion behaviours. The most popular method for doing QD is called MAP-Elites [8], [14], where we tabularize the task-space (for example, discretize the displacement of the robot along the x, y axis) into grids. Each grid corresponds to a policy that leads the robot to this part of task space. Initially the grids are empty, as we have no policies at start, and will be filled during a process of stochastic optimization inspired by natural evolutionary. We start from a few random policies, which are typically neural networks, and execute them in the simulator to get their outcomes in the task space. We also assign a performance score (called fitness) to each policy to quantify the efficiency of the execution. Typically, the fitness can be based on the energy consumption [15] or the shape of the trajectory [16]. Each of these outcomes is mapped to one of the girds (hence these grids are filled). We then begin our main loop of policy generation. In each iteration we select a few policies corresponding to the filled grids as the parents. We then conduct crossover on the parents to mutate their genotypes (for neural networks, these are just the weights of the neurons) to get their off-springs. The crossover operation should balance the exploitation and exploration by ensuring some inheritance of the parents to preserve good genes as well as some random variations for promoting diversity. We then evaluate the off-springs in the simulator to get their outcomes in the task space and identify the grids they belong to. If a grid is already occupied by another policy, we only keep the one with the higher fitness; if it is empty, we fill this grid with the corresponding off-spring. The pseudo code for MAP-Elites QD algorithm is presented below.

Algorithm 1 MAP-Elites

```
procedure MAP-Elites
Discretize the task space into grids
R ← ∅                                          ▷ Initialize repertoire
for i = 1 → G do
  θ ← generate_random_policy()
  Θ, f ← evaluate_policy(θ)                    ▷ Get outcome and fitness
  add_to_repertoire(Θ, f, θ, R)                ▷ Update repertoire
end for
for i = 1 → T do
  ϑ ← select_parents(R)
  θ ← crossover(ϑ, N)                          ▷ Generate N off-springs
  for j = 1 → N do
    Θ, f ← evaluate_policy(θ_j)                ▷ Off-spring evaluation
    add_to_repertoire(Θ, f, θ_j, R)
  end for
end for
```

Evaluating the policies in the simulation is the major source of computationally expense for the MAP-Elites. A merit of this algorithm is that the inner for loop of off-spring evaluation can be conducted in parallel, and the repertoire update can be made after having collected all the results of the off-springs. This enables us to generate a large number of off-springs in each iteration and evaluate them in parallel, fully leveraging the power of modern Cluster computing [15]. By running the algorithm for a few thousand iterations, we will end up having a repertoire of high-performing policies well-covering the task space.

B. Gaussian Process

The Gaussian Process (GP) is a powerful non-parametric machine learning algorithm. It assumes the prior distribution of the outcomes to be a multivariate normal distribution with a prior mean μ and a covariance matrix Σ.

$$f(\boldsymbol{y}) = \frac{exp(-\frac{1}{2}(\boldsymbol{y}-\boldsymbol{\mu})^T\boldsymbol{\Sigma^{-1}}(\boldsymbol{y}-\boldsymbol{\mu}))}{\sqrt{(2\pi)^N det(\boldsymbol{\Sigma})}} \tag{1}$$

In contrast to regular multivariate normal distribution, GP allows the prior mean and the covariance matrix to be input dependent. The prior mean becomes a function $\mu(\boldsymbol{x})$, and the covariance between two data points is given by a covariance function $k(\boldsymbol{x_1}, \boldsymbol{x_2})$ called kernel. Following the fact that data points closer to each other are more correlated, this covariance starts from the prior variance and asymptotically decreases to zero as the distance between the two points tends to infinity. To account for the fact the observations might be noisy, we can incorporate the noise into the covariance matrix by adding the variance of the noise on the diagonal.

$$\boldsymbol{\Sigma} = \boldsymbol{K}(\boldsymbol{x_{1:N}}, \boldsymbol{x_{1:N}}) + \sigma_n^2\mathbf{I} \tag{2}$$

This allows us to build reasonable joint prior distribution for the values of any well-behaved functions. To make prediction with GP, we first use the prior mean function and kernel to build up the joint distribution as the prior. Then we can calculate the posterior for the data we wish to evaluate conditioned on the ones we have observed using Bayes rule. Since the Gaussian distribution is a conjugate distribution, the posterior is still a Gaussian with mean and variance given by [11], [17]:

$$\mu_*(\boldsymbol{x}) = \mu(\boldsymbol{x}) + \boldsymbol{\Sigma}_{\boldsymbol{N},\boldsymbol{x}}^T\boldsymbol{\Sigma}_{\boldsymbol{N},\boldsymbol{N}}^{-1}(\boldsymbol{y_{1:N}} - \boldsymbol{\mu}(\boldsymbol{x_{1:N}})) \tag{3}$$

$$\sigma_*^2(\boldsymbol{x}) = \sigma^2(\boldsymbol{x}) - \boldsymbol{\Sigma}_{\boldsymbol{N},\boldsymbol{x}}^T\boldsymbol{\Sigma}_{\boldsymbol{N},\boldsymbol{N}}^{-1}\boldsymbol{\Sigma}_{\boldsymbol{N},\boldsymbol{x}} \tag{4}$$

where $\boldsymbol{\Sigma}_{\boldsymbol{N},\boldsymbol{N}}$ denotes the covariance matrix of the observed data, and $\boldsymbol{\Sigma}_{\boldsymbol{N},\boldsymbol{x}}$ denotes the column vector with entry on each row equal to the covariance between each observed data and the data we hope to evaluate.

Typically, the prior mean is just a fixed constant like zero, representing the prior knowledge we have about the outcome when there is no neighbouring data to refer to. The typical choice of kernel can be RBF kernel that models the correlations to decrease in the form of a Gaussian function.

$$k(\boldsymbol{x_i}, \boldsymbol{x_j}) = \alpha_0 e^{-\frac{1}{2}d_{ij}^2} \tag{5}$$

RBF kernel assumes the function is infinitely differentiable, which may be factually incorrect. Hence, it might be more desirable to use the Matern kernel:

$$k(\boldsymbol{x_i}, \boldsymbol{x_j}) = \alpha_0 \frac{2^{1-v}}{\Gamma(v)}(\sqrt{2v}d_{ij})^v K_v(\sqrt{2v}d_{ij}) \tag{6}$$

where the Γ is the gamma function, and the K_v is the modified Bessel function of the second kind. The v controls the smoothness, as GP using Matern kernel is $\lceil v \rceil - 1$ times differentiable in the mean-square sense [11], [18]. The distance d_{ij} between two data points does not have to be Euclidean distance. A more general form would be:

$$d_{ij}^2 = \Sigma_{d=1}^D \frac{(x_{d,i} - x_{d,j})^2}{l_d^2} \tag{7}$$

Where this distance metric can be anisotropic, suggesting distance along some axis contributes more uncertainty than others. The l_d in the denominator is a positive number called length scale that determines how slowly we lose uncertainty (bigger length scale means the function is flatter) along the d^{th} axis.

C. Reset-free Trial-and-Error Learning

The work of this paper is based on the Reset-free Trial-and-Error [10] (RTE) algorithm. The RTE uses GP to learn the transformation model online. This is a very clever choice for two reasons. First, the online learning requires very high data-efficiency, which happens to be one of the strengths of GP. Secondly, the true dynamics happens at low level, while the repertoire-based method learns the transformation model at high level. This will inevitably introduce errors, which can be well incorporated by GP as being a probabilistic model. In RTE, the task space is just the displacement of the robot in x and y direction after executing each policy. The prior mean function of RTE is chosen to be the outcomes of the policies evaluated in the simulator, which are the results collected during the MAP-Elites policy generation. Since the policy outcome has two dimensions (x and y), RTE uses a different GP for each dimension. The kernel is chosen to be the RBF kernel with an isotropic distance metric. The input space is chosen to be the same as the task space, namely the x and y displacement evaluated in the simulation. During online adaptation, the RTE first predicts the outcome of each policy in the repertoire using Eq. 3 and Eq. 4, then it uses a Monte Carlo Tree Search [19] (MCTS) to plan its actions. It is worth noting that the RTE uses a periodic set of commands as the elementary policy. This means the policy execution is open-loop control, despite the robot knows its position and orientation. While using such periodic controllers seems to be weaker than using a neural network, it is much more stable than using neural networks. This is because a periodic controller performs all its actions during evaluation, while neural networks are much more complicated and could behave very differently in different situations. Since the repertoire based control aims to learn the distortion with limited number of interactions, the distortion cannot be too large or too complicated. Hence the use of periodic controller is a much better choice.

This method is not very ideal for three reasons. First, the use of outcomes in default simulation setting as the prior mean is not good enough. Since some conditions like broken leg can lead to very large distortions. In this case, such prior would be misleading. Second, the use of the task space as the input space is not ideal. Two policies leading to very close displacements could follow very different trajectories, hence they may react very differently to the same new environment. Also, the kernel in the RTE is manually chosen, which depends

on experiences of the designer and could lead to suboptimal results. Nevertheless, this method successfully enabled a damaged hexapod robot to recover 77.52% of its capability [10] and significantly outperforms the baseline result that does not use the GP as the transformation model. Some other works like Adaptive Prior selection for Repertoire-based Online Learning [20] (APROL) take very similar approach as RTE but use several repertoires generated in different environments to address potential large distortions. Since RTE is easier to implement, and any improvements on the RTE can be easily transferred to other other works, it is selected as the algorithm that our work is based on.

D. Dirichlet Process Mixture Model

Dirichlet process mixture model (DPMM), also called infinite mixture model, is a Bayesian non-parametric model widely used in data clustering. It is based on a stochastic process called Dirichlet Process (DP). DP assumes that the data are sampled from a distribution of distributions. For example, the data points come from a set of Gaussian distributions, while the mean and variance of each Gaussian follow another distribution. To make clear explanation, we first investigate the finite mixture model using DP, and then extend the derivations to the infinite limit. We know that finite mixture models can be represented as:

$$p(\boldsymbol{y}) = \Sigma_{j=1}^{k} \pi_j \cdot p(\boldsymbol{y}|\boldsymbol{\theta_j}) \tag{8}$$

where this model consists of k components, and $\boldsymbol{\theta_j}$ and π_j are the parameters and the mixture weights (also called mixing proportions) for each mixture component. DP assumes the parameters and the mixture weights are independently sampled. The parameters of the mixture components are sampled from a base distribution H, and the mixture weights are sampled from a symmetric Dirichlet distribution [21]:

$$p(\pi_1, \pi_2, \ldots, \pi_k|\alpha) = \frac{\Gamma(\alpha)}{\Gamma(\alpha/k)^k} \prod_{j=1}^{k} \pi_j^{\alpha/k-1} \tag{9}$$

Where the mixture weights are positive and sum up to 1. The constant α is called concentration parameter, which controls how dense the sampling will be. To train a DPMM, the only thing we need to determine is the indicator of each data which points to the mixture component that this data belongs to. We then try to find the configuration of the indicators that maximizes the posterior likelihood (MAP estimate). This is very hard to do directly, typical alternative approaches include Gibbs sampling [22], [23] and the use of variational inference [24]. This paper uses the Gibbs sampling method. In each iteration, we loop through each indicator and resample it based on the other indicators:

$$\begin{gathered} p(c_i = j|y_i, \boldsymbol{y}_{-i}, \boldsymbol{c}_{-i}, \alpha) \propto \\ p(c_i = j|\boldsymbol{c}_{-i}, \alpha) \cdot p(y_i|c_i = j, \boldsymbol{y}_{-i}, \boldsymbol{c}_{-i}) \end{gathered} \tag{10}$$

The subscript $-i$ means all the data except for i. We know the probability of getting a certain configuration is:

$$p(c_1, \ldots, c_n|\pi_1, \ldots, \pi_k) = \prod_{j=1}^{k} \pi_j^{n_j} \tag{11}$$

Where the n_j in the superscript denotes the number of data assigned to the j^{th} component. Using Eq. (9) and standard Dirichlet integral, we can calculate the probability density of such configuration in the prior:

$$\begin{aligned} p(c_1, c_2, \ldots, c_n|\alpha) &= \frac{\Gamma(\alpha)}{\Gamma(\alpha/k)^k} \int \prod_{j=1}^{k} \pi_j^{n_j+a/k-1} d\boldsymbol{\pi} \\ &= \frac{\Gamma(\alpha)}{\Gamma(\alpha+n)} \prod_{j=1}^{k} \frac{\Gamma(n_j+\alpha/k)}{\Gamma(\alpha/k)} \end{aligned} \tag{12}$$

Hence we can find the posterior using Bayes rule:

$$p(c_i = j|\boldsymbol{c}_{-i}, \alpha) = \frac{n_{-i,j} + \alpha/k}{\alpha + n - 1} \tag{13}$$

So far, we have been discussing the case of finite mixture models. If we let the component number k tend to infinity, we will have the equations for infinite mixture models:

$$\begin{gathered} p(c_i = j|\boldsymbol{c}_{-i}, \alpha) = \frac{n_{-i,j}}{\alpha + n - 1} \\ p(c_i \neq j \text{ for any } n_j \neq 0|\boldsymbol{c}_{-i}, \alpha) = \frac{\alpha}{\alpha + n - 1} \end{gathered} \tag{14}$$

For any existing cluster (mixture component with at least one data assigned to it), this probability is proportional to the number of data in that cluster. Note that there is a non-zero probability that this data belongs to a new cluster. This is a very important property of DPMM that new clusters can be automatically generated based on the likelihood. To calculate the last term in Eq. (10), DP takes a full Bayesian approach:

$$\begin{gathered} p(y_i|c_i = j, \boldsymbol{y}_{-i}, \boldsymbol{c}_{-i}) = \int p(y_i|\boldsymbol{\theta}_j) p(\boldsymbol{\theta}_j|\boldsymbol{y}_{-i}, \boldsymbol{c}_{-i}) d\boldsymbol{\theta}_j \\ \text{where} \quad p(\boldsymbol{\theta}_j|\boldsymbol{y}_{-i}, \boldsymbol{c}_{-i}) = \frac{1}{Z} p(\boldsymbol{\theta}_j) \prod_{c_k=j} p(y_k|\boldsymbol{\theta}_j) \end{gathered} \tag{15}$$

where the Z is a normalization factor. In the case of a new cluster, the integration is made over the prior:

$$p(y_i|c_i \neq j \text{ for any } n_j \neq 0) = \int p(y_i|\boldsymbol{\theta}) p(\boldsymbol{\theta}) d\boldsymbol{\theta} \tag{16}$$

To conduct Gibbs sampling, we first start from an initial configuration of indicators (common choice is that every data is a cluster on its own). Then in each iteration, we loop through each indicator and use Eq. (10) to resample it. The $-i$ subscript means we need to remove the data from its current cluster during resampling, which will affect the result of Eq. (15) when calculating the probability of the data remaining in its previous cluster. The Gibbs sampling is a Markov Chain Monte Carlo (MCMC) that satisfies irreducibility, positive recurrence and aperiodicity, hence it will eventually converge to the equilibrium distribution regardless of its initial state [25]. Since we are aiming to maximize the posterior likelihood, we need to calculate the likelihood after each iteration and record the configuration with the highest value. The posterior likelihood is given by:

$$p(\boldsymbol{c}|\boldsymbol{y}) \propto \left[\prod_{n_j \neq 0} \alpha \Gamma(n_j) \right] \prod_{i=1}^{n} p(y_i|c_i = j, \boldsymbol{y}_i, \boldsymbol{c}_{-i}) \tag{17}$$

where the first product is made over each cluster containing at least one data. Note that the probability in the second product doesn't have the $-i$ subscript like in Eq. (15), so the data doesn't need to be removed from its current cluster now.

We also need to define the base distribution H that gives the prior distribution $p(\boldsymbol{\theta})$ and the concentration parameter α which controls the generation of new clusters. Note that under the assumptions of DP, the final data distribution is a biased sample from H unless α tends to infinity. Hence we don't need to ensure our H being close to data distribution, while a rather conservative distribution is encouraged. The α is selected based on Eq. (14). We can see that the probability of getting a new cluster is proportional to α and decreases asymptotically to zero as the number of data increases. This is reasonable as the more data we have, the more certain we are that we have sampled at least one data from each cluster, hence the lesser we need a new cluster. If the α is infinite, DP believes all mixture component has the equal weight, and the data should always belong to a new situation regardless how many data we have collected. In practice, we can estimate the amount of data we need in order to fully cover all situations. Then we can find the corresponding α using Eq. (14) so that the probability decreases to a small value (e.g. 1%) after we have collected this number of data.

DPMM is a very powerful tool to make non-parametric clustering without specifying the number of clusters. It can also easily incorporate the infinite limit than approaches working with finite models of unknown sizes like [26]. Comparing with optimization based methods like EM [27], the use of MCMC can easily overcome the local optimal [28]. For example, if there are duplicated mixture components, EM will still converge but the DPMM will eventually merge them as the Eq. (17) favours the data to be concentrated. Note that the above derivations of DPMM did not put any constrain on the type of distribution, namely the term $p(y|\boldsymbol{\theta})$ in Eq. (15). Such distribution can also be non-parametric distributions like Gaussian Processes. This means we can use DP to cluster the dynamics in RTE, which is GP, to build a mixture model of the real-world dynamics.

III. Methodology

To formulate the problem clearly, we are aiming to leverage the collaboratively collected real-world interactions to assist a robot deployed in an unknown environment to quickly adapt and complete its task. We are provided with some episodes (each task-solving process corresponds to an episode of interaction data) of real-world interactions collected by several robots deployed in an unknown distribution of environments. The most intuitive idea is to label these historical data with their sources (like collected from grass terrain, collected with left leg broken, etc.) and build a simple database in the cloud server. During the adaptation, the robot examines the current environment and uploads the result to the database server. The server then identifies the most similar environment recorded in the database and let the robot download the corresponding data to assist the its adaptation. However, accurately measuring and describing the environment can be expensive and difficult. It is also hard to determine whether a previously encountered environment shares similar dynamics with the current one even if we have accurate descriptions for them. In the case we do know that some historical data come from exactly the same environment as the current one, these data might be insufficient in quantity to assist the current task. Hence, it is reasonable to group the episodes resulting from similar dynamics into clusters, ensuring having enough data for analysis. We will assume that we don't have access to any information of the environment and aim to conduct clustering only according to the dynamics.

Previous works of clustering GP via DP have been made in [29], [30]. But they work with individual data points, while in our case all the interaction data collected from the same task share the same indicator. In this paper, we introduced a novel method to cluster batched data with infinite mixture of GPs while ensuring high flexibility.

There are a few assumptions needs to be made for further analysis. The first assumption is that the simulated results always correspond to real-world outcomes in some environment. For example, we set the friction coefficient to 0.9 in the simulation, but the it underestimated friction and the simulated outcomes are very close to the real-world results for friction coefficient equal to 0.8. Hence we believe that although the simulated results are not accurate, they are still close to some real-world results. This is not always true, but it plays a very important role in the methodology.

$$we\ have\ talked\ about\ prior\ mean\ importance$$

$$we\ have\ talked\ about\ simulation\ assumptions$$

A. Linear Prior Mean Function

The prior mean function gives a prior estimate of the dynamics for a certain environment. This means we need to estimate the outcomes of every elementary policy in this environment. The dynamics can efficiently expressed in the following form:

$$x_d(\boldsymbol{\beta}, \boldsymbol{\epsilon}) = f_d(\boldsymbol{\beta}, \boldsymbol{\epsilon}) + g_d(\boldsymbol{\beta}) + h_d(\boldsymbol{\epsilon}) + c_d \tag{18}$$

where the x_d stands for the d^{th} dimension of the policy rollout outcome (for example, this can be the robot displacement in the x axis). Vector $\boldsymbol{\beta}$ represents the behaviour descriptor of the controller; vector $\boldsymbol{\epsilon}$ is the environment descriptor, representing some quantities of the environment. It can be seen from Eq. (18) that the dynamics consists of a term that depends on both the policy behaviour and the environment plus the terms that only depends on either the behaviour or environment plus a constant c_d. Note that we do not require access to the $\boldsymbol{\beta}$ and $\boldsymbol{\epsilon}$ or knowing their detailed meanings. Eq. (18) is generally non-linear. But in the case where the dynamics is relatively simple, like the use of periodic controller in RTE, we can get an approximation of it by Taylor expanding the $f_d(\boldsymbol{\beta}, \boldsymbol{\epsilon})$ term to the second order:

$$\begin{aligned} f_d(\boldsymbol{\beta}, \boldsymbol{\epsilon}) &\approx f_d(\boldsymbol{\beta}_0, \boldsymbol{\epsilon}_0) + \begin{bmatrix} J_{\boldsymbol{\beta}_0} & J_{\boldsymbol{\epsilon}_0} \end{bmatrix} \begin{bmatrix} \Delta\boldsymbol{\beta} \\ \Delta\boldsymbol{\epsilon} \end{bmatrix} \\ &+ \frac{1}{2} \begin{bmatrix} \Delta\boldsymbol{\beta}^T & \Delta\boldsymbol{\epsilon}^T \end{bmatrix} \begin{bmatrix} H_{\boldsymbol{\beta}_0\boldsymbol{\beta}_0} & H_{\boldsymbol{\beta}_0\boldsymbol{\epsilon}_0} \\ H_{\boldsymbol{\epsilon}_0\boldsymbol{\beta}_0} & H_{\boldsymbol{\epsilon}_0\boldsymbol{\epsilon}_0} \end{bmatrix} \begin{bmatrix} \Delta\boldsymbol{\beta} \\ \Delta\boldsymbol{\epsilon} \end{bmatrix} \end{aligned} \tag{19}$$

where the Jacobian vector J and the Hessian matrix H have been arranged in the partitioned form. We then substitute this result into Eq. (18) to arrive at:

$$\begin{aligned} x_d(\boldsymbol{\beta},\boldsymbol{\epsilon}) \approx{}& \Delta\boldsymbol{\beta}^T H_{\boldsymbol{\beta}_0\boldsymbol{\epsilon}_0}\Delta\boldsymbol{\epsilon} + f_d(\boldsymbol{\beta}_0,\boldsymbol{\epsilon}_0) + c_d \\ &+\frac{1}{2}\Delta\boldsymbol{\beta}^T H_{\boldsymbol{\beta}_0\boldsymbol{\beta}_0}\Delta\boldsymbol{\beta} + J_{\boldsymbol{\beta}_0}\Delta\boldsymbol{\beta} + g_d(\boldsymbol{\beta}) \\ &+\frac{1}{2}\Delta\boldsymbol{\epsilon}^T H_{\boldsymbol{\epsilon}_0\boldsymbol{\epsilon}_0}\Delta\boldsymbol{\epsilon} + J_{\boldsymbol{\epsilon}_0}\Delta\boldsymbol{\epsilon} + h_d(\boldsymbol{\epsilon}) \end{aligned} \tag{20}$$

Comparing Eq. (20) with Eq. (18), we can see our approximation factorizes the correlated term. If the same policy is executed in two different environment, the difference in the dynamics will be:

$$\begin{aligned} x_d(\boldsymbol{\beta},\boldsymbol{\epsilon}_1) - x_d(\boldsymbol{\beta},\boldsymbol{\epsilon}_2) \approx{}& \Delta\boldsymbol{\beta}^T H_{\boldsymbol{\beta}_0\boldsymbol{\epsilon}_0}(\boldsymbol{\epsilon}_1 - \boldsymbol{\epsilon}_2) \\ &+h_d^*(\boldsymbol{\epsilon}_1) - h_d^*(\boldsymbol{\epsilon}_2) \\ \text{where } h_d^*(\boldsymbol{\epsilon}) ={}& \frac{1}{2}\Delta\boldsymbol{\epsilon}^T H_{\boldsymbol{\epsilon}_0\boldsymbol{\epsilon}_0}\Delta\boldsymbol{\epsilon} + J_{\boldsymbol{\epsilon}_0}\Delta\boldsymbol{\epsilon} + h_d(\boldsymbol{\epsilon}) \end{aligned} \tag{21}$$

If we treat the $h_d^*(\boldsymbol{\epsilon})$ term as an additional dimension of the environment vector, then Eq. (21) can be represented as:

$$\begin{aligned} \Delta\boldsymbol{x}_d = \begin{bmatrix} \Delta x_{d,1} \\ \vdots \\ \Delta x_{d,N} \end{bmatrix} &\approx \begin{bmatrix} \Delta\boldsymbol{\beta}_1^T H_{\boldsymbol{\beta}_0\boldsymbol{\epsilon}_0} & 1 \\ \vdots & \\ \Delta\boldsymbol{\beta}_N^T H_{\boldsymbol{\beta}_0\boldsymbol{\epsilon}_0} & 1 \end{bmatrix}\Delta\boldsymbol{\varepsilon} \\ &= W_d^T\Delta\boldsymbol{\varepsilon} \end{aligned} \tag{22}$$

where $\boldsymbol{\varepsilon}$ is our new environment vector. The vector $\Delta\boldsymbol{x}_d$ stands for the difference in dynamics for the entire repertoire with each entry $\Delta x_{d,i}$ denoting the result for the each elementary policy. Combined with the assumption we made about the simulation, Eq. (22) suggests we may approximate the real-world distortion of the entire repertoire using a linear model.

Although this is a very potent result, we still do not have access to the matrix W_d or the environment vector $\boldsymbol{\varepsilon}$. Instead, we can leverage the fact that any environment vector can be linearly represented by a set of environment vectors as the basis. Hence, if we have collected enough distortions, we can represent any new distortion with:

$$\begin{aligned} \Delta\boldsymbol{x}_d &\approx W_d^T\left[\Delta\boldsymbol{\varepsilon}^{(1)},\cdots,\Delta\boldsymbol{\varepsilon}^{(n)}\right]\begin{bmatrix}\omega_1\\ \vdots \\ \omega_n\end{bmatrix} \\ &= \left[\Delta\boldsymbol{x}_d^{(1)},\cdots,\Delta\boldsymbol{x}_d^{(n)}\right]\begin{bmatrix}\omega_1\\ \vdots \\ \omega_n\end{bmatrix} \end{aligned} \tag{23}$$

This means we need to get the distortions of all the elementary policies in several environments. Since typically a repertoire can contain of a few thousand policies, we cannot afford to do such amount of evaluations in the real-world, and hence we need to rely on simulation.

A question is that how many distortion we should collect. If we get too few distortions, we cannot cover the entire space, and the linear model will perform poorly in uncovered environments. On the other hand, if we have too many, it will be computationally expensive and lead to very large dimension. The solution is collect as many as we can and then reduce the dimensionality with singular vector decomposition (SVD). The collection of distortions can be represented as:

$$\boldsymbol{A} = \left[\Delta\boldsymbol{x}_d^{(1)},\cdots,\Delta\boldsymbol{x}_d^{(n)}\right] = \sum_{i=1}^{s}\sigma_i\boldsymbol{v}_i\boldsymbol{u}_i^T \tag{24}$$

where $\boldsymbol{v}_i$ stands for each normalized base vector and $\boldsymbol{u}_i$ is the normalized component of this base vector in each distortion. The number of basis s is much smaller than n as we have collected more than we need to ensure a good coverage. To conduct SVD, we left multiply matrix $\boldsymbol{A}$ by its transpose:

$$\begin{aligned} \boldsymbol{A}^T\boldsymbol{A} &= \sum_{i=1}^{s}\sigma_i^2\boldsymbol{u}_i\boldsymbol{u}_i^T = \boldsymbol{U}\boldsymbol{D}\boldsymbol{U}^T \\ \text{where } \boldsymbol{U} &= \left[\boldsymbol{u}_1,\cdots,\boldsymbol{u}_s\right],\ \boldsymbol{D} = \begin{bmatrix}\sigma_1^2 & & \\ & \ddots & \\ & & \sigma_s^2\end{bmatrix} \end{aligned} \tag{25}$$

According to spectral theorem, entry σ^2 and vectors $\boldsymbol{u}$ are the non-zero eigenvalues and the corresponding eigenvectors of $\boldsymbol{A}^T\boldsymbol{A}$, respectively. The desired base vectors can then be calculated:

$$\left[\boldsymbol{v}_1,\cdots,\boldsymbol{v}_s\right] = \sqrt{\boldsymbol{D}^{-1}}\boldsymbol{A}\boldsymbol{U} \tag{26}$$

The eigenvalues denote the importance of each base vector. If an eigenvalue is zero, we can completely ignore this dimension as it can be linearly represented by other basis. In practice, since evaluations are noisy and our linear relation is approximated, we will never get any zero eigenvalues. To continue with our dimensionality reduction, we can rearrange the eigenvalues in a descending order and keep only the first few terms depending on how many dimensions we wish to have. Thus, we can approximate the real-world distortion using a linear combination of simulated distortions filtered by SVD. Although this is just an approximation, it should serve well as of the prior mean function of GP.

B. Choosing the Dynamics Representation

We can use different representations of the dynamics x_d in Eq. (18). Apart from using x and y displacements (called displacement-based representation in this paper) as in RTE, another type of representation is investigated. Since RTE uses periodic controllers, the robot will experience the same displacement and rotation during each period, and the execution should result in an arc. Based on this analysis, we represent the dynamics as the tangential speed v, angular velocity ω and direction φ (see Fig. 1). This design, called arc-based representation, considers the fact that the different dimensions of the dynamics are modelled by different GPs (hence are decoupled) in RTE. This is not an ideal design since x and y are typically correlated. In arc-based representation, these parameters are more decoupled and more consistent with real-world mechanics. For example, if the robot is carrying heavy payload, it will be slower and get a smaller v for all policies. If it suffers damage in one of its legs, all policies will get a shift in their angular velocities and directions. However, for displacement based representation, reduced speed will lead to decrease for positive displacements and increase for negative

displacements. Rotational and directional change will lead to x and y being heavily correlated.

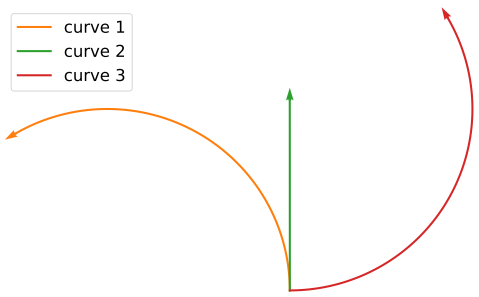

Fig. 1. Arc trajectories with different v, ω and φ. Curve 1 has $v = 0.5$, $\omega = 30°/s$ and $\varphi = 90°$. Curve 2 has $v = 0.25$, $\omega = 0$ and $\varphi = 90°$. Curve 3 shares the same v and ω with curve 1 but having $\varphi = 0$. All the curves are generated after 4 seconds of movement.

To implement arc-based representation, we will need to get the three parameters from the trajectory. During the execution of each policy, we record the x and y positions of the robot as well as the time. We do this by first finding the center and radius of the arc:

$$(x - A)^2 + (y - B)^2 = A^2 + B^2 \tag{27}$$

Where A and B are the x and y coordinates of the center. In Eq. (27) we force the curve to pass through the origin, which is the initial position of the robot. The optimal result should minimize the loss function:

$$\mathcal{L} = \sum_{i=1}^{N} \left[\sqrt{(x_i - A)^2 + (y_i - B)^2} - \sqrt{A^2 + B^2} \right]^2 \tag{28}$$

The optimal solution of this loss function cannot be obtained analytically, hence we use L-BFGS-B [31] optimization algorithm instead. After finding the A and B, we then get the angular change between each time step. This is achieved by taking the cross product between the neighbouring points:

$$\sin(\theta_2 - \theta_1) = \frac{(x_1 - A)(y_2 - B) - (y_1 - B)(x_2 - A)}{\sqrt{(x_1 - A)^2 + (y_1 - B)^2}\sqrt{(x_2 - A)^2 + (y_2 - B)^2}} \tag{29}$$

Because the $\sin(x)$ function is monotonically increasing between $[-\frac{\pi}{2}, \frac{\pi}{2}]$, we can also determine the sign of the angular difference (positive for anti-clockwise). All three parameters can then be calculated:

$$\omega = \frac{\sum_{i=1}^{N-1} \Delta\theta_i \Delta t_i}{\sum_{i=1}^{N-1} \Delta t_i^2}, \; v = \sqrt{A^2\omega^2 + B^2\omega^2}$$
$$\text{and} \;\; \varphi = \operatorname{atan2}(-B, -A) + \begin{cases} \frac{\pi}{2} & \text{if } \omega > 0 \\ -\frac{\pi}{2} & \text{if } \omega \leq 0 \end{cases} \tag{30}$$

IV. Experiment

References

[1] David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. Mastering the game of go with deep neural networks and tree search. *nature*, 529(7587):484–489, 2016.

[2] Oriol Vinyals, Igor Babuschkin, Wojciech M Czarnecki, Michaël Mathieu, Andrew Dudzik, Junyoung Chung, David H Choi, Richard Powell, Timo Ewalds, Petko Georgiev, et al. Grandmaster level in starcraft ii using multi-agent reinforcement learning. *Nature*, 575(7782):350–354, 2019.

[3] Christopher Berner, Greg Brockman, Brooke Chan, Vicki Cheung, Przemysław Dębiak, Christy Dennison, David Farhi, Quirin Fischer, Shariq Hashme, Chris Hesse, et al. Dota 2 with large scale deep reinforcement learning. *arXiv preprint arXiv:1912.06680*, 2019.

[4] Marc Deisenroth and Carl E Rasmussen. Pilco: A model-based and data-efficient approach to policy search. In *Proceedings of the 28th International Conference on machine learning (ICML-11)*, pages 465–472, 2011.

[5] Konstantinos Chatzilygeroudis, Roberto Rama, Rituraj Kaushik, Dorian Goepp, Vassilis Vassiliades, and Jean-Baptiste Mouret. Black-box data-efficient policy search for robotics. In *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 51–58. IEEE, 2017.

[6] Rituraj Kaushik, Konstantinos Chatzilygeroudis, and Jean-Baptiste Mouret. Multi-objective model-based policy search for data-efficient learning with sparse rewards. In *Conference on Robot Learning*, pages 839–855. PMLR, 2018.

[7] E Keogh and A Mueen. Curse of dimensionality. encyclopedia of machine learning. c. sammut and gi webb, eds, 2010.

[8] Antoine Cully and Yiannis Demiris. Quality and diversity optimization: A unifying modular framework. *IEEE Transactions on Evolutionary Computation*, 22(2):245–259, 2017.

[9] Miguel Duarte, Jorge Gomes, Sancho Moura Oliveira, and Anders Lyhne Christensen. Evolution of repertoire-based control for robots with complex locomotor systems. *IEEE Transactions on Evolutionary Computation*, 22(2):314–328, 2017.

[10] Konstantinos Chatzilygeroudis, Vassilis Vassiliades, and Jean-Baptiste Mouret. Reset-free trial-and-error learning for robot damage recovery. *Robotics and Autonomous Systems*, 100:236–250, 2018.

[11] Christopher KI Williams and Carl Edward Rasmussen. *Gaussian processes for machine learning*, volume 2. MIT press Cambridge, MA, 2006.

[12] Thomas Cover and Peter Hart. Nearest neighbor pattern classification. *IEEE transactions on information theory*, 13(1):21–27, 1967.

[13] James MacQueen et al. Some methods for classification and analysis of multivariate observations. In *Proceedings of the fifth Berkeley symposium on mathematical statistics and probability*, volume 1, pages 281–297. Oakland, CA, USA, 1967.

[14] Jean-Baptiste Mouret and Jeff Clune. Illuminating search spaces by mapping elites. *arXiv preprint arXiv:1504.04909*, 2015.

[15] Bryan Lim, Maxime Allard, Luca Grillotti, and Antoine Cully. Accelerated quality-diversity through massive parallelism. *arXiv preprint arXiv:2202.01258*, 2022.

[16] Miguel Duarte, Jorge Gomes, Sancho Moura Oliveira, and Anders Lyhne Christensen. Evorbc: Evolutionary repertoire-based control for robots with arbitrary locomotion complexity. In *Proceedings of the Genetic and Evolutionary Computation Conference 2016*, pages 93–100, 2016.

[17] Thomas J Page Jr. Multivariate statistics: A vector space approach. *JMR, Journal of Marketing Research (pre-1986)*, 21(000002):236, 1984.

[18] Thomas J Santner, Brian J Williams, William I Notz, and Brain J Williams. *The design and analysis of computer experiments*, volume 1. Springer, 2003.

[19] Guillaume Chaslot, Sander Bakkes, Istvan Szita, and Pieter Spronck. Monte-carlo tree search: A new framework for game ai. In *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, volume 4, pages 216–217, 2008.

[20] Rituraj Kaushik, Pierre Desreumaux, and Jean-Baptiste Mouret. Adaptive prior selection for repertoire-based online learning in robotics. *arXiv preprint arXiv:1907.07029*, 2019.

[21] Thomas S Ferguson. A bayesian analysis of some nonparametric problems. *The annals of statistics*, pages 209–230, 1973.

[22] Radford M Neal. Markov chain sampling methods for dirichlet process mixture models. *Journal of computational and graphical statistics*, 9(2):249–265, 2000.

[23] Stuart Geman and Donald Geman. Stochastic relaxation, gibbs distributions, and the bayesian restoration of images. *IEEE Transactions on pattern analysis and machine intelligence*, (6):721–741, 1984.

[24] David M Blei and Michael I Jordan. Variational inference for dirichlet process mixtures. 2006.

[25] Sean P Meyn and Richard L Tweedie. *Markov chains and stochastic stability*. Springer Science & Business Media, 2012.

[26] Sylvia Richardson and Peter J Green. On bayesian analysis of mixtures with an unknown number of components (with discussion). *Journal of the Royal Statistical Society Series B: Statistical Methodology*, 59(4):731–792, 1997.

[27] Arthur P Dempster, Nan M Laird, and Donald B Rubin. Maximum likelihood from incomplete data via the em algorithm. *Journal of the royal statistical society: series B (methodological)*, 39(1):1–22, 1977.

[28] Carl Rasmussen. The infinite gaussian mixture model. *Advances in neural information processing systems*, 12, 1999.

[29] Carl Rasmussen and Zoubin Ghahramani. Infinite mixtures of gaussian process experts. *Advances in neural information processing systems*, 14, 2001.

[30] Shiliang Sun and Xin Xu. Variational inference for infinite mixtures of gaussian processes with applications to traffic flow prediction. *IEEE Transactions on Intelligent Transportation Systems*, 12(2):466–475, 2010.

[31] Ciyou Zhu, Richard H Byrd, Peihuang Lu, and Jorge Nocedal. Algorithm 778: L-bfgs-b: Fortran subroutines for large-scale bound-constrained optimization. *ACM Transactions on mathematical software (TOMS)*, 23(4):550–560, 1997.