Background and Progress report

§1 Introduction

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. The problem is that there is always a gap between simulation and reality, and the policy trained in simulation could suffer from this sim-to-real transfer. To retain a high performance, these deep RL trained agents require significant amount of extra fine-tuning in response to this gap. Apart from the sim-to-real problem, other perturbations in the environment dynamics could also require fine-tuning or even retaining to retain the capability of the agent. For example, the team of OpenAI Five [3] had to conduct three additional amendments followed by 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 can lead to unforeseeable distortions during deployment, and the environment dynamics is typically non-stationary as the robot moves or makes interaction with the environment (could be change in the payload, damage in the robot, or different terrain).

It would be better if we may learn the true dynamics from the interactions with the real-world 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, 5, 6]. However, the amount of data required to learn a model typically scales exponentially with the dimensionality of the input space [7] and fine-tuning the control policy with the learned model might still be too slow 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 [8] to fully cover the task space (e.g., displacement of centre of mass). During the adaptation, we learn the high-level distortion (called transformation model) in the task space from real-world interactions. Because the task space typically has much smaller dimensionality than the low-level state and actions (i.e., joint encoders and torques for all motors), this transformation model may be learned in real time. The transformation model is typically probabilistic models like Gaussian Processes (GP) [9]. Instead of optimizing the policy with the transformation model, we use a repertoire-based control that simply chooses the elementary policy with the outcome predicted by this model that is most aligned with the current goal. This is available thanks to keeping a repertoire of policies rather than a single policy, so that the robot can still perform the same tasks by finding the corresponding policies that compensate the distortion (for example the robot aims to go forward but distortion is left, then a policy aiming right-front can be used for going forward instead).

The repertoire-based method is very data-efficient and can be used for online adaptation and have achieved very impressing results. But this method heavily relies on having a good prior mean for the GP, otherwise the probabilistic modelling might be ineffective or even misleading. Typically,

the prior can simply be the outcome of the elementary policies in the simulation (assuming no distortion). However, this might not be good enough for environments that are distinct from the training environment, like broken legs that can lead to very large distortion. It would be better if we can use the real-world interactions of several robots in different environments to build several priors, and then teach the robot to identify the one that mostly explains its current situation during online adaptation. The difficulty is that, even with this collaborative-learning of priors, it is still not practical to evaluate the outcome of all the policies in the repertoire in the real-world as a repertoire typically contains thousands of elementary policies. Also, the selection of environment is made manually, while some environments could have similar dynamics, and some could be omitted. Hence, we need to identify the dynamics that are distinct, and we need more efficient way to build the priors of those environments with limited number of real-world interactions.

The solution proposed by this paper is that: we let the users deploy the robots in the environment and make online adaptation. After that, we collect the adaptation data and upload them into an archive management system. The system will group the data with similar dynamics into clusters and use a parameterised model to learn the prior for each cluster with their data. Thus, we don't need to manually select the distinct dynamics that we may encounter, and the use of a parameterised model for the prior relaxes the need to evaluate the whole repertoire in real-world.

§2 Related work

2.1 Quality-Diversity Optimisation

The repertoire-based method uses quality diversity (QD) optimization to generate a repertoire of diversified and high-quality locomotion behaviors in simulation. The most popular method for doing QD is called MAP-Elites [8, 10], where we tabularize the task-space (for example, discretize the displacement along the x, y axis of the robot after executing a policy into 2D grids). We then conduct a stochastic optimization inspired by natural evolutionary to fill the grids. We start from a few random policies, which are typically neural networks. We then evaluate them in the simulator to get their outcomes in the task space. Each of these outcomes is mapped to one of the girds (hence these grids are filled). We then conduct a loop of policy generation. In each iteration we select one of the filled grids and then mutate the policy by adding a random variation to the policy parameters (called genotype in evolutionary computation, in the neural network case just the weights of the neurons), leading to a new policy. We then evaluate the policy to get the outcome and identify the grid it belongs to. If the grid is already filled, we only keep the policy with the better performance score; if it is empty, we simply fill this grid with the new policy. The pseudo code for MAP-Elites QD algorithm is presented below.

Algorithm 1: MAP-Elites
Discretize the task space into grids
Initialize a few random policies
Evaluate random policy and map them to grids
For $t = 1 \rightarrow T$ do:
1) Select a filled grid
2) Add a small variation to the policy π in the grid and get π'

3) Evaluate the new policy π' and find the corresponding grid and its policy π''
4) Fill the grid with the new policy if $\pi'' = \emptyset$ or $score(\pi') > score(\pi'')$

This algorithm enabled us to do with novelty search with local competition [11] (NSLC), where we find a variety of novel solutions to solve different tasks, while keeping only the best one in each task domain. This means our robot successfully learns a large repertoire of elementary policies that fully covers the task space. If the distortion shifts the policies, we can easily find ones to compensate such distortion.

2.2 Gaussian Process

The Gaussian Process (GP) assumes the prior distribution of the outcomes to be a multivariate normal distribution that is governed by a prior mean function μ and a covariance matrix Σ.

$$f(y_1, y_2, \dots, y_N) = \frac{exp\{-\frac{1}{2}(\vec{y}-\vec{\mu})^T\Sigma^{-1}(\vec{y}-\vec{\mu})\}}{(2\pi)^{N/2}\cdot\sqrt{det(\Sigma)}} \quad (1)$$

Typically, the prior mean is just a fixed constant like zero, but in our case this prior mean is a non-constant function that represents the prior knowledge we have about the outcome of a policy when there is no neighbouring data to refer to. The covariance matrix is a symmetric matrix that quantifies the covariance between two data points and is generated using a covariance function (called kernel). The kernel takes in the coordinates of two data points in the input space and return their correlation. This correlation starts from the prior variance and asymptotically decreases to zero as the distance between the two points tends to infinity, following the fact that data points closer to each other are more correlated. The typical choice of kernel can be RBF kernel that the correlation decreases in the form of a Gaussian function.

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

RBF kernel assumes the function is infinitely differentiable, which may be factually incorrect. Hence, it might be more desirable to use a Matern kernel that assumes the function is ⌈v⌉-1 times differentiable in the mean-square sense [9, 12].

$$k_v(\boldsymbol{x_i}, \boldsymbol{x_j}) = \alpha_0 \frac{2^{1-v}}{\Gamma(v)}(\sqrt{2v}d_{ij})^v K_v(\sqrt{2v}d_{ij}) \quad (3)$$

The distance d_{ij} between the two data points does not have to be Euclidean distance. This distance generally takes the following form:

$$d_{ij} = (x_i - x_j)^T\Delta^T\Delta(x_i - x_j) \quad (4)$$

For simplicity, if we assume the Δ matrix is diagonal, (4) becomes:

$$d_{ij} = \sum_{i=1}^{d}\frac{(x_i - x_j)^2}{l_i} \quad (5)$$

Where this distance metric is anisotropic, suggesting distance along some axis contributes more uncertainty than others. The l_i 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 that axis.

To account for the fact the evaluation might be noisy, we can incorporate the noise into our covariance matrix by adding a diagonal term that quantifies the scale of the noise.

$$\Sigma = k(\boldsymbol{x}_{1:N}, \boldsymbol{x}_{1:N}) + \sigma_n^2\boldsymbol{I} \quad (6)$$

Where the first term is the kernel and the σ_n^2 in the second term is the variance of the noise, where we have assumed that the noise is also Gaussian.

When making prediction, GP first combines the point that we wish to evaluate along with the points that we have already evaluated into a multivariate normal distribution. Then, we can calculate the posterior distribution of the point we aim to evaluate conditioned on the results of the ones that have been evaluated using Bayes rule.

$$y(x_{N+1})|\{\boldsymbol{x}_i, y_i\}_{i=1}^{N} \sim \mathcal{N}(\mu_*(\boldsymbol{x}), \sigma_*^2(\boldsymbol{x})) \tag{7}$$

The resultant posterior is also a Gaussian with mean and variance presented below [9, 13]:

$$\mu_*(\boldsymbol{x}) = \mu(\boldsymbol{x}) + \Sigma(\boldsymbol{x}, \boldsymbol{x}_{1:N})\Sigma^{-1}(\boldsymbol{x}_{1:N}, \boldsymbol{x}_{1:N})(\vec{y}_{1:N} - \vec{\mu}(\boldsymbol{x}_{1:N})) \tag{8}$$

$$\sigma_*^2(\boldsymbol{x}) = \Sigma(\boldsymbol{x}, \boldsymbol{x}) - \Sigma(\boldsymbol{x}, \boldsymbol{x}_{1:N})\Sigma^{-1}(\boldsymbol{x}_{1:N}, \boldsymbol{x}_{1:N})\Sigma(\boldsymbol{x}_{1:N}, \boldsymbol{x}) \tag{9}$$

2.3 Reset-free Trial-and-Error learning

The work of this paper is based on the Reset-free Trial-and-Error [14] (RTE) algorithm. In RTE, the task space is just the displacement of the robot in x and y direction after executing each policy. The RTE then uses a GP to build probabilistic inference between the policies and make Bayesian prediction about the outcome of unevaluated policies based on the ones that have been evaluated during adaptation. The prior mean function is chosen to be the evaluation outcome of these policies in the simulator with the default environment settings, which is the result that were used to run the MAP-Elites. The RBF kernel is used with a diagonalised Δ matrix, and the input space that were used by the kernel is chosen to be the same as the task space, namely the x and y displacement of executing the policy in the default simulation settings. Since the policy outcome has several dimensions, RTE requires the use of a different GP for each dimension (hence two GPs for 2D task space). During online adaptation, the RTE first predicts the outcome of each policy in the repertoire using GP and then uses a Monte Carlo Tree Search [15] (MCTS) to plan its actions.

This method is not very ideal for two 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, and such prior would be misleading. Second, the use of the task space as the input space is not ideal, two-dimensional input space may be inadequate, and displacement in y axis does not mean a lot for the outcome in the x axis. Nevertheless, this method successfully enabled a damaged hexapod to recover 77.52% of its capability [14] 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 [16] (APROL) uses several repertoires that are generated in different environments to overcome the first problem mentioned above. The APROL is better than RTE, but it still suffers from the second problem, and the environments to build the priors needs to be manually selected and are discrete. Also, running MAP-Elites to get so many repertoires can be computationally expensive. This work hopes to overcome these problems via clustering and then leveraging the real-world adaptation data collected during user deployment.

2.4 Dirichlet Process

Since we don't know how many clusters need to be generated to group the real-world dynamics, we

need to take a non-parametric approach that makes no assumption for the cluster number. Dirichlet Process (DP) is ideal for this and is used in this work to cluster the adaptation experiences. DP is an extension of Dirichlet distribution. We assume the current data distribution is a mixture model that each mixture component is sampled from a distribution of distributions (for example, the mean of Gaussian distributions with fixed a variance follows another Gaussian distribution), and the mixture weights follow the following distribution:

$$p(\pi_1, \dots, \pi_M|\alpha) \sim \frac{\Gamma(\alpha)}{\Gamma(\alpha/M)^M} \prod_{j=1}^{M} \pi_j^{\alpha/M-1} \tag{10}$$

This is called symmetric Dirichlet distribution. The term π_m stands for the mixture weight for the m^{th} mixture component, and they sum up to 1. M is the number of clusters, and α/M is the concentration parameter that controls how centred the weights are. The prior probability to find the data points in a certain configuration of assignment can be calculated using the standard Dirichlet integral [17].

$$p(c_1, \dots, c_N|\alpha) = \frac{\Gamma(\alpha)}{\Gamma(\alpha+N)} \prod_{j=1}^{M} \frac{\Gamma(\alpha/M+N_j)}{\Gamma(\alpha/M)} \tag{11}$$

This can then be used to calculate the posterior of a new data point belonging to a cluster given the existing data assignment:

$$p(c_i = j|\boldsymbol{c}_{-i}, \alpha) = \frac{N_{-i,j}+\alpha/M}{N-1+\alpha} \tag{12}$$

Where the subscript $-i$ denotes all indexes except for i, and $N_{-i,j}$ stands for the number of data points in the j^{th} cluster except for the i^{th} data. If we let the number of clusters tend to infinity, where we assume the data comes from an infinite number of clusters (which is reasonable for real-world dynamics), this becomes the Dirichlet Process, and the posterior becomes:

$$p(c_i = j|\boldsymbol{c}_{-i}, \alpha) = \frac{N_{-i,j}}{N-1+\alpha} \tag{13}$$

$$p(new\ cluster|\boldsymbol{c}_{-i}, \alpha) = \frac{\alpha}{N-1+\alpha} \tag{14}$$

Where the data has certain probability given by (14) to be a completely new cluster on its own. This special property of DP allows the adaptive generation of new clusters, and it allows us to group data without specifying the number of clusters. The parameter α controls the growth of new clusters. We can see from (14) that the more data we have, the slower the new clusters are generated. This is reasonable as more data we have, the more likely that our sampling has covered the truth distribution. To determine the α, we first set a threshold of probability (for example, 0.5%) that we believe is small enough for the prior that is almost impossible be sampled. Then we set the number of data size (for example, 1000) that after having sampled such amount of data we will be confident that no further clusters are needed. Then we can use (14) to calculate the α.

§3 Methodology

3.1 Intuition

To improve based on RTE via collecting adaptation data, we are provided with some (episodes of) real-world interactions of some robots of the same design in some unknown environments, and we don't have access to the parameters (like fiction, payload, etc.) of those environments. We are aiming to leverage

those real-world interactions (with limited information) to assist a robot deployed in an unknown environment to quickly adapt its locomotion.

The most important thing is that although some of the collected experiences may come from the same (or very similar) environment, we don't have access to the parameters of the environment to determine whether this experience can be used to assist the current situation that the robot is in. Even if we know that one experience results from the same situation as the current situation, this experience many contain only a limited number (say 5) of evaluation results, hence it might not be very useful to assist the adaption. The solution to this problem, also the key concept of this paper, is to group the experiences as clusters and use a different GP for each cluster. Thus, we have several experiences for each cluster, hence enough data for analysis. During deployment, the robot will assume that the current situation is in one of the clusters, and it will try to narrow down which cluster it is in and hence use the GP of that cluster for its adaptation.

3.2 Assumptions

There are a few assumptions needs to be made for further analysis. The first assumption is that the actual coordinates in the task space can efficiently expressed in a linear form, hence we parametrise the model for the prior:

$$x = \beta^T \epsilon + \beta_0 + \epsilon_0 + c \tag{15}$$

Where the β represents a vector with elements of some quantities of the robot behaviour; the vector ϵ is the environment descriptor, representing some quantity of the environment. The term β_0 and ϵ_0 are scalers that only depend on the robot behaviour and the environment, respectively. Note that we neither have access to the β and ϵ vector directly nor do we know their detailed meaning.

The linear assumption seems a little bit weak, but it can be proved that by Tylor expanding the potential form of the real-world dynamics to its second order term, we will get exactly such a linear form. If we assume that the real-world dynamics is a sum of terms consisting of factors from the behavior of the robot and the environment, we have the dynamics in the following form:

$$x_d = \sum_{i=1}^{T} f_{i,d}(\beta_i, \epsilon_i) + f_{\beta,d}(\beta) + f_{\epsilon,d}(\epsilon) + c \tag{16}$$

Where the subscript d denotes the dimension in the task space, and the subscript i in $f_{i,d}(\beta_i, \epsilon_i)$ means that each term can depend on a different quantify of the robot behaviour and the environment. If we Taylor expand the terms which considers both the behaviour factor and the environment factor, namely $f_{i,d}(\beta_i, \epsilon_i)$, to the second order, we get an approximation:

$$f_{i,d}(\beta, \epsilon) \approx f_{i,d}(\beta_{i,0}, \epsilon_{i,0}) + \frac{\partial f_{i,d}}{\partial \beta_i}\Delta\beta_i + \frac{\partial f_{i,d}}{\partial \epsilon_i}\Delta\epsilon_i + \frac{\partial^2 f_{i,d}}{\partial \beta_i \partial \epsilon_i}\Delta\beta_i\Delta\epsilon_i + \frac{\partial^2 f_{i,d}}{\partial \beta_i^2}\Delta\beta_i^2 + \frac{\partial^2 f_{i,d}}{\partial \epsilon_i^2}\Delta\epsilon_i^2 \tag{17}$$

If we examine this form, we can see that there is only one term, $\frac{\partial^2 f_{i,d}}{\partial \beta_i \partial \epsilon_i}\Delta\beta_i\Delta\epsilon_i$, that considers both the two factors, and the rest terms are either constant or only consider only one of the two factors. Hence, if we substitute (17) into (16), we will get the following formulas:

$$x_d \approx \sum_{i=1}^{T} \frac{\partial^2 f_{i,d}}{\partial \beta_i \partial \epsilon_i}\Delta\beta_i\Delta\epsilon_i + f'_{\beta,d}(\beta) + f'_{\epsilon,d}(\epsilon) + c' \tag{18}$$

$$f'_{\beta,d}(\beta) = f_{\beta,d}(\beta) + \sum_{i=1}^{T}\left(\frac{\partial^2 f_{i,d}}{\partial \beta_i^2}\Delta\beta_i^2 + \frac{\partial f_{i,d}}{\partial \beta_i}\Delta\beta_i\right) \tag{19}$$

$$f'_{\epsilon,d}(\epsilon) = f_{\epsilon,d}(\epsilon) + \sum_{i=1}^{T}\left(\frac{\partial^2 f_{i,d}}{\partial \epsilon_i^2}\Delta\epsilon_i^2 + \frac{\partial f_{i,d}}{\partial \epsilon_i}\Delta\epsilon_i\right) \tag{20}$$

$$c' = c + \sum_{i=1}^{T} f_{i,d}(\beta_{i,0}, \epsilon_{i,0}) \quad (21)$$

We can see that (18) is exactly in the linear form that we have assumed in (15).

The second assumption is that similar environments have similar dynamics, and we can use the same GP for each cluster we find, meaning that they share the same prior mean and the same kernel. This seems trivial, but a very important point that we only ask that they use the same kernel for the data in their experience. Here, we have made a huge relaxation that different experiences in the same cluster only share the same covariance function but are mutually decoupled. This means that the covariance matrix for the entire cluster is a blocked matrix (see Fig. 1 below).

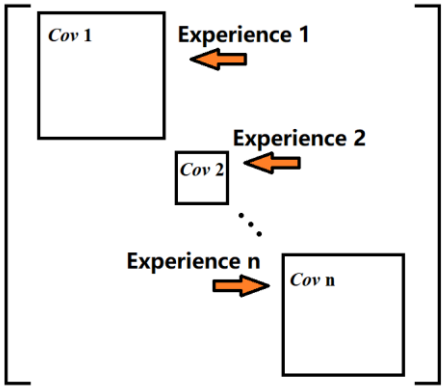

Fig. 1. The covariance matrix for the GP of a cluster. Each block is a covariance matrix for a particular experience, where the size of the block represents the data in this experience.

This assumption ensures two things. One is that the computational expense for optimising the kernel (via MLE) wouldn't be too large, as the matrix inversion scales to the cubic of the matrix size. The second is that the evaluation results of the same policy are allowed to be different for different experiences. The reason for making the second point is that although some environments have similar dynamics, they might not be the same everywhere. Also, Dirichlet Processes with Gibbs sampling could assign experiences to the wrong cluster in the intermedia steps. If we do not factorise the individual experiences, hence assuming all experiences in the same cluster come from the same environment, we might get contradictory results for different experiences. The kernel found using MLE for these data will have very small length scales and very large noise, indicating that the underlying dynamics is highly fluctuating and noisy, and we quickly lose certainty as we walk in the behaviour space. In this case, the prediction made by the GP using such kernel might give us little information. Hence, we relax the requirement that experiences in the same cluster strictly share the same dynamics, and we only assume they share the same kernel. The reason we are still insisting they have the same kernel is because we have reasons to believe that: for similar environments the dynamics might be different, but they should share similar probabilistic influences from the behaviours of the robot (see Fig. 2). This is guaranteed by using the same kernel for all experiences in a cluster, which ensures the same probabilistic dependency on the behaviour descriptors for the same cluster.

The third assumption is that we believe that the simulated results are nothing more than a shifted version of real-world outcomes. This is not always true, as there could be bugs in the simulator that may lead to very large distortion, but it plays a very important role in the methodology.

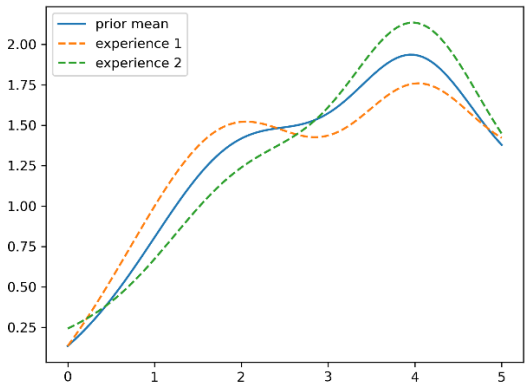

Fig. 2. The assumption of the decoupled experiences. Here, the two experiences have the similar but not the same dynamics, and they have the same probabilistic dependencies on the behaviour of the robot (x-axis).

3.3 Clustering experiences

A very important part is that we don't know how many types of situations that the robot will encounter hence we don't have any idea how many clusters we should use. The solution is to use a DP with Gibbs sampling to determine the clusters. The merit of using DP is that it allows the experience to either belong to one of the existing clusters or to form its own cluster, based on the likelihood. Hence, we are assuming the ensemble of experiences we collected is an infinite mixture of GPs, where each GP represents an environment. This is done using a Gibbs sampling to assign each experience to a cluster. The Gibbs sampling starts from one experience and discard it from its current assignment. Then, we calculate the probability of this experience belonging to each existing clusters and being a new cluster. We use (13) and (14) to calculate the prior and use Bayes rule to calculate the posterior:

$$p(z = i|x_{1:N}) = \frac{p(x_{1:N}|z=i)\cdot p(z=i)}{\sum_{j=1}^{M} p(x_{1:N}|z=j)\cdot p(z=j)+p(new)\cdot p(x_{1:N}|new)} \tag{22}$$

We then randomly assign this data point based on the above probability distribution and move to the next one. This sampling is looped through all experiences and repeated several times to ensure convergence. Each time the assignment is changed, we will update the prior and the kernel accordingly. The prior update is cheap to do, but fitting the kernel is rather computational expensive, hence we may need to do rank one update in the sake of computational cost.

3.4 Determining the GP

The GP needs a prior mean function and a covariance function (kernel). The first challenging question is how to find the prior function for the cluster, since each cluster will only contain a small subset of evaluated policies, while we need a prior function for entire repertoire. This gives a constrain on the prior function that it must be highly parametric so we can determine its form with only a few data points in a cluster. We have already made a very parametric assumption of the linear relation, so we only need to determine the behaviour descriptors as the base functions for each policy and use linear regression to find the relation for each cluster.

The behaviour descriptors are hard to find, but we can use a result-based approach that fully leverages

the linear assumption. Since the actual outcome is assumed to obey a linear relation with the behaviour descriptors, this means we can also linearly represent the behaviour descriptors using the outcomes. So, the design is: we first evaluate the entire repertoire on several cases in simulator and use those evaluation results as base functions, and then we use linear regression using the data in each cluster to find the linear relation for each cluster. Hence, we are representing the prior for each cluster as a linear combination of simulated outcomes via conducting linear regression using the real-world interaction data:

$$\mu_i = x_0 + \sum_{j=1}^{m} \alpha_{i,j} \cdot \Delta x_j + c_i \tag{23}$$

Where the μ_i stands for the prior mean for the i^{th} cluster, x_0 is the baseline which can be chosen to be the result in the default simulation settings, and Δx_j is the simulated outcome evaluated in the j^{th} environment setting subtracted by the baseline result. The subtraction is here to remove the $f'_{\beta,d}(\beta)$ term and the c' term in (18), but the $f'_{\epsilon,d}(\epsilon)$ is still there in the c_i term.

This leads to a balancing problem that evaluating the repertoire in many environments is computationally expensive, and that also means we have many coefficients to be determined via linear regression hence we need many data for each cluster; reducing the number of environments to evaluate the repertoire, on the other hand, might give poor representation and is also unstable depending on the environments that are picked. The solution is that we evaluate the repertoire in as many (and as much diverse) environments as we can afford, and then we use singular vector decomposition (SVD) to reduce the dimensionality to the level we want. To do SVD, we organise the differences between these evaluation results and the baseline result into a big rectangular matrix A, in the form below:

$$A = \begin{bmatrix} x_{1,1} & x_{1,2} & \cdots & x_{1,2000} \\ x_{2,1} & x_{2,2} & & x_{2,2000} \\ \vdots & & \ddots & \vdots \\ x_{64,1} & x_{64,2} & \cdots & x_{64,2000} \end{bmatrix} = \sum_{i=1}^{64} \sigma_i \cdot u_i v_i^T \tag{24}$$

Where $x_{i,j}$ stands for the outcome of the j^{th} elementary policy evaluated in the i^{th} environment setting subtracted by the baseline result. Here, we have evaluated a repertoire consisting of 2000 policies in 64 different environments. After arranging the terms based on σ_i (which are the eigenvectors of matrix AA^T) in a descending order, we can reduce the dimensionality while having the least squared error by keeping the first few terms, and the v_i in these terms are the new base functions for the behaviour descriptors. Thus, for each cluster we only need a few data to fit the prior mean.

As for the kernel to build the GP, we can use a *Matern* kernel with $v = 2.5$, assuming twice differentiable, with a diagonalised Δ matrix. Then we can use maximum likelihood estimate (MLE) to determine the length scales, the prior variance in the kernel, and the noise variance σ_n^2.

The trickiest part is the mean and kernel for a new cluster, where it might be hard to do since this new cluster only has one adaptation experience which could contain very limited amount of data. In this case, the above approach could lead to overfitting. Hence, we can select the simulation evaluated result that is closest to it as the prior. As for the kernel, we can just use a default set of length scales.

3.5 Adaptation

This consists of two parts: figuring out which situation robot is facing and make planning to achieve the goal. Instead of using the DP, we assume that the current situation belongs exactly to one of the clusters we have built. The estimation of policy outcome is now a probabilistic linear combination of the posterior made by each GP.

$$p(x_{N+1}|x_{1:N}) = \sum_{i=1}^{M} p(z = i|x_{1:N}) \cdot p(x_{N+1}|x_{1:N}, z = i) \tag{25}$$

Where the z is the discrete latent variable that represents the index of the cluster that the data belongs

to. The probability of the current environment belonging to each cluster $p(z = i|x_{1:N})$ can be easily found using Bayesian rule since we have the GP for each cluster. The prior probability of cluster assignment $p(z = i)$ is determined using the data collected during clustering.

$$p(z = i|x_{1:N}) = \frac{p(x_{1:N}|z=i)\cdot p(z=i)}{\sum_{j=1}^{M} p(x_{1:N}|z=j)\cdot p(z=j)} \tag{26}$$

$$p(x_{1:N}|z = i) = \frac{1}{\sqrt{(2\pi)^N \cdot det\ (\Sigma_i)}} e^{-\frac{1}{2}(x_{1:N}-\mu_i)^T {\Sigma_i}^{-1}(x_{1:N}-\mu_i)} \tag{27}$$

$$p(z = i) = \frac{N_i}{N} \tag{28}$$

The planning part we can just stick to RTE that we use a MCTS to determine the best action.

§4 Evaluation

The performance of such approach can be compared with RTE in simulator. This is majorly because RTE is easier to implement than APROL, and we cannot afford testing on real-world robots. Although real-world data leveraging is the major improvement of introducing the collaborative learning, this is not practical due to the lack of time for this project.

It is believed with strong confidence that the proposed method will significantly outperform the RTE, majorly for having a better prior mean function. In the future work, this method may hopefully be implemented in real robots.

§5 Other improvements

There are promising further improvements that I can think of for now, but it is going to require significant amount of effort and is not likely to be finished by the end of this project.

One big regret of this approach is that we are never directly using the real-world interactions to for the GP. We are only using them for finding the clusters, fitting the prior, and fitting the kernel. This is due to the concern that the experiences in the same cluster might not come from exactly the same environment. This can be further improved by grouping the experiences in the same cluster if their GPs agree with each other. This is enabled with a second DP that form sub-clusters within each cluster. In this setting, the robot will have not just a prior mean and kernel, but a posterior mean and variance matrix to begin with its adaptation. Since the DP also allows new clusters to be formed, the robot can reject this posterior and stick to the prior only if it finds the posterior fails to explain its observations. This might give much quicker adaptation for adapting visited environments. But the workload might be quite high.

The other improvement I have is about the SVD. As the SVD is to minimize the error of using a subset of linear transformed vectors to linearly represent the original matrix, this minimization could suffer from a covariate shift. I mean, the SVD minimizes the error of representing simulated outcomes, and the distribution of environments that we choose to evaluate the repertoire matters. So, we can improve by not minimizing the error of simulated outcomes but that of the real-world outcomes. We can do this by fine-tuning the SVD results.

§6 Ethical concern

There is no obvious ethical concern that should be considered in this work. This paper studies the adaptive locomotion of robots using advanced probabilistic modelling methods like Gaussian Process and Dirichlet Process. The only element of potential ethical concern would be the collection of user data of their robots making real-world adaptation in the future work. However, since little information (no personal information) can be extracted from such data, such collection of data raises no ethical danger if comes with user consent.

§7 References

[1] Silver D, Huang A, Maddison C J, et al. Mastering the game of Go with deep neural networks and tree search[J]. nature, 2016, 529(7587): 484-489.

[2] Vinyals O, Babuschkin I, Czarnecki W M, et al. Grandmaster level in StarCraft II using multi-agent reinforcement learning[J]. Nature, 2019, 575(7782): 350-354.

[3] Berner C, Brockman G, Chan B, et al. Dota 2 with large scale deep reinforcement learning[J]. arXiv preprint arXiv:1912.06680, 2019.

[4] Deisenroth M, Rasmussen C E. PILCO: A model-based and data-efficient approach to policy search[C]//Proceedings of the 28th International Conference on machine learning (ICML-11). 2011: 465-472.

[5] Chatzilygeroudis K, Rama R, Kaushik R, et al. Black-box data-efficient policy search for robotics[C]//2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS). IEEE, 2017: 51-58.

[6] Kaushik R, Chatzilygeroudis K, Mouret J B. Multi-objective model-based policy search for data-efficient learning with sparse rewards[C]//Conference on Robot Learning. PMLR, 2018: 839-855.

[7] Keogh E, Mueen A. Curse of Dimensionality. Encyclopedia of Machine Learning. C. Sammut and GI Webb, eds[J]. 2010.

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

[9] Rasmussen C E, Williams C K I. Gaussian processes for machine learning[M]. Cambridge, MA: MIT press, 2006.

[10] Mouret J B, Clune J. Illuminating search spaces by mapping elites[J]. arXiv preprint arXiv:1504.04909, 2015.

[11] Lehman J, Stanley K O. Evolving a diversity of virtual creatures through novelty search and local competition[C]//Proceedings of the 13th annual conference on Genetic and evolutionary computation. 2011: 211-218.

[12] Santner T J, Williams B J, Notz W I, et al. The design and analysis of computer experiments[M]. New York: Springer, 2003.

[13] Eaton M L. Multivariate statistics: a vector space approach[J]. JOHN WILEY & SONS, INC., 605 THIRD AVE., NEW YORK, NY 10158, USA, 1983, 512, 1983.

[14] Chatzilygeroudis K, Vassiliades V, Mouret J B. Reset-free trial-and-error learning for robot damage recovery[J]. Robotics and Autonomous Systems, 2018, 100: 236-250.

[15] Chaslot G, Bakkes S, Szita I, et al. Monte-carlo tree search: A new framework for game ai[C]//Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment. 2008, 4(1): 216-217.

[16] Kaushik R, Desreumaux P, Mouret J B. Adaptive Prior Selection for Repertoire-based Online Learning in Robotics[J]. arXiv preprint arXiv:1907.07029, 2019.

[17] Foster A, Li H, Maierhofer G, et al. An extension of standard latent Dirichlet allocation to multiple corpora[J]. SIAM Undergraduate Research Online, 2016, 9.