Behavioral Repertoire via Generative Adversarial Policy Networks

Behavioral Repertoire via Generative Adversarial Policy Networks

Marija Jegorova[1], Stéphane Doncieux[2], and Timothy M. Hospedales[1]

***Abstract*—Learning algorithms are enabling robots to solve increasingly challenging real-world tasks. These approaches often rely on demonstrations and reproduce the behavior shown. Unexpected changes in the environment or in robot morphology may require using different behaviors to achieve the same effect, for instance to reach and grasp an object in changing clutter. An emerging paradigm addressing this robustness issue is to learn a diverse set of successful behaviors for a given task, from which a robot can select the most suitable policy when faced with a new environment. In this paper, we explore a novel realization of this vision by learning a generative model over policies. Rather than learning a single policy, or a small fixed repertoire, our generative model for policies compactly encodes an unbounded number of policies and allows novel controller variants to be sampled. Leveraging our generative policy network, a robot can sample novel behaviors until it finds one that works for a new scenario. We demonstrate this idea with an application of robust ball-throwing in the presence of obstacles, as well as joint-damage-robust throwing. We show that this approach achieves a greater diversity of behaviors than an existing evolutionary approach, while maintaining good efficacy of sampled behaviors, allowing a Baxter robot to hit targets more often when ball throwing in the presence of varying obstacles or joint impediments.**

I. Introduction

Robots are increasingly able to solve challenging tasks by learning controllers. While reinforcement or imitation learning approaches can be effective, they typically learn a single ideal solution to a given control problem, and the robustness of that solution to challenging situational variants (e.g., changing/complex obstacles, such as in Fig. 1, or damage to the robot) is hard to guarantee. If a control policy fails due such an unexpected environmental change, robots can try to adapt their control policy to a new situation through re-planning [1] or adapting a learned policy [2], [3]. Beyond such adaptation, when animals face a challenging environment in which a previously learned behavior fails, they also draw on an additional capability: leveraging a suite of other known behaviors that are expected to solve the task at hand [4]. Exploration within a set of diverse historical behaviors that solved a task can quickly lead to a solution that succeeds in a new environment [4]. Such behavioral repertoire-based approaches are emerging as promising techniques for robustly solving tasks [4], [5], [6].

Existing realizations of this robustness-through-diversity vision are often based on evolutionary algorithms that train a diverse set (population) of controllers that solve a given task [5], [6]. However this approach has several drawbacks: storing a large database of controllers is not compact, and there is only as much diversity as is contained in the population of controllers. We argue that a preferable instantiation of this vision is to learn a generative model over controllers. Firstly, it is compact – only the parameters of the generative model rather than a large list of controllers need to be stored. Secondly, the available diversity is not limited to the instances in a fixed length list. By sampling a generative model over controllers, an unlimited number of distinct controllers can be obtained. And with a sufficiently flexible generative model, sampled controllers need not be simple interpolations between controllers used to train the generative model. Samples could encode novel solutions to the problem by drawing diverse aspects of multiple training policies.

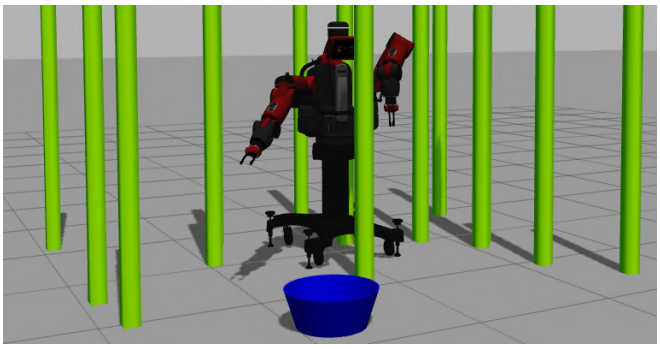

Fig. 1. An example obstacle-occluded environment, for which the GPN-generated policy repertoire can offer solutions.

This approach is coherent with the exploratory behavior of infants (and other animals) – specifically their ability to perform a behavior in high variation so there is a distribution of actions associated with each behavior [7]. Our method models this distribution. Following the example of other progressive sequential architectures – [8], [9], we propose a simple two-staged developmental framework where one first builds up the initial repertoire of actions (using methods such as quality-diversity search [10]), and then generalizes beyond this repertoire via our proposed generative model. Our progression from a library-based approach to a generative-model can also be considered a representational re-description [11], between developmental waves [7].

While conceptually appealing, training generative models over policies is non-trivial. The space of reasonable policies likely to solve a given task is a complicated manifold within the space of all policies, considering actuator redundancy, non-linearities and so on. We therefore propose to apply generative adversarial networks (GANs) [12] to model the distribution over policies that solve a given task using a neural network, thus defining a generative policy network (GPN). In our framework the GPN models the distribution over policy parameters, so that each sample from the GPN defines a specific robot controller. Multiple samples from the GPN therefore correspond to different solutions to the task that the

[1] School of Informatics, University of Edinburgh, UK. {m.jegorova@sms.,t.hospedales@}ed.ac.uk. [2] Sorbonne Université, CNRS, ISIR, Paris, France. stephane.doncieux@upmc.fr.

GPN is trained on. To generate training data for the GPN we exploit quality-diversity (QD) search evolutionary algorithms [10] to find a diverse set of policies that solve a task in various contexts. The GPN network is built with a (i) target-conditional DCGAN [13] for learning a contextual policy repertoire, upgraded with (ii) an outcome prediction regularisation for improved accuracy. Compared to a conventional GAN, we find it beneficial to regularize GPN-training by requiring it to generate not only a controller but the outcome of running that controller (i.e. to simulate the forward model, or reconstruct the input goal state), and this is also useful as a way to pick promising policies (e.g., sample the GPN until a policy is drawn which is expected to work in the current environment). Once trained, a GPN then provides a compact source of diverse and novel policies likely to solve variants of the task.

We demonstrate our approach through the specific application of target-conditional ball-throwing [14], [15] in the presence of confounding obstacles or joints impediments. Throwing is often formalized as a contextual policy problem where a movement primitive for throwing is synthesized conditionally on the desired target position [14], [15]. In the presence of obstacles however, the most 'natural' way to throw to a given target may be blocked. Nevertheless, there are multiple throwing movements that hit a given target. We show that the ability to model – and sample from a distribution of controllers allows the robot to find throwing controllers that can avoid any given motion constraint. A preliminary version of this work appeared in [16].

II. Related Work

Learning Robot Control Typical approaches to learning robot control include learning by demonstration [17], reinforcement learning to maximize some extrinsic reward [14], or demonstration-based initialization followed by policy search-based reinforcement learning to fine-tune the demonstrated policy. Those policy search algorithms in turn can often be categorized into gradient-based [18], [19] and gradient-free methods such as Bayesian optimization [20] and evolutionary search [21]. An advantage of evolutionary methods is that they often provide a population of policies as a byproduct, rather than a single best controller.

Where obstacles can impede behavior, a standard robotics approach is to localise the obstacle and plan a movement that avoids it [22]. However this requires both (i) accurate 3D obstacle localisation and (ii) appropriate adaptive planning capabilities. One or other of these sensing and reasoning capabilities may not be available at the required efficacy level at a given developmental stage in an animal or robot. In contrast generating diverse behaviors and exploring them until one works has lower prerequisites and hence is suitable for earlier developmental stages.

Behavioral Diversity in Robot Control For a robot to be able to deal rapidly with new and unanticipated situations, a recently proposed approach consists of building a large repertoire of behaviors in which it should be possible to find one adapted to a newly arising situation or environment. The repertoire creation step can be done in a preliminary phase and a learned repertoire subsequently used to accelerate the adaptation to an unanticipated situation by relying on a selection process instead of a full learning process [4]. Promoting behavioral diversity is a key feature of a repertoire creation process. Driven by research on novelty search [23], evolutionary approaches have been adapted to generate a behaviorally diverse set of solutions instead of converging to a single solution optimizing a given fitness function [5]. These algorithms are called Quality Diversity algorithms [24], [25] and are used here to bootstrap the proposed method. Our proposed GPN builds on QD-search by leveraging its results as training data. However, in contrast to the selection-from-repertoire paradigm of QD, it has several interrelated benefits: (i) We can more compactly store a large repertoire by storing instead the parameters of a generative model that represents that repertoire of behaviors. (ii) Rather than a fixed size database of behaviors, the generative model can continue to sample unlimited new behaviors until a suitable one is found. (iii) Samples drawn from the generative model of behaviors can discover novelty beyond the initial training repertoire, by combining aspects from different training behaviors. (iv) Importantly the GPN approach is better suited for contextual policies. To solve a contextual policy task like diverse throwing to different targets, a library-based approach increases the required data collection and repertoire storage size dramatically because it would need to keep samples of many different throwing targets, and for each of those targets, samples of many different ways to throw there. In contrast, given a few samples of different throwing targets, a contextual policy GPN can extrapolate and draw many different controllers for throwing to any given target.

Generating Diverse Policies Another somewhat related work [26] covers diverse policy generation in a model-based framework. DIAYN [26] learns diverse skills (policies), assessing the diversity by the variety of states they visit in the process of RL-style unsupervised exploration. The main difference is that DIYAN tries to learn a small set of very distinct skills. While our GPN focuses on one skill type, but learns an infinite smoothly-varying manifold of controllers covering both all the potential goals (e.g., movement targets) and ways to achieve those goals. DIYAN also focuses more on initial exploration (and is thus analogous to QD-search in our pipeline), while we focus on compactly representing and exploiting the results of such an exploration process.

Generative Adversarial Networks Generative Adversarial Networks were proposed [12] to address the challenge of learning a neural network-based generative model for complex high-dimensional data. The key idea being that generator training is enabled by a second discriminator network that is simultaneously adversarially trained to distinguish true training data and the generator's synthetic examples. To improve its ability to fool the discriminator the generator must generate increasingly realistic synthetic samples. There have since been numerous extensions including convolutional GANs [13], conditional GANs [27], [28], disentanglement and interpretable latent codes [29], and improvements of GAN training stability with regards to challenges such as non-convergence and mode-

collapse [30], [31].

Generative Adversarial Network Applications The vast majority of GAN applications are in image generation tasks [12], [13], [32], [30], [28]. In robotics, GANs have been applied in robot haptic recognition [33]. Autoencoding VAE-GAN has been used for visual representation learning to process visual input in support of vision-based actuation in control [34]. The most related application of GANs has been in an imitation or inverse reinforcement learning context by GAIL and InfoGAIL, [35], [36]. GAIL trains a *single policy* by imitation learning by matching generated and demonstrated state-action pairs distribution. In contrast, GPN trains a *distribution over policies* by matching generated and demonstrated policy distribution. Furthermore, all GPN distribution generation is target-conditional, while GAIL has no conditioning. InfoGAIL extends GAIL to a multiple expert setting, training a *small discrete set of policies* by matching a generated policy distribution conditioned on a discrete latent variable, and a demonstrated set of policies. In contrast GPN trains a *continuous distribution over policies* by distribution matching. GPN enables conditioning on a continuous variable such as target, while InfoGAIL conditions by selecting a discrete policy. To our knowledge neural network generators have not previously been applied to sample diverse continuously distributed control policies, or to the generation of diverse robot behaviors, as we explore here.

III. Method

A. Background: Generative Adversarial Networks

Generative adversarial networks are neural network generative models for complex high-dimensional data. In this framework, a generator G is trained to produce samples representative of a training data distribution $p_{data}(\mathbf{x})$. G takes as input a random noise vector $\mathbf{z}$, and for a given noise distribution $p(\mathbf{z})$, samples $\mathbf{x} = G(\mathbf{z})$, $\mathbf{z} \sim p(\mathbf{z})$ should follow the same distribution as the observed data $\mathbf{x} \sim p_{data}(\mathbf{x})$. Such generative neural networks are challenging to train, but [12] showed that they can be trained via a min-max game between the generator and an adversary (the discriminator) D:

$$\begin{aligned}\min_G \max_D V(G,D) = E_{\mathbf{x}\sim p_{data}(\mathbf{x})}[\log D(\mathbf{x})] \\ + E_{\mathbf{z}\sim p(\mathbf{z})}[\log(1 - D(G(\mathbf{z})))]\end{aligned} \tag{1}$$

where $\mathbf{x}$ stands for a data example, $\mathbf{z}$ a random noise vector, $D(\mathbf{x})$ represents the discriminator's estimate of the probability that $\mathbf{x}$ came from real data rather than the generator. and $D(G(\mathbf{z}))$ - a probability that data came from a generator. GANs can also be extended to model the distribution of data conditional on some observed context vector [13], in which case both the generator and the discriminator also take the conditioning data $\mathbf{c}$ as input. GANs are most commonly applied to generate images (e.g., $\mathbf{x}$ is a person image and $\mathbf{c}$ is the gender of that person). In the following we adapt them to generate policies $\mathbf{x}$ conditional on goals $\mathbf{c}$.

B. Generative Policy Networks

Unconditional Policies Robot behaviors are defined by a control policy π operating in some state space $\mathscr{S}$ and action space $\mathscr{A}$. Thus while generative models are conventionally used to define a distribution $p(\mathbf{x})$ over data instances $\mathbf{x}$, our GPN defines a distribution $p(\pi)$ over policies π, which are themselves functions $\pi : \mathscr{S} \rightarrow \mathscr{A}$. Given a set of training policies $D_{train} = \{\pi_i\}$, we train our GPN to estimate the distribution over observed policies.

Assuming the policies in question lie in some parametric family, then each is identified by some parameter vector (e.g., weights in a neural network [3], radial basis function (RBF) kernels in a dynamic movement primitive (DMP) [17], [15]). By training a generator G to generate such parameters, samples from the generator are interpretable as controllers. In this case the discriminator enables the training of the generator by learning to distinguish between real policies in D_{train} and generator synthesized policies $\pi = G(\mathbf{z})$. Once the generator learns to fool the discriminator, and assuming it does not mode collapse, then samples from the generator represent diverse control policies that are novel yet statistically indistinguishable from the training policies. We denote sampling policies from the distribution implied by the generator $G(\mathbf{z})$ under a given noise distribution $p(\mathbf{z})$ as $\pi \sim p_G(\pi)$.

Contextual Policies We aim to go beyond simple fixed behaviors to work with contextual policies that are parameterized by a goal condition to achieve [14], [17]. For a goal directed policy such as our intended application of robot throwing, we need not just a controller (e.g., a throwing movement), but a conditioning mechanism [27], [29] that generates a controller that achieves the right goal (e.g., a throwing movement that hits a specific target). As described above, if the training set D_{train} consists of controllers throwing to multiple different locations, then sampled policies $\pi \sim p_G(\pi)$ will throw to new locations within the distribution of training targets. If the training set D_{train} consists of multiple controllers that throw in different ways to the same location, then $\pi \sim p_G(\pi)$ will sample novel policies that throw to that same location. In the contextual policy case we want a policy that achieves a specifiable goal. Thus we define a conditional generator $\pi = G(\mathbf{z}, \mathbf{c})$, $\pi \sim p_{G(\mathbf{c})}(\pi)$ to sample policies π that target a specific landing point $\mathbf{c}$. Thus we can both throw at a specified target (set the generator condition), and also find multiple ways to throw there (sample the generator).

C. Application to Throwing

For application to throwing, we assume a set of training policies, and denote sampling these as $\pi \sim p_{data}(\pi)$ and $\pi, \mathbf{c} \sim p_{data}(\pi, \mathbf{c})$. We then train a conditional generator network $G(\mathbf{z}, \mathbf{c})$ as below. The third term is an added regularizer that requires the generator to reconstruct the landing point of the policy that it just sampled. Here $G(\mathbf{z}, \mathbf{c})_T$ means sample the policy and its target point and take only the target point term, and $G(\mathbf{z}, \mathbf{c})_{\neg T}$ means the opposite.

$$\begin{aligned}\min_G \max_D V(G,D) = E_{\pi,\mathbf{c}\sim p_{data}(\pi,\mathbf{c})}[\log D(\pi, \mathbf{c})] \\ + E_{\mathbf{z}\sim p(\mathbf{z}),\mathbf{c}\sim p_{data}(\mathbf{c})}[\log(1 - D(G(\mathbf{z}, \mathbf{c})_{\neg T}))] \\ + \|G(\mathbf{z}, \mathbf{c})_T - \mathbf{c}\|_2\end{aligned} \tag{2}$$

Policy Representation We have applied our framework suc-

cessfully to many different policy representations including sampling the RBF parameters of the forcing term of a DMP, but we found the following simple representation effective and easy to tune. For our Baxter robot arm, we represent the π as a 15D vector defining a high-level open-loop controller in terms of the ball release time, and effector position and velocity at release. Specifically $\pi = [\boldsymbol{\theta}_{t_T}, \dot{\boldsymbol{\theta}}_{t_T}, t_T]$, where $\boldsymbol{\theta}_{t_T}$ and $\dot{\boldsymbol{\theta}}_{t_T}$ are 7D robot arm joint angles and joint velocities at launch time, and t_T is the launch time.

The goal condition $\mathbf{c}$ is a 2-dimensional Cartesian coordinate of the ball landing point. There are multiple launch configurations as described above, that result in the same landing point $\mathbf{c}$, and the trained GPN will sample this space of configurations.

Policy Execution With the policy definition above, samples from our GPN constitute a high-level action plan of how to launch the ball. To actually actuate this we map the high-level action into an open loop controller for low-level actuation via the following third-order polynomial function of time:

$$\boldsymbol{\theta}_{t_i} = \alpha_4 \left(\frac{t_i}{t_T}\right)^3 + \alpha_3 \left(\frac{t_i}{t_T}\right)^2 + \alpha_2 \left(\frac{t_i}{t_T}\right) + \alpha_1 \qquad (3)$$

with $\dot{\boldsymbol{\theta}}_{t_i} = \frac{d\boldsymbol{\theta}_{t_i}}{dt_i}$ and $\ddot{\boldsymbol{\theta}}_{t_i} = \frac{d\boldsymbol{\theta}_{t_i}}{dt_i}$, and parameters:

$$v_{t_i} = \frac{1}{t_T}\left(3\alpha_4 \left(\frac{t_i}{t_T}\right)^2 + 2\alpha_3 \left(\frac{t_i}{t_T}\right) + \alpha_2\right),\ \alpha_1 = \boldsymbol{\theta}_{t_0},$$
$$\alpha_2 = \dot{\boldsymbol{\theta}}_{t_0} t_T, \quad \alpha_3 = 3\boldsymbol{\theta}_{t_T} - \dot{\boldsymbol{\theta}}_{t_T} t_T - 2\alpha_2 - 3\alpha_1,$$
$$\alpha_4 = \boldsymbol{\theta}_{t_T} - \alpha_1 - \alpha_2 - \alpha_3,$$

where $\boldsymbol{\theta}_{t_i}, \dot{\boldsymbol{\theta}}_{t_i}, \ddot{\boldsymbol{\theta}}_{t_i}$ are the positions, velocities and accelerations of joints at time t; $\boldsymbol{\theta}_{t_0}$ and $\dot{\boldsymbol{\theta}}_{t_0}$ are initial positions and velocities at time t_0 and t_T is the time of launch. We assume the starting robot arm configuration is the same for each trial. Baxter is then actuated by sending the above joint position, velocity and acceleration plan to a ROS control node.

D. Data Collection with QD Search

We describe the collection of data used to train our GPN. Please note that, our main contribution of GPN is agnostic to the specific method used to collect training data. In our experiments we use data collected by archive-based Quality Diversity (QD) search [37], following the principles of Novelty Search with Local Competition (NSLC) [10], throughout, although other data sources could be used. Specifically, we use the evolutionary QD search method [10] to find a set of genotypes (high level policies π) that have diverse behavior-space effects (e.g., arm trajectories and landing positions) when actuated, while being of high quality (low torque during the entire actuation). Specifically behaviour-space effect is measured by describing each motion in terms of an 11D phenotype feature consisting of landing point of the ball, and 3 sets of 3D arm position descriptors recorded during the throwing. Diversity is then measured via the negative exponential distance between a behaviour and the dataset so far. QD search then optimizes for a high quality an diverse dataset. Please see [38] for full details on QD data generation for throwing. Overall, we use use a realistic Baxter simulation (Gazebo) to obtain around 15,000 throwing episodes. Each training episode records the arm trajectory, ball trajectory, and ball landing point. Since landing points are continuous variables, there is only one trajectory per landing point.

E. Baselines for Comparison

Evolutionary Repertoire Baseline We use QD search to generate training data for our GPN as described above, and will exploit our trained GPN to solve a robust throwing task later. For quantitative comparison, we consider an alternative evolutionary-style approach to exploiting this data. Such a repertoire-based approach to robust throwing would treat the dataset as a large library (repertoire), and then solve a new task by selection from the repertoire [4], [5], [6]. To throw to a specific target, the closest memorized landing point is recalled, and the associated policy is executed. If an environmental change (e.g., an obstacle) causes that known solution to fail, a lookup can be performed to find and execute some other policy with approximately the same landing point, but potentially different arm/ball trajectory. The problem is that this scales badly: although a large number of throwing episodes (15,000) covers the space of landing points reasonably well, it is not enough to cover many diverse ways to throw to each individual landing point. So the lookup-based approach may fail to effectively find diverse ways to throw to a specific point.

Other Alternatives Besides QD, we also compare two general purpose alternatives to GPN for robust throwing. KDE: As a non-parametric alternative to our GPN, we define a target-conditional Kernel Density Estimation [39] model over the same QD-based training set used by our GPN, so $p(\pi|\mathbf{c})$ is a Gaussian mixture model. We can then sample this mixture instead of our GPN. BayesOpt: Bayesian Optimization [40] is an established approach to adaptive behaviors in robotics [20]. We use the the best QD-trajectory for the given targets as the starting condition, and then perform Bayesian optimization for 10 trials for direct comparison to the diversity-based models.

IV. Experiments

A. Training data and settings

We apply our GPN to enable a Baxter robot to robustly throw a ball in different environmental obstacle conditions, as well as within various joint impediment scenarios. Since the Baxter arm has 7 joints, there are 15 parameters for any policy (Eq. 3) – position and velocity for each joint and the launch time. The training data is illustrated in Figure 2 in terms of a heat map of ball landings at different positions on the floor around the robot (left), and some example training episodes represented by their arm trajectories, ball trajectories and landing points (right).

Settings Our GPN is built upon DCGAN framework [13], and has a 4-layer RELU-activated convolutional architecture that maps a 100-dimensional noise vector $\mathbf{z}$ to a 15-dimensional output vector representing π. It is trained using 15000 episodes of data using learning rate 0.0002 and 1000 epochs with batch size 250. We used 20 generator updates for each discriminator update. Both QD and GPN use the same policy representation π, and underlying actuation strategy.

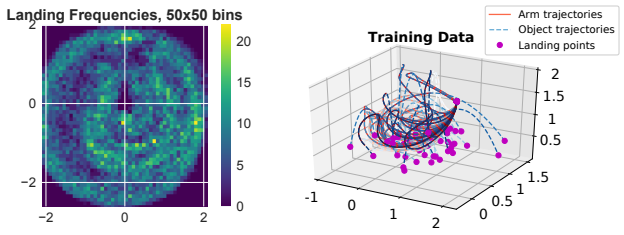

Fig. 2. Training data for throwing. Baxter robot is located at $(0,0)$. Left: Frequency of ball landing points at different floor positions (50×50 bins). Right: Example trajectories where red line is the end-effector, blue lines are the ball in flight, and magenta points are the landing points on the floor.

Evaluation Metrics We evaluate the methods with two sets of metrics. For evaluating simple throwing, we compute: **RMSE** between the target and actual landing point for all the trials; **Diversity** of the trials by taking the ball trajectory, computing equidistant waypoints along it, and then using these to compute a standard deviation of all trajectories towards a given target; **Harmonic Mean** aggregates the other two metrics - accuracy $(1-RMSE)$ and diversity (standard deviation) in order to provide a single quantitative measure of performance. Note that harmonic mean aggressively penalizes failure in either metric.

When evaluating the ability of repertoires to perform robust throwing in the presence of obstacles, or with broken joints, we consider a trial as a success if the ball lands within radius τ of the intended target. Our metric is $SuccessesProportion(k,\tau)$: How many of the target coordinates does the ball hit successfully (within τ radius), at least k out of 10 times, when sampling from the repertoire? The idea is that even if obstacles block some particular throws, or a physical malfunction causes it to fail, a model that can generate multiple diverse behaviors that all solve the task (i.e., throwing trajectories that hit the same target) should be able to find at least some (i.e., k) successful solutions.

B. Experiment 1: Target-conditional throwing

Setup We aim to achieve robust throwing by learning to hit a target in diverse ways. We therefore first evaluate the ability of our GPN and QD alternative to: (i) accurately throw to a given position, and simultaneously (ii) find diverse ways of throwing to each position around the robot. For this purpose we grid the floor space around the robot into a 5×5 grid (25 target landing points). We experiment both in simulation (Gazebo Baxter) where we attempt to throw to each of those points 10 times, and then corroborate those results on the real Baxter where we throw to each coordinate 3 times, tracking the ball using an OptiTrack system. We compare results from our GPN with the standard evolutionary strategy that treats the GPN-training set as a repertoire library (Sec. IV-A).

Results The results in Figure 3 plot the landing error and diversity metrics at each grid point on the floor around the robot. The first two columns compare QD/Library-based approach with our GPN in simulation; the third column evaluates our GPN results on the real Baxter robot. From the results of this experiment we make the following observations: (i) In general QD search has higher accuracy. This is expected as

Fig. 3. Throwing to a 5×5 grid of points on the floor around the robot located at $(0,0)$ and facing right. Local throwing accuracy (top), diversity (middle), and their harmonic mean (bottom) when throwing by QD trajectories in simulation, GPN sampled controllers in simulation, GPN controllers on a real Baxter robot. GPN generally has higher diversity, and better overall performance (harmonic mean).

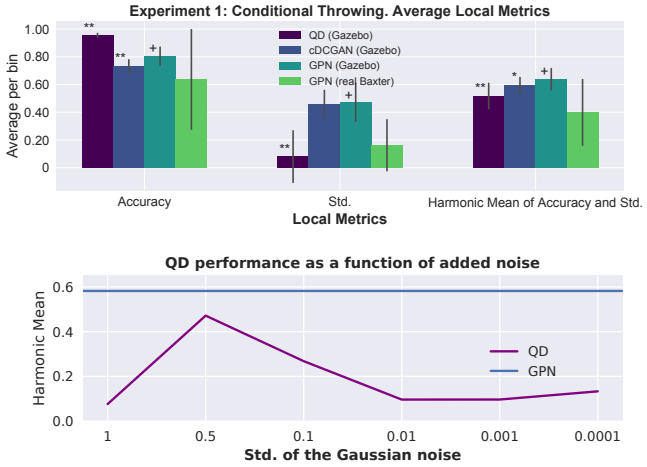

Fig. 4. Top: Summary statistics of throwing to all points on a 5×5 floor grid. The difference between harmonic means of the simulated data for QD and GPN is statistically significant according to unequal variances t-test with significance level $\alpha = 0.01$ $(p_{value} < 2.78\cdot10^{-12})$ [2]. Bottom: Comparison to QD with varying levels of added noise.

it is simply recalling previously memorized movements and replaying them exactly, which unsurprisingly leads to very similar outcomes, and hence high accuracy. In contrast our GPN is a predictive model that must infer the right policy to throw to any given target point, so its slightly lower accuracy is understandable. (ii) However, GPN has much higher diversity. It models the distribution of policies that throw to a conditioning target, and samples that distribution for each trial. (iii) Aggregating these metrics via harmonic mean, we see that our GPN performs favorably compared to QD. (iv) When executing our GPN-sampled controllers on the physical Baxter robot, the results are comparable to the simulated case (third vs second column). Note that the grey areas to the left of the map on the results of the real robot are because of walls in the physical Baxter environment preventing the data collection in these regions.

These results are summarized over all the spatial coordinates in Figure 4 (top), where GPN significantly outperforms QD in terms of harmonic mean. To be as fair as possible to the QD search alternative, we also considered boosting its diversity by adding Gaussian noise to the executed policies at each trial. The result in Figure 4 (bottom) shows that noise can improve QD performance. However it must be carefully tuned as too much noise quickly degrades QD's accuracy. Overall this result is understandable as uniformly adding noise to known throwing behaviors can quickly move off the manifold of good throwing policies. In contrast GPN learns the distribution over good throwing policies so it can sample novel throwing controllers from within that distribution. Finally we mention that, as per common safety practice, all our movement plans are checked for self-collision before execution. We also note that despite lacking a model of robot kinematics, the vast majority (98.4%) of the diverse plans generated by GPN are collision free. This indicates that GPN has also learned about the manifold of reasonable controllers in the sense of non-colliding as well as ability to hit a target.

C. Experiment 2.1: Target-conditional throwing with obstacles: GPN vs QD

Setup Our motivating scenario was to use the learned conditional distribution over controllers to achieve robust throwing in the presence of obstacles. In this experiment, we evaluate this quantitatively using Gazebo Baxter simulator. Specifically, we consider a 5×5 grid of floor targets as before, and we throw to each of these targets with 10 diverse sampled controllers as before. For each of those throws, we simulate obstacles and calculate whether an attempted throwing trajectory fails due to robot or ball collision with the obstacle. To systematically explore these issues we run the simulation for $k = 1 \ldots 9$, $\tau = 0, 0.1, \ldots, 1.0$, and repeat for different occlusion rates $= 1\%, \ldots, 8\%$ when calculating $SuccessesProportion(k, \tau)$. For simplicity we model occlusions as a randomly selected set of inaccessible floor areas, where the total proportion of blocked floor area is the specified occlusion rate. We compute results averaging over a 5×5 target grid, 10 throws per target, and 1000 maps with random obstacle scenarios.

Results Figure 5 (right) shows the Gazebo simulation of Baxter attempting to throw to a specific target in the presence of randomly generated obstacles. Figure 5 (left) shows heatmaps of $SuccessesProportion(k, \tau = 0.2)$ for various occlusion rates and minimum hit requirements k. From these we can make the following observations: (i) Both QD and GPN methods have higher success rate in the easier bottom left (low occlusion, low hit ratio required for success), and vice-versa in the harder top right. (ii) The GPN result is much higher than that of QD for low k values (e.g., $k = 1$). This means that the GPN can often find at least one way to hit the target, for this whole range of occlusion rates. (iii) At very high minimum hit (e.g., $k = 9$) QD performance is slightly better than GPN. This is because GPNs slightly lower accuracy means that it's rarely the case that as many as 9 out of 10 attempts hit the target. However, at this stringent hit rate requirement, we note that the success rate of QD in absolute terms is also very low (around 10%). Finally, Figure 5 (middle) shows the success rate averaged over occlusion rates as a function of different hit-radius requirements. We see that for a stringent accuracy requirement ($\tau < 0.1$), neither method succeeds. While for all larger values of τ, GPN consistently outperforms QD in success rate. Overall the results validate our goal: GPN can throw accurately enough that it often hits the target, but it does so in diverse enough ways that at least one way can usually be found to dodge any given obstacle configuration.

[2]Symbolic representations of significance levels in this paper are as follows: '+' stands for the method or parameter against which the rest are compared (usually GPN for methods or the parameter we are using for the parameter sensitivity assessment), '*' stands for significance level $\alpha = 0.05$, and '**' - for $\alpha = 0.01$. Grey bars in all the bar-plots represent the standard deviation.

D. Experiment 2.2: Target-conditional throwing with obstacles: Baseline Comparisons

Setup In this experiment, we compare two further alternative approaches to obstacle-robust throwing: KDE [39] and BayesOpt [40] (Section III-E). We use a simulated setup where there is a wall randomly placed between Baxter and the target (see supplementary video and Section IV-E).

Results The results in Figure 6 average over four random goal/wall positions and compare the different methods in terms of error, diversity, collision rate and $SuccessesProportion(k = 3, \tau = 0.2)$. We see that while GPN is not the most accurate model, its success rate is best overall. This is because (i) it has good diversity enabling it to dodge obstacles more often, (ii) it has learned the manifold of reasonable policies, so it usually also avoids self-colliding or unsafe movements. In contrast, KDE and BayesOpt often generate self-colliding movements or hit the obstacle. Kernel Density Estimation suffers from being an inefficient/inaccurate model of relatively high-dimensional (15D) policies. Bayesian Optimization purposefully adapts the actuator movement to avoid the obstacle, but cannot succeed in the relatively small number (10) of available trials.

E. Experiment 2.3: Physical Baxter robust conditional throwing with obstacles

To test our method, we sample the conditional GPN ten times to generate ten diverse controllers that should throw to the required point. We simulate them to check for collisions with the obstacle, and found three of these avoided collision and landed into the basket in simulation. This validates that the GPN has indeed learned to generalize and samples novel behaviors. Figure 7 shows the successful execution of two of these controllers. We can see that the ability to generate diverse trajectories enables the robot to successfully hit the required target with the ball while dodging the given obstacle.[3]

[3]For a video of this experiment please refer to the supplementary material: https://youtu.be/2LCnaa89erM

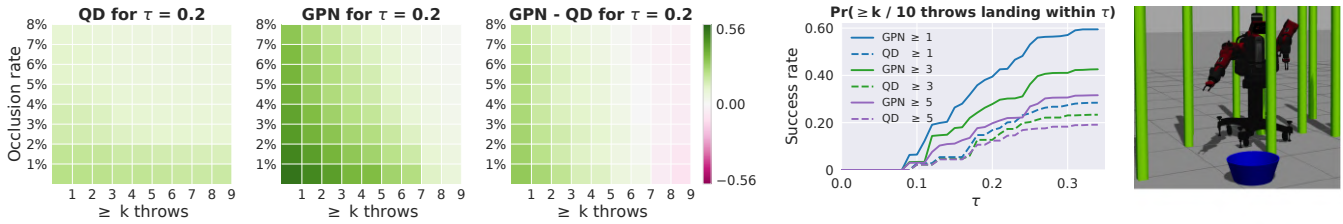

Fig. 5. Robustness of target-conditional throwing in obstacle-occluded environments. Left: Heat maps illustrate the probability of at least k of 10 throws landing within $\tau = 0.2$ of the target for different levels of occlusion. QD lookup method. Our GPN method. Difference between GPN and QD result. Middle: Success rate for varying accuracy requirements τ. Right: Example of the random obstacle environment.

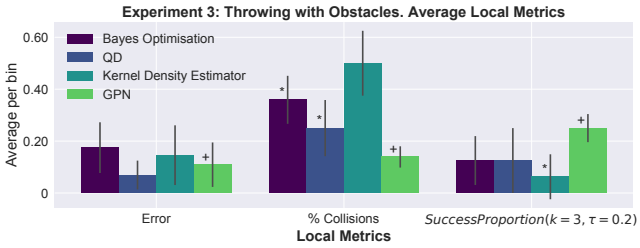

Fig. 6. Target-conditional throwing robust to obstacles. Comparison of GPN, QD, Bayesian Optimisation, and Kernel Density Estimation approaches. GPN is not the most accurate model, however is very low on collisions due to the diversity of its policies. Its greater diversity translates into higher *SuccessProportion* due to better dodging of random obstacles compared to other methods. [2]

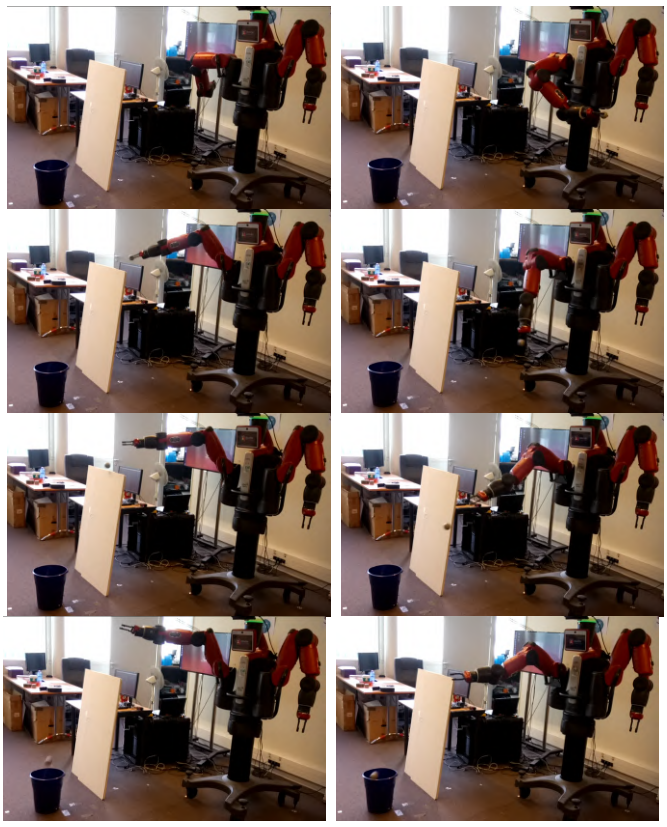

Fig. 7. Two examples of obstacle robust throwing behaviors obtained by sampling our learned distribution over policies. The GPN is conditioned on the target location, and samples controllers for throwing there until samples are drawn that generate neither robot nor ball collisions. For a video of this experiment please refer to the supplementary material: https://youtu.be/2LCnaa89erM

F. Experiment 3: Throwing with damaged joints

Setup Another potential motivation for modeling behavioural diversity is to enable a robot to adapt its behaviour in response to physical damage [4]. In this experiment we explore this idea by evaluating the ability of GPN to find a behaviour that enables Baxter robot to complete its throwing task despite a hardware failure in one or more joints. There are various ways in which a joint can be damaged, for example: being jammed in a fixed position (unlikely in practice), full or partial joint sensor or actuator failure causing complete or semi-random motion. We are focusing on the latter. To simulate hardware sensor failure for a selected joint, random uniform noise is inserted to replace the planned velocity while performing a movement. This event would cause many (but perhaps not all) potential throwing plans to fail. So the aim is to show that GPN can generate diverse enough behaviours that at least some attempts succeed despite the hardware malfunction.

As before, the quantitative assessment has been run using Gazebo Baxter simulator. In this experiment Baxter is operating on a fixed subset of 5 targets from a 5×5 grid of floor targets as described in subsection IV-D. The controllers are sampled by the GPN from the target-conditional distribution learned from the training data. Baxter throws to each of the floor targets with 10 different controllers sampled by the GPN. For comparison we conduct the same set of experiments using the behavioural library assembled with the QD-search. The assessment metrics are near-identical to the subsection IV-C.

We assess the $SuccessesProportion(k, \tau = 0.2)$ in terms of how many throws can be land within τ distance of the desired target. The value of τ for the first set of results is selected arbitrarily and there are the further results are provided, demonstrating how the $SuccessesProportion(k, \tau)$ changes with respect to τ. In this experiment the *SuccessesProportion* is expressed in terms of the of the number of successful throws for each individual damaged joint scenario. The Baxter robot has 7 joints, (denoted 1 to 7, where 1 is the closest to the body and 7 the closest to the end-effector, Figure 8, left). We repeat this experiment, simulating each joint being broken in turn, as well as some combinations of broken joints.

Results: Joint-wise Results are presented in Figures 8 and 9. First of all, similar to Figure 5 for the obstacle occlusion experiments we compare QD and GPN's ability to deal with impediments, in this case individual damaged joints. Evidently from Figure 8 heat-maps of $SuccessesProportion(k, \tau = 0.07)$, there is some variability in how well each model deals with individual broken joints, e.g., QD is relatively weaker for joint

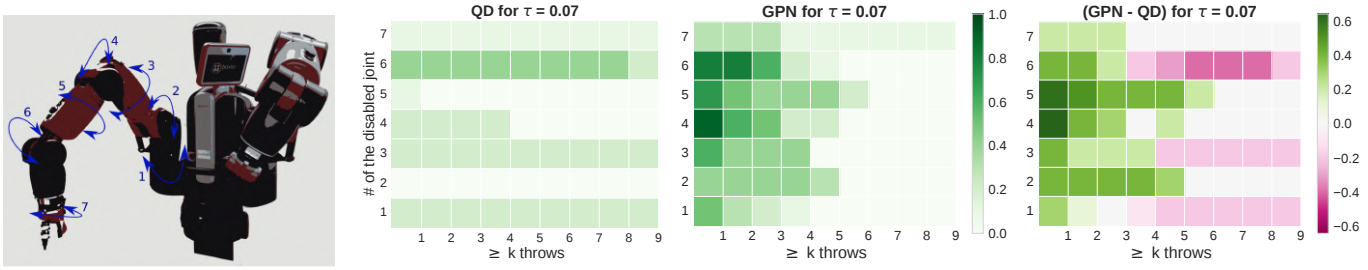

Fig. 8. Robustness of target-conditional throwing broken down by damaged joint. Left: the numbering of Baxter actuator joints. Here and in all the previous experiments Baxter only uses its right arm for throwing. Since it is symmetric, results should be the same for either side. Right: Heat maps illustrate the probability of at least k out of 10 throws landing within $\tau = 0.07$ of the target for different individual joints impeded. The first heat-map corresponds to the QD library lookup method, the second - to our GPN method, third shows the difference between the GPN and QD results. Our GPN sampling is equal or better than QD lookup for the majority of the scenarios, with the exception of larger k values for joints 1, 3, and 6. (Picture of Baxter (left) comes from [41].)

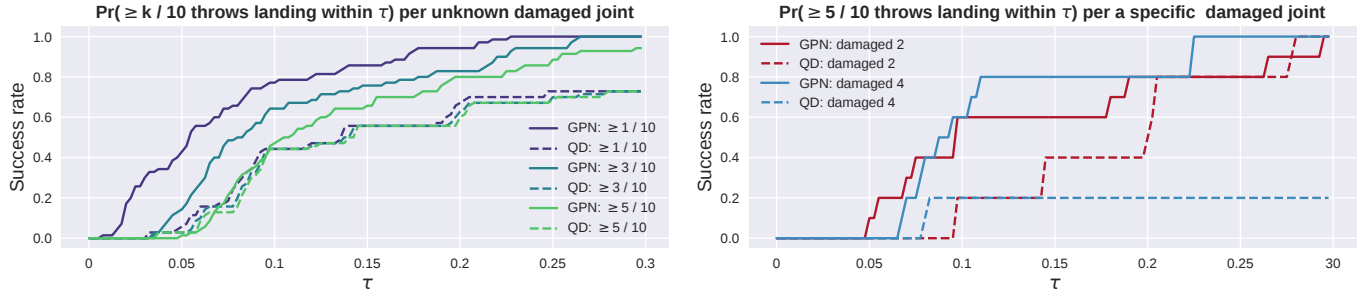

Fig. 9. Left: Throwing success rate with respect to the allowed distance from target τ, where 'success' counts as at least $k = 1, 3, 5$ minimum number of hits (out of 10) within radius τ. Average result across all single joint failures. Right: the success rates of half of the throw landing within τ of the target, for example damaged joints 2 and 4.

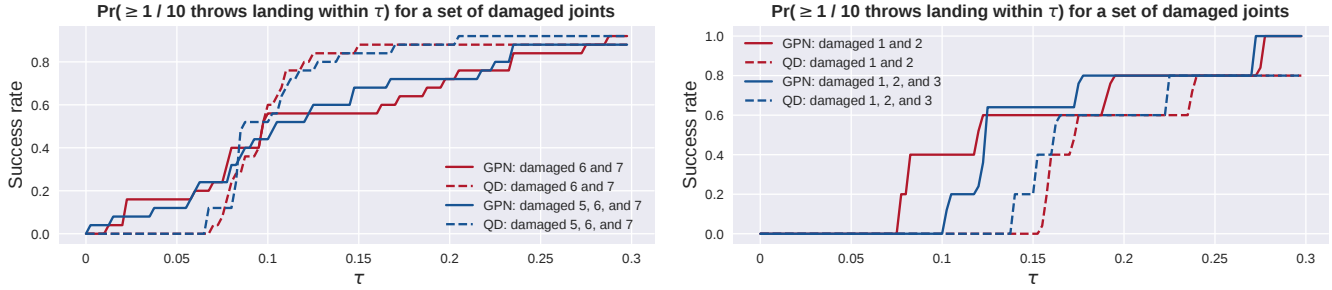

Fig. 10. Robust target-conditional throwing with multiple actuator joints disabled. Left: damaging 3 last actuator joints, one after the other, starting with the one closest to the end-effector. There is not much difference between the compared methods in this case: For very small values of τ, i.e. for a very precise throwing requirement our GPN is slightly better. However, for the less precise throwing requirement, QD is slightly better. Right: "damaging" three first actuator joints, starting with the closest one to the body. The data suggests that the GPN-sampled policies usually outperform the QD ones.

2 and 7; GPN is weaker for joint 1. Nevertheless, in aggregate the GPN-sampled controllers are more often successful than QD ones, especially for the lower values of required minimum successful hits. The QD policies, however precise in the initial environment, provide no diversity of execution per fixed target. The diversity of the GPN policies translates into a higher chance of at least one out of ten of its policies overcoming the difficulty presented by an impaired joint.

Further, the following observations can be made: (i) the QD library is very evenly successful (or otherwise) over k for each damaged joint. This can be attributed to its high precision and low diversity - if the chosen trajectory happens to not be affected by the damaged joint very much and hits the target, it is likely to hit it for all 10 attempts. Unlike the averaging over the obstacle occlusion scenarios the stochasticity of the broken joints appears to represent a much more uniform constraint for the QD library. Thus, leading to performance less dependent

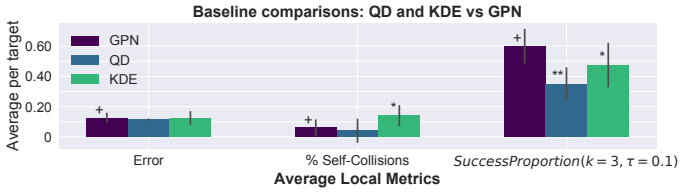

Fig. 11. **Baseline Comparison.**[2] Comparison of GPN, to QD and KDE, averaged across the multiple targets and damaged joints cases. All three have comparable Error rates, however the ratio of self-collisions is noticeably higher for KDE. The better success-rate of the GPN comes from the higher diversity of its controllers, overcoming the joints impediments.

on the minimal number of throws required. (ii) In contrast, the GPN offers a set of diverse solutions - so one of them being successful does not guarantee the success of the other ones - leading to relatively weaker performance at high k. (iii) Despite a small subset of scenarios where the QD library is

more efficient, GPN shows comparable or better performance in the majority of cases, especially for the smaller amount of the minimal successful hits required (Figures 8, right). In principle, if one is not limited in the number of attempts, the GPN can be thus sampled until success is achieved.

The previous results are for an arbitrarily chosen accuracy threshold, τ. We next present a different view of the results by fixing the required hits k and varying the required accuracy τ. The plots in Figure 9 (right) show the probability of at least half of the executed throws landing in the basket, for a few example joints.

Results: Aggregate Finally, we present aggregate results averaged over all the potential damage points. Figure 9 (left) shows success rate as a function of accuracy threshold τ for some hit-rate requirements $k = 1, 3, 5$, when averaged across all the affected joint cases. Across all accuracy thresholds τ, the GPN policies are overall equally or more successful than QD at functioning successfully despite broken joints. In summary, the results confirm that a GPN-based diversity strategy can successfully overcome joint failures for most of the scenarios considered. This in turn demonstrates the generalisation of GPN beyond the QD library upon which it was trained.

Baseline comparisons As in the experiments with the obstacle-occluded environment, we compare our GPN to alternative approaches in terms of aggregate statistics. Besides the library-based QD, we also consider sampling the Kernel Density Estimate defined over the same QD-based training set used by our GPN. The results presented next come from the same experiment above, averaging across the variety of damaged joint situations. As before we assess the methods in terms of the of error, diversity, collision rate and $SuccessesProportion(k = 3, \tau = 0.1)$. We conduct this experiment for one damaged joint at a time, performing 10 throws each to a selection of target points. Finally we compute the average of each metric over all the target points and damaged joints.

The results in Figure 11 suggest that for throwing with an impeded joint, the average accuracy of the models is very similar. KDE exhibits a slightly higher self-collision rate compared to the other two. And our GPN has the better Success Rate, facilitated by the higher diversity of its policies.

Multiple Damaged Joints A more challenging scenario is one in which multiple joints are simultaneously affected. We evaluate two settings. First is "damaging" the joints from the wrist joint 7 (in practice the most frequent to get damaged) up to the joint 5. Figure 10 (left) suggests that there is no clear difference between the compared methods. The second setting, is "damaging" joints starting with the joint 1 and going up to the joint 3. The results presented in Figure 10 (right) show that the GPN is usually equally or more successful than the QD in overcoming the multi-joint failures for the joints closer to the body.

G. Further Analysis

We finally evaluate the sensitivity of GPN to various hyper-parameters. Unless stated otherwise, the results in this section are averaged across test runs of 250 trajectories - 10 throws to 25 different targets, where targets remain the same for all the models to ensure the fair comparison[4].

Noise Parameters GAN-type models such as our GPN uniquely exploit a noise source to generate their diversity. We evaluate the impact of two parameters: the dimensionality and distribution of our latent noise source. Our experiments thus far were carried out with a $100D$ noise vector. Figure 12 (left) shows the results obtained from the same GPN configuration, but trained with noise inputs of various sizes. The results show that our 100D noise vector model exhibits the highest diversity, while maintaining one of the lowest error- and collision-rates. The larger 200D candidate also works well, implying that our model is not very sensitive to the size of the noise vector once it is above a minimum threshold. We also compared spherical versus uniformly distributed noise, but detected no clear difference in performance.

Training Epochs and Dataset Size Another factor to consider is the number of training epochs. The results in Figure 12 (right) show that performance generally improves with more epochs, suggesting that our GPN is converging to a good solution without overfitting or diverging. Notably, a sufficiently well-trained model produces nearly zero self-collisions.

Finally, we ask the question of how much data is necessary to train our model? The effect of training dataset size on the performance metrics is shown in Figure 13. The sizes compared are 1000, 5000, 10000, and 15000 (the total number of the training data available). Performance is best with the largest 15000 sample dataset. However, we can see that it degrades relatively slowly with the decreasing amount of data, suggesting that the large scale training data is not crucial for our framework[4].

V. Discussion

A. Summary

To summarise, the GPN deployment consists of (i) acquiring the initial behavioural repertoire, in our examples QD library, but can be acquired in other ways, (ii) target-conditional DCGAN for the generalisation of a contextual policy repertoire, (iii) upgraded with a regularisation on the predicted outcome.

The results show that the GPN is capable of generating significantly more diverse conditional throwing policies compared to the initial training data at a small cost of accuracy (Experiment 1). This enables the GPN to successfully perform in two major applications - obstacle avoidance robust throwing (Experiment 2) and robust throwing in unknown broken joints setting (Experiment 3).

In case of the obstacle avoidance, we declare a success if at least k/10 trials results in a hit. Here the GPN approach is clearly better when a smaller amount of successful hits is acceptable. The lookup-table of QD data outperforms it when at least 80% success level required (Figure 5), because it will just repeat the same looked-up trajectory 10 times, hence $\approx 80-90\%$ success, *if* that trajectory happens to be successful

[4]Please note that unlike in the rest of the paper these results are based on training the GPN for 100 training epochs (for speed sake). This is justifiable as there is no statistically significant difference between 100 and 1000 training epochs according to Fig. 12, right.

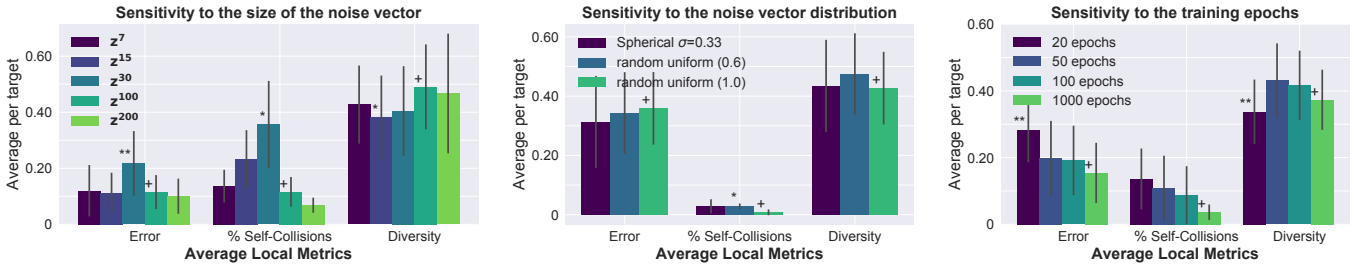

Fig. 12. **Parameter sensitivity.**[2] Left-to-right: **1.** Sensitivity of the GPN to the size of the generator noise vector in terms of the average local metrics (average error, ratio of self-collisions, and diversity of the trajectories).The GPN with 100D noise vector has been used for the rest of the results in this work, due to high diversity and low error. **2.** Sensitivity to distribution of the noise GPN noise vector. Both spherical and uniform random noise perform similarly, so we stick with the random uniform noise. **3.** GPN performance after various training epochs. The results confirm, that once trained sufficiently GPN produces next to no self-collisions.

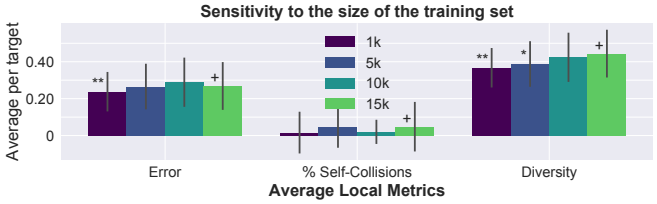

Fig. 13. **Dataset Size.**[2] Comparative performance of the same GPN configuration trained with datasets of different sizes. Despite the difference in the sizes of the training datasets, the performance is very comparable.[4]

in a particular obstacle setting. But in absolute terms both methods perform badly against this stringent requirement. These results are valid assuming no feedback is available about the result (hit or miss). If such feedback is available, the GPN could repeat its first successful trajectory in each setting, which will would further improve its success rates.

In the broken joints application, GPN similarly improves on QD when a smaller numbers of successful hits are required (Figure 8). It seems like QD policies are relatively more robust to certain joint failures (1, 3, and 6). However overall there is still an obvious benefit to using GPN in the settings of 1-3 unknown broken joints (Figures 8-12).

B. Significance

Overall our contribution fits in with and extends the successful existing line of work on learning repertoire-based robust behaviours [4], [5], [6], [38]. Rather than learning a fixed repertoire, where available diversity is set after training; we learn a conditional generative model for behaviours that can sample additional diversity online at run-time, and do so conditionally on a specified goal. This both allows increased robustness through greater available diversity, and potentially better scalability to very large repertoires due to carrying a parametric behaviour generator in place of an exhaustive behaviour library. Although we evaluated our technique on throwing, the methodology is generic and could be applied to any setting addressable by open-loop controller. Thus this provides a valuable tool that could help realise the vision of achieving robot robustness through behavioural diversity.

C. Future Work

In future, we intend to explore applying the proposed generative policy network framework for generating low-dimensional closed-loop controllers such as dynamic movement primitives [17] rather than our current open-loop controllers, and application to different kinds of tasks besides throwing [38]. Rather than relying on a fixed training set, we would also like to close the loop between generator learning and training set collection, and gather data more efficiently by leveraging the GPN's ability to condition on a desired target.

VI. Conclusion

We introduced the idea of generative policy networks, for defining a generative model over policies. We showed that our generative policy network provides a way to compactly encode a large set of known behaviors, and that sampling the GPN provides a way to draw unlimited novel controllers that are related-to but different-from known training behaviors. We showed how to apply this novel idea to provide an effective repertoire-based solution to challenging tasks including obstacle-robust throwing and joint-damage-robust throwing. This adds to the existing promising line of research on behavioural diversity and brings us one step closer to achieving robust behaviour through behavioural repertoire.

Acknowledgement

This work is supported by the DREAM project through the European Unions Horizon 2020 research and innovation under grant agreement No 640891, EPSRC grant EP/R026173/1, and NVIDIA Corporation GPU donation. We would also like to express our gratitude to Joshua Smith for the constant technical support provided throughout the project.

References

[1] G. Fagogenis, V. D. Carolis, and D. M. Lane, "Online fault detection and model adaptation for Underwater Vehicles in the case of thruster failures," in *ICRA*, 2016.

[2] K. Fang, Y. Bai, S. Hinterstoisser, S. Savarese, and M. Kalakrishnan, "Multi-Task Domain Adaptation for Deep Learning of Instance Grasping from Simulation," in *ICRA*, 2018.

[3] C. Zhao, T. M. Hospedales, F. Stulp, and O. Sigaud, "Knowledge Transfer Across Skill Categories for Robot Control," *IJCAI*, 2017.

[4] A. Cully, J. Clune, D. Tarapore, and J. B. Mouret, "Robots that can adapt like animals," *Nature*, vol. 521, pp. 503–507, 2015.

[5] A. Cully and J. B. Mouret, "Evolving a behavioral repertoire for a walking robot," *Evolutionary Computation*, vol. 24, pp. 59–88, 2016.

[6] M. Duarte, J. Gomes, S. M. Oliveira, and A. L. Christensen, "Evolution of Repertoire-Based Control for Robots With Complex Locomotor Systems," *IEEE Transactions on Evolutionary Computation*, vol. 22, pp. 314–328, 2018.

[7] F. Guerin, N. Kruger, and D. Kraft, "A survey of the ontogeny of tool use: From sensorimotor experience to planning," *IEEE Transactions on Autonomous Mental Development*, vol. 5, no. 1, pp. 18–45, 2013.
[8] E. Ugur, Y. Nagai, E. Sahin, and E. Oztop, "Staged development of robot skills: Behavior formation, affordance learning and imitation with motionese," *IEEE Transactions on Autonomous Mental Development*, vol. 7, no. 2, pp. 119–139, June 2015.
[9] D. Kraft, R. Detry, N. Pugeault, E. Baseski, F. Guerin, J. H. Piater, and N. Kruger, "Development of object and grasping knowledge by robot exploration," *IEEE Transactions on Autonomous Mental Development*, vol. 2, no. 4, pp. 368–383, Dec 2010.
[10] J. Lehman and K. O. Stanley, "Evolving a diversity of virtual creatures through novelty search and local competition," *GECCO*, 2011.
[11] A. Clark and A. Karmiloff-Smith, "The cognizer's innards: A psychological and philosophical perspective on the development of thought," *Mind & Language*, vol. 8, no. 4, pp. 487–519, 1993.
[12] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative adversarial networks," *NIPS*, pp. 2672–2680, 2014.
[13] A. Radford, L. Metz, and S. Chintala, "Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks," *ICLR*, 2016.
[14] A. G. Kupcsik, M. P. Deisenroth, J. Peters, and G. Neumann, "Data-efficient Generalization of Robot Skills with Contextual Policy Search," in *AAAI*, 2013.
[15] J. Kober, E. Oztop, and J. Peters, "Reinforcement Learning to adjust Robot Movements to New Situations," in *RSS*, 2010.
[16] M. Jegorova, S. Doncieux, and T. Hospedales, "Generative adversarial policy networks for behavioural repertoire," in *ICDL*, 2019.
[17] F. Stulp, G. Raiola, A. Hoarau, S. Ivaldi, and O. Sigaud, "Learning compact parameterized skills with a single regression," in *IEEE-RAS International Conference on Humanoid Robots (Humanoids)*, 2013.
[18] R. J. Williams, "Simple Statistical Gradient-Following Algorithms for Connectionist Reinforcement Learning," *Machine Learning*, 1992.
[19] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, "Continuous Control with Deep Reinforcement Learning," in *ICLR*, 2016.
[20] T. Lopez-Guevara, N. K. Taylor, M. U. Gutmann, S. Ramamoorthy, and K. Subr, "Adaptable Pouring: Teaching Robots Not to Spill using Fast but Approximate Fluid Simulation," in *CORL*, 2017.
[21] F. P. Such, V. Madhavan, E. Conti, J. Lehman, J. Clune, and K. O. Stanley, "Deep Neuroevolution: Genetic Algorithms are a Competitive Alternative for Training Deep Neural Networks for Reinforcement Learning," in *arXiv:1712.06567*, 2018.
[22] F. Stulp, E. Oztop, P. Pastor, M. Beetz, and S. Schaal, "Compact models of motor primitive variations for predictable reaching and obstacle avoidance," in *IEEE-RAS International Conference on Humanoid Robots*, 2009.
[23] J. Lehman and K. O. Stanley, "Abandoning objectives: Evolution through the search for novelty alone," *Evolutionary Computation*, vol. 19, pp. 189–222, 2011.
[24] A. Cully and Y. Demiris, "Quality and Diversity Optimization: A Unifying Modular Framework," *IEEE Transactions on Evolutionary Computation*, vol. 22, no. 2, pp. 245–259, 2018.
[25] J. K. Pugh, L. B. Soros, and K. O. Stanley, "Quality Diversity: A New Frontier for Evolutionary Computation," *Frontiers in Robotics and AI*, vol. 3, p. 40, 2016.
[26] B. Eysenbach, A. Gupta, J. Ibarz, and S. Levine, "Diversity is all you need: Learning skillswithout a reward function," *ICLR*, 2019.
[27] M. Mirza and S. Osindero, "Conditional Generative Adversarial Nets," *arXiv*, 2014.
[28] S. Reed, Z. Akata, X. Yan, L. Logeswaran, B. Schiele, and H. Lee, "Generative Adversarial Text to Image Synthesis," *ICML*, 2016.
[29] X. Chen, Y. Duan, R. H. nad John Schulman, I. Sutskever, and P. Abbeel, "Infogan: Interpretable representation learning by information maximizing generative adversarial nets," in *NIPS*, 2016.
[30] L. Arjovsky, M.; Chintala S.; Bottou, "Wasserstein GAN," *ICML*, 2017.
[31] T. Salimans, I. Goodfellow, W. Zaremba, V. Cheung, A. Radford, and X. Chen, "Improved Techniques for Training GANs," *NIPS*, 2016.
[32] I. Gulrajani, F. Ahmed, M. Arjovsky, V. Dumoulin, and A. Courville, "Improved Training of Wasserstein GANs," *NIPS*, 2017.
[33] Z. Erickson, S. Chernova, and C. C. Kemp, "Semi-Supervised Haptic Material Recognition for Robots using Generative Adversarial Networks," *CORL*, 2017.
[34] R. Rahmatizadeh, P. Abolghasemi, L. Bölöni, and S. Levine, "Vision-Based Multi-Task Manipulation for Inexpensive Robots Using End-To-End Learning from Demonstration," in *ICRA*, 2018.
[35] J. Ho and S. Ermon, "Generative Adversarial Imitation Learning," in *NIPS*, 2016.
[36] Y. Li, J. Song, and S. Ermon, "Infogail: Interpretable imitation learning from visual demonstrations," in *Advances in Neural Information Processing Systems 30*, I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, Eds., 2017.
[37] A. Cully and Y. Demiris, "Quality and diversity optimization: A unifying modular framework," *IEEE Transactions on Evolutionary Computation*, vol. 22, no. 2, pp. 245–259, 2018.
[38] S. Kim, A. Coninx, and S. Doncieux, "From exploration to control: learning object manipulation skills through novelty search and local adaptation," *arXiv preprint arXiv:1901.00811*, 2019.
[39] M. Rosenblatt, "Remarks on Some Nonparametric Estimates of a Density Function," *Ann. Math. Statist.*, vol. 27, no. 3, 1956.
[40] D. R. Jones, M. Schonlau, and W. J. Welch, "Efficient Global Optimization of Expensive Black-Box Functions," *J. of Global Optimization*, vol. 13, no. 4, pp. 455–492, dec 1998.
[41] N. Jaquier, "Improving the drawing skills of a humanoid robot with visual feedback," Master's Thesis, École Polytechnique Fédérale de Lausanne, 2016.

Marija Jegorova is a post-doctoral Research Associate at School of Engineering at the University of Edinburgh. As a part of the DREAM FET H2020 project (2014 - 2018), her research has been focused around realistic artificial data synthesis for machine learning applications to robotics and automation. Current research interests extend to privacy in machine learning, specifically membership attacks for generative models in differential privacy setting.

Stephane Doncieux is Professor in Computer Science at ISIR (Institute of Intelligent Systems and Robotics), Sorbonne University, CNRS, in Paris, France. He is deputy director of the ISIR, a multidisciplinary robotics laboratory with researchers in mechatronics, signal processing computer science and neuroscience. Until that date, he was in charge of the AMAC multidisciplinary research team (Architectures and Models of Adaptation and Cognition). He was coordinator of the DREAM FET H2020 project from 2015 to 2018 (http://robotsthatdream.eu/). His research is in cognitive robotics, with a focus on learning and adaptation with a developmental approach.

Timothy Hospedales is a Professor at the University of Edinburgh. He is Associate Editor of TPAMI, and has served as Area Chair of several major events (ICCV, ECCV, CVPR, AAAI, IJCAI) and Program Chair of BMVC 2018. His work has been funded by UK EPSRC and the European Commission, and has led to over 75 publications in major venues, as well as best paper awards or nominations at ICML, ICPR and BMVC. His research interests include data efficient and robust machine learning with applications in computer vision, language, robot control and beyond.