ROBOT SKILL ACQUISITION VIA REPRESENTATION SHARING AND REWARD CONDITIONING

by

Mete Tuluhan Akbulut

B.S., Electrical and Electronics Engineering, Boğaziçi University, 2018

Submitted to the Institute for Graduate Studies in
Science and Engineering in partial fulfillment of
the requirements for the degree of
Master of Science

Graduate Program in Computer Engineering
Boğaziçi University

2021

ACKNOWLEDGEMENTS

First, I would like to thank my advisor Assoc. Prof. Emre Uğur, who guided me throughout my master's study. His endless support, encouragement, and motivation did not only make this thesis possible but also helped me a lot in my academic journey. I feel extremely lucky to have worked with him. I would like to thank Prof. Erhan Öztop for his support and for giving helpful advice in my research. My sincere gratitude to Assoc. Prof. Arzucan Özgür and Prof. Çağatay Başdoğan for kindly accepting to be in my thesis jury.

I would like to thank my friends Ahmet Tekden, Yunus Şeker, and Utku Bozdoğan for their incredible contributions to this thesis and for making this research enjoyable. Many thanks to Alper Ahmetoglu for our visionary brainstorming sessions and his friendship. My sincerest gratitude to my friends, Mert Imre, Safa Andaç, Hamit Başgöl, Melisa İdil Şener, Serkan Buğur, Suzan Ece Ada for their support and joyful coffee breaks. It was a great pleasure to be a member of the CoLoRs research group. I also want to express my appreciation for Selen Yapraklı for her genuine support and encouragement.

Finally, I would like to thank my parents, my brother, and my sister for their love and encouragement during my whole life. I would have never been able to persevere in my education without them.

This research has been supported by the Tubitak 2210-A scholarship awarded to me for my master's studies.

ABSTRACT

ROBOT SKILL ACQUISITION VIA REPRESENTATION SHARING AND REWARD CONDITIONING

Skill acquisition is a character trait of intelligent behavior, which Robot Learning aims to give to robots. An effective approach is to teach an initial version of the skill by demonstrating as a form of Supervised Learning (SL), called Learning from Demonstrations (LfD), then let the robot improve it and adapt to novel tasks via Reinforcement Learning (RL). In this thesis, we first propose a novel LfD+RL framework, Adaptive Conditional Neural Movement Primitives (ACNMP), that simultaneously utilizes LfD and RL together during adaptation and makes demonstrations and RL guided trajectories share the same latent representation space. We show through simulation experiments that (i) ACNMP successfully adapts the skill using order of magnitude fewer trajectory samples than baselines; (ii) its simultaneous training method preserves the demonstration characteristics; (iii) ACNMP enables skill transfer between robots with different morphologies. Our real-world experiments verify the suitability of ACNMP in real-world applications where non-linearity and the number of dimensions increases. Next, we extend the idea of using SL in reward-based skill learning tasks and propose our second framework called Reward Conditioned Neural Movement Primitives (RC-NMP), where learning is done using only SL. RC-NMP takes rewards as input, generates trajectories conditioned on desired rewards. The model uses variational inference to create a stochastic latent representation space from where varying trajectories are sampled to create a trajectory population. Finally, the diversity of the population is increased using crossover and mutation operations from Evolutionary Strategies to handle environments with sparse rewards, multiple solutions, or local minima. Our simulation and real-world experiments show that RC-NMP is more stable and efficient than ACNMP and two other robotic RL algorithms.

ÖZET

ROBOTLARIN TEMSİL PAYLAŞIMI VE ÖDÜL KOŞULLANMASI YOLUYLA BECERİ KAZANMALARI

Beceri kazanımı zeki davranışın karakteristik özelliklerinden biridir ve robot öğrenmesi bu özelliği robotlara kazandırmayı amaçlar. Etkili yöntemlerden birisi becerinin basit halini örnekleyerek, gözetimli öğrenmenin bir çeşidi şeklinde, robota öğretmek ve daha sonra robotun kendi kendine beceriyi geliştirmesini ve yeni görevlere pekiştirmeli öğrenme ile uygun hale getirmesini sağlamaktır. Biz bu tezde ilk olarak Uyarlanır Koşullu Nöral Hareket Primitifleri (ACNMP) adlı, güdümlü ve pekiştirmeli öğrenmeyi uyarlama esnasında eşzamanlı kullanarak örneklemeler ve keşif hareketlerini aynı temsil uzayına kodlayan yapıyı sunacağız. Simulasyon deneylerimiz bize (I) ACNMP'nin en gelişmiş yapılara göre en az on kat daha verimli olduğunu; (II) eş zamanlı öğrenme yönteminin örnekleme özelliklerini yeni hareketlerde koruduğunu; (III) vücut yapıları farklı robotlar arasında beceri aktarımına olanak sağladığını gösterdi. Gerçek robot deneyleri de ACNMP'nin daha yüksek boyutlu ve karmaşık gerçek dünya koşullarına uyum sağlayabildiğini gösterdi. Daha sonra güdümlü öğrenmeyi ödül bazlı uyarlama görevlerinde kullanma fikrini ilerleterek oluşturduğumuz Ödülle Koşullu Nöral Hareket Primitifleri (RC-NMP) adlı ikinci yapıyı sunacağız. Bu yapı ödülleri girdi olarak alarak, istenen ödülü veren hareket güzergahları oluşturabilmektedir. RC-NMP varyasyon çıkarımı yöntemiyle olasılıksal bir temsil uzayı oluşturup, bu uzaydan çeşitli hareket güzergahları çekerek bir populasyon oluşturmaktadır. Son olarak bu populasyonun çeşitliliği seyrek ödüllü, birden fazla çözümlü veya lokal çözümlere sahip ortamlarla başa çıkmak için evrimsel stratejilerden krosover ve mutasyon yöntemleriyle arttırılmaktadır. Simulasyon ve gerçek dünya deneylerimiz RC-NMP'nin ACNMP ve diğer iki robotik pekiştirmeli öğrenme yöntemlerine göre daha istikrarlı ve verimli olduğunu gösterdi.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF SYMBOLS

a	action
A	Aggregation layer and action space
c	context parameter
$e(t)$	Robot position error at time t
E	Encoder network
I_{cross}	Boolean crossing value
K_d	Controller parameter
K_p	Controller parameter
$L_{trajectory}$	Length of the trajectory
N	Normal distribution
$N_{bottlesdown}$	Number of robots that robot collided with
n_{max}	Observation hyperparameter
Q	Query network
R	Reward function
r_i	Deterministic representation of observation i
r	Deterministic trajectory representation and reward
s	state
$SM(t)$	Sensorimotor trajectory value at time t
t	time
U	Uniform distribution
W	Wedge number
$x(t)$	Robot position at time t
x	X coordinate
y	Y coordinate
z	Stochastic trajectory representation
λ	Weighting parameter
γ	External task parameters

μ_q	Mean value
π	Policy
σ_q	Standard deviation
ϕ	Neural network weights
τ	Trajectory
θ	Neural network weights

LIST OF ACRONYMS/ABBREVIATIONS

2D	Two Dimensional
3D	Three Dimensional
ACNMP	Adaptive Conditional Neural Movement Primitives
ANN	Artifical Neural Network
CNMP	Conditional Neural Movement Primitives
CNP	Conditional Neural Processes
CVAE	Conditional Variational Autoencoders
DMP	Dynamic Movement Primitives
DoF	Degrees of Freedom
DREPS	Dual Relative Entropy Search
ELBO	Evidence Lower Bound
ES	Evolutionary Strategies
GP	Gaussian Processes
IEEE	Institute of Electrical and Electronics Engineers
KL	Kullback–Leibler
LfD	Learning from Demonstrations
MSE	Mean Squared Error
NP	Neural Processes
ProMP	Probabilistic Movement Primitives
RC-NMP	Reward Conditioned Neural Movement Primitives
REPS	Relative Entropy Search
RL	Reinforcement Learning
ROS	Robot Operating System
SL	Supervised Learning
SM	Sensorimotor
UR10	Universal Robot 10
VAE	Variational Autoencoder

1. INTRODUCTION

Robot Learning aims to develop robots that are capable of adapting to different situations and can be functionally good at a variety of tasks. Skill acquisition, which can be defined as the learning of a task to give accuracy and speed after practice, is an important stepping stone to achieve such an intelligent robot behavior. A fruitful skill learning approach is called Learning from Demonstrations (LfD), where an expert provides demonstrations of the skill to the robot by controlling it via teleoperation or letting the robot observe the skill via its sensors. Then, the robot learns to reproduce the demonstrations and develops a policy to use in the environment [3]. If the demonstrations are provided in the intrinsic space of the robot, they can be played back on the robot with a controller or a model which can be an Artificial Neural Network (ANN) that captures the mapping from states to actions can perform the skill (e.g. [4,5]). LfD is often used with Reinforcement Learning (RL) [6] to provide a good start for policy search, to increase learning efficiency by encouraging found policy trajectories to be close to demonstrations, and to preserve demonstration characteristics like user preferences and concepts hard to specify formally [7]. Although RL algorithms coupled with ANNs are shown to synthesize complex skills from scratch in simulation [8–10], they are not directly implemented in robotic systems due to limited time and resources as the number of trajectory samples for successful learning can be excessive [11]. Therefore again, LfD approaches are used together with RL algorithms.

Even though this LfD and RL idea seems really promising and is well studied, there are still many open challenges to develop robots that can be used flexibly in different real-world environments. In this thesis, we first propose Adaptive Conditional Neural Movement Primitives (ACNMP) that addresses three important challenges in this context:

Sample Efficiency is still the most important challenge in real-world learning problems. The relationships between task parameters and the robot's policy are non-linear and high dimensional. Therefore, learning a mapping that captures these relationships requires excessive trajectory samples and we need sample-efficient machine learning methods to use with real robots.

Interference of novel goals with demonstrations is another important challenge that we face when we use LfD together with RL algorithms. The demonstrations provide skill characteristics that are difficult to specify formally in a reward function. These skill characteristics can reflect safe behaviors, demonstrator's preferences, or robot and environment constraints. Thus, the learning algorithms should produce trajectory samples that preserve these skill characteristics while updating the policy to adapt to novel goals or environments.

Skill transfer between agents with different embodiments is another challenge when the demonstrated skill is not provided in the robot's own action space. The demonstration trajectories should be translated to intrinsic coordinates of the robot with an additional step to complete the initial skill transfer. This can be problematic if the mapping is not well defined and multiple solutions exist. This problem may also arise in multi-robot systems where one skill is needed to be transferred to another robot. Hence, an automatic transfer mechanism between agents that does not require the design of an explicit coordinate mapping is required.

Our proposed method ACNMP [12] addresses the aforementioned challenges by (i) introducing a novel LfD+RL framework that captures the complex relationships between task parameters and sensorimotor trajectories order of magnitude more sample efficient than the state-of-the-art algorithms, (ii) sampling the exploratory trajectories of RL algorithm from the same latent representation space where demonstrations are encoded so that maintaining the demonstrated skill characteristics in new trajectories is ensured, and (iii) forming a common latent representation space between robots with different morphologies, that allows automatic skill transfer between them.

ACNMP uses a recently developed robotics model called Conditional Neural Movement Primitives (CNMP) [13], which is built on top of Conditional Neural Processes [14], as the LfD model, and it uses a novel RL component to step in case the LfD model fails when it is queried with inputs from outside of the training range or it is placed in a novel environment. The LfD model of ACNMP consists of an encoder and decoder network that learns the distribution of sensorimotor trajectories in relation to task parameters from a few demonstrations. The RL component employs the same network for adaptation but it is trained in a novel way using simultaneous RL and LfD (SL) objectives. Concretely, the model uses error-based SL loss for the demonstration set and policy gradient loss for RL-guided actions in alternating steps. After adaptation is successful, the newfound trajectory is regarded as a new demonstration and the generalization capacity of the framework is increased. To realize skill transfer between robots with different embodiments, two ACNMPs are trained together to construct the common latent space by using proxy skill demonstrations provided to both robots. Afterward, one robot can autonomously learn a novel task by observing the skill execution of its peer robot. We showed the superiority of ACNMP over other state-of-the-art methods in three simulation experiments, and its suitability for real-world applications in two real robot experiments.

The advantages of using Supervised Learning loss in Reinforcement Learning training motivated us to benefit further from SL during reward-based policy learning. We investigated whether we can train the robot fully using Supervised Learning in Reinforcement Learning tasks so that the robot can learn a new skill or improve an old skill for a new environment in a more efficient and stable way.

Barto and Dietterich [15] argued that such an approach is not possible. In Supervised Learning, inputs and outputs are given to the model, and the model is trained to find the unknown mapping. The training is implemented using error signals between desired and produced outputs. However, in Reinforcement Learning, there is a reward function and the model tries to find an unknown input that maximizes the obtained reward. Here, only evaluation signals (rewards) are provided based on the performance of the current input and there is no direct knowledge about the desired input.

Recently, Schmidhuber [16] proposed to use the environmental feedback (reward) together with desired time horizon as input to predict actions to convert the problem to a mapping problem as in the Supervised Learning. This method is called Upside Down RL and realized in [17] in Atari environments. Here, it is shown that URDL performs better than the traditional RL algorithms. However, robotic systems bring additional challenges and requirements that hinder the direct implementation of the same method. This method is shown to produce discrete actions for a given state, desired reward, and time horizon but robotic tasks require complex continuous sensorimotor trajectories. In addition, contrary to used Atari environments, the reward is typically sparse in robotic systems, i.e. the reward is obtained after the execution of the full trajectory in robotic systems. Moreover, many robotic tasks have multiple solutions and multiple local minima difficult to escape with standard policy search algorithms.

In our second proposed method, namely Reward Conditioned Neural Movement Primitives (RC-NMP) [18], we study reward-based skill learning in robotics systems as a supervised learning problem and address the aforementioned challenges. Our framework is built on top of a recent ANN architecture called Neural Processes [19], that encodes multi-modal trajectory distributions and their non-linear relation with external task parameters. Differently from [14], the trajectory distribution is captured in latent space using variational inference [20] instead of the task space. Our framework constructs the latent representation distribution conditioned on rewards, uses stochastic sampling to generate varying continuous trajectories for the desired reward, and is fully trained in SL settings. Stochastic sampling from different reward landscapes contributes to the exploration of the environment but we further benefit from Evolutionary Strategies (ES) [21] to effectively deal with multiple solutions and multiple local minima in the search space. First, we propose a novel crossover operation, that temporally blends the generated trajectories in the latent representation space. We observed that this operation accelerates the learning, because sub-parts of undesired trajectories, which receive rewards from different portions, could possibly be combined to achieve high rewards at early steps. We further implemented a mutation operation to increase the diversity of trajectories by adding a smoothed Gaussian noise in the task space.

We compared our framework's encoding and trajectory generation capability with CNMP [13], and its skill learning performance with ACNMP [12] and two other state-of-the-art RL algorithms with three experiments.

In Chapter 2 we introduce the robot, its gripper, and software tools that we used in this thesis. In Chapter 3, we provide an analysis of the literature together with the state-of-the-art methods in the areas that we worked on. Chapter 4 provides the background information that may help to comprehension of this study. The proposed frameworks, their experimentation results, and their comparison with state-of-the-art models will be introduced in Chapter 5 and Chapter 6 respectively. Chapter 7 presents the discussion and the conclusion.

2. EXPERIMENT PLATFORM

In this chapter, we present the robot, simulation environments, and other tools that we used in this thesis to provide a better understanding. Section 2.1 introduces the robot arm and the gripper that we used in our experiments, and Section 2.2 introduces the software components that we benefit from in our thesis.

2.1. Physical Components

2.1.1. UR10 Robot

In this thesis, we used the UR10 robot of Universal Robots [22] in real-world experiments. UR10 is a versatile collaborative industrial robot, that can carry a 12.5 kg load and has 1.3 meters reach. It is designed to perform with high accuracy and reliability in a wide range of applications like machine tending, palletizing, and packaging. The robot has 6 rotational joints, thus 6 degrees of freedom (DoF). The robot is able to manipulate objects in three-dimensional (3D) space in all directions. In our experiments, the robot is used via Robot Operating System (ROS) [23] in 3D and planar manipulation, and high precision water pouring tasks.

2.1.2. 3F Robotiq Gripper

The gripper attached to the UR10 robot is a three-fingered Robotiq gripper [24]. The gripper has 20 to 155 mm object diameter for encompassing, it can apply 30 to 70 N grip force, and carry a 10 kg load. It can operate in 4 modes: basic mode, pinch Mode, scissor mode, and wide mode. Basic mode is used for grasping objects in our experiments. The gripper is connected to ROS using present Python packages.

2.2. Software Components

2.2.1. Robot Operating System (ROS)

ROS [23] is not an actual operating system but a very flexible open-source framework to develop robotic software. It is widely used among the robotic community and includes a wide variety of libraries, tools, and conventions. ROS assigns nodes to different processes that control different components of a robotic system, like the robot, the gripper, sensors; and connects them via a message-passing system. Each node can publish information using a topic, and other nodes can listen and process this information. It also eases robot control by providing forward and inverse kinematic chains and path planning. We used ROS to record demonstrations and control movements of the UR10 robot and the Robotiq gripper.

2.2.2. CoppeliaSim

CoppeliaSim [25] is a robotic simulator. It inherited most of the features from its ancestor, V-Rep simulator [26]. It has a distributed control architecture: embedded scripts, plugins, ROS nodes, remote API clients, and custom solutions are used to control each individual object/model separately. It supports various computer languages and platforms for the development of controllers: Python, Java, Lua, C/C++, Matlab, or Octave. We used its ROS backend to control robots in simulation.

2.2.3. Tensorflow and Keras

Tensorflow is a free open-source library developed for machine learning applications. It particularly eases the design and training of ANNs in deep learning. It builds common knowledge in machine learning by providing comprehensive, flexible ecosystem tools and libraries that allow easy development of machine learning models and the usage of state-of-the-art models. Keras is an API designed to reduce the cognitive human effort in machine learning applications. Its guides, documentation, clear error messages accelerate the development of deep learning architectures.

2.2.4. Pyrep

Pyrep is a robot learning toolkit for robot learning researchers who use the CoppeliaSim simulator. Instead of using slower remote procedure calls to reach the simulator, Pyrep python codes can be run synchronously with the simulation loop, and increase the speed of interaction with the simulator. Therefore, it is very practical to be used in reinforcement learning and learning from demonstration.

2.2.5. Jupyter Notebook

The Jupyter Notebook is an open-source python editor, where codes can be stored and run as independent blocks. Text and equation blocks can be added as well to build a more complete project. The source codes of our both frameworks are written in the Jupyter Notebook and shared with the community with its format.

2.3. External Library and Packages

Additionally, Matplotlib [27] is used for data visualization, scikit-learn [28] and numpy [29] are used for data processing.

3. RELATED WORK

3.1. Learning Movement Primitives

Learning from Demonstration (LfD) [3] is a widely used technique in robotic problems [7]. Some examples include object grasping and manipulation [30–34]. Popular learning frameworks include dynamic systems [35], statistical modeling [36] and their combination [37,38]. Dynamic Movement Primitives (DMP) [35] represent the demonstrated trajectory with a set of differential equations. It uses a spring-mass-damper system with a non-linear function, and it can learn non-linear movements in one shot. It also provides a guarantee for reaching the goal point under perturbations thanks to its point attractor. The expressiveness of DMP trajectories is determined by the number of basis functions, a hyper-parameter needs additional tuning efforts. Probabilistic Movement Primitives (ProMP) [39] learns the distribution of these basis functions for a trajectory distribution and therefore, can generate stochastic policies. Conditional Neural Movement Primitives (CNMP) [13], built on a recent ANN architecture, Conditional Neural Processes (CNP) [14], can also represent trajectory distributions. It extracts non-linear relationships between task parameters and complex sensorimotor trajectories from a few demonstrations. Our first proposed framework (ACNMP) uses CNMP as the LfD model, and its performance and generalization capabilities of ACNMP are compared against adaptive ProMPs. Our second proposed framework (RC-NMP) is built on a similar ANN architecture called Neural Processes [19] that adds variational inference on top of CNP. This architecture allows capturing trajectory distributions in latent representation space so that full trajectories can be sampled, whereas an explicit trajectory shaping mechanism is required to combine trajectory points for each time point for CNMP.

3.1.1. Variational Inference in Movement Primitives

In recent years, variational inference has been used to develop generative models that produce movement primitives [40–42]. [40] benefited from Deep Variational Bayes Filtering [43] to encode the model parameters of DMPs in a Variational Autoencoder (VAE). In [41], Temporal Convolution Networks [44] were used to represent trajectories, and a VAE conditioned on task parameters was trained as the generative model. [42] combined discrete and continuous latent variables to train a VAE, and the decoder network was conditioned on goal parameters to produce trajectories. All these methods require training trajectories on a scale of thousands but RC-NMP is sufficient to perform well from only a few demonstrations because Neural Processes takes random observation samples to learn trajectory representations.

3.2. Adapting Primitives

Sample efficiency is the bottleneck of RL algorithms that are aimed to be used in real-world environments. Expert demonstrations are often combined with an RL agent which later generates its own experiences [11]. These efforts can be categorized into two. The first one utilizes the RL agent at the core and uses demonstrations to provide an initial experience and reference trajectories for RL training [45–47]. For instance, [45] defines a trade-off cost function that balances the effect of obtained rewards from the environment with the knowledge obtained from demonstrations. However, [48] states that trade-off weighting hyper-parameters should be carefully tuned. Otherwise, the agent either can not benefit from expert trajectories or imitates only the expert and cannot show adaptation.

The second category, where our frameworks belong, uses an LfD engine at the core and extends its capabilities by RL training. ACNMP is closely related to [1, 2], which used ProMP as the LfD model and adapted it to novel environments using RL. [1] used ProMP to learn demonstrations of a planar robot arm that push an object to a goal point. Here, the framework requires separate ProMPs to learn demonstrations for different goal points, and RL is used to find trajectories for an unseen goal point.

The algorithm, namely, Relative Entropy Search (REPS) [49] defines an additional metric, KL divergence, to stay close to previous parameters of the model with the purpose of preserving the shape of the movement. [2] used a single ProMP to condition with different task parameters by combining it with Gaussian Processes. For the RL part, relations between task parameters and trajectories were found using a relevance metric. In comparison with the studies above, ACNMP and RC-NMP do not benefit from any additional/weighted loss, relevance metrics, or KL-divergence term. The training procedure of ACNMP, which uses demonstrations and RL-guided trajectories simultaneously, allows preserving demonstration characteristics inherently. On the other hand, RC-NMP regards all trajectories (demonstrations and exploration trajectories) similarly to produce a trajectory population without additional efforts.

3.2.1. Use of Learning Data with Varying Quality

An important challenge in LfD and trajectory adaptation is to exploit trajectories fully from different reward landscapes. RC-NMP addresses this challenge by learning a mapping between reward values and trajectories. In this way, the model benefits from all trajectories in the population to understand the trajectory reward relation. In this respect, Double REPS [50] particularly addresses this problem by improving the REPS [49]. High reward trajectories and low reward trajectories are clustered during learning, and policy updates are constrained to make the policy closer to the high reward cluster and further away from the low reward cluster. In other words, these clusters work as force fields. The experiments showed that this approach is highly beneficial for problems, that include multiple solutions and local minima. In this thesis, RC-NMP is compared with both DREPS and REPS in terms of performance in a similar problem.

3.2.2. Transfer Learning

Another way of task adaption is taking additional demonstrations for the novel task. These demonstrations can come from the expert or another agent proficient in that particular task. Therefore, transfer learning in robotics is an important topic for task adaptation [51]. A common method is to pair all the corresponding states of different agents [52]. Another practice is constructing a common feature space manually instead of direct pairing [53], or aligning states via unsupervised manifold alignment [54]. [55] learned the common feature space using ANN after aligning states by expectation-maximization-based dynamic time warping. In comparison with other studies, our method, ACNMP, finds the common feature space to transfer skills at the trajectory level rather than the state level. Thus, it does not require any explicit state pairing and aligning efforts.

3.2.3. Evolutionary Policy Optimization

Recently, Evolutionary Strategies (ES) have been used first as an alternative to RL [56], [57], and later approaches using them together came forward. While ES approaches are quick for a small number of parameters and effective to deal with challenging search spaces with multiple local minima, they suffer from exponential complexity for a large number of parameters. So, they are used together with RL algorithms, which are able to handle a large number of parameters. Genetic-Gated Networks [58] employed a chromosome vector, to which crossover and mutation operations are applied to control gating choices of the policy network. The chromosome vector that corresponds to the best solution is selected, and the resulting network is optimized based on the gradient. Genetic Policy Optimization [59] exploits both imitation learning and deep reinforcement learning. Offspring networks of two parent policy networks are trained with imitation learning to share a similar state distribution with parents. Policy gradient methods are applied for mutation operation with high variance in order to create enough diversity for better exploration. [60] used ES directly on network parameters. Differently from the above studies, [60] studied problems in dynamic environments, and the learning is adjusted using instance weighting that favors quality and novelty.

In [61], robustness against deceptive rewards is investigated and it is shown that the ES approach performs better on such tasks than RL, since it encourages different behaviors more through novelty search [62], avoiding the local optima that RL gets stuck on. [63–65] combined deep RL algorithms with ES. Compared to these studies, RC-NMP introduces a novel crossover operator that temporally blends parent trajectories to create offspring using the self-organized latent representations.

4. BACKGROUND

In this chapter, we will introduce the models that we inspired from and used while developing our frameworks ACNMP and RC-NMP to provide a better understanding. First, we will explain Conditional Neural Processes and Neural Processes, two recent ANN architectures. Then, we will talk about Conditional Neural Movement Primitives that extend Conditional Neural Processes to the robotics domain.

4.1. Conditional Neural Processes

Neural networks have been widely used as function approximators and showed great success. Yet, they usually require big datasets and are trained from scratch. If different functions are needed to use without retraining models, Bayesian approaches are really useful. For instance, Gaussian Processes exploit the prior knowledge about a function set and infer specific functions at test time. Prior selection and training time for large datasets are the main disadvantages of Gaussian Processes. Conditional Processes claim to bring the best of the two worlds. This model family is inspired by the flexibility of Gaussian Processes (GP) but organized and trained like neural networks. Here, the model extracts the prior knowledge directly from the data and can infer the underlying function distribution from observations. The structure of the model, which is adopted from [14] is shown in Figure 4.1. The observations (x_i and y_i pairs) are passed through a parameter-sharing encoder network to get the corresponding representations r_i. Then, these representations are merged with an arbitrary differentiable aggregation A operation to get the function representation r. In the experiments, this operation is selected as averaging. The query network, denoted with Q, uses this representation r together with query inputs x_q to predict the corresponding output y_q with a Gaussian distribution parametrized with μ_q and σ_q. The model is trained with the log-likelihood loss in the following equation:

$$\mathcal{L}(\theta, \phi) = -\log P(y_q \mid \mu_q, \text{softplus}(\sigma_q)) \tag{4.1}$$

θ and ϕ denote the parameters of the encoder and query network respectively. At each training iteration, random observations are selected from a random function and the output for the target input is predicted.

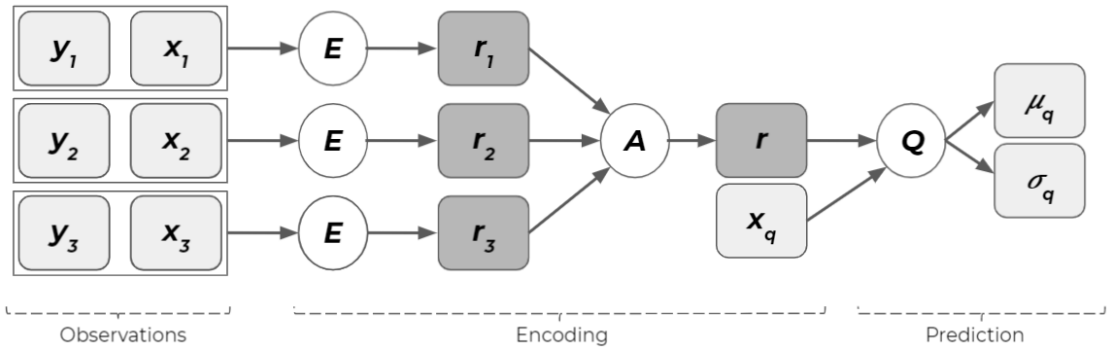

Figure 4.1. Architecture of the CNP.

4.2. Neural Processes

Conditional Neural Processes work arguably well to model function distributions in the task space. It produces a mean and variance to model the likelihood of the function output for the given query inputs. Since the model lacks a global latent variable that allows different function samples for the given observations, an external shaping mechanism is required to connect sampled outputs to obtain a full function. Neural Process is developed as the generalized version of CNP with a global latent variable. Variational inference is used to learn the latent representation distribution, and representation samples from this distribution are used to generate function samples as in Variational Autoencoders (VAE) [66]. The model architecture adopted from [19] can be seen in the Figure 4.2. Similarly to CNP, a parameter-sharing encoder network E extracts the representations r_i of corresponding observations (x_i and y_i pairs). These representations are again merged with a differentiable aggregation operation A. Differently here, the resulting representation r is used to construct the parameters of a latent variable Gaussian distribution N_z. The sample from this distribution z is fed to the query network along with the query inputs x_q to predict the corresponding outputs y_q.

The error between the true and the predicted outputs is the reconstruction error but the variational model also needs to tighten the variational bound between the true posterior and the variational posterior. The combination of these two errors is called the evidence lower bound (ELBO), the training objective. Normally, the prior is modeled with a standard Normal distribution $p(z)$ in practice, and the ELBO would look like this:

$$\log p(y_{1:n}|x_{1:n}) = \mathbb{E}_{q(z|y_{1:n},x_{1:n})}[\sum_{i=1}^{n} \log p(y_i|x_i,z) + \log \frac{p(z)}{q(z|y_{1:n},x_{1:n})}] \tag{4.2}$$

For the purpose of predicting a function from random observations, the dataset is divided into two: the observation set $(y_{1:m}, x_{1:m})$ and the target set $(y_{m+1:n}, x_{m+1:n})$. We can use the prior as $p(z|y_{1:m}, x_{1:m})$ using the observations for our purposes. Since calculating this probability will be intractable, it can be approximated with $q(z|y_{1:m}, x_{1:m})$. The equation becomes:

$$\log p(y_{m+1:n}|y_{1:m}, x_{1:n}) = \mathbb{E}_{q(z|y_{1:n},x_{1:n})}[\sum_{i=m+1}^{n} \log p(y_i|x_i,z) + \log \frac{q(z|y_{1:m},x_{1:m})}{q(z|y_{1:n},x_{1:n})}] \tag{4.3}$$

Here, the latent variable is sampled with the reparametrization trick. The training of the model is very similar to the training of CNMP. The only difference is the sampling of the latent variable for each observation set.

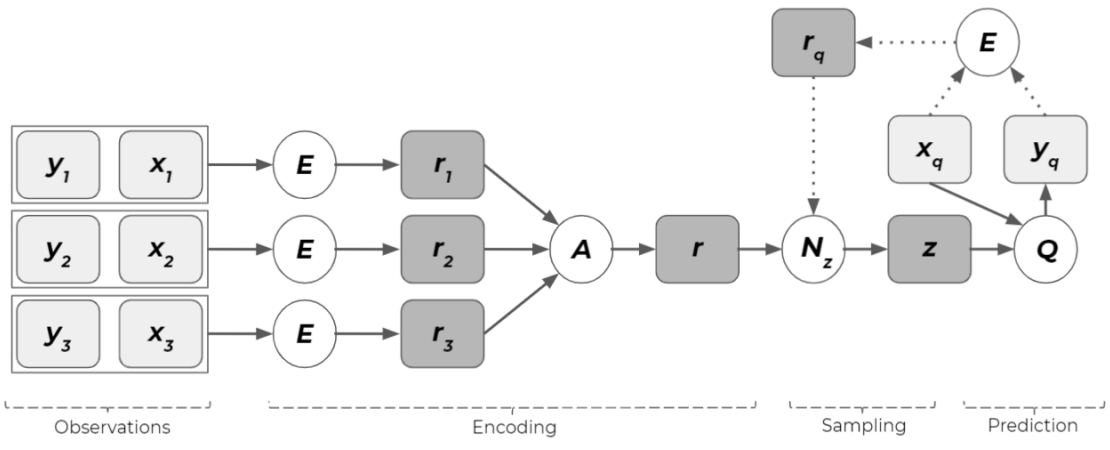

Figure 4.2. Architecture of the NP.

4.3. Conditional Neural Movement Primitives

Conditional Neural Movement Primitives [13] extend the CNP model to the robotics domain. The illustration of the model can be seen in Figure 4.3. Here, the data set consists of sensorimotor trajectories with respect to time. The time interval of the trajectories is scaled into the interval $[0, 1]$ to make the model time-invariant. Although the trajectories are shown in one dimension here, they can be multidimensional. The observations are sensorimotor values $SM(t)$ and time points t, and the output is again mean and the variance of possible sensorimotor values for the query time point t_q. Here, both encoder and decoder networks are queried with the external task parameters γ to learn the non-linear relationships between trajectories and task parameters. At test time, condition points (via-points that the trajectory should follow) and task constraints in form of parameters are given to the model. The query network takes all time points in the time horizon $[0, 1]$ to produce the mean sensorimotor trajectory and the variance around it.

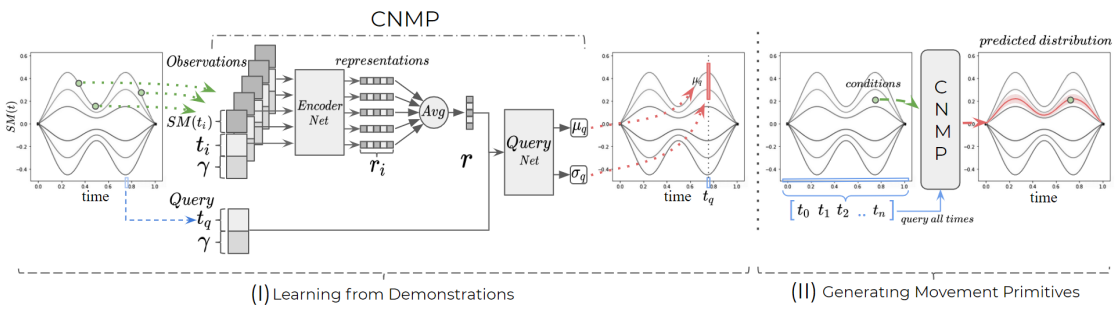

Figure 4.3. Architecture of the CNMP.

5. ADAPTIVE CONDITIONAL NEURAL MOVEMENT PRIMITIVES

5.1. Proposed Method

Adaptive Conditional Movement Primitives (ACNMP) is a robot skill learning framework that consists of one LfD component and one RL component. Given expert demonstrations, the LfD component learns the initial version of the skill. Then, the RL component steps in and improves the skill whenever a failure occurs due to a new environment of new task constraints. ACNMP also allows transfer of skills between morphologically different robots by finding a common latent space between the agents. This becomes quite useful when the reward is too sparse for the RL component and there exists another robot that acquired the skill before.

We use CNMP as the LfD component of ACNMP, and its underliying model CNP and CNMP were introduced in Chapter 4. We employ a policy gradient approach for the RL component because our action space consists of continuous sensorimotor trajectory values. The stochastic policy of the agent is denoted with $\pi_\theta(a, s) = P(a_t \in A | s_t \in S, \theta)$, where θ represents the policy parameters (network parameters), A and S represents action and state spaces. We generalize policy to depend on a context signal c, which is often *time* augmented with some task parameters. The aim is to maximize the expected reward $R(\tau) = \sum_{t=1,..T} R(s_t, a_t, c_t)$ obtained through the course of a trajectory $\tau = \{s_0, a_0, ...s_T, a_T\}$ in policy gradient algorithms [67]. In this setting, the gradient of the policy gradient objective with respect to policy parameters becomes:

$$\nabla_\theta J(\theta) = \mathbb{E}_{\pi_\theta}[\sum_{t=1}^{T} \nabla_\theta \log \pi_\theta(s_t, a_t, c_t)(\sum_{t'=t}^{T} R(s_{t'}, a_{t'}, c_{t'}))] \tag{5.1}$$

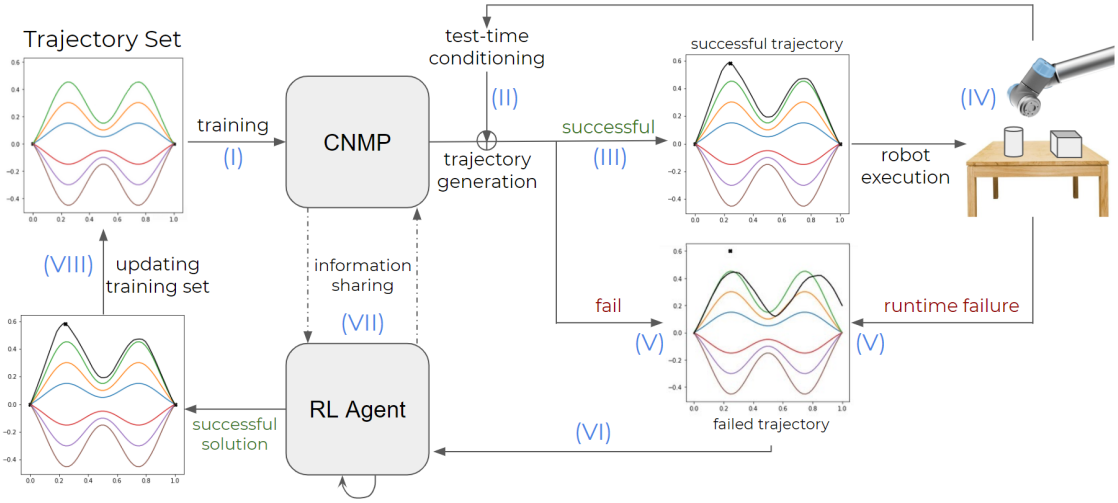

Figure 5.1. The ACNMP architecture and the learning loop.

Furthermore, since the RL agent produces actions with respect to context parameters, and there is no natural state transitions with respect to actions, the problem becomes a generalized multi-armed bandit problem where the policy distribution depends on the context parameters c. In this case, the gradient simplifies to:

$$\nabla_\theta J(\theta) = \mathbb{E}_{\pi_\theta}[\sum_{t=1}^{T} \nabla_\theta \log \pi_\theta(a_t, c_t)(\sum_{t'=t}^{T} R(a_{t'}, c_{t'}))] \tag{5.2}$$

The proposed framework ACNMP can be seen in Figure 5.1. First, the LfD model (CNMP) is trained with demonstration trajectories (I) by following the explained procedure in Section 4.3. Then, the model is conditioned with task parameters and via-points (II) to generate the desired behavior (III), which is carried out by the real robot (IV). If the generated trajectory is found unsatisfactory or a runtime failure occurs during execution in extrapolation cases (V), the RL agent is triggered (VI). RL agent uses the same network architecture as the policy network and updates it via policy gradient while it is also being trained with demonstrations in alternating steps (VII). After the RL agent finds a satisfactory trajectory based on the rewards of the new task, this trajectory is added to the trajectory set as a new demonstration (VIII). The LfD framework updates itself using the new trajectory set, and the loop continues.

Simultaneous Training with SL and RL: The critical contribution of our proposed model is that while RL agent explores the environment to find a sufficient trajectory, Supervised Learning using expert demonstrations is applied simultaneously to preserve expert knowledge. This training method does not only find new trajectories that satisfy the novel tasks but also makes those new trajectories share similar characteristics with expert demonstrations. The reason is that the SL updates avoid catastrophic forgetting of the formed representation space by demonstrations, and the representation of the novel trajectory is placed to an appropriate location in the same space. In short, discovered trajectories by RL agent share the same representation space and therefore similar characteristics with demonstrations.

The network updates itself using two loss functions for SL and RL objectives: likelihood loss of CNMP, Equation (4.1), and the policy gradient loss, Equation (5.2). The former is used to train the network for the input data (observation and task parameters) coming from demonstrations. The latter is used to guide actions of RL agent for extrapolation cases, e.g. unseen task constraints or novel environments. Although we are saying that there are two loss functions, they are actually quite similar. If we manipulate the loss function of CNMP, Equation (4.1), it will look like another policy gradient loss:

$$\nabla_\theta J(\theta) = \mathbb{E}_{\text{CNMP}_\theta}[(\sum_{t=1}^{T} \nabla_\theta \log_{\text{CNMP}_\theta}(a_{t,\text{demo}}, c_t))] \tag{5.3}$$

where $R(\tau) = 1$ because the ground truth action ($a_{t,\text{demo}}$) is known. This manipulation can be summarized as follows: CNMP generates a trajectory distribution to make the demonsration trajectories more likely whereas policy gradient updates a policy distribution to make the most rewarding action sequences or trajectories more likely. policy gradient adjusts policy distribution to make the most rewarding action most likely whereas CNMP adjusts trajectory distribution to make the observed actions most likely.

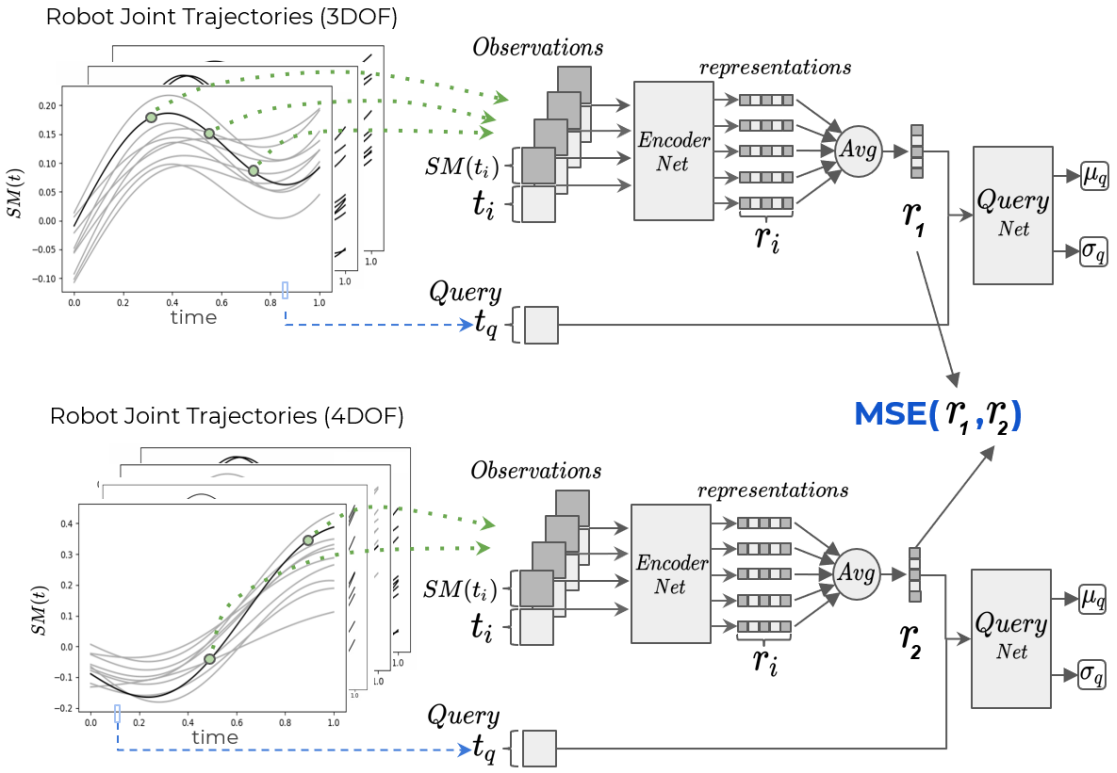

Figure 5.2. The transfer learning architecture.

Assimilating New RL Solution into the CNMP: The RL agent only uses the provided via-points as inputs while discovering the new trajectory. However, the LfD model is trained by sampling random observations from a full trajectory at each training step. For that reason, the solution of the RL agent is added to the dataset of the LfD model for further training. In the end, the capability of the model is extended from the range of demonstrations to the range of trajectories that include RL solutions and demonstrations.

Transfer Learning: For knowledge transfer between robots, two ACNMP models are deployed in two robots. These two models learn the proxy skills, demonstrations of which are available to both of them, together to form a common representation space. The Figure 5.2 illustrates learning of common space for two agents. Here task parameters are not used to find task invariant latent space, only the joint trajectories of robots are used as inputs to show agent to agent transfer capability.

The training of a single CNMP here is same as in Section 4.3 but the found representations by encoders for the corresponding trajectories of the proxy skills are forced to be as close as possible by adding a distance metric to the loss:

$$\mathcal{L}(\theta_1, \phi_1, \theta_2, \phi_2) = \mathcal{L}_{\text{CNMP1}}(\theta_1, \phi_1) + \mathcal{L}_{\text{CNMP2}}(\theta_2, \phi_2) + \mathcal{L}_{\text{MSE}}(r_1, r_2) \tag{5.4}$$

Here, r_1 stands for the representation produced by the encoder of the first CNMP, r_2 stands for the representation produced by the encoder of the second CNMP, and $\mathcal{L}_{\text{MSE}}$ denotes the mean squared error. We would like to attract attention to the fact that the observations coming from two robots have different numbers, different dimensions, and different time points. Our framework can successfully find trajectory to trajectory transfer by encoding the whole trajectory into latent space from observations.

For the test task, problem formulation assumes that the source agent can demonstrate the skill that the target cannot perform. Skill transfer occurs as follows. The encoder network of the first ACNMP extracts the representations of skill trajectories. Then, these trajectories are fed to the decoder of the second ACNMP to find the translation of the trajectory of the source robot to the joint space of the target robot. Afterward, the RL agent of the target robot can optimize these translation trajectories to learn the skill perfectly.

5.2. Experiments

We evaluated the performance of ACNMP in four simulation experiments, where the state-of-the-art comparisons and two real robot experiments were made. The simulation models, dataset, and code can be found in [68]. We were not able to find the source codes of two state of art algorithms that we used for comparison [1,55], and our implementations were based on the descriptions (including reward functions) in the papers.

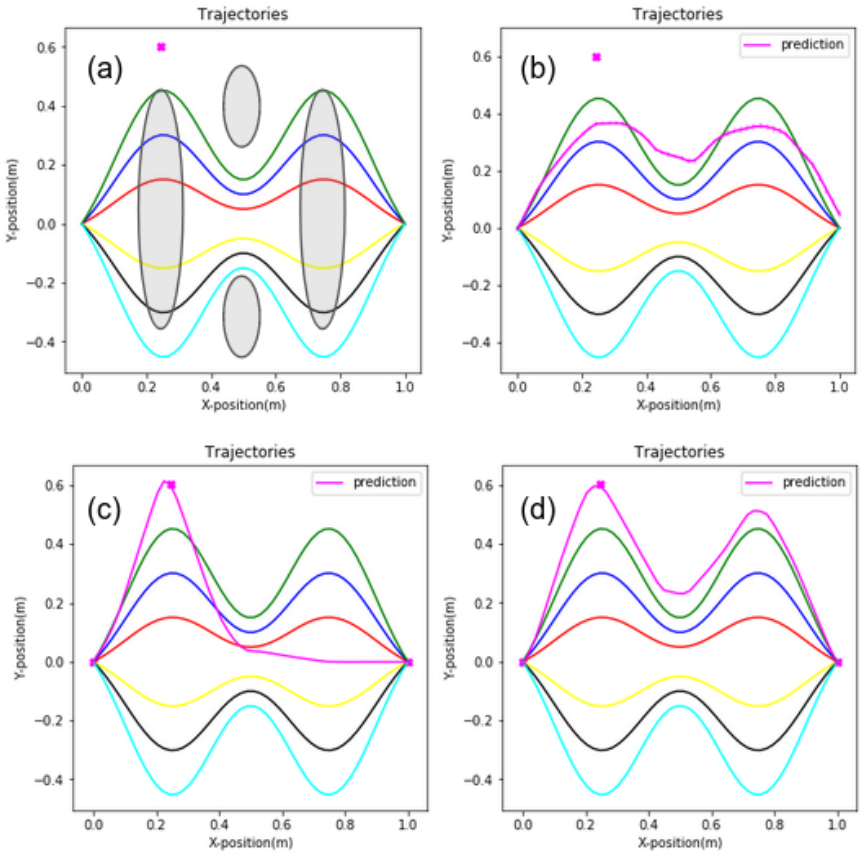

Figure 5.3. (a) The simulated obstacle avoidance task. The six demonstrations are colored, obstacles are represented with gray ellipses, and pink point shows the via-point. (b) CNMP model fails to extrapolate. (c) The trajectory found by RL-only model. (d) Simultaneous LfD+RL based ACNMP.

5.2.1. Extrapolation in Obstacle Avoidance

This toy experiment shows in which situations the LfD (CNMP) model fails and RL agent is triggered and evaluates the performance of ACNMP. For this purpose, we placed 4 elliptic obstacles and provided 6 different expert demonstrations that avoid the obstacle and share the same starting and endpoint. The experiment is illustrated in Figure 5.3 (a) where different demonstrations have different colors. Given a conditioning point (pink dot in the figure), the model is expected to produce a trajectory that avoids the obstacles and goes through the via-point. The LfD-only model fails to generate a sufficient trajectory if the via-point is outside of the training range 5.3 (b).

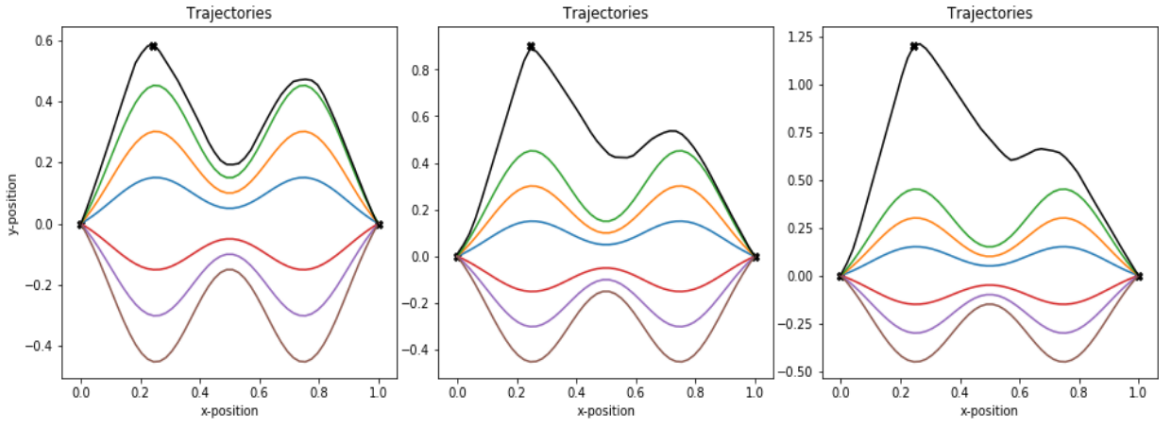

Figure 5.4. ACNMP performance with respect to the increasing amount of extrapolation. Black dot is the via-point, black line is the generated trajectory, and colored lines are demonstrations.

Figure 5.3 (c) shows the result if we employ only an RL agent to solve the task where the reward function calculates the minus distance between the trajectory and the via-point, and it does not give any feedback about the obstacles. The RL-only trajectory does not preserve the demonstrated shape while going through the via-point. Figure 5.3 (d) illustrates the performance of ACNMP. The found trajectory of ACNMP after 35 rollouts both preserves the demonstrated shape and avoids the obstacles with the same reward function due to its training procedure that uses demonstrations and discovered trajectories by RL simultaneously.

Next, we checked the limit of this shape preservation by placing the via-point further away from demonstrations. It seems that the algorithm had more difficulty in expressing the same shape with demonstrations when the extrapolation point move away although the trajectories continue to meet the via-point. Still, we should note that what is the right trajectory shape is not a well-defined question.

Assimilating new constraints into CNMP representations: This part of the experiment investigates how demonstrations and RL trajectories are encoded in the representation space after ACNMP training.

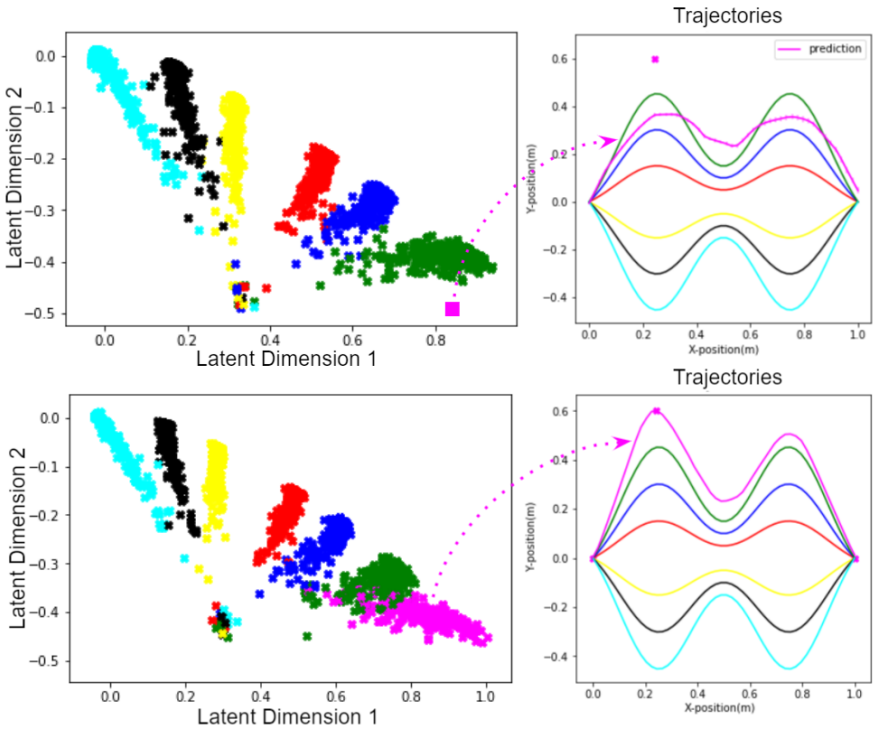

Figure 5.5. Illustration of the representations for each trajectory in the latent representation space. The top and bottom parts show latent representation before and after RL solution is assimilated into the model, respectively.

For this visualization, the bottleneck of the neural network, representation layer of the CNMP, is decreased to 2 neurons, and Figure 5.5 shows color-coded trajectories and their corresponding representations before and after ACNMP training. It is shown that different representation clusters are formed for different trajectories. The top figure illustrates that if we take a sample from a point in representation space, where we would expect the solution trajectory would be, we see that the corresponding trajectory does not satisfy task constraints. However, the novel training method of ACNMP ensures that the interpretation of representation space is changed and the RL solution is encoded into the right spot. This analysis suggests that ACNMP robustly integrates new knowledge into the existing representations.

5.2.2. Adaptation to Novel Environments in Simulation Experiments

We evaluate the adaptation capabilities of ACNMP in a simulated tabletop pushing and 2D obstacle avoidance task, and compare its performance with the baseline studies [1, 2]. For the first experiment, we replicated the experiment setup of [1] in CoppeliaSim [25]. Figure 5.6 (a) illustrates the task. A joint-controlled 3-DoF planar robot arm is placed on the table along with a black cylinder, which the robot should push to 10 different target locations. The robot is provided with demonstrations for the 9 target locations except for the test target location, and the experiment measures how effective the learning model can use expert knowledge for other target points to learn how to push to the test target location. The reward that both [1] and we set is the minus distance between the object positions during the execution of the demonstration and the generated trajectory. [1] had to use one ProMP to represent the demonstrations for each target point because the initial version of ProMP cannot learn relations between trajectories and task parameters.

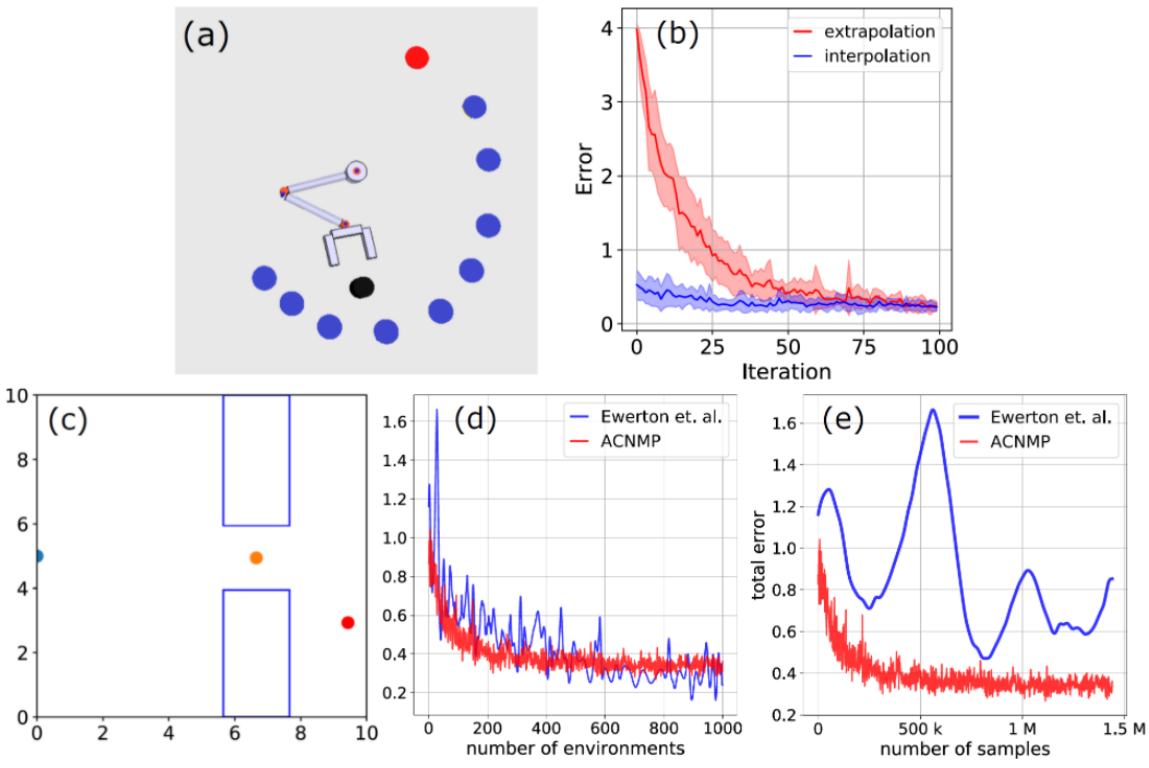

Figure 5.6. Tasks and performance comparisons for [1] (a-b) and [2] (c-e)

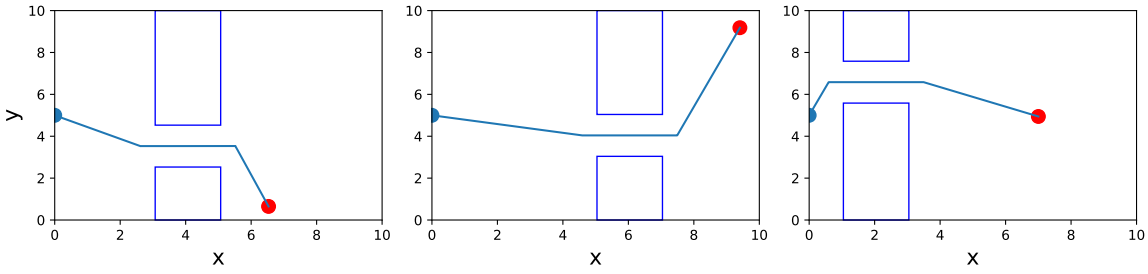

Figure 5.7. The demonstrations for the 2D obstacle avoidance experiment are shown. The blue circle shows the start point and the red circle shows the end point.

On the other hand, ACNMP can learn that non-linear relationships between trajectories and task parameters and can generate trajectories conditioned on the desired target position. Thus, a single ACNMP is used to learn all the demonstrations. We consider the target points in two categories: interpolation and extrapolation, which are shown in Figure 5.6 (a) with blue and red respectively. Figure 5.6 (b) provides the analysis. As shown, ACNMP almost did not require any adaptation for the interpolation cases and used 130 rollouts (2 per iteration) for adaptation for the extrapolation cases. In [1], it is indicated that their model solved the task using around 1200 rollouts where interpolation cases dominated the dataset (80%).

Table 5.1. Comparison summary.

	ACNMP	ProMP+RL
# of trajectories	130	1200 [1]
# of trajectories (single environment)	1.4K	20K [2]
# of trajectories (total)	0.7M	10M [2]

In the second experiment, we replicated a 2D obstacle avoidance task [69] of [2] that is shown in Figure 5.6 (c). Here, where the model is required to produce trajectories that start from a fixed point (shown with blue), go through the aperture of the wall without collision, and end at a randomly located goal point (shown with red).

The task parameters are coordinates of the center of the aperture in the wall (x_c, y_c), and coordinates of the goal point (x_g, y_g). The objective is determined as minimizing the distance to the start and goal point and the signed distance to the walls. The coordinates of the wall and goal point are sampled uniformly from the following ranges to create various environments: $2 \leq x_c \leq 8$, $1 \leq y_c \leq 9$, $x_c + 1.5 \leq x_g \leq 10$ and $0 \leq y_g \leq 10$. 30 environments with demonstrations are used to initialize both models. Figure 5.7 shows sample trajectories that ACNMP used. Note that all trajectories are traveled uniformly in 200 time points and aperture behaviors are not aligned in time. Therefore, ACNMP is successful at preserving non-aligned demonstrated shapes as well. Both models use RL to create a self-improvement loop to be able to generate sufficient trajectories for any random points from the specified range. Figures 5.6 (d) and (e) show performance comparisons with respect to environments used in the self-improvement loop and samples used in total for 1000 unseen test environments. Although ACNMP and [2] show similar performance increase with respect to environments, ACNMP is much more efficient in terms of trajectory samples used for training. On average, ACNMP uses 1400 trajectories at maximum for each new environment whereas [2] uses 20k trajectories. In the end, the performance level achieved in 700k samples by ACNMP can be achieved in 10 Million samples by [2]. The Table 5.1 summarizes the comparison of ACNMP with the two baseline ProMP+RL methods. This difference makes ACNMP more suitable for real-world applications.

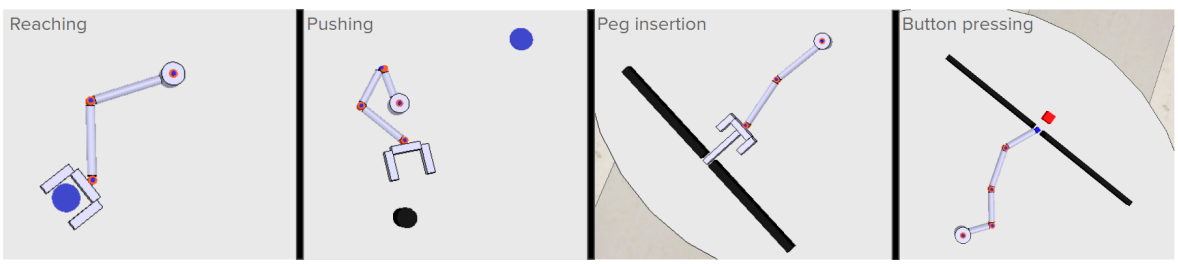

Figure 5.8. The three training tasks (on the left) and the test task (on the right) for transfer learning.

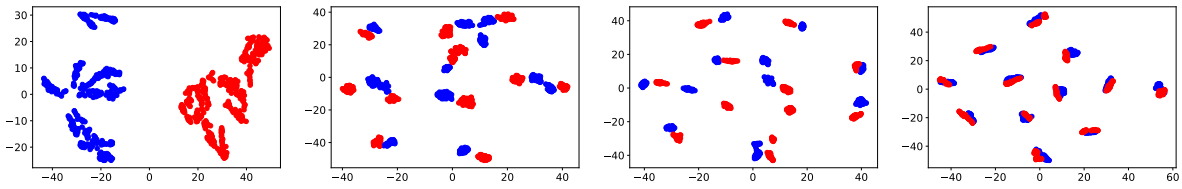

Figure 5.9. Snapshots of overlaid latent space representations obtained during learning.

5.2.3. Skill Transfer between Robots with Different Morphologies

The last simulation experiment analyzes the transfer learning capability of ACNMP. This experiment setup is adopted from [55] based on the descriptions in the paper. There are 2 robots with different numbers of DoFs (3 and 4) and different link lengths simulated in CoppeliaSim [25] using the PyRep toolkit [70]. Both robots are provided with 4 demonstrations of 3 proxy skills: reaching, pushing, and peg insertion as shown in Figure 5.8. Figure 5.9 illustrates how a common latent space is formed using these proxy skills for the two robots. Here, t-SNE visualizations [71] of the learned representations at different phases of training are plotted from left to right. For the test task, named button pressing and shown at the right of Figure 5.8, a demonstration is only provided to the 3 DoF robot and the skill transfer to the 4 Dof robot is tested. This task, where the robot should reach the blue cube through a narrow opening and push it to the location of the red cube, is extremely hard to learn from scratch because the reward is determined as the minus distance of the blue cube to the red cube. The robot does not get any feedback unless it can touch the blue cube, which has a very low probability due to obstacles around it. So, the knowledge transfer should generate a trajectory for the 4 DoF robot close enough to the optimal one. Indeed, this was the case for both models, and RL algorithms optimize this skill in comparable iterations. The difference is that [55] used an explicit time-warping mechanism to align and pair the states before the learning but ACNMP was able to learn the correspondences at trajectory level and did not require any state aligning or state matching mechanism.

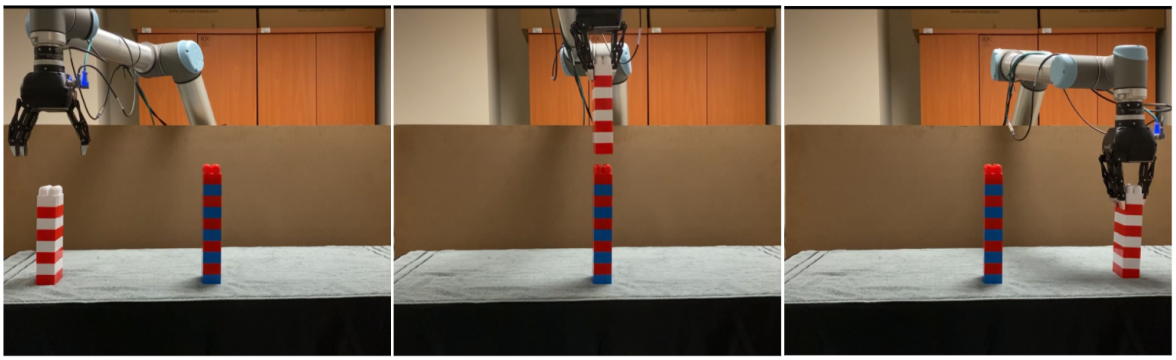

Figure 5.10. 3 snapshots from the pick & place task.

5.2.4. Extrapolation in Real-Robot Adaptation

The first real robot experiment, which is adopted from [13] using [72], targets to evaluate the adaptation ability of ACNMP in a non-linear and high-dimensional task. Here, a 6 DoF UR10 robotic arm mounted with a Robotiq gripper should carry an object with varying sizes over a tower with varying sizes (Figure 5.10). LfD-only model (CNMP) was successful to generalize the skill within the demonstration range (interpolation range) but it failed to produce successful trajectories when it was faced with objects and towers higher than the demonstrated ones [13].
ACNMP was also trained in similar settings to see the performance improvement. 8 demonstrations are provided for object heights between 2 and 6 cm and tower heights between 2 and 8 cm. The reward is determined as the sum of minus distances to obstacle avoidance and grasp points. The left part of the Table 5.2 shows that ACNMP successfully adapted to novel task requirements that have non-linear relationships with the trajectories of 6 joints. Figure 5.12 shows the solution trajectories of ACNMP for the task parameters the furthest away from the demonstration range. Note that ACNMP solutions preserve the demonstrated shapes even though the reward function does not include a term for them.

The second real robot experiment shows that ACNMP can also adapt to novel task constraints outside of the demonstration range using the reward coming from the real-world that is not directly related to the demonstration trajectories.

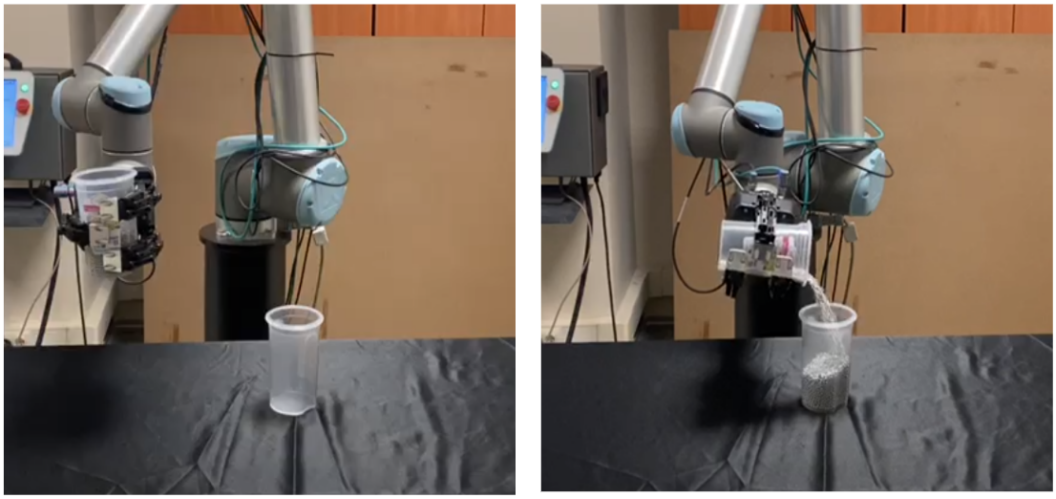

Figure 5.11. 2 snapshots from the pouring task.

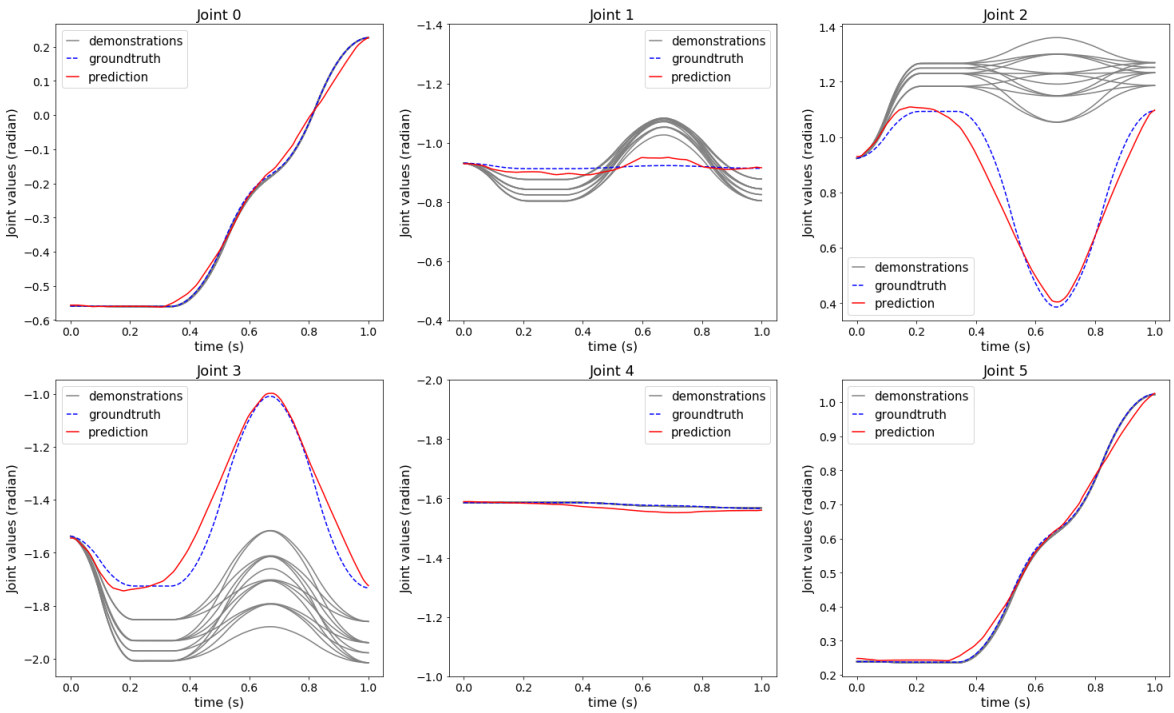

Figure 5.12. ACNMP predictions for pick and place task. Gray lines show expert demonstrations. Blue lines show the expert behaviour, and the red lines show ACNMP prediction for the extrapolation task parameters.

Table 5.2. Left: Average joint errors in pick & place task. Right: Error change in pouring during RL.

		Joint Errors (deg)					
Method	**Parameters**	**Base**	**Shoulder**	**Elbow**	**Wrist1**	**Wrist2**	**Wrist3**
CNMP	Obj: 8* Obs.: 10*	1.038	2.325	8.879	8.094	0.360	0.830
ACNMP	Obj: 8* Obs.: 10*	0.889	1.589	4.668	3.465	0.845	0.861
CNMP	Obj: 9* Obs: 11*	1.861	5.380	17.978	14.894	0.590	1.386
ACNMP	Obj: 9* Obs: 11*	0.994	0.507	1.780	1.514	0.814	1.007

	Error in Grams			
1-4	591	447	435	175
5-8	511	604	305	42
9-12	417	284	144	194
13-16	367	364	79	177
17-18	133	**27**		

The robot is provided with a cup that contains marbles with varying weights. It is supposed to pour the varying target amount into a second cup (Figure 5.11). Therefore, the task parameters are the target pouring amount and weight of the cup, and ACNMP is conditioned with them to produce the required trajectory for the wrist joint. The LfD model was trained with 16 expert demonstrations with initial cup weights in the range of 85-95 degrees. The extrapolation task includes 1400 gram initial cup weight of which 1322 grams need to be poured. ACNMP was able to find a suitable trajectory using only 18 rollouts. The learning progress is shown in the right part of the Table 5.2. In the beginning, the RL agent was triggered with 600 grams error, which was dropped to 27 grams after training. In the two real robot experiments, the built-in controller that comes with robot software is used because ACNMP produces trajectories.

6. REWARD CONDITIONED NEURAL MOVEMENT PRIMITIVES

6.1. Proposed Method

Reward Conditioned Neural Movement Primitives (RC-NMP) is a robot skill learning framework, which is trained fully using supervised learning. It captures the trajectory distribution in the latent space using variational inference. For the given target reward, it can generate various trajectories to create a trajectory population thanks to the stochastic sampling from the latent space. The diversity of this population is increased by benefiting from Evolutionary Strategies with a novel crossover operation organized in the latent space and a mutation operation. This diversity helps with the exploration of the environment in order to deal with local minima, sparse rewards, and multiple solutions.

The learning loop of the proposed framework is illustrated in Figure 6.1. Replay buffer saves the training data, previous trajectory executions (I). The ANN uses this data to learn to produce various trajectories for desired rewards (II). A trajectory population is generated by conditioning the network with the maximum reward that is experienced. Diversity of trajectories is increased with a crossover operation applied in the latent space (IV) and a mutation operation applied in the task space (V). The generated trajectories are given to a PD controller for execution (VI), and the resulting trajectories are added to the replay buffer. In the following, the details of the architecture are provided in detail.

Replay Buffer (I). The replay buffer consists of sensorimotor (SM) trajectories and the reward values obtained after execution of the trajectories. The replay buffer can be initialized with demonstrations or random trajectories with the corresponding rewards. Our system uses these reward trajectory pairs to learn a model that can generate SM trajectories that give the desired reward as follows.

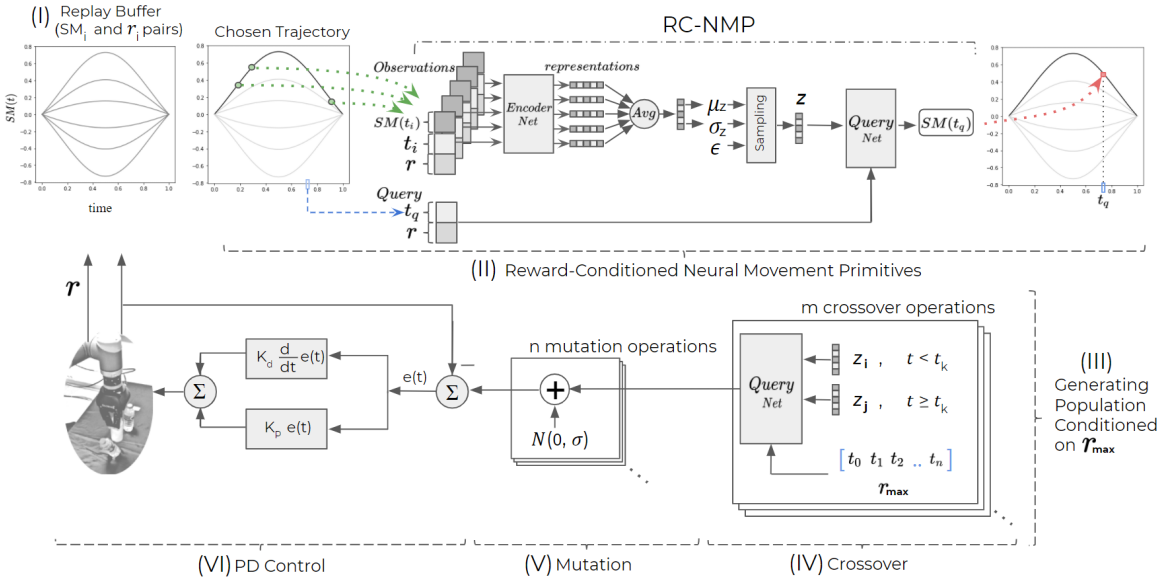

Figure 6.1. Training the RC-NMP by sampling observations from the demonstration set and predicting the trajectory value over the queried time-step and generating a trajectory according to the best reward parameter at test time. © 2021 IEEE

Reward Conditioned Neural Movement Primitives (RC-NMP) (II). Before getting to the training model, we would like to introduce the problem formulation where there is a reward function (R) that maps trajectories (τ) to real-valued rewards: $R(\tau) = r, r \in \mathbb{R}$. Standard reinforcement learning algorithms maximize expected reward $\mathbb{E}_{\tau \sim \pi_\theta(\tau)}[R(\tau)]$ by updating the policy parameters θ. On the other hand, RC-NMP aims to generate the trajectory distribution that returns the global maximum reward value or a satisfactory local maximum (r^*). Our approach takes the direct path by conditioning the trajectory generation with rewards r, and it is trained using the reconstruction error between the generated trajectory and the original trajectory. The network updates its parameters to find a trajectory distribution conditioned on r^*.

Figure 6.1 (II) illustrates the training of our ANN using (SM, r) pairs. The neural network architecture is built upon Neural Processes [19] explained in the Section 4.2. At each training iteration, a random SM trajectory and its reward value are selected from the replay buffer. A random number of observations (O) consist of SM value and time tuples (the green dots) are sampled from random locations along the trajectory.

The parameter sharing encoder network finds the corresponding representations for these observations as in CNP and NP. Then, these representations are merged to find the parameters of a Gaussian distribution in the latent representation space (μ_z, σ_z). Differently from NP, we are using a Gaussian prior $N(0, I)$ for the latent space. The model is trained using variational inference similar to Conditional Variational Autoencoders (CVAE) [73] and β-Variational Autoencoders [74]. Reconstruction error is calculated with log-likelihood of the trajectory, and it is composed of the log-likelihoods of individual data points: $\log p_\theta(\tau|t,r) = \sum_{t=1}^{T} \log p_\theta(x_t|t,r)$.

By following the steps in [66], this log-likelihood can be expressed as:

$$\begin{aligned} \log p_\theta(x_t|t,r) \quad &= \mathbb{E}_{q_\phi(z|O,r)}[\log p_\theta(x_t|t,r)] \\ &= \mathbb{E}_{q_\phi(z|O,r)}[\log \frac{p_\theta(x_t,z|t,r)}{q_\phi(z|O,r)}] \\ &+ \mathbb{E}_{q_\phi(z|O,r)}[\log \frac{q_\phi(z|O,r)}{p_\theta(z|x_t,t,r)}] \end{aligned} \tag{6.1}$$

Evidence lower bound (ELBO) is the first term before the plus, and the non-negative Kullback-Leibler (KL) divergence term is right after the plus. Therefore, we can conclude that ELBO is a lower bound on the likelihood of data, and it will be equal to the log-likelihood when the KL term goes to 0. Thus, it can be used as a training objective. By following steps in [74], ELBO can also be expressed as:

$$\begin{aligned} \mathcal{L}(\theta,\phi) \quad &= \mathbb{E}_{q_\phi(z|O,r)}[\log p_\theta(x_t|z,t,r)] \\ &- \beta D_{KL}\left(q_\phi(z|O,r) \| p_\theta(z|t,r)\right) \end{aligned} \tag{6.2}$$

Gaussian $N(0, I)$ can be used in the place of $p_\theta(z|t,r)$ because we sample trajectories from this Gaussian independently at the test time [75]. Our proposed model is different from CNMP [13], where trajectories are mapped to deterministic representations, and the trajectory distribution is represented in the task space. RC-NMP learns to represent the trajectory distribution in the latent space, and differently, RC-NMP uses rewards as external task parameters for conditioning.

Generating population of individual solutions (III). After RC-NMP is trained and the mapping between trajectory distributions and reward values is learned, our system attempts to search the environment to find better trajectories to increase the reward gradually. For this purpose, multiple trajectories are produced by conditioning the ANN with the maximum reward that has been experienced until that training iteration. Various latent representations z_i are sampled from the latent distribution, and the diversity is increased to help exploration by applying two operations from evolutionary algorithms as follows evolutionary operations as follows.

Crossover operation (IV). The decoder network produces a full trajectory when it is queried with all time points, the latent representation that explains the trajectory characteristics, and the target reward. In order to create offspring from previously generated representations, the trajectories are temporally blended as a crossover operation through representations. Two random representation pairs (z_i, z_j), and a random time point t_k from the time horizon $t_0 - t_n$ are selected. Afterward, the query network generates the part of the trajectory until t_k using z_i and the rest of the trajectory using with z_j. The resulting trajectory carries one trajectory part of each of its parents. The symmetric operation is also applied (first z_j, then z_i), and 2 offspring trajectories are obtained in total from 2 parents. This operation is repeated for m trajectories, another m trajectories are generated.

Mutation operation (V). We also apply a mutation operation in task space to diversify solutions and increase the exploration further. A smoothed Gaussian noise with zero mean that shares the same time horizon is added to each trajectory. The standard deviation (σ) of the noise is a hyperparameter tuned for each movement dimension and gradually decreased during training. Since the latent representation distribution fits the trajectory distribution in the replay buffer, traversing in latent space provides limited exploration. Therefore, we used mutation operation in the task space.

PD controller (VI). We also employed a PD controller to ensure that the trajectories will reach the target point and they will be smooth enough for robotic applications. All trajectories obtained after mutation operation are given to PD controller for execution. Since our PD controller has two objectives: ending at the goal point and smooth trajectory following, we use 2 weight parameters to balance the behavior:

$$\begin{aligned} u(t) &= K_p * e(t) - K_d * \frac{de(t)}{dt} \\ e(t) &= \lambda_1 * (g - x(t)) + \lambda_2 * (\tau(t) - x(t)) \end{aligned} \tag{6.3}$$

where g stands for the goal position. $\tau(t)$ and $x(t)$ stand for desired and current positions respectively, and $e(t)$ denotes the error between the two at time t. K_p and K_d are PD parameters that define control behavior. λ_1 is the coefficient of the objective goal-reaching, and λ_2 is the coefficient for smooth trajectory following. Since we want λ_1 to be active around the end of the trajectory, λ_1 grows exponentially over the time horizon between 0 and 1. Similarly, λ_2 does the opposite to ensure that the given trajectory is followed as much as possible.

After trajectories are played on the robot, best-performing trajectories and a few random trajectories are added to the replay buffer, the process repeats, and the framework improves itself progressively.

6.2. Experiments

We evaluated the performance of RC-NMP in three experiments, where the state-of-the-art comparisons were made. The dataset and code can be found in [76].

6.2.1. Stochastic Sampling in Latent Space

This experiment investigates how well RC-NMP generates trajectories that cover the demonstration distribution by stochastic sampling in comparison with its deterministic counterpart CNMP [13]. Here, 6 colored demonstrations are provided to both models as in Figure 6.2.

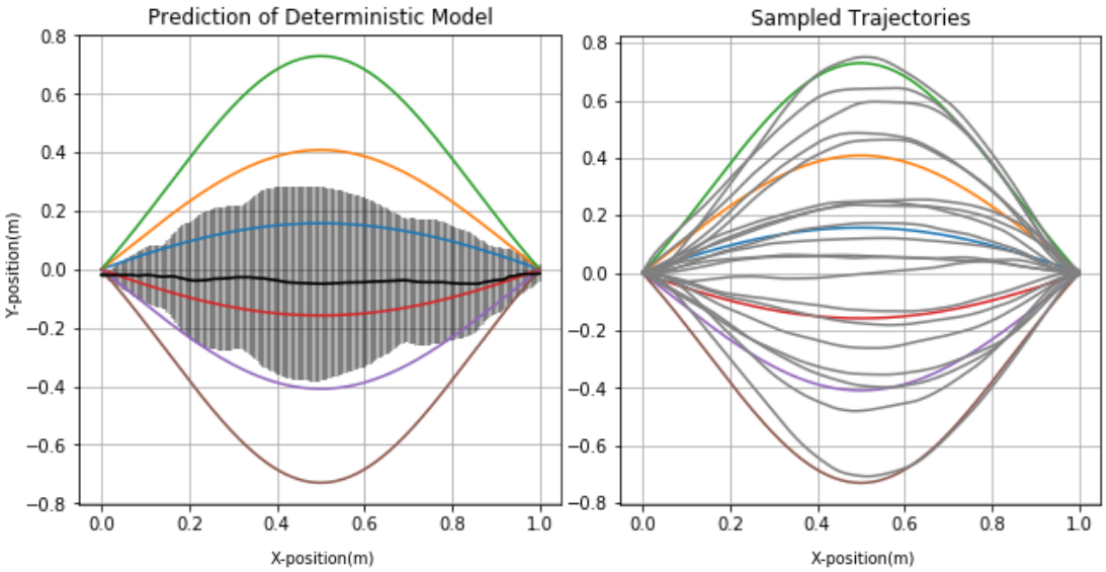

Figure 6.2. 6 demonstrations are shown with colored curves in both figures. Given the initial position, the generated trajectories for CNMP (left) and RC-NMP (right) are compared. © 2021 IEEE

The trajectories start from the same $(0, 0)$ point, follow a curvy path with different lengths, and end at the same $(1, 0)$ point. Both models are trained without any task parameters like the reward because the purpose is to analyze the contribution of variational inference in trajectory generation. At test time, both models are conditioned with the $(0, 0)$ start point, and this point is shared by all demonstrations. Therefore, the models are expected to output the whole demonstration distribution. The left part of Figure 6.2 shows the performance of CNMP, which represented the trajectory distribution in task space by finding the mean of the demonstrations with the bold black line and the variance of the trajectories with the gray shaded area. Note that this model lacks a global latent variable to sample a trajectory from the shaded region. One possible forced approach would be sampling SM points for each time point and connecting them via a trajectory shape mechanism. On the other hand, RC-NMP can generate trajectories that cover the demonstration range as shown in the right part of Figure 6.2. RC-NMP does possess a global latent distribution to sample representations that correspond to trajectories. Note that RC-NMP can generate trajectories that look similar to demonstrations at the edges of the demonstration range.

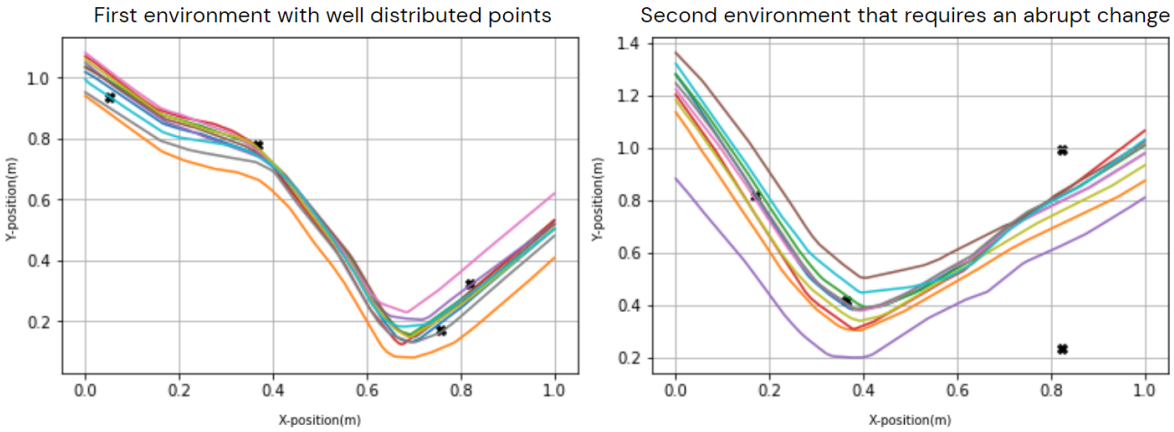

Figure 6.3. Example environment configurations for the task generating complex trajectories. The complexity is determined how random via-points are placed, and it affects the model's performance. © 2021 IEEE

6.2.2. Performance in Generating Complex Trajectories

This experiment investigates the limit of our model in finding and generating the most rewarding complex trajectories in comparison with our first proposed method ACNMP. In addition, the contribution of our genetic operations was investigated by ablation experiments, where either the mutation operation or the crossover operation was removed from the learning cycle. In the experiment, a random integer between 1 and 6 is generated indicating the number of locations (via-points) to go through. To give context, the expert may not be able to provide full trajectories but via-points, guiding the robot to follow some important locations or to avoid collisions while learning. The x and y positions are sampled from a uniform distribution: $X \sim U(0,1)$ and $Y \sim U(0,1)$), and the x position of trajectory is changing linearly with time. The reward function is defined as the average of minus distances from the trajectory to all the via-points. Here, PD controllers are not used not to create any biases. Trajectories generated by RC-NMP are shown for 2 example task configurations with 4 points in Figure 6.3. The black dots represent the via-points. For each of the number of via-points, 10 different environment configurations were created, and all the models were trained until they sampled 300 trajectories.

The left part of Figure 6.4 illustrates performances of all models (RC-NMP, RC-NMP without crossover, RC-NMP without mutation, and ACNMP) in terms of error decrease. The colored bold lines indicate the mean performances and shaded areas indicate the standard error throughout different runs. It is shown that crossover operation accelerates the learning in the beginning because the trajectory population has a high genetic diversity although they are far from optimal. This genetic diversity increases the probability of combining 2 locally well-performing parts of trajectories through the crossover operation and having a rapid performance increase. On the other hand, mutation operation is more useful at the later stages of learning. We see that the progress stops without small changes made by mutation operation when generated trajectories are close to the optimal solutions. When both operations are included, RC-NMP can show fast progress at the beginning and produce optimal trajectories at the end. Our first method, ACNMP, progressed fast at the beginning as well but its performance was marginally worse than RC-NMP at the end. The most important difference is that ACNMP shows oscillatory progress like most of the RL algorithms but RC-NMP offers more stable learning because it uses error-based Supervised Learning. A mapping from reward values to trajectory distributions is learned by utilizing both poor and well-performing trajectories fully. In addition, the most rewarding trajectories can be transferred to the next generations to avoid performance drop. On the other hand, ACNMP updates its parameters using gradient descent to make well-performing trajectories more likely and poor-performing trajectories less likely. The gradient is dependent on the current batch of experiences, and it may become noisy. Old trajectories can not be transferred to the next batch because then, the policy that produced the trajectories and the current policy would be different. In this case, the gradients may become even noisier.

The right part of Figure 6.4 plots the error of RC-NMP with respect to the number of via-points. The performance of RC-NMP drops if the number of points increases because challenging configurations are more likely to happen. For instance, RC-NMP achieves the perfect score in environments with 2 points because it is possible to find a simple line. Figure 6.3 illustrates this fact more in detail.

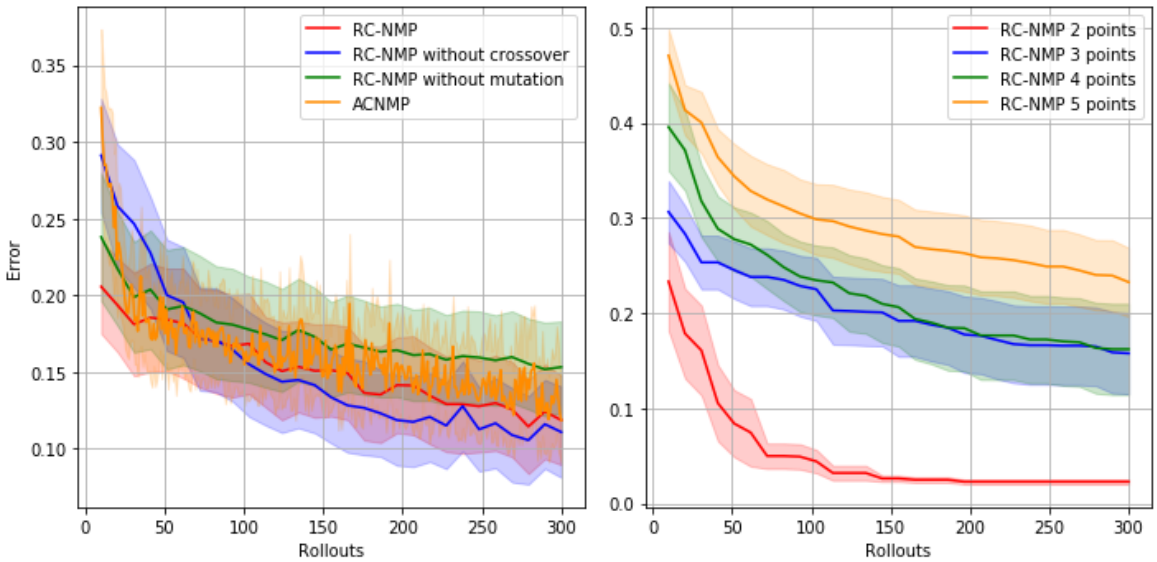

Figure 6.4. Learning performance of RC-NMP, RC-NMP without crossover, RC-NMP without mutation, and ACNMP for Experiment 6.2.2 on the left, and learning performance of RC-NMP for different number of via-points on the right. © 2021 IEEE

The via-points in the left environment are well-spaced along both the x and y-axis and the model can produce trajectories that meet the via-points. On the other hand, the last two points in the last figure are very close to each other along the x-axis but very distant from each other along the y-axis. In addition, the second via-point is at the bottom side and the trajectories require a sharp turn to go through all the via-points. RC-NMP could not generate successful trajectories for this challenging configuration.

6.2.3. Multi-modal problem: Passing through Linearly Aligned Objects

This experiment aims to analyze our method in a challenging planar obstacle avoidance task, and compare its performance with two baseline RL algorithms, DREPS [50] and REPS [49]. This task was introduced in [50] to create a problem with multiple solutions and a challenging reward function. The task is on a tabletop setting where there are two bottles placed close to each other, but with enough space so that the end-effector can pass between them safely.

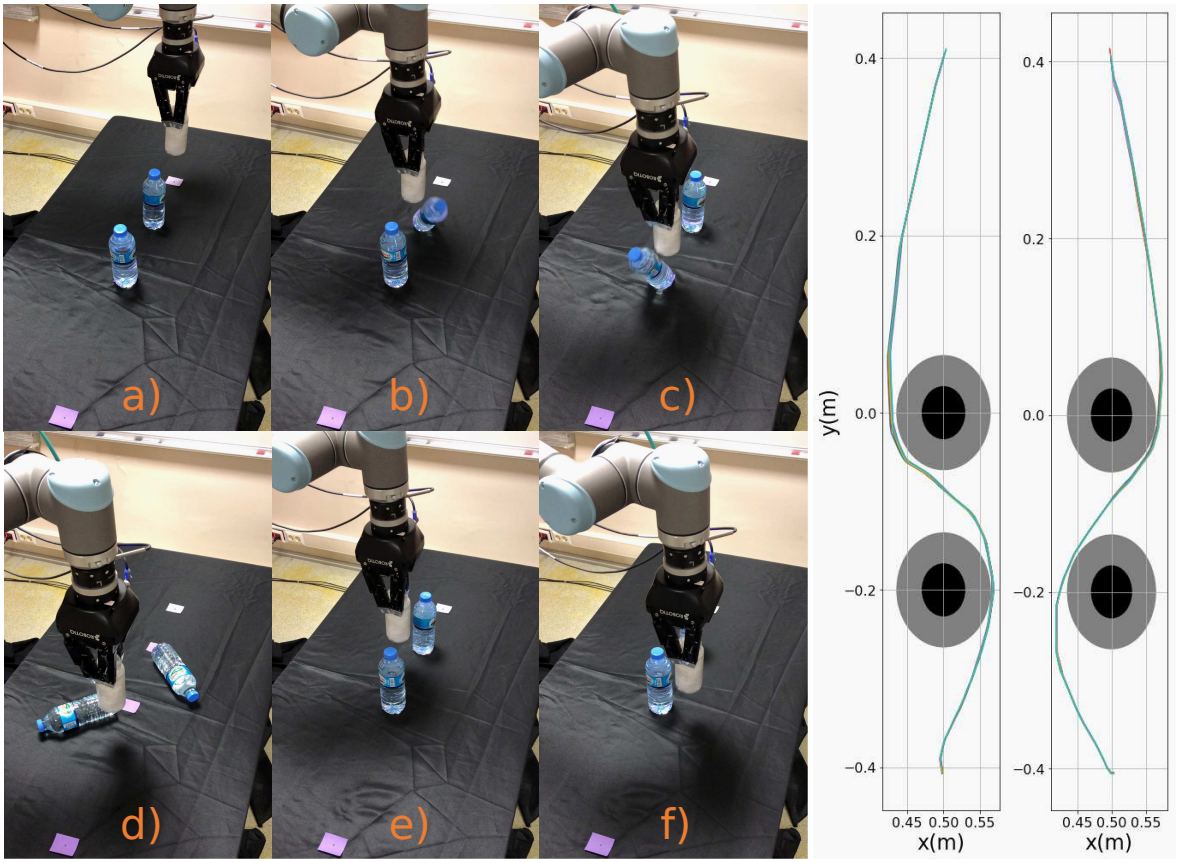

Figure 6.5. Snapshots from the 'passing through linearly aligned objects experiment. (a) gives the setup; (b,c,d) show snapshots from collisions observed during learning, and (e,f) show snapshots from successful execution after learning is accomplished. The produced trajectories for 2 modes are shown in the plots on the right. The black circle represents the bottle on the table and it is expanded with the radius of the bottle at the hand of the robot to the gray circle.

The end-effector of the robot should start from one side of the table and should pass between these bottles without knocking them over. The reward function for the task encourages quick completion by punishing longer trajectories. Other than that, it gives sparse signals, which makes the problem harder. There is a binary penalty for knocking each bottle over, and there is also a binary penalty for not passing between the bottles; which is a symmetric reward considering that this way the robot can pass from left to right or from right to left. The expected result is similar to an S-shaped or reverse S-shaped trajectory, so this is in fact a multi-modal problem.

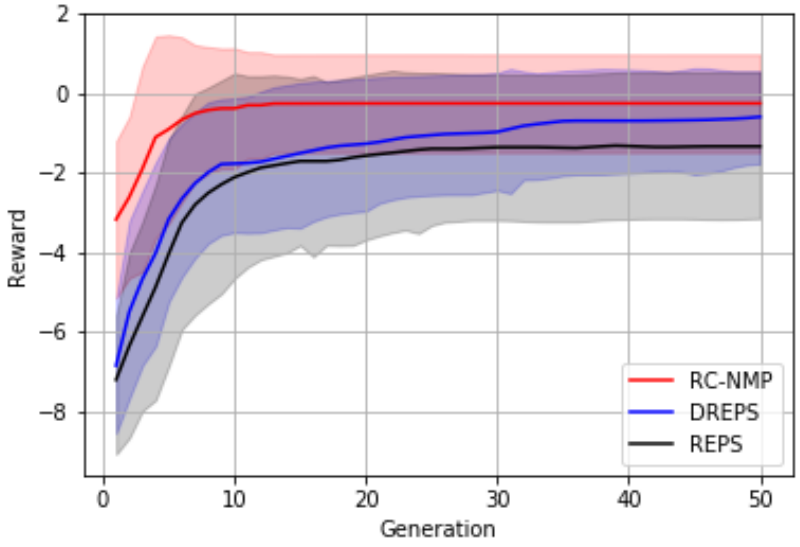

Figure 6.6. Learning curves of RC-NMP, DREPS, and REPS for the multi-modal problem for 50 runs. © 2021 IEEE

Many approaches like [49] get stuck between two alternative minima, different modes of solutions as reported in [50]. The reward function is:

$$\mathrm{R} = -2N_{\mathrm{bottlesdown}} - 4\mathrm{I}_{\mathrm{cross}} - 0.15L_{\mathrm{trajectory}} \tag{6.4}$$

where $N_{\mathrm{bottlesdown}}$ stands for the number of bottles that the robot collided with, $\mathbf{I}_{\mathrm{cross}}$ denotes whether a crossing between bottles happened, and $L_{\mathrm{trajectory}}$ stands for length of the trajectory in meters. Note that there is no term that indicates the direction of the crossing.

In the experiment, the sum of radii of the bottle and the end effector was set as 6.5 cm like in [50]. Similar to DREPS, we used simulated experiences to learn the model and then transferred the end results to the real robotic system. We realized the experiment setup using the UR10 robot. To create a similar environment in the real world, we used bottles with a radius of 3 cm as obstacles and gave another bottle with a similar size to the gripper. The remaining 0.5 cm left is the error margin because the bottles can not be placed inch-perfect by us on the table between training runs.

RC-NMP uses a straight line to initialize the replay buffer like the two other models (REPS and DREPS). Both baselines use DMP as the movement primitive architecture and they fit DMP to the trajectory that goes straight from the initial point to the endpoint. On top of this, DREPS and REPS learn the weights of the basis functions of DMP to solve the task. DMP can satisfy initial and end position requirements naturally because of its internal PD controller. Therefore, we employ the PD controller described in 6.1, to run the trajectories found by RC-NMP on the robot. The models are trained with 100 runs repeatedly in 50 experiments, and RC-NMP found a satisfactory solution, that ends up at the final point after crossing between bottles without touching them, in 48 of 50 trials. DREPS and REPS had success 47 and 35 times respectively. Figure 6.6 illustrates this performance comparison. The bold lines show the mean reward obtained in 50 experiments and the shaded area indicates the standard deviation. The higher bold line means the method found a satisfactory solution more in 50 experiments. When we check the graph with respect to generation number, we see that RC-NMP finds the sufficient trajectory in around 10 generations and obtains almost the maximum reward, whereas DREPS requires 50 generations to reach the same reward level, and REPS could not solve the task in many cases. At each generation, our method samples 20 trajectories for the initial population, and 20 more trajectories are obtained using the crossover operation, whereas DREPS and REPS use 50 trajectories for each policy update. As a result, RC-NMP uses 5 times fewer samples than the two baselines. Figure 6.5 provides snapshots of robot trajectory execution at different generations during training and illustration of solution trajectories found by RC-NMP for two modes. Note that trajectories for two modes are obtained in different runs, and the variance of trajectory distribution is small since reward values are continuous and they correspond to a single trajectory during learning.

7. DISCUSSION AND CONCLUSION

In this thesis, we proposed two skill learning methods using the movement primitives approach. The first one, ACNMP, employs an LfD network at core, improves the learned skill whenever the network fails because of new task constraints or a new environment. It utilizes Reinforcement Learning and Supervised Learning simultaneously to outperform two other state-of-the-art adaptive movement primitive approaches in terms of adaptation performance and sample efficiency while still preserving the key characteristics of the demonstration data. Furthermore, ACNMPs are shown to construct a common latent space for different robots to transfer skills even if they have different morphology. The knowledge transfer is used to solve spare reward tasks, which would be extremely hard to solve from scratch. While the proof-of-the-concept realization of ACNMP in real robots has been provided, ACNMP is planned to be verified in more complex real-robot problems and settings in the future. Next, we extended the idea of using Supervised Learning in task adaptation and proposed another framework, RC-NMP, that learns to generate trajectory distributions conditioned on the desired reward, and trained fully by Supervised Learning. Variational inference is used to form latent representation distribution where samples are used to generate varying full trajectories for creating a population. The diversity of this population is increased via a crossover operation that blends the trajectories w.r.t. time and a mutation operation implemented in task space as additive noise. The genetic diversity is shown to be quite beneficial for environments with sparse rewards, multiple solutions, and local minima. The experiments verified that RC-NMP is more stable than its policy gradient variant ACNMP, and it is more sample efficient than two other RL algorithms used to find parameters of another movement primitives architecture. In the future, we plan to extend the latent space with discrete variables which capture different modalities in the data.

REFERENCES

1. Stark, S., J. Peters and E. Rueckert, "Experience Reuse with Probabilistic Movement Primitives", *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, Macau, China, November 4-8, 2019.

2. Ewerton, M., O. Arenz, G. Maeda, D. Koert, Z. Kolev, M. Takahashi and J. Peters, "Learning Trajectory Distributions for Assisted Teleoperation and Path Planning", *Frontiers in Robotics and AI*, Vol. 6, p. 89, 2019.

3. Argall, B. D., S. Chernova, M. Veloso and B. Browning, "A Survey of Robot Learning from Demonstration", *Robotics and Autonomous Systems*, Vol. 57, No. 5, pp. 469–483, 2009.

4. Atkeson, C. G. and S. Schaal, "Memory-Based Neural Networks for Robot Learning", *Neurocomputing*, Vol. 9, No. 3, pp. 243–269, 1995.

5. Peternel, L., T. Petric, E. Oztop and J. Babic, "Teaching Robots to Cooperate with Humans in Dynamic Manipulation Tasks Based on Multi-Modal Human-in-the-Loop Approach", *Autonomous Robots*, Vol. 36, No. 1-2, pp. 123–136, 2014.

6. Sutton, R. S. and A. G. Barto, *Reinforcement Learning: An Introduction*, MIT Press, 2018.

7. Kroemer, O., S. Niekum and G. Konidaris, "A Review of Robot Learning for Manipulation: Challenges, Representations, and Algorithms", *Journal of Machine Learning Research*, Vol. 22, No. 30, pp. 1–82, 2021.

8. Mnih, V., K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski *et al.*, "Human-Level Control through Deep Reinforcement Learning", *Nature*, Vol. 518, No. 7540, pp. 529–533, 2015.

9. Silver, D., A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. Van Den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot *et al.*, "Mastering the Game of Go with Deep Neural Networks and Tree Search", *Nature*, Vol. 529, No. 7587, pp. 484–489, 2016.

10. Lillicrap, T., J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver and D. Wierstra, "Continuous Control with Deep Reinforcement Learning", *CoRR*, Vol. abs/1509.02971, 2016.

11. Dulac-Arnold, G., D. J. Mankowitz and T. Hester, "Challenges of Real-World Reinforcement Learning", *ArXiv*, Vol. abs/1904.12901, 2019.

12. Akbulut, M. T., E. Oztop, M. Y. Seker, H. Xue, A. E. Tekden and E. Ugur, "ACNMP: Skill Transfer and Task Extrapolation through Learning from Demonstration and Reinforcement Learning via Representation Sharing", *Conference on Robot Learning*, Massachusetts, USA, November 16-18, 2020.

13. Seker, M. Y., M. Imre, J. Piater and E. Ugur, "Conditional Neural Movement Primitives", *Proceedings of Robotics: Science and Systems*, Freiburg, Germany, July 13-17, 2019.

14. Garnelo, M., D. Rosenbaum, C. Maddison, T. Ramalho, D. Saxton, M. Shanahan, Y. W. Teh, D. Rezende and S. M. A. Eslami, "Conditional Neural Processes", *International Conference on Machine Learning*, pp. 1704–1713, Stockholm, Sweden, July 10-15, 2018.

15. Barto, A. G. and T. G. Dietterich, "Reinforcement Learning and Its Relationship to Supervised Learning", *Handbook of Learning and Approximate Dynamic Programming*, Vol. 10, 2004.

16. Schmidhuber, J., "Reinforcement Learning Upside Down: Don't Predict Rewards - Just Map Them to Actions", *ArXiv*, Vol. abs/1912.02875, 2019.

17. Srivastava, R., P. Shyam, F. W. Mutz, W. Jaśkowski and J. Schmidhuber, "Training Agents using Upside-Down Reinforcement Learning", *ArXiv*, Vol. abs/1912.02877, 2019.

18. Akbulut, M. T., U. Bozdogan, A. Tekden and E. Ugur, "Reward Conditioned Neural Movement Primitives for Population Based Variational Policy Optimization", *International Conference on Robotics and Automation*, Xi'an, China, May 30-June 5, 2021.

19. Garnelo, M., J. Schwarz, D. Rosenbaum, F. Viola, D. J. Rezende, S. Eslami and Y. Teh, "Neural Processes", *ArXiv*, Vol. abs/1807.01622, 2018.

20. Blei, D. M., A. Kucukelbir and J. D. McAuliffe, "Variational Inference: A Review for Statisticians", *Journal of the American Statistical Association*, Vol. 112, No. 518, p. 859–877, April 2017.

21. Nolfi, S., D. Floreano and D. D. Floreano, *Evolutionary Robotics: The Biology, Intelligence, and Technology of Self-Organizing Machines*, MIT Press, 2000.

22. Universal Robots, *UR10 Website*, `https://www.universal-robots.com/products/ur10-robot/`, accessed at June 2021.

23. Roswiki, *Robot Operating System*, `http://wiki.ros.org/`, accessed at June 2021.

24. Universal Robots, *3-Finger Robotiq Gripper Website*, `https://robotiq.com/products/3-finger-adaptive-robot-gripper`, accessed at June 2021.

25. Coppelia Robotics, *CoppeliaSim Robot Simulator*, `http://www.coppeliarobotics.com/`, accessed at June 2021.

26. Rohmer, E., S. P. N. Singh and M. Freese, "V-REP: A Versatile and Scalable Robot Simulation Framework", *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 1321–1326, Tokyo, Japan, November 3-7, 2013.

27. Hunter, J. D., "Matplotlib: A 2D Graphics Environment", *Computing in Science & Engineering*, Vol. 9, No. 3, pp. 90–95, 2007.

28. Pedregosa, F., G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg *et al.*, "Scikit-Learn: Machine Learning in Python", *Journal of Machine Learning Research*, Vol. 12, No. Oct, pp. 2825–2830, 2011.

29. Oliphant, T. E., *A Guide to NumPy*, Vol. 1, Trelgol Publishing USA, 2006.

30. Calinon, S., P. Evrard, E. Gribovskaya, A. Billard and A. Kheddar, "Learning collaborative manipulation tasks by demonstration using a haptic interface", *2009 International Conference on Advanced Robotics*, pp. 1–6, Munich, Germany, June 22-26, 2009.

31. Asfour, T., P. Azad, F. Gyarfas and R. Dillmann, "Imitation Learning of Dual-Arm Manipulation Tasks in Humanoid Robots", *International Journal of Humanoid Robotics*, Vol. 5, No. 02, pp. 183–202, 2008.

32. Pastor, P., L. Righetti, M. Kalakrishnan and S. Schaal, "Online Movement Adaptation Based on Previous Sensor Experiences", *International Conference on Intelligent Robots and Systems*, pp. 365–371, San Francisco, USA, September 25-30, 2011.

33. Ben Amor, H., O. Kroemer, U. Hillenbrand, G. Neumann and J. Peters, "Generalization of Human Grasping for Multi-Fingered Robot Hands", *International Conference on Intelligent Robots and Systems*, pp. 2043–2050, Algarve, Portugal, October 7-12, 2012.

34. Mühlig, M., M. Gienger and J. J. Steil, "Interactive Imitation Learning of Object Movement Skills", *Autonomous Robots*, Vol. 32, No. 2, pp. 97–114, 2012.

35. Schaal, S., "Dynamic Movement Primitives-a Framework for Motor Control in

Humans and Humanoid Robotics", *Adaptive Motion of Animals and Machines*, pp. 261–280, Springer, 2006.

36. Calinon, S., "A Tutorial on Task-Parameterized Movement Learning and Retrieval", *Intelligent Service Robotics*, Vol. 9, No. 1, pp. 1–29, 2016.

37. Girgin, H. and E. Ugur, "Associative Skill Memory Models", *International Conference on Intelligent Robots and Systems*, pp. 6043–6048, Madrid, Spain, October 1-5 2018.

38. Ugur, E. and H. Girgin, "Compliant Parametric Dynamic Movement Primitives", *Robotica*, Vol. 38, No. 3, pp. 457–474, 2020.

39. Paraschos, A., C. Daniel, J. Peters and G. Neumann, "Probabilistic Movement Primitives", C. J. C. Burges, L. Bottou, M. Welling, Z. Ghahramani and K. Q. Weinberger (Editors), *Advances in Neural Information Processing Systems*, pp. 2616–2624, Curran Associates, Inc., 2013.

40. Chen, N., M. Karl and P. van der Smagt, "Dynamic Movement Primitives in Latent Space of Time-Dependent Variational Autoencoders", *International Conference on Humanoid Robotics*, Cancun, Mexico, November 15-17, 2016.

41. Noseworthy, M., R. Paul, S. Roy, D. Park and N. Roy, "Task-Conditioned Variational Autoencoders for Learning Movement Primitives", *Conference on Robot Learning*, Osaka, Japan, October 30-November 1, 2019.

42. Osa, T. and S. Ikemoto, "Goal-Conditioned Variational Autoencoder Trajectory Primitives with Continuous and Discrete Latent Codes", *SN Computer Science*, Vol. 1, No. 5, 2020.

43. Karl, M., M. Soelch, J. Bayer and P. Van der Smagt, "Deep Variational Bayes Filters: Unsupervised Learning of State Space Models from Raw Data", *International Conference on Learning Representations*, Toulon, France, April 24-26, 2017.

44. van den Oord, A., S. Dieleman, H. Zen, K. Simonyan, O. Vinyals, A. Graves, N. Kalchbrenner, A. Senior and K. Kavukcuoglu, "WaveNet: A Generative Model for Raw Audio", *The 9th ISCA Speech Synthesis Workshop*, Sunnyvale, USA, Sep 13-15, 2016.

45. Hester, T., M. Vecerik, O. Pietquin, M. Lanctot, T. Schaul, B. Piot, D. Horgan, J. Quan, A. Sendonaris, I. Osband *et al.*, "Deep Q-Learning from Demonstrations", *Thirty-Second AAAI Conference on Artificial Intelligence*, New Orleans, USA, February 2-7, 2018.

46. Vecerik, M., O. Sushkov, D. Barker, T. Rothörl, T. Hester and J. Scholz, "A Practical Approach to Insertion with Variable Socket Position Using Deep Reinforcement Learning", *International Conference on Robotics and Automation*, pp. 754–760, Montreal, Canada, May 20-24, 2019.

47. Rajeswaran, A., V. Kumar, A. Gupta, J. Schulman, E. Todorov and S. Levine, "Learning Complex Dexterous Manipulation with Deep Reinforcement Learning and Demonstrations", *ArXiv*, Vol. abs/1709.10087, 2018.

48. Li, Y., I. Kash and K. Hofmann, "Learning Good Policies from Suboptimal Demonstrations", *The 14th European Workshop on Reinforcement Learning*, Lille, France, October 1-3, 2018.

49. Peters, J., K. Mülling and Y. Altun, "Relative Entropy Policy Search", *Twenty-Fourth AAAI Conference on Artificial Intelligence*, Vol. 10, pp. 1607–1612, Atlanta, USA, July 11-15, 2010.

50. Colome, A. and C. Torras, "Dual REPS: A Generalization of Relative Entropy Policy Search Exploiting Bad Experiences", *IEEE Transactions on Robotics*, Vol. 33, No. 4, pp. 978–985, 2017.

51. Taylor, M. E. and P. Stone, "Transfer Learning for Reinforcement Learning Domains: A Survey", *Journal of Machine Learning Research*, Vol. 10, No. 1, pp.

1633–1685, 2009.

52. Taylor, M. E., N. K. Jong and P. Stone, "Transferring Instances for Model-Based Reinforcement Learning", W. Daelemans, B. Goethals and K. Morik (Editors), *Machine Learning and Knowledge Discovery in Databases*, pp. 488–505, Springer Berlin Heidelberg, 2008.

53. Ammar, H. B. and M. E. Taylor, "Reinforcement Learning Transfer via Common Subspaces", P. Vrancx, M. Knudson and M. Grześ (Editors), *Adaptive and Learning Agents*, pp. 21–36, Springer Berlin Heidelberg, 2012.

54. Bou-Ammar, H., E. Eaton, P. Ruvolo and M. E. Taylor, "Unsupervised Cross-Domain Transfer in Policy Gradient Reinforcement Learning via Manifold Alignment", B. Bonet and S. Koenig (Editors), *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence, January 25-30, 2015, Austin, Texas, USA*, pp. 2504–2510, AAAI Press, 2015.

55. Gupta, A., C. Devin, Y. Liu, P. Abbeel and S. Levine, "Learning Invariant Feature Spaces to Transfer Skills with Reinforcement Learning", *ArXiv*, Vol. abs/1703.02949, 2017.

56. Salimans, T., J. Ho, X. Chen, S. Sidor and I. Sutskever, "Evolution Strategies as a Scalable Alternative to Reinforcement Learning", *ArXiv*, Vol. abs/1703.03864, 2017.

57. Such, F. P., V. Madhavan, E. Conti, J. Lehman, K. O. Stanley and J. Clune, "Deep Neuroevolution: Genetic Algorithms Are a Competitive Alternative for Training Deep Neural Networks for Reinforcement Learning", *ArXiv*, Vol. abs/1712.06567, 2017.

58. Chang, S., J. Yang, J. Choi and N. Kwak, "Genetic-Gated Networks for Deep Reinforcement Learning", S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi and R. Garnett (Editors), *Advances in Neural Information Processing Sys-*

tems, pp. 1747–1756, Curran Associates, Inc., 2018.

59. Gangwani, T. and J. Peng, "Policy Optimization by Genetic Distillation", *ArXiv*, Vol. abs/1711.01012, 2017.

60. Wang, Z., C. Chen and D. Dong, "Instance Weighted Incremental Evolution Strategies for Reinforcement Learning in Dynamic Environments", *ArXiv*, Vol. abs/2010.04605, 2020.

61. Conti, E., V. Madhavan, F. P. Such, J. Lehman, K. Stanley and J. Clune, "Improving Exploration in Evolution Strategies for Deep Reinforcement Learning via a Population of Novelty-Seeking Agents", S. Bengio, H. M. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi and R. Garnett (Editors), *Advances in Neural Information Processing Systems*, pp. 5027–5038, 2018.

62. Lehman, J. and K. O. Stanley, "Novelty Search and the Problem with Objectives", *Genetic programming theory and practice IX*, pp. 37–56, Springer, 2011.

63. Khadka, S. and K. Tumer, "Evolution-Guided Policy Gradient in Reinforcement Learning", S. Bengio, H. M. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi and R. Garnett (Editors), *Advances in Neural Information Processing Systems*, pp. 1196–1208, 2018.

64. Bodnar, C., B. Day and P. Lió, "Proximal Distilled Evolutionary Reinforcement Learning", *ArXiv*, Vol. abs/1906.09807, 2019.

65. Pourchot, A. and O. Sigaud, "CEM-RL: Combining Evolutionary and Gradient-Based Methods for Policy Search", *ArXiv*, Vol. abs/1810.01222, 2018.

66. Kingma, D. P. and M. Welling, "An Introduction to Variational Autoencoders", *CoRR*, Vol. abs/1906.02691, 2019.

67. Kober, J., "Learning Motor Skills: From Algorithms to Robot Experiments", *In-*

formation Technology, Vol. 56, No. 3, pp. 141–146, 2014.

68. Akbulut, M. T., *ACNMP implementation*, `https://github.com/mtuluhanakbulut/ACNMP`, accessed at June 2021.

69. Ewerton, M., *Source code of Ewerton et al. (2019)*, `https://github.com/marcoewerton`, accessed at June 2021.

70. James, S., M. Freese and A. J. Davison, "PyRep: Bringing V-REP to Deep Robot Learning", *ArXiv*, Vol. abs/1807.01622, 2019.

71. Maaten, L. v. d. and G. Hinton, "Visualizing Data Using t-SNE", *Journal of Machine Learning Research*, Vol. 9, pp. 2579–2605, 2008.

72. Seker, M. Y., *CNMP implementation*, `https://github.com/myunusseker/CNMP`, accessed at June 2021.

73. Sohn, K., H. Lee and X. Yan, "Learning Structured Output Representation using Deep Conditional Generative Models", C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama and R. Garnett (Editors), *Advances in Neural Information Processing Systems 28*, pp. 3483–3491, Curran Associates, Inc., 2015.

74. Higgins, I., L. Matthey, A. Pal, C. Burgess, X. Glorot, M. Botvinick, S. Mohamed and A. Lerchner, "Beta-VAE: Learning Basic Visual Concepts with a Constrained Variational Framework", *International Conference on Learning Representations*, Toulon, France, April 24-26, 2017.

75. Doersch, C., "Tutorial on Variational Autoencoders", *ArXiv*, Vol. abs/1606.05908, 2016.

76. Akbulut, M. T., *RC-NMP implementation*, `https://github.com/mtuluhanakbulut/RC-NMP`, accessed at June 2021.