Data-Efficient Robot Learning using Priors from Simulators

Rituraj Kaushik

UNIVERSITÉ DE LORRAINE

École doctorale IAEM Lorraine

Data-Efficient Robot Learning using Priors from Simulators

THÈSE

présentée et soutenue publiquement le 23 Juillet 2020

pour l'obtention du

Doctorat de l'Université de Lorraine
(mention informatique)

par

Rituraj Kaushik

Composition du jury

Rapporteurs :	Stéphane DONCIEUX	Professeur Sorbonne University, France.
	Dongheui LEE	Professeur Technical University of Munich, Germany.
Examinateurs :	Bruno SCHERRER	Chercheur Université de Lorraine, CNRS, Inria, France.
	Ville KYRKI	Professeur associé Aalto University, Finland.
Directeur :	Jean-Baptiste MOURET	Directeur de recherche Université de Lorraine, CNRS, Inria, France.

Laboratoire Lorrain de Recherche en Informatique et ses Applications — UMR 7503

English Abstract

As soon as the robots step out in the real and uncertain world, they have to adapt to various unanticipated situations by acquiring new skills as quickly as possible. Unfortunately, on robots, current state-of-the-art reinforcement learning (e.g., deep-reinforcement learning) algorithms require large interaction time to train a new skill. In this thesis, we have explored methods to allow a robot to acquire new skills through trial-and-error within a few minutes of physical interaction. Our primary focus is to incorporate prior knowledge from a simulator with real-world experiences of a robot to achieve rapid learning and adaptation.

In our first contribution, we propose a novel model-based policy search algorithm called Multi-DEX that (1) is capable of finding policies in sparse reward scenarios (2) does not impose any constraints on the type of policy or the type of reward function and (3) is as data-efficient as state-of-the-art model-based policy search algorithm in non-sparse reward scenarios.

In our second contribution, we propose a repertoire-based online learning algorithm called APROL which allows a robot to adapt to physical damages (e.g., a damaged leg) or environmental perturbations (e.g., terrain conditions) quickly and solve the given task. In this work, we use several repertoires of policies generated in simulation for a subset of possible situations that the robot might face in real-world. During the online learning, the robot automatically figures out the most suitable repertoire to adapt and control the robot. We show that APROL outperforms several baselines including the current state-of-the-art repertoire-based learning algorithm RTE by solving the tasks in much less interaction times than the baselines.

In our third contribution, we introduce a gradient-based meta-learning algorithm called FAMLE. FAMLE meta-trains the dynamical model of the robot from simulated data so that the model can be adapted to various unseen situations quickly with the real-world observations. By using FAMLE with a model-predictive control framework, we show that our approach outperforms several model-based and model-free learning algorithms, and solves the given tasks in less interaction time than the baselines.

French Abstract

Quand les robots doivent affronter le monde réel, ils doivent s'adapter à diverses situations imprévues en acquérant de nouvelles compétences le plus rapidement possible. Les algorithmes d'apprentissage par renforcement (par exemple, l'apprentissage par renforcement profond) pourraient permettre d'apprendre de telles compétences, mais les algorithmes actuels nécessitent un temps d'interaction trop important. Dans cette thèse, nous avons exploré des méthodes permettant à un robot d'acquérir de nouvelles compétences par essai-erreur en quelques minutes d'interaction physique. Notre objectif principal est de combiner des connaissances acquises sur un simulateur avec les expériences réelles du robot afin d'obtenir un apprentissage et une adaptation rapides.

Dans notre première contribution, nous proposons un nouvel algorithme de recherche de politiques basé sur un modèle, appelé Multi-DEX, qui (1) est capable de trouver des politiques dans des scénarios aux récompenses rares, (2) n'impose aucune contrainte sur le type de politique ou le type de fonction de récompense et (3) est aussi efficace en termes de données que l'algorithme de recherche de politiques de l'état de l'art dans des scénarios de récompenses non rares.

Dans notre deuxième contribution, nous proposons un algorithme d'apprentissage en ligne basé sur un répertoire, appelé APROL, qui permet à un robot de s'adapter rapidement à des dommages physiques (par exemple, une patte endommagée) ou à des perturbations environnementales (par exemple, les conditions du terrain) et de résoudre la tâche donnée. Dans ce travail, nous utilisons plusieurs répertoires de politiques générés en simulation pour un sous-ensemble de situations possibles auxquelles le robot pourrait être confronté dans le monde réel. Au cours de l'apprentissage en ligne, le robot détermine automatiquement le répertoire le plus approprié pour adapter et contrôler le robot. Nous montrons qu'APROL surpasse plusieurs lignes de base, y compris l'algorithme d'apprentissage par répertoire RTE (Reset Free Trial and Error), en résolvant les tâches en un temps d'interaction beaucoup plus court que les algorithmes avec lesquels nous l'avons comparé.

Dans notre troisième contribution, nous présentons un algorithme de méta-apprentissage basé sur les gradients appelé FAMLE. FAMLE permet d'entraîner le modèle dynamique du robot à partir de données simulées afin que le modèle puisse être adapté rapidement à diverses situations invisibles grâce aux observations du monde réel. En utilisant FAMLE pour améliorer un modèle pour la commande prédictive, nous montrons que notre approche surpasse plusieurs algorithmes d'apprentissage basés ou non sur un modèle, et résout les tâches données en moins de temps d'interaction que les algorithmes avec lesquels nous l'avons comparé.

Acknowledgements

The work presented in this thesis would not have been possible without the motivation and support from many people. I take this opportunity to extend my sincere gratitude and appreciation to all those without whom the completion of my Ph.D. would not have been possible.

First and foremost, I would like to thank my supervisor, Jean-Baptiste Mouret, for being my guide on this wonderful journey. Without him, this would not have been possible. I am fortunate to have worked under the guidance of somebody whose passion and knowledge towards the field pushed me everyday to give my best. On days when things did not go very well, he was my constant source of motivation and support. Other than his extensive academic guidance throughout, his patience, respect, and friendship towards me as a Ph.D. candidate was something I will treasure forever.

In this three and a half years journey, some days have been more challenging than the others. In all those days, Ashmita, my lovely wife, has always been my constant source of motivation and happiness. She has been not only curious to know about my work but also patient enough during the long evenings, the nights and the weekends that I spent to finish a paper or writing codes. I would like to thank her from the bottom of my heart for being a wonderful partner during this journey. This thesis would not be the same without her support.

I have been fortunate to find wonderful colleagues within the ResiBots project and the LARSEN team in general. I would like to thank some people with whom I had chance to spend some time during my Ph.D.: Serena Ivaldi, Francis Colas, Pauline Maurice, Oriane Dermy, Adrien Malaisé, Yassine El-Khadiri, Iñaki Fernández Pérez, Adrian Bourgaud, Brice Clement, Lucien Renaud, Vaios Papaspyros, Jamie Waugh, and Valerio Modugno. I would also like to thank Debaleena Misra, Kapil Sawant, Luigi Penco, Vladislav Tempez, Waldez Azevedo Gomes, Jonathan Spitz, Eloïse Dalin, Glenn Maguire, and Pierre Laclau for all the wonderful discussions (both scientific and non-scientific) that we had over the time. In particular, this thesis definitely would not be the same without those long scientific conversations that I had with Konstantinos Chatzilygeroudis and Vassilis Vassiliades almost everyday during the coffee breaks. Also, during these years, I got the opportunity to have wonderful scientific collaborations with Konstantinos Chatzilygeroudis, Vassilis Vassiliades, Roberto Rama, Dorian Goepp, Timothée Anne, and Pierre Desreumaux for which I am grateful towards them.

My friends in Nancy had a crucial role to play in the journey. They were like my second family, and without their support, this journey would have been a difficult one. I would like to thank Harshad Mishra, Annya Kar, Pragya Tripathi, Rahul Mishra, Tithi Ghosh, Divyesh Arora, Himanshu Maheshwari, Neelam Rout, Anant Shirsath, Saylee Chavan, Sunit Sivasankaran, Anupama Venugopal, Tulika Bose, Shantanu Misra, Mainak Mukherjee, Shashank Singh, and Alka Singh for making these years beautiful in Nancy.

My family has been a constant support system. Without them, this would have never been possible. They have been very patient and understanding throughout this journey. I would like to extend my gratitude towards my parents

(Bina Devi and Bipin Sarma), my sister (Rajashree Sarma), and my parents-in-law (Manjula Bhattacharya and Patanjali Bhattacharya).

I would also like to thank the jury members of my Ph.D. defense for accepting to be part of the jury. It is truly an honor to be reviewed by such experienced and respected researchers. Finally, I would like to thank the European Research Council and the University of Lorraine, who gave me the opportunity to pursue this Ph.D. thesis.

Summary in French

L'une des plus grandes aspirations des scientifiques et des philosophes au cours des derniers siècles a été de créer des machines intelligentes semblables à la vie, des robots capables de percevoir le monde et d'agir de manière transparente dans une société humaine. Après l'avènement des ordinateurs numériques programmables vers le milieu du 20ème siècle, le domaine de la robotique a commencé à prospérer en même temps que l'exploration scientifique visant à relier l'intelligence humaine et les machines. Aujourd'hui, directement ou indirectement, des robots ou des systèmes autonomes font partie de notre vie. Aujourd'hui, les robots sont déployés dans de nombreux domaines différents, tels que les industries manufacturières, les applications spatiales, les applications médicales, la défense et les applications domestiques [Hollerbach et al., 2016]. Par exemple, la société Amazon a plus de 200 000 robots mobiles qui travaillent dans son réseau d'entrepôts, aux côtés de centaines de milliers de travailleurs humains. Dans les applications domestiques, le robot le plus populaire est probablement l'aspirateur robotisé qui peut cartographier et nettoyer la maison de manière autonome. De même, dans les applications spatiales, les rovers déployés sur Mars par la NASA peuvent planifier leurs trajectoires de manière autonome en évitant les zones dangereuses de la surface martienne.

En général, un robot peut être défini comme une machine programmable qui peut détecter son environnement et effectuer une séquence d'actions en fonction de ces informations sensorielles. Une grande partie des résultats impressionnants obtenus en robotique ces dernières années ont été obtenus grâce à l'application de la théorie moderne de la commande en conjonction avec la mécanique des corps rigides et les algorithmes de planification [Murray et al., 1994; Lynch and Park, 2017]. Grâce à ces approches, par exemple, les robots humanoïdes peuvent désormais marcher sur un terrain plat [Sellaouti et al., 2006] même en cas de perturbations externes [Padois et al., 2017; Griffin et al., 2018], et les robots de Boston Dynamics (une société américaine de robotique), comme l'Atlas, le Spot et le Big Dog, peuvent démontrer des capacités de locomotion robustes sur divers terrains, comme la neige, l'herbe et les routes rocheuses dans les forêts.

Bien que ces approches permettent à un robot d'effectuer de nombreuses tâches de manière autonome, elles nécessitent encore de nombreuses compétences humaines. Tout d'abord, ces approches sont basées sur des contrôleurs conçus à partir d'une modélisation mathématique précise du robot ainsi que de la connaissance complète de la tâche à accomplir. En conséquence, lorsque la spécification de la tâche change, le système de commande doit être réajusté ou entièrement modifié par un expert humain. En outre, ces contrôleurs sont généralement conçus en tenant compte de certaines hypothèses préalables sur les situations futures auxquelles le robot pourrait être confronté pendant son déploiement, comme les conditions du terrain ou l'altitude de la surface. Par

conséquent, toute situation imprévue, par exemple une nouvelle condition de terrain, une défaillance de l'actionneur ou un défaut des capteurs, peut finalement rendre ces contrôleurs inutilisables tant que les experts humains n'ont pas pris les mesures appropriées. Lors des récents défis de la DARPA, par exemple, de nombreux robots sophistiqués ont dû être sauvés par des humains en raison de diverses défaillances survenues dans les robots pendant le défi [Krotkov et al., 2017].

Une autre approche consiste à laisser le robot apprendre sa commande par lui-même, sans aucune supervision humaine, tout en interagissant dans l'environnement. Cette façon de penser découle du fait que les robots ne sont pas seulement des agents intelligents, mais qu'ils ont aussi des corps physiques comme les animaux qui façonnent leur façon d'agir et de sentir l'environnement. Rodney Brooks soutient dans une série de travaux [Brooks, 1991b,a] que l'intelligence nécessite toujours un corps, et qu'il faut capitaliser sur l'interaction entre le système et l'environnement plutôt que de se concentrer sur des processus de raisonnement sophistiqués. Il appelle cette perspective de voir l'intelligence artificielle comme "Hypothèse physiquement fondée" [Brooks, 1990], tandis que d'autres la désignent comme "Cognition incorporée" [Pfeifer and Bongard, 2006]. Ces dernières années, les algorithmes d'apprentissage automatique [Bishop, 2006; Kober et al., 2013; Deisenroth et al., 2013], la robotique évolutionniste [Nolfi et al., 2000; Doncieux et al., 2015, 2011] et développementale [Lungarella et al., 2003] ont ouvert la voie vers la conception de robots qui peuvent apprendre leur contrôleur par eux-même.

L'apprentissage autonome en robotique est désormais l'un des principaux axes de recherche en robotique. Grâce à leur capacité d'auto-apprentissage, les robots peuvent être capables d'effectuer une variété de tâches sans avoir à concevoir manuellement leurs contrôleurs pour les tâches spécifiques, pour chaque robot [Brooks, 2014]. De plus, ces robots peuvent s'adapter à des situations imprévues, comme être endommagés ou interagir avec un nouvel objet, et peuvent être capables de trouver des stratégies alternatives pour effectuer la tâche en fonction de la situation [Bongard et al., 2006; Cully et al., 2015].

L'apprentissage sur les robots peut se faire de trois manières différentes : (1) apprentissage supervisé (2) apprentissage non supervisé, et (3) apprentissage par renforcement (RL). Dans un cadre d'apprentissage supervisé, on peut fournir plusieurs exemples étiquetés au robot sur certaines tâches et le robot apprend à généraliser ces exemples à diverses tâches inconnues. L'apprentissage supervisé est principalement utilisé pour la perception ; en robotique, l'idée la plus proche est l'apprentissage à partir de la démonstration, qui combine l'apprentissage supervisé et l'optimisation locale. Par exemple, à partir des démonstrations fournies pour la saisie de divers objets, un robot peut apprendre à saisir de nombreux objets différents [Calinon et al., 2007; Billard et al., 2008b]. Dans un cadre d'apprentissage non supervisé, un robot peut apprendre une représentation abstraite de son environnement pour comprimer ses informations sensorielles de haute dimension [Jonschkowski and Brock, 2015; Doncieux et al., 2018]. L'apprentissage non supervisé peut également être observé dans la robotique développmentale [Cangelosi and Schlesinger, 2015], où un robot interagit avec l'environnement d'une manière ouverte pour développer des compétences

progressivement plus complexes [Oudeyer et al., 2007; Goff et al., 2019; Paolo et al., 2020; Santucci et al., 2020]. Avec l'approche d'apprentissage par renforcement [Sutton and Barto, 1998], un robot peut apprendre à effectuer une tâche en rassemblant des expériences et en maximisant sa récompense à long terme pour ses actions [Kober et al., 2013; Deisenroth et al., 2013]. Dans ce manuscrit, nous nous concentrons principalement sur les scénarios d'apprentissage du renforcement en robotique.

Récemment, les algorithmes d'apprentissage par renforcement ont donné de nombreux résultats prometteurs grâce à la récente percée dans la recherche sur l'apprentissage profond [Arulkumaran et al., 2017]. Par exemple, les agents peuvent apprendre à jouer à de nombreuses parties de l'Atari 2600 en observant directement les pixels [Mnih et al., 2015b] et peuvent battre les meilleurs joueurs du monde de Go [Silver et al., 2016] et d'échecs [Silver et al., 2017] en utilisant un minimum d'expertise humaine. Cependant, derrière ces succès impressionnants se cachent des algorithmes de RL gourmands en données qui nécessitent des millions d'essais pour acquérir ces compétences. Par exemple, 4,8 millions de parties ont été nécessaires pour apprendre à jouer au Go à partir de zéro [Silver et al., 2016], 38 jours de jeu (en temps réel) pour les jeux Atari 2600 [Mnih et al., 2015b], et environ 100 heures de simulation (plus pour le temps réel) pour un mannequin de 9 degrés de liberté qui apprend à marcher [Heess et al., 2017].

Malheureusement, contrairement aux agents RL abstraits simulés, les robots ont un corps physique qui est limité par les lois de la physique. Par conséquent, ils sont nettement plus lents que les agents simulés et sont susceptibles de s'user et de se casser avec le temps. Ces limites physiques rendent difficile l'application des algorithmes de RL sur les robots, car ces algorithmes nécessiteraient un grand nombre d'essais (c'est-à-dire plusieurs heures à plusieurs jours de tests) pour apprendre une nouvelle compétence [Kober et al., 2013; Chatzilygeroudis et al., 2020]. À titre d'exemple, Levine et al. [2018] a récemment proposé un algorithme pour l'apprentissage de la coordination œil-main pour la saisie robotique en utilisant l'apprentissage par renforcement combiné à l'apprentissage profond. L'algorithme a nécessité environ 800 000 saisies, qui ont été collectées en l'espace de deux mois à l'aide de 6 à 14 manipulateurs robotiques fonctionnant en parallèle. Malgré les résultats prometteurs, cela n'a été possible que parce que les manipulateurs sont généralement abordables et faciles à automatiser. Cependant, il est difficile d'imaginer faire la même chose avec une "ferme" de robots humanoïdes.

Pour des applications pratiques, un robot doit être capable d'apprendre une compétence en quelques minutes, et non en quelques jours. L'apprentissage devient encore plus difficile lorsque le robot doit apprendre une nouvelle compétence en ligne pendant sa mission. Par exemple, un robot déployé sur un site post-catastrophe pour une mission de sauvetage [Nagatani et al., 2013] doit être capable d'apprendre à gérer des situations imprévues presque immédiatement. Ce type d'adaptation en ligne devient crucial pour les robots dès qu'ils sortent des environnements contrôlés des industries et des usines de fabrication pour entrer dans notre vie quotidienne [Bellingham and Rajan, 2007; Trevelyan et al., 2016; Antonelli et al., 2008]. Dans le monde réel, un robot peut avoir à faire

face à de nombreux défis, comme se retrouver sur un nouveau terrain, endommager ses articulations, etc. Dans de telles situations, au lieu d'interrompre sa mission, il doit apprendre à s'adapter le plus rapidement possible et à accomplir sa mission [Cully et al., 2015].

Malgré les récents succès des algorithmes RL, le long temps d'interaction est devenu une préoccupation majeure lors de l'utilisation de ces algorithmes sur les robots. En RL, le temps d'interaction est proportionnel à la quantité de données requises par l'algorithme pour entraîner le robot. Ainsi, en d'autres termes, un algorithme doit nécessiter un minimum de données pour des applications pratiques dans l'apprentissage du robot ; c'est-à-dire que les algorithmes doivent être *data-efficient*. L'efficacité en données est un terme relatif qui peut être quantifié comme le rapport entre la performance et le nombre d'échantillons de données requis. En ce sens, un algorithme ne peut être efficace en termes de données que par rapport à un autre algorithme, et cela ne signifie pas nécessairement que l'algorithme convient pour des applications pratiques d'apprentissage de robots. Récemment, Mouret [2016] a proposé un terme absolu appelé *micro-data learning*, qui signifie essentiellement que l'apprentissage doit se faire si rapidement (c'est-à-dire en une minute, voire en quelques secondes, d'interaction) que le robot peut apprendre en ligne et poursuivre sa mission dans des scénarios du monde réel. C'est donc dans un esprit similaire que, dans ce manuscrit, nous nous intéressons à l'apprentissage des micro-data pour permettre aux robots d'apprendre en quelques minutes ou secondes d'interaction.

Les algorithmes d'apprentissage par renforcement les plus utilisés pour l'apprentissage des robots sont les algorithmes *policy search* qui apprennent les paramètres d'un contrôleur, appelé politique, qui met en correspondance les entrées des capteurs aux positions et couples des articulations [Deisenroth et al., 2013]. Ces algorithmes permettent d'utiliser des politiques bien adaptées à la commande de robots, comme les primitives de mouvement dynamiques [Ijspeert et al., 2003] ou les réseaux de neurones [Levine and Abbeel, 2014]. En outre, l'approche de recherche de politiques est bien adaptée à la robotique car elle peut traiter des espaces d'état et d'action continus et à haute dimension, l'un des principaux défis de l'apprentissage en robotique [Deisenroth et al., 2013].

Pour un apprentissage efficace en robotique, l'approche *model-based reinforcement learning* (MBRL) est une direction prometteuse [Polydoros and Nalpantidis, 2017; Atkeson and Santamaria, 1997; Chatzilygeroudis et al., 2020]. Dans le MBRL, un robot apprend de manière itérative un modèle dynamique de l'environnement en utilisant les données de ses expériences passées. Le robot utilise ce modèle comme substitut ou comme simulateur interne pour prédire comment une action pourrait modifier l'environnement. Ce modèle permet au robot d'évaluer et d'optimiser les actions sans interagir physiquement avec le monde réel. Cette approche permet donc de réduire le nombre d'essais ou d'observations nécessaires à l'apprentissage d'une nouvelle compétence. Par exemple, en utilisant l'approche PILCO [Deisenroth and Rasmussen, 2011], un chariot peut apprendre à équilibrer un pendule en seulement 17,5 secondes d'interaction. Un inconvénient de la MBRL est que pour des robots relativement complexes, l'apprentissage d'un modèle suffisamment bon pour l'optimisation

des politiques en utilisant uniquement les données observées précédemment nécessiterait encore un grand nombre d'échantillons. En fait, ce nombre augmente de manière exponentielle avec la dimension d'entrée (c'est-à-dire la dimension spatiale état-action) du modèle. Par conséquent, le robot aurait besoin d'une interaction plus longue avec le monde pour apprendre.

Un long temps d'interaction avec le monde réel est inévitable pour tout algorithme d'apprentissage de robot qui n'utilise pas les connaissances préalables sur le robot, son environnement ou la tâche. En d'autres termes, l'absence de connaissances préalables, ou simplement d'a priori, constituent un obstacle majeur à l'apprentissage efficace des données en robotique pour tout algorithme d'apprentissage par essais et erreurs. Heureusement, dans tout problème d'apprentissage en robotique, nous disposons au moins de quelques informations préalables qui peuvent être utilisées efficacement soit pour amorcer le processus d'apprentissage, soit pour réduire la complexité du problème d'apprentissage. En insérant les connaissances préalables appropriées au bon endroit dans l'algorithme, l'apprentissage des robots peut être rendu plus rapide d'un ordre de grandeur par rapport à l'apprentissage sans connaissances préalables [Chatzilygeroudis et al., 2020].

Dans un algorithme d'apprentissage de robot, les connaissances préalables peuvent être insérées dans une ou plusieurs étapes pour accélérer le processus d'apprentissage. Par exemple, dans le cadre de MBPS, on peut utiliser les antécédents du modèle dynamique du robot pour apprendre un modèle utile avec seulement quelques observations du monde réel [Cutler and How, 2015; Saveriano et al., 2017; Chatzilygeroudis and Mouret, 2018]. En utilisant un simulateur paramétré comme précédemment pour le modèle, Chatzilygeroudis and Mouret [2018] permet à un robot hexapode endommagé de marcher efficacement après seulement 16 à 30 secondes de temps d'interaction. Pour un manipulateur robotique, la politique peut également être fournie avec des antécédents sous forme de démonstrations humaines, que l'algorithme peut utiliser comme point de départ ou de référence pour optimiser les paramètres de la politique pour une tâche similaire mais invisible [Calinon et al., 2007]. De plus, le préalable peut également être utilisé pour restreindre l'expressivité de la politique afin de générer des comportements plus utiles et spécifiques aux tâches. Par exemple, les politiques largement utilisées avec des manipulateurs, les primitives de mouvement dynamiques (DMP), sont conçues pour produire des trajectoires fluides et ne comportent que quelques paramètres par rapport aux politiques plus expressives telles qu'un réseau de neurones. En utilisant de telles DMP (31 paramètres), ainsi que des démonstrations initiales et des réglages fins avec RL, un bras robotique pourrait apprendre à jouer au bilboquet en environ 75 essais (environ 4 minutes) [Kober and Peters, 2009]. De même, en apprenant préalablement sur diverses tâches, les paramètres initiaux de la politique ou du modèle peuvent être optimisés par le méta-apprentissage [Schaul and Schmidhuber, 2010; Finn et al., 2017] de telle sorte que l'apprentissage sur une nouvelle tâche ne nécessite que quelques observations du monde réel.

Néanmoins, l'efficacité des connaissances préalables dépend de la proximité entre ces a priori et la réalité [Pautrat et al., 2018; Chatzilygeroudis and Mouret, 2018]. Dans le pire des cas, un a priori erroné ou trompeur pourrait nuire au

processus d'apprentissage et réduire l'efficacité des données. Par exemple, des connaissances préalables sur la façon de marcher sur une glace glissante peuvent ne pas être un bon a priori pour apprendre à marcher sur un terrain rocheux. En revanche, des connaissances préalables sur la manière de marcher sur un sol irrégulier pourraient être un a priori efficace à cette fin. Il est donc crucial d'utiliser les a priori afin d'accélérer le processus d'apprentissage en robotique.

Dans ce manuscrit, nous étudions comment les modèles et les connaissances préalables peuvent être utilisés efficacement pour permettre aux robots d'apprendre une nouvelle compétence en quelques minutes, voire quelques secondes, d'interaction dans le monde réel. Plus précisément, nous proposons des algorithmes pour un apprentissage efficace en données en robotique en utilisant une approche d'apprentissage par essais et erreurs, où nous utilisons un ou plusieurs a priori de sorte que le modèle puisse être appris avec seulement quelques échantillons du monde réel. Notre objectif est d'atteindre une vitesse d'apprentissage adaptée à l'adaptation en ligne, c'est-à-dire que l'apprentissage doit se faire en quelques minutes. En d'autres termes, dans ce manuscrit, nous nous concentrons sur *micro-data learning* [Mouret, 2016] en robotique.

Notre principale motivation est l'application d'algorithmes d'apprentissage par essais et erreurs pour la récupération des dommages en ligne en robotique [Cully et al., 2015; Bongard et al., 2006]. Nous pensons que c'est un problème intéressant pour la communauté d'apprentissage des robots, car il n'y a actuellement aucun consensus sur la meilleure façon analytique de s'adapter à des dommages subis par les robots. En tant que source de connaissances préalables, nous nous concentrons sur l'utilisation des simulateurs basse fidélité. À notre avis, les simulateurs constituent le moyen le plus générique, le plus souple et le plus rentable d'intégrer les connaissances préalables dans les algorithmes d'apprentissage des robots.

Dans le **Chapitre 2** de ce manuscrit, nous donnons un aperçu général de la formulation de problèmes d'apprentissage par renforcement dans le contexte de l'apprentissage des robots. Ensuite, nous décrivons brièvement les différents algorithmes d'apprentissage par renforcement sans modèle. Ensuite, nous orientons notre discussion vers les algorithmes d'apprentissage par renforcement basés sur un modèle, plus précisément les approches de recherche de politiques. Enfin, nous expliquons comment les différents a priori peuvent être utilisés en conjonction avec un cadre d'apprentissage par renforcement en robotique pour obtenir un apprentissage en ligne efficace en termes de données sur des robots physiques.

Dans le **Chapitre 3**, nous considérons l'approche de recherche de politiques basée sur des modèles dans des scénarios aux récompenses rares. Comme nous l'avons déjà mentionné ci-dessus, les algorithmes les plus efficaces en matière de données pour l'apprentissage par renforcement en robotique sont les algorithmes de recherche de politiques basées sur un modèle, qui alternent entre l'apprentissage d'un modèle de la dynamique du robot et l'optimisation d'une politique pour maximiser la récompense attendue compte tenu du modèle et de ses incertitudes. Toutefois, ces algorithmes supposent implicitement que

la plupart des états peuvent être associés à une récompense positive ou négative. Ainsi, ces algorithmes sont intrinsèquement gourmands et surtout exploiteurs. L'hypothèse ci-dessus est valable pour de nombreux scénarios de test dans l'apprentissage des robots. En revanche, dans le monde réel, les récompenses peuvent être beaucoup plus rares, c'est-à-dire que de nombreuses paires état-action ne peuvent pas être associées intuitivement à des récompenses positives ou négatives. Les algorithmes actuels de recherche de politiques basés sur des modèles manquent d'une stratégie d'exploration efficace pour traiter les scénarios de récompenses rares ou trompeuses. Si ces algorithmes ne connaissent aucun état avec une récompense positive lors de l'exploration aléatoire initiale, il est très peu probable qu'ils résolvent le problème. Dans ce chapitre, nous proposons un nouvel algorithme de recherche de politiques basé sur un modèle, Multi-DEX (Multi-objective Data-Efficient eXploration), qui exploite un modèle dynamique probabiliste appris de la dynamique du robot pour explorer efficacement l'espace d'état et résoudre des tâches avec des récompenses rares en quelques épisodes. Nous comparons Multi-DEX avec des algorithmes de RL récents pour des scénarios de récompenses rares VIME [Houthooft et al., 2016] et GEP-PG [Colas et al., 2018], un algorithme de pointe de recherche de politiques sans modèle TRPO [Schulman et al., 2015], un optimiseur de boîte noire de pointe CMA-ES [Hansen, 2006], et un algorithme de recherche de politiques basé sur un modèle Black-DROPS [Chatzilygeroudis et al., 2017]. Nous montrons que le Multi-DEX résout les tâches avec ordre de grandeur de temps d'interaction en moins que les autres approches.

Dans **Chapitre 4**, nous considérons un problème où un robot relativement complexe doit s'adapter en ligne à des situations imprévues, comme être endommagé ou interagir avec de nouveaux objets, etc. Bien que l'approche d'apprentissage basée sur un modèle comme Multi-DEX soit une direction prometteuse, elle nécessiterait encore un grand nombre d'échantillons du monde réel afin d'apprendre un modèle dynamique efficace en raison de la dimension élevée de l'espace état-action dans les robots complexes. Une alternative intéressante consiste plutôt à apprendre un modèle dans l'"espace des tâches" (qui est généralement de dimension inférieure) du robot qui prédit comment les résultats des politiques élémentaires stockées dans un répertoire changent dans le monde réel par rapport au robot simulé. Cette approche est appelée apprentissage par répertoire, où le répertoire des politiques élémentaires appris dans la simulation sert de préalable à l'apprentissage du modèle en utilisant des observations du monde réel [Chatzilygeroudis et al., 2018; Cully et al., 2015]. Avec un tel modèle, un algorithme de planification peut ensuite sélectionner une séquence de ces politiques élémentaires dans le répertoire. Dans ce chapitre, nous assouplissons l'hypothèse des œuvres précédentes selon laquelle un seul antécédent basé sur le répertoire suffit pour l'adaptation dans le monde réel. Au lieu de cela, nous générons des répertoires pour de nombreuses situations différentes (par exemple, avec une jambe manquante, sur différents étages, etc.) et nous laissons notre algorithme sélectionner l'a priori le plus utile pour l'exécution de la tâche. Nous proposons un algorithme, APROL (Adaptive Prior selection for Repertoire-based Online Learning), pour planifier la prochaine action en intégrant ces antécédents lorsque le robot ne dispose d'aucune information sur

la situation actuelle (par exemple, une jambe cassée ou un terrain accidenté, etc.) Nous comparons l'APROL au "Reset-free Trial and Error" (RTE) [Chatzilygeroudis et al., 2018] ainsi qu'à diverses approches basées sur un répertoire unique, et nous montrons que l'APROL résout les tâches en moins de temps d'interaction que ses concurrents.

L'apprentissage du modèle de transition dans l'espace des tâches réduit la dimension du problème d'apprentissage du modèle, mais, pour de nombreux robots, il n'est pas possible d'apprendre un modèle dans l'espace des tâches. Par exemple, pour un robot mobile, l'espace des tâches peut être sa position 2D au sol et l'orientation le long de l'axe vertical. Grâce à ces informations, il est possible d'apprendre un modèle de transition du robot dans l'espace des tâches, c'est-à-dire la position et l'orientation dans l'étape suivante en fonction d'une action (par exemple, la vitesse des roues). Au contraire, pour un manipulateur robotique, ne connaissant que la position et l'orientation de l'effecteur final, il n'est pas possible de prédire avec précision l'état de l'effecteur final suivant étant donné les vitesses des articulations, car la transition dépend de l'état initial de l'articulation du robot.

Dans le **Chapitre 5**, nous étendons l'idée d'utiliser plusieurs a priori différents (dans le chapitre 4) et proposons une approche plus générale dans le cadre de l'apprentissage du renforcement basé sur le modèle. Dans ce chapitre également, nous envisageons l'adaptation en ligne sur des robots relativement complexes, où l'apprentissage du modèle dynamique à l'état complet à partir de zéro nécessiterait un grand nombre d'échantillons du monde réel. Une façon d'accélérer le processus d'apprentissage du modèle est le méta-apprentissage des paramètres initiaux du modèle, comme le méta-apprentissage agnostique du modèle (MAML) [Finn et al., 2017]. En utilisant les données de transition d'état de la simulation, le méta-apprentissage peut trouver un ensemble initial de paramètres pour le modèle dynamique de telle sorte que le modèle peut être formé pour correspondre à la dynamique réelle du système avec seulement quelques points de données. Dans ce chapitre, nous montrons d'abord que lorsque les situations de méta-apprentissage (les situations précédentes) ont des dynamiques diverses, utiliser un unique ensemble de paramètres de méta-apprentissage comme point de départ nécessite encore un grand nombre d'observations du système réel pour apprendre un modèle dynamique utile. Ensuite, nous proposons un algorithme appelé FAMLE (Fast Adaption through meta-learning Embeddings of simulated priors) qui permet d'atténuer cette limitation en méta-apprenant plusieurs points de départ initiaux (c'est-à-dire des paramètres initiaux) pour le modèle. FAMLE permet ensuite au robot de sélectionner le point de départ le plus approprié pour adapter le modèle à la situation réelle en quelques étapes de gradient seulement. Nous comparons FAMLE à MBRL, MBRL avec un modèle méta-formé avec MAML et l'algorithme de recherche de politique sans modèle PPO [Schulman et al., 2017] pour diverses tâches robotiques et nous montrons que FAMLE permet aux robots de s'adapter à de nouveaux dommages en un nombre de pas de temps nettement inférieur à celui des lignes de base.

Dans le **chapitre 6**, nous discutons en détail les avantages et les limites

des algorithmes que nous proposons. Nous explorons la portée future des algorithmes que nous proposons du point de vue de l'application pratique et suggérons des améliorations possibles à cet égard. Enfin, dans le **Chapitre 7**, nous résumons brièvement nos travaux et concluons le manuscrit.

Contents

List of Figures

Chapter 1

Introduction

One of the highest aspirations of the scientists and philosophers in recent centuries was to create life-like intelligent machines – robots that can perceive the world and act seamlessly in a human society. After the advent of programmable digital computers around the mid of 20^{th} century, the field of robotics started flourishing along with the scientific exploration to connect human intelligence and machines. In the present time, directly or indirectly, robots or autonomous systems have become a part of our life. Today, robots are being deployed in many different fields, such as manufacturing industries, space applications, medical applications, defense, and household applications [Hollerbach et al., 2016]. For example, the company Amazon has more than 200,000 mobile robots working inside its warehouse network, alongside hundreds of thousands of human workers. In household applications, the most popular robot is probably the robotic vacuum cleaner that can map and clean the house autonomously. Similarly, in space applications, the rovers deployed on Mars by NASA can plan their trajectories autonomously by avoiding hazardous regions on the Martian surface.

Generally, a robot can be defined as a programmable machine that can sense its environment and perform a sequence of actions based on this sensory information. Much of the impressive results in robotics in recent years have been achieved through the application of modern control theory in conjunction with rigid body mechanics and planning algorithms [Murray et al., 1994; Lynch and Park, 2017]. With such approaches, for instance, now humanoid robots can walk on flat terrain [Sellaouti et al., 2006] even under external perturbations [Padois et al., 2017; Griffin et al., 2018], the robots from Boston Dynamics (a US robotics company), like the Atlas, the Spot and the Big Dog, can demonstrate life-like robust locomotion abilities in various terrains, such as snow, grass and rocky roads in forests.

Although these approaches allow a robot to perform many tasks autonomously, they still require human expertise in many ways. First of all, these approaches are based on controllers designed using precise mathematical modelling of the robot as well as the complete knowledge of the task at hand. As a consequence, when the specification of the task changes, the control system has to be retuned or changed entirely by a human expert. Furthermore, these controllers are typically designed while keeping in mind some prior assumptions about the future situations that the robot might face during its deployment, such as the terrain conditions or surface elevations. Therefore, any unanticipated situation, for instance, a new terrain condition, actuator failure or fault in sensors, can

ultimately make these controllers useless until proper measures are not taken by the human experts. In the recent DARPA challenges, for instance, many sophisticated robots had to be rescued by humans due to various failures occurred in the robots during the challenge [Krotkov et al., 2017].

An alternative approach is to let the robot learn its controller by itself without any human supervision, while interacting in the environment. This way of thinking stems from the fact that robots are not only intelligent agents, but they also have physical bodies like animals that shape the way they act and sense the environment. Rodney Brooks argues in a series of work [Brooks, 1991b,a] that intelligence always requires a body, and one should capitalize on the system-environment interaction rather than focusing on sophisticated reasoning processes. He calls this perspective of seeing artificial intelligence as "Physically Grounded Hypothesis" [Brooks, 1990], while others refer to it as "Embodied Cognition" [Pfeifer and Bongard, 2006]. In the recent years, machine learning algorithms [Bishop, 2006; Kober et al., 2013; Deisenroth et al., 2013], evolutionary robotics [Nolfi et al., 2000; Doncieux et al., 2015, 2011] and developmental robotics [Lungarella et al., 2003] have paved the path towards this new paradigm of designing robots that can learn their controllers by themselves.

Autonomous learning in robotics (i.e., robot learning) is now one of the core research directions in robotics. With a self-learning ability, robots can be able to perform a variety of tasks without any need to manually designing their controllers for the specific tasks, for each robot [Brooks, 2014]. What is more, such robots can be adaptive to unanticipated situations, such as being damaged or interacting with a novel object and can be able to figure out alternative strategies to perform the task depending upon the situation [Bongard et al., 2006].

Learning on robots can happen in three different ways: (1) supervised learning (2) unsupervised learning, and (3) reinforcement learning (RL). In a supervised learning framework, one can provide several labelled examples to the robot on some tasks and robot learns to generalize these examples to various unseen tasks. Supervised learning is mostly used for perception; in robotics, the closest idea is learning from demonstration, which blends supervised learning with local optimization. For instance, from the demonstrations provided for grasping various objects, a robot can learn to grasp many different objects [Calinon et al., 2007; Billard et al., 2008b]. In an unsupervised learning framework, a robot might learn an abstract representation of its environment to compress its high dimensional sensory information [Jonschkowski and Brock, 2015; Doncieux et al., 2018]. Unsupervised learning can also be seen in the developmental robotics [Cangelosi and Schlesinger, 2015], where a robot interacts with the environment in an open-ended way to develop progressively more complex skills [Oudeyer et al., 2007; Goff et al., 2019; Paolo et al., 2020; Santucci et al., 2020]. With the reinforcement learning approach [Sutton and Barto, 1998], a robot can learn to perform a task by gathering experiences and maximizing its long-term *reward* for its actions [Kober et al., 2013; Deisenroth et al., 2013]. In this manuscript, we are primarily focused on reinforcement learning scenarios in robotics.

Recently, reinforcement learning algorithms have shown many promising results thanks to the recent breakthrough in deep learning research [Arulkumaran et al., 2017]. For instance, deep RL agents can learn to play many of the Atari 2600 games by directly observing the pixels [Mnih et al., 2015b] and could beat the world's best players in Go [Silver et al., 2016] and chess [Silver et al., 2017] using minimal human expertise. However, behind these impressive success stories are data-hungry RL algorithms that require millions of trials to learn these skills. For example, 4.8 million game-plays were required to learn to play Go from scratch [Silver et al., 2016], 38 days of game-play (real-time) for Atari 2600 games [Mnih et al., 2015b], and about 100 hours of simulation time (more for real-time) for a 9-DOF mannequin that learns to walk [Heess et al., 2017].

Unfortunately, unlike simulated abstract RL agents, robots have a physical body that is bounded by the laws of physics. As a result, they are significantly slower compared to simulated agents, and they are susceptible to wear and tear over time. These physical limitations make the application of state-of-the-art RL algorithms difficult on robots since these algorithms would require a large number of trials (i.e., several hours to days of training) to train a new skill [Kober et al., 2013; Chatzilygeroudis et al., 2020]. As an example, recently Levine et al. [2018] proposed an algorithm towards learning hand-eye coordination for robotic grasping using deep learning. The algorithm required approximately 800,000 grasps, which were collected within a period of 2 months using 6-14 robotic manipulators running in parallel. Despite the promising results, it was possible only because the manipulators are generally affordable and easy to automate. However, it is difficult to imagine performing the same with a farm of humanoids.

For practical applications, a robot must be able to learn a skill within a few minutes, not in a few days. Learning becomes even more challenging when the robot has to learn a new skill online during its mission. For instance, a robot deployed in a disaster site for rescue mission [Nagatani et al., 2013] must be able to learn to deal with unanticipated situations almost immediately. This type of online adaptation becomes crucial for robots as soon as they step out of the controlled environments of the industries and manufacturing plants to our day-to-day life [Bellingham and Rajan, 2007; Trevelyan et al., 2016; Antonelli et al., 2008]. In the real world, a robot might have to deal with many challenges, such as finding itself on a new terrain condition, damaging its joints, and so on. In such situations, instead of aborting its mission, it has to learn to adapt as quickly as possible and accomplish its mission [Cully et al., 2015].

Despite the recent successes of the RL algorithms, the long interaction time has become a primary concern while using these algorithms on robots. In RL, the interaction time is proportional to the amount of data required by the algorithm to train the robot. Thus, in other words, an algorithm should require a minimum amount of data for practical applications in robot learning; i.e., the algorithms must be *data-efficient*. Data-efficiency is a relative term which can be quantified as the ratio of the performance to the number of data-samples required. In that sense, an algorithm can only be data-efficient compared to another algorithm, and it does not necessarily mean that the algorithm is suitable for practical robot learning applications. Recently, Mouret [2016] coined

an absolute term called *micro-data learning*, which essentially means that the learning has to happen so quickly (i.e., within a minute, if not seconds, of interaction) that the robot can learn online and continue its mission in real-world scenarios. Therefore, with a similar spirit, in this manuscript, we are concerned about micro-data learning to allow the robots to learn within minutes or seconds of interaction.

The most-successful reinforcement learning algorithms towards robot learning are the *policy search* algorithms that learn the parameters of a controller, called the policy, that maps sensor inputs to joint positions/torque [Deisenroth et al., 2013]. These algorithms make it possible to use policies that are well-suited for robot control, such as dynamic movement primitives [Ijspeert et al., 2003] or general-purpose neural networks [Levine and Abbeel, 2014]. In addition, the policy-search approach is well suited for robotics as it can deal with high-dimensional and continuous state and action spaces, one of the main challenges in robot learning [Deisenroth et al., 2013].

Towards data-efficient learning in robotics, *model-based reinforcement learning* (MBRL) is a promising direction [Polydoros and Nalpantidis, 2017; Atkeson and Santamaria, 1997; Chatzilygeroudis et al., 2020]. In MBRL, a robot iteratively learns a dynamical model of the environment by using the data from its past experiences. The robot uses this model as a surrogate or as an internal simulator to predict how an action might change the environment. This model allows the robot to evaluate and optimize the actions without physically interacting with the real world. As a result, this approach reduces the number of trials or observations required to learn a new skill. For instance, using the model-based policy search (MBPS) approach PILCO [Deisenroth and Rasmussen, 2011], a cart could learn to balance a pole in just 17.5 seconds of interaction. One downside of MBRL is that for relatively complex robots, learning a model that is good enough for policy optimization using only the previously observed data would still require large number of samples. In fact, this number grows exponentially with the input dimension (i.e., state-action space dimension) of the model. Consequently, the robot would require longer interaction with the world for learning.

Long real-world interaction time is inevitable for any robot learning algorithm that does not utilize prior knowledge about the robot, its environment or the task. Put it differently, lack of prior knowledge, or simply *priors*, sets a central bottleneck towards data-efficient learning in robotics for any trial-and-error learning algorithm [Thrun and Mitchell, 1995]. Fortunately, in any robot learning problem, we have at least some prior information that can be utilized effectively either to bootstrap the learning process or to reduce the complexity learning problem. By inserting the suitable prior knowledge at the right place in the algorithm, robot learning can be made an order of magnitude faster compared to learning without any prior knowledge [Chatzilygeroudis et al., 2020].

In a robot learning algorithm, the priors can be inserted into one or more stages to speed up the learning process. For instance, in the MBPS framework, one can use priors on the dynamical model of the robot to learn a useful model with only a few real-world observations [Cutler and How, 2015; Saveriano et al., 2017; Chatzilygeroudis and Mouret, 2018]. Using a parameterized simulator

as prior for the model, Chatzilygeroudis and Mouret [2018] allows a damaged hexapod robot to walk effectively after only 16 to 30 seconds of interaction time. Alternatively, for a robotic manipulator, the policy can also be provided with priors in the form of primitive motion patterns using human demonstrations, which the algorithm can use as starting point or reference to optimize the policy parameters for a similar but unseen task [Calinon et al., 2007]. What is more, the prior can also be used to restrict the expressiveness of the policy in order to generate more useful behaviours specific to the tasks. For instance, widely used policies with manipulators, the dynamical movement primitives (DMPs), are designed to produce smooth trajectories and have only a few parameters compared to the more expressive policies such as a neural network. Using such DMPs (31 parameters), together with initial demonstrations and fine-tuning with RL, a robotic arm could learn to swing a small in a cup in around 75 trials (approx 4 minutes) [Kober and Peters, 2009]. Similarly, by learning beforehand on a variety of tasks, the initial policy parameters or the initial model parameters can be optimized through meta-learning [Schaul and Schmidhuber, 2010; Finn et al., 2017] such that learning on a new task requires only a few observations from the real world.

Nevertheless, the effectiveness of the prior knowledge depends on the closeness between the prior and reality [Pautrat et al., 2018; Chatzilygeroudis and Mouret, 2018]. In the worst case, a wrong or misleading prior might adversely affect the learning process and reducing the data-efficiency. For instance, prior knowledge on how to walk on slippery ice may not be a right prior for learning to walk on a rocky terrain. Instead, prior knowledge on how to walk on an uneven soil might be an effective prior for this purpose. Thus, it is crucial to use the right prior in order to accelerate the learning process in robotics.

In this manuscript, we investigate how models and prior knowledge can be used effectively to allow robots to learn a new skill in a few minutes, if not seconds, of interaction in the real world. More specifically, we provide algorithms for data-efficient learning in robotics using trial-and-error learning approach, where we use one or several priors so that the model can be learned with only a few samples from the real world. Our goal is to achieve learning speed that is suitable for *online adaptation*; i.e., the learning has to happen in a few minutes. In other words, in this manuscript, we focus on *micro-data learning* [Mouret, 2016] capability in robotics.

Our primary motivation is the application of trial-and-error learning algorithms for online damage recovery in robotics [Cully et al., 2015; Bongard et al., 2006]. We believe that this is an interesting problem for the robot learning community, as there is presently no consensus about the best analytic way to recover from damages in robots. As the source of prior knowledge, we focus on using the low fidelity simulators. In our opinion, simulators are the most generic, flexible and cost-effective way to integrate prior knowledge into robot learning algorithms.

In **Chapter 2** of this manuscript, we start with a general overview of reinforcement learning problem formulation in the context of robot learning. Then we briefly describe the various model-free reinforcement learning algorithms. After that, we move our discussion towards model-based reinforcement learning

algorithms, more specifically, the policy search approaches. Finally, we elaborate on how various priors can be used in conjunction with a reinforcement learning framework in robotics to achieve data-efficient online learning on real physical robots.

In **Chapter 3**, we consider model-based policy search approach in sparse reward scenarios. As already discussed above, the most data-efficient algorithms for reinforcement learning in robotics are model-based policy search algorithms, which alternate between learning a dynamical model of the robot and optimizing a policy to maximize the expected return given the model and its uncertainties. However, these algorithms implicitly assume that most states can be associated with a positive or negative reward. Thus, these algorithms are inherently greedy and mostly exploiting. The above assumption is valid for many test scenarios in robot learning. By contrast, in the real world, the rewards can be much more sparse, i.e., many state-action pairs cannot be associated intuitively with positive or negative rewards. The current model-based policy search algorithms lack an effective exploration strategy to deal with sparse or misleading reward scenarios. If these algorithms do not experience any state with a positive reward during the initial random exploration, it is very unlikely to solve the problem. In this chapter, we propose a novel model-based policy search algorithm, Multi-DEX (Multi-objective Data-Efficient eXploration), that leverages a learned probabilistic dynamical model of the dynamics of the robot to efficiently explore the state-space and solve tasks with sparse rewards in a few episodes. We compare Multi-DEX with recent RL algorithms for sparse reward scenarios VIME [Houthooft et al., 2016] and GEP-PG [Colas et al., 2018], a state-of-the-art model-free policy search algorithm TRPO [Schulman et al., 2015], a state-of-the-art black-box optimizer CMA-ES [Hansen, 2006], and a model-based policy search algorithm Black-DROPS [Chatzilygeroudis et al., 2017]. We show that Multi-DEX solves the tasks in order of magnitude less interaction time than the baseline.

In **Chapter 4**, we consider a problem where a relatively complex robot has to adapt online to unanticipated situations, such as being damaged or interacting with novel objects, etc. Although, the model-based learning approach as Multi-DEX is a promising direction, it would still require a large number of samples from the real world in order to learn an effective dynamical-model due to the high dimensionality of state-action space in complex robots. Instead, an interesting alternative is to learn a model in the "task-space" (which is usually lower-dimensional) of the robot that predicts how the outcomes for elementary policies stored in a repertoire change in the real world compared to the simulated robot. This approach is called repertoire-based learning, where the repertoire of elementary policies learned in simulation serve as prior for learning the model using real-world observations [Chatzilygeroudis et al., 2018; Cully et al., 2015]. With such a model, a planning algorithm can then select a sequence of these elementary policies from the repertoire. In this chapter, we relax the assumption of previous works that a single repertoire-based prior is enough for adaptation in the real world. Instead, we generate repertoires for many different situations (e.g., with a missing leg, on different floors, etc.) and let our algorithm select the most useful prior for performing the task. We

propose an algorithm, APROL (Adaptive Prior selection for Repertoire-based Online Learning), to plan the next action by incorporating these priors when the robot has no information about the current situation (e.g., where a broken leg or a rough terrain, etc.). We compare APROL to "Reset-free Trial and Error" (RTE) [Chatzilygeroudis et al., 2018] as well as various single repertoire-based baselines, and show that APROL solves the tasks in less interaction time than the baselines.

Although learning the transition model in the task-space reduces the dimensionality of the model learning problem, for many robots, it is not possible to learn a model in the task-space. For instance, for a mobile robot, the task-space can be its 2D position on the floor and the orientation along the vertical axis. With this information, it is possible to learn a transition model of the robot in the task-space, i.e., the position and orientation in the next step given an action (e.g., wheel velocities). On the contrary, for a robotic manipulator, knowing only the position and orientation of the end effector, it is not possible to accurately predict the next end effector state given the joint velocities as the transition depends upon the initial joint state of the robot.

In **Chapter 5**, we extend the idea of using multiple priors from the simulator (in chapter 4) and propose a more general approach in the model-based reinforcement learning framework. In this chapter too, we consider online adaptation on relatively complex robots, where learning the full state dynamical model from scratch would require a large number of samples from the real world. One way to accelerate the model learning process is meta-learning the initial parameters of the model, such as Model-agnostic Meta-Learning (MAML) [Finn et al., 2017]. By using state transition data from the simulation, meta-learning can find an initial set of parameters for the dynamical model such that the model can be trained to match the actual dynamics of the system with only a few data-points. In this chapter, first, we show that when meta-training situations (the prior situations) have diverse dynamics, a single set of meta-trained parameters as a starting point still requires a large number of observations from the real system to learn a useful dynamical model. Next, we propose an algorithm called FAMLE (Fast Adaption through meta-learning Embeddings of simulated priors) that alleviates this limitation by meta-training several initial starting points (i.e., initial parameters) for the model. FAMLE then allows the robot to select the most suitable starting point to adapt the model to the real situation with only a few gradient steps. We compare FAMLE to MBRL, MBRL with a meta-trained model with MAML and model-free policy search algorithm PPO [Schulman et al., 2017] for various robotic tasks and show that FAMLE allows the robots to adapt to novel damages in significantly fewer time-steps than the baselines.

In **Chapter 6**, we discuss the advantages and limitations of our proposed algorithms in detail. We explore the future scope of our proposed algorithms from a practical application standpoint and suggest possible improvements on the same. Finally, in **Chapter 7**, we briefly summarize our work and conclude the manuscript.

Chapter 2

Background

We consider the learning problem in robotics as a Reinforcement Learning (RL) problem, where an RL agent interacts with the environment by applying actions and it receives rewards either immediately or in the future. The goal of the agent is to maximize the accumulated reward over time by optimizing its actions. In our case, this reinforcement learning agent is an embodied agent, which means that it has a physical body, the robot itself. Thus, any action performed by the agent can potentially change the state of the robot as well as its surroundings. Traditionally, the RL problem is studied under the assumption that the state-space and action-space are either discrete in nature or can be discretized. Under this assumption, many RL algorithms were proposed based on value-function estimation to efficiently solve the problem [Sutton and Barto, 1998]. However, in many real-world problems, including robotics, this assumption does not hold. One of the recent successes in RL based on value-function estimation for continuous state-space is deep Q-leaning [Mnih et al., 2013, 2015b] which was later extended to continuous action-spaces in deep deterministic policy gradient approach (DDPG) [Lillicrap et al., 2016]. Nevertheless, much of the recent progress in RL, especially in robotics, have been possible using policy search approaches.

In this chapter, we will discuss various ways of solving the reinforcement learning problem in the context of robotics. The rest of this chapter is organized as follows: section 2.1 presents the formulation of the reinforcement learning problem. In section 2.2 we discuss briefly about value-function based approaches and then in section 2.3 we elaborate various policy search approaches in reinforcement learning. Section 2.4 focusses on model-based reinforcement learning approach and discusses various model-learning methods and policy optimization methods under this framework. Finally, section 2.5 elaborates on various ways of incorporating prior information to improve the data-efficiency of reinforcement learning algorithms in robotics.

2.1 Reinforcement Learning Problem

The aim of a reinforcement learning (RL) agent is to maximize the cumulative reward by performing actions in the environment. Formally, a reinforcement learning problem is represented by a Markov decision process (MDP) which is defined by the tuple $(\mathcal{S}, \mathcal{A}, p, r, \gamma, \rho_0, H)$; where, $\mathcal{S}$ is the set of states, $\mathcal{A}$ is the set of actions, $p(s'|s, a)$ is the state transition probability for given state s and

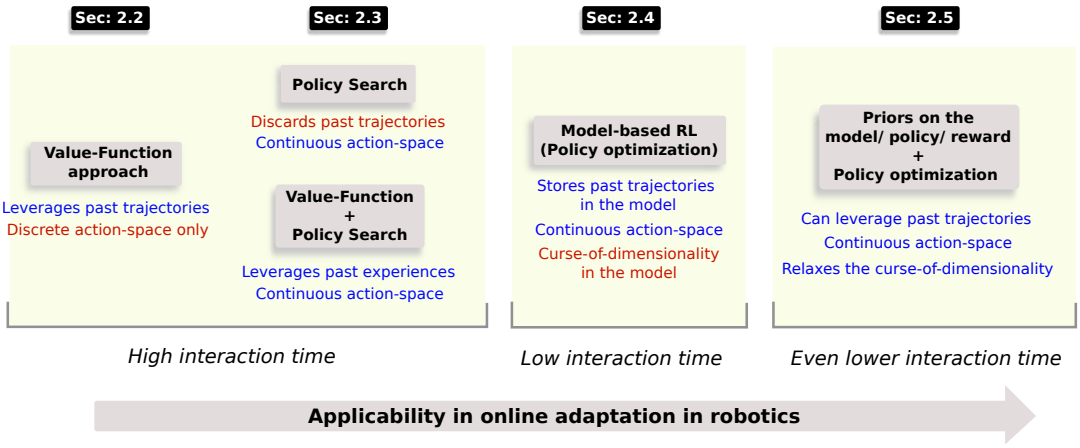


FIGURE 2.1: Overview of the reinforcement learning approaches discussed in this chapter.

action a, $r : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \mapsto \mathbb{R}$ is the reward function, ρ_0 is the initial state distribution, γ is the discount factor, and H is the horizon. An RL agent tries to find a policy $\pi : \mathcal{S} \mapsto \mathcal{A}$ such that the it maximizes the expected return R given by:

$$J = \mathbb{E}_\pi\Big[\sum_{t=0}^{H-1} \gamma^t r(s_t, a_t, s_{t+1})\Big] \tag{2.1}$$

The state transition probability $p(s'|s, a)$ and the reward function $r(s, a, s')$ altogether is also called the environment of the agent. Typically, there are 2 main strategies of solving a reinforcement learning problem [Sutton and Barto, 1998]:

1. Value-function approach (section 2.2)
2. Policy-search approach (section 2.3)

There is one more approach called Actor-Critic approach that utilizes both value-function (critic) approach and the policy (actor) search approach. We will discuss this approach with policy search approach in section 2.3. The overview of the reinforcement learning approaches that we are going to discuss in this chapter is given in the figure 2.1.

2.2 Value-Function Approach

A *state-value-function* or simply *value-function* $V_\pi(s)$ is the expected return when the system is controlled using the policy π from the state s onwards:

$$V_\pi(s) = \mathbb{E}_\pi\Big[\sum_{t=0}^{H-1} \gamma^t r(s_t, a_t, s_{t+1})|_{s_0=s}\Big] \tag{2.2}$$

An optimal value-function is the value-function $V_{\pi^*}(s)$ (simply $V^*(s)$) is the value-function corresponding to the policy π^* that maximizes the expected return for any initial state s.

$$V^*(s) = \arg\max_{\pi} V_{\pi}(s) \tag{2.3}$$

In value-function approach, the agent first tries to find the optimal value-function $V^*(s)$ of the environment. Once optimal value-function is found, it extracts the optimal policy from the optimal value-function $V^*(s)$ as:

$$\pi^*(s) = \arg\max_{a} \mathbb{E}_{s'}\Big[r(s,a,s') + \gamma V^*(s')\Big] \tag{2.4}$$

When, state-transition probability is unknown, it is not possible to know apriori the next state s' when action a is applied in the state s. Thus, it is not possible to compute the optimal policy π^* directly from the optimal state-value-function $V^*(s)$. In such cases, it is more convenient to find out the optimal action-value-function $Q : \mathcal{S} \times \mathcal{A} \mapsto \mathbb{R}$ instead of state-value V. An action-value $Q_\pi(s,a)$ for a particular action a in a state s under a policy π is the expected return when the action a is applied at state s and then onwards the policy π is followed:

$$Q_\pi(s,a) = \mathbb{E}_\pi\Big[\sum_{t=0}^{H-1} \gamma^t r(s_t, a_t, s_{t+1})|_{s_0=s, a_0=a}\Big] \tag{2.5}$$

Similar to optimal state-value, optimal action-value $Q_{\pi^*}(s,a)$ (or simply $Q^*(s,a)$) is the action-value corresponding to the optimal policy π^*. Thus, once the optimal action-value-function is found, the optimal policy can be found as:

$$\pi^*(s) = \arg\max_{a} Q^*(s,a) \tag{2.6}$$

In robotics, typically, the environment is unknown to the agent, i.e., the state transition probability $p(s'|s,a)$ and reward function $r(s,a,s')$ are unknown. In that case, the agent must *estimate* the optimal state-value-function $\hat{V}^*$ or the action-value-function $\hat{Q}^*$ from the experiences in the environment. In such cases, there are primarily two different methods that fall under the value-function based approach: (i) Monte-Carlo methods and (ii) Temporal-difference methods.

2.2.1 Monte-Carlo Methods

Monte-Carlo methods use sampling to estimate the value-function [Sutton and Barto, 1998]. These methods mainly involve two steps which are iteratively performed to optimize the value function. First, the policy evaluation, where

several samples of the of state action trajectories are drawn for the states using the current policy and the value function (or the action-value function) is estimated by taking the mean of the returns for each observed state. Then, the current policy is updated according to the current value function. In the next iteration, the updated policy is used to draw new state-action trajectories to estimate the value function. Iteratively performing these steps drives the value-function or the action-value function towards the optimal value V^* or Q^*.

2.2.2 Temporal-difference (TD) methods

Unlike Monte-Carlo method, TD methods [Sutton, 1988] do not have to wait till the end of an episode to update the value function. TD methods update the value function after applying each action by computing a quantity called *TD error*, which is the difference between the old estimate $V_t(s)$ and the new estimate $V_{t+1}(s)$ of the value function taking into account the instantaneous reward received $r(s_t, a_t, s_{t+1})$ in the current step (t represents the time-step). Thus, the simplest TD update with learning rate α can be written as:

$$V(s_t) := V(s_t) + \alpha[\underbrace{r(s_t, a_t, s_{t+1}) + \gamma V(s_{t+1}) - V(s_t)}_{\text{TD error}}] \tag{2.7}$$

The equivalent TD learning update for action-value can be written as:

$$Q(s_t) := Q(s_t) + \alpha[r(s_t, a_t, s_{t+1}) + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)] \tag{2.8}$$

Where, a_t is the ϵ-greedy action based on Q values at state s_t. This is called SARSA algorithm [Rummery and Niranjan, 1994; Sutton, 1996]. SARSA is an On-policy learning algorithm as the action is obtained from the policy derived from the current Q function that is being updated at every step. Thus, to promote exploration in SARSA, stochastic policies such as ϵ-greedy or ϵ-soft policies are used.

One of the early breakthrough in RL came with an off-policy algorithm called *Q-Learning* [Watkins and Dayan, 1992]. Here, the action-value is updated independently of the policy being followed by the agent as:

$$Q(s_t) := Q(s_t) + \alpha[r(s_t, a_t, s_{t+1}) + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a_t)] \tag{2.9}$$

The methods discussed above so far come under the tabular methods in RL. These methods are applicable when state and action spaces are discrete or can be discretized. However, state-action space in robotics is often extremely large as they grow exponentially with the dimensionality of state and action space, making these approaches less efficient due to the *curse of dimensionality* [Bellman, 1957]. For example, Q-learning can estimate the Q value for a particular state-action pair only when that state-action pair is observed during the learning process. As state-action space grows, it becomes less and less likely

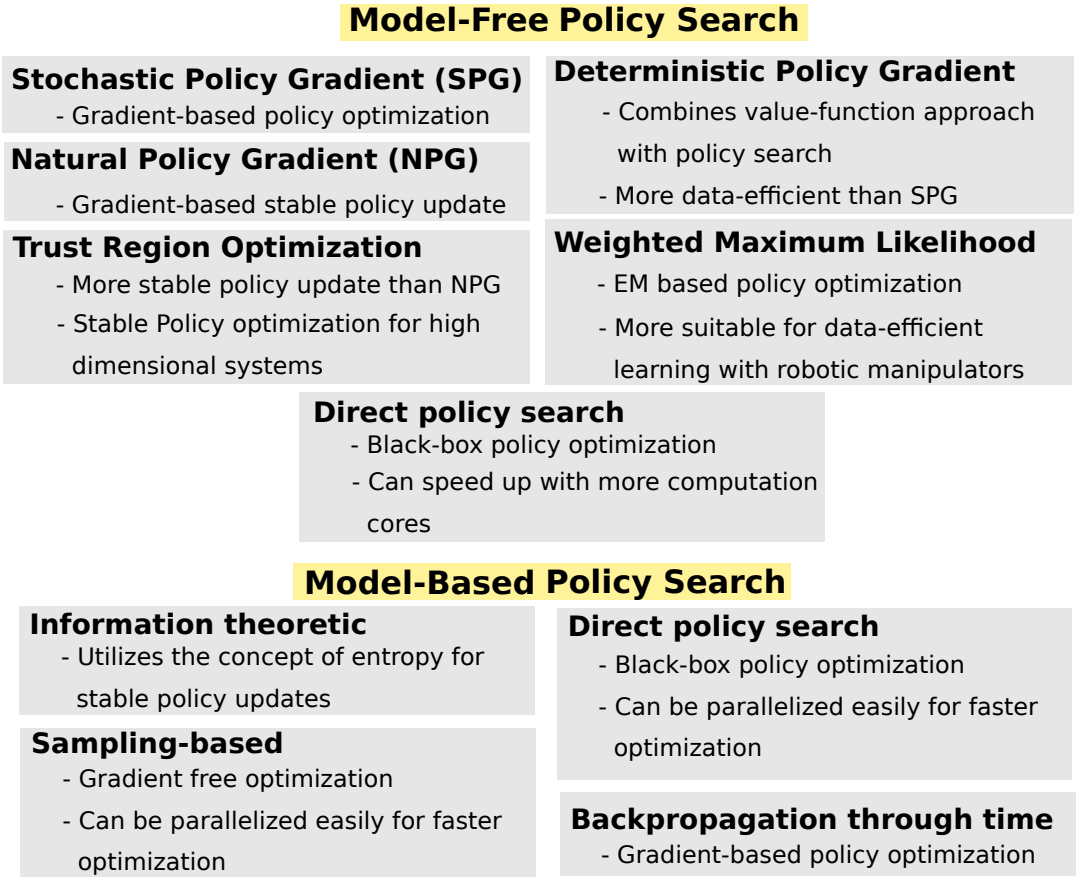


FIGURE 2.2: Overview of the policy search approaches in robotics.

to observe all the state-action pairs in the environment, which makes Q learning less efficient.

To be able to generalize the update of the Q values for the unseen state-action pairs, recently Mnih et al. [2015b] proposed an algorithm based on Q-learning that uses a deep neural network to approximate the Q function [Mnih et al., 2015b]. This method was able to show unprecedented performance of reinforcement learning for very high dimensional state space (e.g., learning to play Atari games by observing pixels). Nevertheless, these methods are limited to simulated environments, such as games, due to their data-hungry nature. For example, Mnih et al. [2015b] required 38 days of game experience to lean to play human-level Atari 2600 games using DQN (Deep Q Network).

2.3 Policy Search Approach

Much of the recent successes in reinforcement learning for robotics have been possible using policy search approach. The goal of policy search approach is to optimize the parameters θ of a policy $\pi_\theta(s)$ so that expected return is maximized:

$$\theta^* = \arg\max_{\theta} \mathbb{E}_{\pi_\theta}\Big[\sum_t \gamma^t r(s_t, a_t, s_{t+1})\Big] \tag{2.10}$$

In the context of robotics, the policy $\pi_\theta(s)$ maps the sensor inputs (i.e., the state) to motor commands (i.e., actions) for the robot. There are several advantages of policy search over the value-function approach, especially in robotics which made it more popular in recent years in the robot-learning field. For example, the most important advantage of using a parametric policy is that it makes it possible to integrate priors such as expert knowledge or domain appropriate structuring of the policy to the bias actions for faster learning. Typically, unlike the value function approach that attempts to find the global optimal value function and thereby the globally optimal policy, policy search instead attempts to optimize the policy locally. Thus, good initialization of the policy parameters or a domain-specific structuring of the policy can make the policy search converge significantly faster than the value-function based approach.

Policy search approaches can be classified into model-free and model-based policy search approaches based on whether policy improvement is done directly using the data obtained from the interaction of the agent with the environment or optimization is done on a learned model (or a simulator or a mathematical model) of the environment. Model-based policy search algorithms are not only more data-efficient than the model-free algorithms, but also provide additional flexibility of integrating prior knowledge in the dynamics model too, which can allow faster learning of the model and thereby improve the data-efficiency. An overview of different policy search approaches is shown in the figure 2.2.

2.3.1 Stochastic Policy Gradient Method

In continuous control regime of reinforcement learning, one of the most popular class of model-free policy search methods is *policy gradient methods* [Sutton et al., 2000]. Policy gradient methods use the gradient of the expected return (the objective) with respect to the policy parameter to perform gradient-ascent on the policy parameters. The fundamental result of these algorithms is the *stochastic policy gradient theorem* [Sutton et al., 2000]. If the trajectory distribution $p^\pi(\tau)$ corresponding to the stochastic policy $\pi(a|s,\theta)$ is given by:

$$p^\pi(\tau) = p(s_0) \prod_0^{H-1} p(s_{t+1}|s_t, a_t)\pi(a_t|s_t, \theta) \tag{2.11}$$

Then, the stochastic policy gradient theorem can be written as:

$$\nabla_\theta J(\theta) = \int p^\pi(\tau) \int \nabla_\theta \log \pi(a|s,\theta) Q^\pi(s,a) ds da \tag{2.12}$$

$$= \mathbb{E}\Big[\sum_{t=0}^{H-1} \nabla_\theta \log \pi(a_t|s_t,\theta) Q^\pi(s_t, a_t)\Big] \tag{2.13}$$

The policy gradient theorem has been used later to derive several policy gradient algorithms using sample-based estimation of the expectation. These algorithms proposed various ways of estimating the action-value function $Q^\pi(s_t, a_t)$. The simplest one is to use the return $R(\tau)$ of the trajectory as the estimate of the true Q value. In REINFORCE [Williams, 1992], G(PO)MDP [Baxter and Bartlett, 2001], PGPE [Sehnke et al., 2008] etc. improved the sample return based estimation of this Q value in various ways.

The search for a better estimation of the Q value in policy gradient theorem opened the frontiers for a new class of policy gradient algorithms called *actor-critic* algorithms [Sutton et al., 2000; Peters and Schaal, 2008a; Bhatnagar et al., 2008]. The actor-critic algorithms connect the gap between policy search approach and value function approach where a learned function approximator is used to approximate the action-value which is updated, typically, with temporal-difference learning (section 2.2.2). It is worth mentioning here that policy gradient approaches based on the actor-critic formulation can be more sample-efficient compared to direct policy gradient approaches (i.e., without Q value estimation) since actor-critic approaches can use past experiences for policy updates. In this direction, Haarnoja et al. [2018] proposed soft actor-critic (SAC) approach, which uses a stochastic policy, and policy updates try to maximize both the expected reward and the expected entropy of the policy. As a result, unlike previous actor-critic methods, SAC is much more stable, and the performance is less dependent upon the hyperparameter tuning while retaining the superior sample-efficiency compared to the state-of-the-art direct policy gradient approaches. Another actor-critic based approach called Asynchronous Normalized Advantage Function algorithm (A-NAF) [Gu et al., 2017] decoupled the policy improvement from the off-policy experience gathering rollouts. Here, the rollouts are performed in parallel on several robots performing the same task, which reduces the amount of time required to learn a skill. However, using several robots makes it not only less cost-effective but also makes it virtually infeasible on many complex robots such as humanoids.

2.3.2 Deterministic Policy Gradient Method

Silver et al. [2014] introduced *Deterministic Policy Gradient (DPG) theorem* which extended the actor-critic stochastic policy gradient methods to deterministic policies:

$$\nabla_\theta J(\theta) = \int p^\pi(\tau) \nabla_\theta Q^\pi(s, a = \pi(s|\theta)) ds \tag{2.14}$$

$$= \mathbb{E}\Big[\sum_{t=0}^{H-1} \nabla_\theta \pi(a_t|s_t, \theta) \nabla_{a_t} Q^\pi(s_t, a_t)|_{a_t=\pi(s_t|\theta)} \Big] \tag{2.15}$$

Although DPG algorithms provide a better theoretical basis for continuous control using policy gradient method, naïve application of this actor-critic method with neural function approximators is unstable for challenging problems. In fact, Silver et al. [2014] showed that the function approximator for the action-value (the critic) in DPG should be linear to avoid instability in

the optimization process. However, following the breakthrough in the deep Q-learning [Mnih et al., 2013, 2015b], where a deep neural network was used to approximate the action-value function, Lillicrap et al. [2016] proposed an algorithm called *Deep Deterministic Policy Gradient (DDPG)* that used deep neural network for the critic in DPG approach. Similar to Deep Q-learning, DDPG was able to learn the critic in a stable and robust manner primarily due to two ideas: (1) the network was trained off-policy with samples from a replay buffer to minimize correlations between samples; (2) the network is trained with a target Q network to give consistent targets during temporal difference backups.

2.3.3 Natural Policy Gradient Method

The policy gradient methods discussed in the previous sections assume that all the dimensions of the policy parameter θ have similar effects on the resulting distribution $p_\theta(\tau)$ and thus take a small step $\Delta\theta$ along the policy gradient $\nabla_\theta J(\theta)$ where

$$\Delta\theta = \alpha \nabla_\theta J(\theta), \ \alpha \text{ is the learning rate} \tag{2.16}$$

However, a small change in θ might case a large change in the resulting distribution $p_{\theta+\Delta\theta}(\tau)$ compared to the old distribution $p_\theta(\tau)$. This kind of update can be catastrophic and lead to unstable behavior in the learning process since the policy gradient $\nabla_\theta J(\theta)$ is computed based on the samples according to the distribution $p_\theta(\tau)$. To have a stable learning process, it is desirable that the new distribution $p_{\theta+\Delta\theta}(\tau)$ is closer to the old distribution $p_\theta(\tau)$, i.e.,

$$D_{KL}[p_\theta(\tau)||p_{\theta+\Delta\theta}(\tau)] \approx \Delta\theta^T F(\theta)\Delta\theta \leq \epsilon, \text{ for some small } \epsilon \tag{2.17}$$

where, $D_{KL}[\cdot||\cdot]$ is the Kullback–Leibler divergence, and $F(\theta)$ captures how a single parameter influence the distribution $p_\theta(\tau)$. $F(\theta)$ is called the Fisher information matrix (FIM) and is given by:

$$\begin{aligned} F(\theta) &= \mathbb{E}_{p_\theta(\tau)}\Big[\nabla_\theta \log p_\theta(\tau) \nabla_\theta \log p_\theta(\tau)^T\Big] \\ &= \mathbb{E}_{p_\theta(\tau)}\Big[\sum_{t=0}^{H-1} \nabla_\theta \log \pi(a_t|s_t,\theta) \nabla_\theta \log \pi(a_t|s_t,\theta)^T\Big] \end{aligned} \tag{2.18}$$

The *Natural Policy Gradient* (NPG) [Kakade, 2002; Peters and Schaal, 2008a; Bhatnagar et al., 2008] formulates this idea as an optimization problem:

$$\arg\max_\theta \mathbb{E}_{p_{\theta_{old}}(\tau)}\Big[\frac{\pi_\theta(a|s)}{\pi_{\theta_{old}}(a|s)} Q_{old}(s,a)\Big] \tag{2.19}$$

$$\text{subject to } D_{KL}[p_{\theta_{old}}(\tau)||p_\theta(\tau)] \leq \epsilon \tag{2.20}$$

Taking first order Taylor series approximation of the objective and second order approximation of the constraint, the optimization problem can be rewritten as:

$$\arg\max_{\Delta\theta} \Delta\theta^T \underbrace{\nabla_\theta J(\theta)}_{\text{Policy Gradient}} \tag{2.21}$$

$$\text{subject to } \Delta\theta^T F(\theta)\Delta\theta \leq \epsilon \tag{2.22}$$

The resulting natural policy gradient can be computed in closed form as:

$$\nabla_\theta^{NPG} J(\theta) = F(\theta)^{-1}\nabla_\theta J(\theta) \tag{2.23}$$

The corresponding policy update can be written as:

$$\theta_{new} = \theta_{old} + \sqrt{\frac{2\epsilon}{\nabla_\theta J(\theta)^T F(\theta)^{-1}\nabla_\theta J(\theta)}} F(\theta)^{-1}\nabla_\theta J(\theta) \tag{2.24}$$

This learning process stabilizes the policy update by avoiding aggressive steps and premature convergence on plateaus as well. Natural policy gradient can be used to update the policy at every step as well as in episodic learning framework. Policy gradient methods dominated the policy search approach for quite a long time. These methods have been used in many robot learning problems successfully in the past, such as [Peters et al., 2003; Kohl and Stone, 2004; Peters and Schaal, 2008b]. However, most of these works used policies that are specific to manipulators, e.g., DMPs [Ijspeert et al., 2003], with a small number of parameters and often with initialization through demonstrations. For instance, Peters and Schaal [2008b] could train a robotic arm to hit a baseball with initial demonstration and around 200 trials for policy improvement. Moreover, these methods need careful tuning of the learning rate as well as the exploration parameter.

2.3.4 Weighted Maximum Likelihood Methods

The basic idea behind this class of methods is to change the current policy $\pi(a|s, \theta_{old})$ such that in the new policy $\pi(a|s, \theta_{new})$ the probability of high rewarding actions is higher [Dayan and Hinton, 1997]:

$$\pi(a|s, \theta_{new}) \propto f(r(s, a))\pi(a|s, \theta_{old}) \tag{2.25}$$

Where, $f(r(s, a))$ is the success probability. Several algorithms have been proposed based on this idea using expectation maximization (EM) formulation. Peters and Schaal [2007] proposed *Reward Weighted Regression (RWR)* where $f(r(s, a))$ is taken as parametric function of the reward whose parameters are updated along with the policy parameters. Then Kober and Peters [2009] improved RWR by proposing a new algorithm called *PoWER*, which has

better exploration compared to RWR. Similarly, *MCEM* [Vlassis et al., 2009] generalized the PoWER algorithm and used Monte-Carlo approach for expectation maximization. Although these approaches alleviate the need to specify the learning rate, still EM-based approach does not guarantee a stable update of the policy, i.e., staying closer to the data. The information-theoretic approach *Relative Entropy Policy Search(REPS)* [Peters et al., 2010] combined the advantages of both natural policy gradient method and weighted maximum likelihood method to allow stable update of the policy parameters without the need of specifying the learning rate.

2.3.5 Trust Region Optimization Method

In the previous section we have seen that for a stable learning process, it is desirable to keep the updated policy close to the old policy so that the data distribution of the new policy is not very different from the old policy. This objective was achieved using a KL constraint on the policy in NPG method (section 2.3.3). However, the KL constraint was approximated in this method by using a second-order Taylor series expansion of the KL. Due to this approximation, it is still possible that the new policy violates the KL constraint. To guarantee that the KL constraint is always satisfied, Schulman et al. [2015] proposed an algorithm called *Trust Region Policy Optimization (TRPO)* which is very similar to NPG. The main difference is that TRPO performs a simple line-search on the step-size parameter to guarantee that the KL constraint is always satisfied after the policy update. Thus, TRPO can be thought of as "NPG + Line-search on the step size". To be more precise, if Δ_θ is the proposed policy update given by NPG, then TRPO update is given by:

$$\begin{aligned} \Delta_\theta &= \sqrt{\frac{2\epsilon}{\nabla_\theta J(\theta)^T F(\theta)^{-1} \nabla_\theta J(\theta)}} F(\theta)^{-1} \nabla_\theta J(\theta) \\ \theta_{new} &= \theta_{old} + \alpha^n \Delta_\theta; \quad n \in \{0, 1, 2, 3, ..., L\} \end{aligned} \tag{2.26}$$

Where, n is optimized using simple line-search, i.e., iteratively trying values from $\{0, 1, 2, 3, ..., L\}$ so that KL constraint (equation 2.20) is satisfied. With this extension, TRPO seemed to perform much better (faster convergence, higher asymptotic performance) than its predecessors. In the domain of robotic locomotion, TRPO successfully learned controllers for swimming, walking and hopping in a physics simulator, using general-purpose neural networks and minimally informative rewards. Before TRPO, no prior work has learned controllers from scratch using gradient-based policy optimization for all of these tasks [Schulman et al., 2015], using a general policy search method and non-engineered, general-purpose policy representations.

Nevertheless, TRPO is relatively complicated (2^{nd} order optimization), and is not compatible with architectures that include noise (such as dropout) or parameter sharing (between the policy and value function, or with auxiliary tasks) Schulman et al. [2017]. Schulman et al. [2017] proposed a class of algorithms called *Proximal Policy Optimization (PPO)* that borrows the similar idea of TRPO and attains the data-efficiency and reliable performance of TRPO, while

using only first-order optimization. PPO considers the constrained optimization problem given by Eq. 2.20 as an unconstrained optimization problem using a penalty as:

$$\arg\max_{\theta} \mathbb{E}_t\Big[\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)}Q_{old}(s_t,a_t) - \beta D_{KL}[\pi_{\theta_{old}}(.|s_t)||\pi_\theta(.|s_t)]\Big], \text{ for some } \beta \tag{2.27}$$

The *adaptive KL penalty* variant of PPO, optimize the policy using several steps of SGD for the objective 2.27. Then it adjusts the value of β depending upon whether $D_{KL}[\pi_{\theta_{old}}(.|s_t)||\pi_\theta(.|s_t)]$ less than or more than ϵ. In the next iteration, this updated value of β is used while optimizing the policy. The initial value of β is a hyperparameter here, but it is not important in practice because the algorithm quickly adjusts it after a few iterations.

Another variant of PPO called *clipped surrogate objective* uses a surrogate objective which is optimized at every iteration by taking several gradient steps using SGD:

$$\arg\max_{\theta} \mathbb{E}_t\Big[\min\Big\{\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)}Q_{old}(s_t,a_t), \\ clip\Big(\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)}, 1-\delta, 1+\delta\Big)Q_{old}(s_t,a_t)\Big\}\Big] \tag{2.28}$$

where, δ is a hyperparameter such that $\delta \in [0,1]$. PPO algorithms have the stability and reliability of trust-region methods, but are much simpler to implement, applicable in more general settings (such as using a joint architecture for the policy and value function), and have better overall performance. PPO outperformed previous approaches in many simulated high dimensional continuous control tasks.

Model-free reinforcement learning (RL) algorithms have achieved impressive successes during the last few years, from learning to play games from pixels to beating professional Go players, but at the expense of enormous interaction time with the system. For example, they required up to 38 days of game-play (real-time) for Atari 2600 games [Mnih et al., 2015a], 4.8 million games for Go [Silver et al., 1 28], or about 100 hours of simulation time (more if real-time) to train a 9-DOF mannequin to walk [Heess et al., 2017]. This makes these algorithms suitable only for policy *synthesis*, that is, creating a policy for a robot in simulation, but difficult to use for *online learning* in robotics, that is, adapting online to a new system or a new situation. By the term "situation" we refer to any perturbation in the dynamics of the robot caused by physical damages, faults in the actuators or environmental changes such as terrain condition.

2.3.6 Direct policy search

Direct policy search approaches consider policy search as an optimization problem. Thus, these algorithms do not consider the entire state-action-reward trajectory for optimizing the policy. Instead, they only require the final reward of the episode for a given policy. As a result, direct policy search algorithms do not compute the analytical gradient of the reward to perform gradient ascent on the policy parameters. Below, we discuss some of the noteworthy approaches that fall into the direct policy search category.

Evolutionary algorithms: The potential advantages of evolutionary algorithms compared to standard RL methods are that they (1) are often easier to apply and are more robust with respect to the tuning of the hyperparameters (2) can be applied if the function approximators (learned dynamical model, policy or reward function) are non-differentiable, and (3) can also optimize the underlying structure of the function approximator used for the policy [Stanley and Miikkulainen, 2002; Such et al., 2017].

Recent researches on evolutionary algorithms applied to RL problems revealed many promising insights [Salimans et al., 2017]. For example, [Salimans et al., 2017] reported that Evolutionary Strategies, a special class of evolutionary algorithms, exhibited better exploration behaviour than policy gradient methods like TRPO: on the MuJoCo humanoid task, ES has been able to learn a very wide variety of gaits (such as walking sideways or walking backwards). These unusual gaits are never observed with TRPO, which suggests a qualitatively different exploration behavior. They were also able to achieve linear speedup on the computation time with the number of workers. In particular, using 1,440 workers, they were able to solve the MuJoCo 3D humanoid task in under 10 minutes. Moreover, they were able to match the final performance of A3C [Mnih et al., 2 04] on most of the Atari environments while using between 3x and 10x as much data.

Some popular evolutionary algorithms used in robotics (both in model-based and model-free frameworks) are explained below:

CMA-ES: Covariance Matrix Adaptation Evolutionary Strategy (CMA-ES) [Hansen and Ostermeier, 1996] is an evolutionary algorithm for difficult non-linear non-convex black-box optimization problems in continuous domain. It is considered as state-of-the-art [Hansen, 2009a,b] in evolutionary computation and has been adopted as one of the standard tools for continuous optimization when gradient-based optimization (supposedly) fails due to a rugged search landscape (e.g. discontinuities, sharp bends or ridges, noise, local optima, outliers). Being a population-based algorithm, CMA-ES is robust to uncertainties present in the function. In particular, the rank-based selection operation in the variant UH-CMAES [Hansen et al., 2009] allows for a robust quantification and handling of uncertainties in the cost function. Thus, CMA-ES can effectively be used to optimize the policy when the learned dynamical model of the system is probabilistic.

CMA-ES performs roughly the following steps at each generation k:

1. Sample λ policies from a multivariate normal distribution with mean μ_k and covariance $\sigma_k\Sigma_k$, i.e., $\theta_i \sim \mathcal{N}(\mu_k, \sigma_k\Sigma_k)$, $i = 1, 2, ..., \lambda$. Here $\sigma_k > 0$ is the step size.

2. Evaluate the policies on the model and compute the new mean μ_{k+1} from the n best policies: $\mu_{k+1} = \sum_{i=1}^{n} w_i\theta_i$ such that $w_i > w_j$ if $J(\theta_i) > J(\theta_j)$ and $\sum w_i = 1$

3. Based on the n best policies, their mean μ_{k+1} and the old mean μ_k compute the parameters σ_{k+1} and Σ_{k+1} to sample a new population.

Iteratively performing the above steps drive the distribution $\mathcal{N}(\mu_k, \sigma_k\Sigma_k)$ towards the optimal region the policy space.

NES: Natural Evolution Strategy (NES) is a class of evolution strategies that performs stochastic gradient ascent steps on the parameters of a probability distribution of the policy [Wierstra et al., 2008]. This approach is closely related to a model-free policy search algorithm called *Policy Gradients with Parameter-based Exploration (PGPE)* [Sehnke et al., 2010]. In NES, at every generation k, the population $\pi^{(k)}$ is parameterized by $\theta = \langle\mu, \Sigma\rangle$, where μ is the mean and Σ is the covariance matrix of a multi-variate Gaussian distribution. If z is any search point (policy) sampled from the current population parameters $\theta = \langle\mu, \Sigma\rangle$, then probability density of the search point z will be:

$$p(z|\theta) = \mathcal{N}(z|\mu, \Sigma) \tag{2.29}$$

Now if $f(z)$ is the fitness at any point z, then expected fitness of the population:

$$J(\theta) = \mathbb{E}_z[f(z)] = \int p(z|\theta)f(z)dz \tag{2.30}$$

Using "likelihood-ratio trick", the gradient of the expected fitness can be computed as:

$$\nabla_\theta J(\theta) = \mathbb{E}_z[\nabla_\theta \log p(z|\theta)f(z)] \tag{2.31}$$

Thus, gradient with respect to μ and Σ will be:

$$\nabla_\mu \log p(z|\theta) = \Sigma^{-1}(z - \mu) \tag{2.32}$$

$$\nabla_\Sigma \log p(z|\theta) = \frac{1}{2}\Sigma^{-1}(z - \mu)(z - \mu)^T\Sigma^{-1} - \frac{1}{2}\Sigma^{-1} \tag{2.33}$$

By representing $\Sigma = A^TA$ to ensure positive variance and keep it positive semi-definite, the gradients $\nabla_A \log p(z|\theta)$ can be computed as:

$$\nabla_A \log p(z|\theta) = A[\nabla_\Sigma \log p(z|\theta) + \nabla_\Sigma \log p(z|\theta)^T] \tag{2.34}$$

Now, the population can be updated as:

$$\theta = \langle \mu, \Sigma = A^T A \rangle \leftarrow \theta + \beta \nabla_\theta J(\theta), \ \beta \text{ is the learning rate} \tag{2.35}$$

$\nabla_\theta J(\theta)$ in Eq.2.35 called *vanilla gradient for evolution strategy*. Since, new population parameter is computed here using the information from the last population, it is important that the new population stay closer to the old population, i.e., the probability distribution $p(z|\theta_{old})$ and $p(z|\theta_{new})$ should be closer to each other. To ensure this, NES uses natural gradient instead of the vanilla gradient and computes the update $\delta\theta$ as:

$$\delta\theta = (\Phi^T \Phi)^{-1} \Phi^T R \tag{2.36}$$

where

$$\Phi = \begin{bmatrix} \nabla_\theta \log p(z_1|\theta) & 1 \\ \vdots & \vdots \\ \nabla_\theta \log p(z_n|\theta) & 1 \end{bmatrix} \tag{2.37}$$

$$R = [f(z_1), \ldots, f(z_n)]^T \tag{2.38}$$

NES has competitive performance as CMA-ES on unimodal tasks, while outperforms it on some multimodal tasks that are rich in deceptive local optima [Wierstra et al., 2008]. It is worth mentioning here that both NES and CMA-ES are closely related to each other. CMA-ES follows an approximate natural gradient, whereas NES follows estimated natural gradient [Akimoto et al., 2010].

Neuroevolution: Neuroevolution [Yao, 1999; Stanley et al., 2019] is a class of evolutionary algorithms to optimize the neural networks' weights, architecture, or both at the same time. These algorithms are inspired by the fact that natural brains themselves are the products of an evolutionary process. In the context of RL, the simplest form of neuroevolutionary algorithm optimizes only the weights of the neural network policy, i.e., the topology of the network remains fixed. The more sophisticated class of neuroevolutionary algorithms, collectively called TWEANNs (topology and weight evolving artificial neural networks) [Stanley et al., 2019], can simultaneously search in the policy parameters space and the policy representation (network topology) space. The basic steps of a neuroevolutionary algorithm are as follows. First, a random population of neural network policies is initialized. Then, in each generation, each policy network in the population is evaluated on the task. Next, the best performing policies are selected, e.g., via rank-based selection, roulette wheel

selection, or tournament selection [Goldberg and Deb, 1991]. Then, by performing crossover and mutation on the selected population, the new population is formed, and the process repeats.

Perhaps the most popular TWEANN is "neuroevolution of augmenting topologies" (NEAT) [Stanley and Miikkulainen, 2002]. To represent networks of varying topologies, NEAT employs a flexible genetic encoding. Each network is described by a list of edge genes, each of which describes an edge between two node genes. Each edge gene specifies the in-node, the out-node, and the weight of the edge. During mutation, a new structure can be introduced to a network via special mutation operators that add a new node or edge genes to the network. To avoid catastrophic crossover, NEAT relies on innovation numbers, which track the historical origin of each gene. Whenever a new gene appears via mutation, it receives a unique innovation number. Thus, the innovation numbers can be viewed as a chronology of all the genes produced during evolution. Neuroevolution has a profound history in learning policies/controllers in robotics, starting from evolving neural controllers for hexapod robot [Gallagher et al., 1996] to producing the state-of-the-art performance on simulated robot control tasks, outperforming complex, modern policy gradient methods [Mania et al., 2018]. Huang et al. [2014] used neuroevolution as a training method to enable a real robotic system to grasp novel objects in an unstructured environment. Neuroevolution has also been successful in training deep neural network policies for many challenging tasks, such as Atari games and simulated humanoid locomotion task. Such et al. [2017] showed that neuroevolution is a competitive alternative to deep reinforcement learning algorithms.

Quality-diversity optimization approaches: These algorithms are based on the fact that natural evolution does not produce one effective solution but rather an impressively large set of different organisms, all well adapted to their respective environment. Quality-Diversity optimization algorithms aim to produce a large collection of solutions that are both as diverse and high-performing as possible, which covers a particular domain, called the descriptor space [Cully and Demiris, 2017].

In this direction, the "Novelty Search" algorithm [Lehman and Stanley, 2011a; Doncieux et al., 2019] completely drops the objective of maximizing the performance, and searches only for the solutions that are different (i.e., novel) from the previously found solutions in the descriptor space. For instance, in biped locomotion task, the novelty search algorithm would give reward for simply falling down in a different way, regardless of whether it is closer to the objective behaviour (walking as far as possible) or not. After a few ways to fall are discovered, the only way to be rewarded is to find a behavior that does not fall immediately. In this way, behavioral complexity rises from the bottom up. Eventually, to do something new, the biped would have to successfully walk for some distance even though it is not an objective. Using novelty search, a simulated biped robot (6-DOF) could learn to walk much faster than using the fitness-objective-based approach with the same number of evaluations [Lehman and Stanley, 2011a].

Among the algorithms that search for diverse and high performing solutions, MAP-Elites (Multi-dimensional Archive of Phenotypic Elites) [Mouret

and Clune, 2015; Cully et al., 2015; Vassiliades et al., 2017] has shown many promising results. For instance, a damaged hexapod robot could learn to walk within 2 minutes of real-world interaction using MAP-Elites in conjunction with Bayesian optimization [Cully et al., 2015]. The MAP-Elites algorithm divides the behavior space or the descriptor space into discrete bins and aims at finding high-quality solutions in each bin of the behavior space (the behavior repertoire). This local elitism in MAP-Elites promotes an increase in quality throughout the behavior repertoire, as existing solutions can be replaced by newer ones with higher quality (refer to section 2.5.4 for more details on MAP-Elites).

Bayesian Optimization: Bayesian optimization (BO) is a framework for online, black-box, gradient-free global search [Shahriari et al., 2015; Brochu et al., 2010]. It is a state-of-the-art optimization method that can be applied to problems where it is vital to optimize a performance criterion while keeping the number of evaluations of the system small, e.g., when an evaluation requires an expensive interaction with a robot. Bayesian optimization makes efficient use of past interactions (experiments) by learning a probabilistic surrogate model of the function to optimize. Subsequently, the learned surrogate model is used for finding optimal parameters without the need to evaluate the expensive (true) function. By exploiting the learned model, Bayesian optimization, therefore, often requires fewer interactions (i.e., evaluations of the true objective function) than other optimization methods. Thus, BO has been used extensively in the literature for optimizing controllers for robots, e.g., snake robots [Tesch et al., 2011], AIBO quadrupeds [Lizotte et al., 2007], hexapods [Cully et al., 2015; Pautrat et al., 2018], bipedal robots [Rai et al., 2019]. More details on BO can be found in the section 2.5.3.

2.4 Model-Based Policy Search

The core of the model-based policy search (MBPS) approach is continuous *self-modeling*. A robot that continuously learns its self-model, i.e., how its action is going to change its state, can not only be adaptive to the changes in its morphology due to damages but also can learn new skills in a data-efficient manner. Bongard et al. [2006] were among the first ones to combine self-modeling with policy search in robotics. The self-modeling aspect of their approach is very close to model-identification approach [Hollerbach et al., 2016]. It consists of three steps: 1) performing action and collecting data, 2) synthesizing 15 candidate models to explain the data, and 3) actively selecting the actions that will extract the new information from the robot. Following a few cycles of these steps (i.e. about 15), the most accurate model is selected, and policy search is performed to produce a desired behavior. Using this approach, the authors were able to control a four-legged robot in less than 20 episodes. The robot was also able to adapt to damages in a few trials by rerunning the self-modeling procedure.

In MBPS approach, the robot iteratively learns a model of the dynamics from past experiences and uses that model for policy optimization [Deisenroth

and Rasmussen, 2011; Chatzilygeroudis et al., 2017; Kaushik et al., 2018]. Here, by repeatedly evaluating the optimized policies on the robot, more observations are collected to improve the model during the learning process. Since MBPS algorithms optimize the policy using the model, they can be highly data-efficient compared to model-free RL algorithms. It is to be noted here that, in contrast to the MBPS algorithms, classic model-identification approaches are based on two steps: first, learning an effective dynamical model with proper excitation of the system, and then using the model for controlling the system [Hollerbach et al., 2016]. Whereas, in MBPS approach, the model does not need to be perfect for all the possible policies, and the model can be discarded once a good policy is found.

In the context of MDP formulation, MBPS algorithms approximate the environment model $p(s'|s,a)$ (and reward model $r(s,a,s')$ if reward cannot be computed from the estimated states) using observed data. Then, the MBPS agent uses this model as its internal simulator of the environment to optimize the policy. However, this learned model of the dynamics may not represent the true environment or the system. As a result, the policy optimized on this model may not be optimal for the real system. Therefore, much of the literature on MBPS is focused on how to build a better model of the dynamics of the system using the observed data and how to use this model for policy optimization in a robust manner in spite of the presence of model inaccuracies.

In robotics, MBPS typically considers the following set up where state s of the system evolves according to Markovian dynamics:

$$s_{t+1} = f(s_t, a_t) + w, \;\; s_0 \sim p(s_0) \tag{2.39}$$

where $f(\cdot,\cdot)$ is the non linear forward dynamics of the system, a_t is the action at time t, w is the additive i.i.d Gaussian noise of the system and $p(s_0)$ is the initial state distribution. MBPS algorithms try to find the policy π_{θ^*} that maximizes the expected return:

$$\pi_{\theta^*} = \arg\max_{\theta} \mathbb{E}\Big[\sum_{t=0}^{H-1} \gamma^t r(s_t, a_t, s_{t+1})|\pi_\theta\Big] \tag{2.40}$$

where $r(\cdot,\cdot,\cdot)$ is the reward function, H is the horizon and $\gamma \leq 1$ is the discount factor.

2.4.1 Learning the Model

To learn an effective forward dynamics model (and the reward model if required) from the observed data, several approaches have been proposed in the literature [Nguyen-Tuong and Peters, 2011]. Those approaches can be broadly classified into two types: deterministic model and probabilistic model. A deterministic dynamical model predicts the next state s_{t+1} of the system based on the current state s_t and the applied action a_t on the system. Whereas, a probabilistic

dynamical model predicts a probability distribution $p(s_{t+1}|s_t, a_t)$ over the next state given the current state s_t and the applied action a_t.

A deterministic model does not say anything about the uncertainty or confidence of the prediction. As a result, using deterministic models as a "global model" of the dynamics for policy optimization might produce sub-optimal performance since the optimizer can exploit the model greedily. However, a deterministic model can be used as a local approximation of the dynamics around the trajectory distribution $p_\theta(\tau)$ of the current policy π_θ, which can be used to improve the policy in an iterative manner. This type of local model has been used in many recent work [Levine and Koltun, 2013; Levine and Abbeel, 2014; Kumar et al., 2016] to train policies for high dimensional tasks such as 24-DOF dextrous manipulation task [Kumar et al., 2016] in a data-efficient manner.

The effectiveness of a policy function learned using a model depends upon the accuracy of the model in multiple steps ahead prediction. When the model is imperfect due to the scarcity of data, the predicted trajectories deviate significantly from the true trajectories by accumulating model prediction errors. In that case, the optimized policy will behave differently on the real system compared to the behaviour on the model. Instead of optimizing a policy function, a more robust option is to optimize the action at every step (like model predictive control [Camacho and Alba, 2013]) to reduce the dependency on accurate long horizon prediction using the model. Thus, many recent works incorporated model predictive control framework with an iteratively learned deterministic model [Nagabandi et al., 2018, 2019].

When the data is scarce and it is difficult to learn a perfect model of the dynamics from the data, probabilistic models are usually more effective. Probabilistic models can provide uncertainty information (as prediction variance) in the prediction, which can be incorporated into a long-horizon prediction for robust optimization of the policy, i.e., finding a policy that is free from bias due to model inaccuracies. For example, PILCO [Deisenroth and Rasmussen, 2011] uses Gaussian Process regression [Deisenroth et al., 2015] to model the forward dynamics using only a few data points from the real system. PILCO was able to find optimal policies for many tasks such as cart-pole swing up, pendulum swing up etc. by iteratively learning the dynamical model from the observed data with only a few trials on the real systems.

Non-parametric regression models such as Gaussian processes are computationally expensive (to learn as well as to query) and thus increases the optimization time with the size of data. One alternative is to learn an ensemble of parametric models (such as neural networks) using the data sampled from the data distribution. In this direction, Lakshminarayanan et al. [2017] proposed *deep ensembles*, an ensemble of probabilistic neural networks, a type of neural networks whose output neurons simply parameterize a probability distribution function, capturing aleatoric uncertainty. Recently, Chua et al. [2018] used such ensemble deep neural networks with model predictive control (MPC) framework and showed that it matches the asymptotic performance of model-free algorithms such as PPO on several challenging benchmark tasks while requiring significantly fewer samples. They show that this kind of ensemble model is capable of capturing aleatoric (inherent system stochasticity) and epistemic

(subjective uncertainty, due to limited data) uncertainty in the model, which contributes to data-efficiency as well as the robustness of the MPC controller.

Designing a good Bayesian neural network (BNN) models have been an open problem in the literature [MacKay, 1992; Neal, 2012; Osband, 2016; Guo et al., 2017]. Recently, some promising progress has been seen in this direction by Gal et al. [2017] using *Monte-Carlo dropout* in the neural network. Such BNNs have been used in many recent MBPS approaches [Gal et al., 2016; Higuera et al., 2018] and showed the scalability of such models with state-action space dimensionality and data-set sizes compared to Gaussian processes models in complex control tasks.

2.4.2 Policy optimization

Once the model of the dynamics (and the reward function if required) is available, then that model can be used as a surrogate of the real system to optimize the policy. However, due to the presence of model error, the gradient-based model-free RL algorithms that we discussed before are less likely to be efficient enough for policy optimization in this case. These algorithms will greedily exploit the model while optimizing the policy by ignoring the presence of error/uncertainty and will not contribute towards sample efficiency. Several approaches have been proposed in the literature to efficiently use the dynamical model in MBPS framework. We classify these algorithms into 4 categories:

1. Back-propagation through time
2. Direct policy search
3. Information-theoretic
4. Sampling-based

In the following sub-sections, we explain some popular algorithms for model-based policy search under these 4 categories.

Back-propagation through time

When the dynamics model, the reward function and the policy are differentiable, then it is possible to compute the gradient of the long-term reward with respect to the policy parameters and accordingly update the policy parameters with gradient ascent. However, directly using this approach on a learned dynamical model will greedily exploit the model ignoring the model error. Additionally, gradient with respect to the initial actions might explode due to irregular transitions predicted by the model in the unseen region of the state-space. Because of these reasons, a naïve gradient ascent through time on a leaned dynamics model is not an effective approach for policy optimization.

PILCO: To avoid the model bias, PILCO (Probabilistic Inference for Learning Control) [Deisenroth and Rasmussen, 2011] uses Gaussian processes regression to construct a probabilistic model of the dynamics so that the model uncertainties can be expressed during the prediction. Then, with that model,

PILCO applies a deterministic approximate inference technique for policy evaluation, which allows it to perform policy search based on analytic policy gradients. The training process of PILCO iterates between learning the GP model of the dynamics from the observed data and optimizing the policy on that model to execute it on the real system. PILCO reported unprecedented data efficiency on relatively high dimensional tasks. For example, PILCO was able to train a 28-dimensional parametric policy within 20 seconds (more including optimization time) of trials to balance a unicycle with 12D state-space and 5D action space in simulation. Additionally, PILCO was able to find a policy to balance a physical cart-pole system in just 17.5 seconds of interactions. Nevertheless, PILCO's scalability is limited by the Gaussian processes model, whose query time has a cubic complexity with the data size.

Guided Policy Search (GPS): Guided policy search [Levine and Koltun, 2013] essentially performs regularized importance sampled policy update (a model-free policy search method) by computing the analytical gradient of the policy. The main innovation of GPS is that for the policy update, the sample trajectories are drawn using iLQG controller policy π_g, a model predictive control (MPC) approach. To perform the MPC, GPS uses fitted local time-varying dynamics models [Levine and Abbeel, 2014] around the last trajectories. Thus, GPS is a hybrid approach where the model-based iLQG policies π_g are used to perform the model-free update of the parameterized policy π_θ (e.g., a neural network). The iLQG ensures that the new trajectories are always better than the old trajectories and stays closer to the last trajectories. Alternatively, it can be seen as a distillation of good iLQG policies into the parameterized policy π_θ using behavioral cloning.

Although GPS is more-data efficient than model-free approaches, it is less data-efficient than model-based policy search algorithms based on global model learning such as PILCO [Deisenroth and Rasmussen, 2011], provided a reasonable global model can be learned using the data. However, when the dynamics contain discontinuity, such as contacts, GPS is not only data-efficient but also has better asymptotic performance than algorithms like PILCO. The reason is that in contact-rich scenarios, it is difficult to learn a reasonable global dynamics model using the data on which the algorithms like PILCO rely [Levine and Abbeel, 2014].

Direct policy search

Similar to model-free direct policy search in section 2.3.6, here also policy search is seen as an optimization of an objective function which is estimated using the learned dynamical model. Direct policy search does not consider the trajectory information (e.g., actions, state transitions and reward information at every step). These algorithms require only the accumulated reward at the end of the episode only. Thus, these algorithms do not require any specific property on the type of the model, the policy representation, or the reward function. In this category, there are primarily two different approaches: (1) Evolutionary strategies, and (2) Bayesian optimization. For instance, neuroevolution has been used to find policies for hovering helicopters (in simulation) using model-based policy search framework [Koppejan and Whiteson, 2009]. Similarly, Chatzilygeroudis

et al. [2017] and Chatzilygeroudis and Mouret [2018] used the evolutionary strategy CMA-ES [Hansen and Ostermeier, 1996] for policy optimization in many robot learning tasks with Gaussian processes modeling of the dynamics. Another work [Ha and Schmidhuber, 2018] learned a recurrent model of the world (in simulation), which was then used to for policy search using CMA-ES. In Bayesian optimization, [Wilson et al., 2014] used an iteratively learned dynamical model to find policies for simulated robots in a data-efficient manner.

Direct policy search approaches are generally flexible and faster than gradient-based approaches due to ease of parallelization [Chatzilygeroudis et al., 2017]. For instance, Chatzilygeroudis et al. [2017] proposed a model-based policy search algorithm called Black-DROPS and showed that when several computational cores are available, an evolutionary strategy can be much faster than the gradient-based optimization algorithm PILCO [Deisenroth and Rasmussen, 2011]. Black-DROPS iteratively learns a probabilistic dynamical model of the system using Gaussian process regression [Rasmussen and Williams, 2006] and optimizes the policy on this model. At its core, Black-DROPS exploits the implicit averaging property [Jin and Branke, 2005] of the population and rank-based optimizers, like CMA-ES [Hansen and Ostermeier, 1996], in order to perform sampling-based evaluation of the policies efficiently. The key idea in Black-DROPS is that the policy optimization on a probabilistic dynamical model can be seen as a noisy optimization problem. Thus, with CMA-ES, there is no need to estimate the expected long-term reward with multiple samples, as this expectation is implicitly computed by the optimizer. In more detail, instead of performing deterministic long-term predictions, like PILCO, or Monte-Carlo evaluation, like PEGASUS [Ng and Jordan, 2000], Black-DROPS stochastically generates trajectories, but considers that each of these trajectories (or rollouts) is a measurement of a function $G(\theta)$, which is the actual function $\hat{J}(\theta)$ perturbed by a noise $N(\theta)$:

$$G(\theta) = \hat{J}(\theta) + N(\theta) \tag{2.41}$$

It is easy to verify that maximizing $\mathbb{E}[G(\theta)]$ is equivalent to maximizing $\hat{J}(\theta)$, when $\mathbb{E}[N(\theta)] = \text{constant}$. The Black-DROPS uses a recent variant of CMA-ES (i.e., one of the most successful algorithms for optimizing noisy and black-box functions [Hansen et al., 2009], Hansen [2009a]) that combines random perturbations with re-evaluation for uncertainty handling along with restart strategies for better exploration [Auger and Hansen, 2005]. While Black-DROPS has the same data-efficiency as PILCO, it has the added benefit of being able to exploit multi-core architectures, thus, greatly reducing the computation time. Similar to most Monte-Carlo methods, Black-DROPS is a purely black-box model-based policy search algorithm; i.e., one can swap the model types, reward functions and/or initialization procedure with minimal effort. BlackDROPS was able to learn in less than 20 seconds of interaction time to solve the cartpole swing-up task as well as to control a physical 4-DOF physical manipulator in less than 5-6 episodes.

Information-theoretic

Information-theoretic approaches for policy optimization utilize the concept of entropy for stable policy updates using the last observed trajectories. More specifically, these approaches try to set an upper bound on the *relative entropy* between the old policy and the new policy so that policy update does not change the old policy aggressively. In this context, when we say policy, we refer to the data or trajectory distribution generated by the policy. Information-theoretic approaches try to keep the new trajectories closer to the last trajectory distribution since the gradient for the policy parameters is estimated based on the last trajectories and it is expected to be valid only around that trajectory distribution. It is to be noted here that constraining the policy update in the data distribution is completely different from constraining the policy update directly on its parameters. The reason is that a small change in the parameters of the policy might produce a significant change in the data or trajectory distribution under the new policy.

ME-TRPO: Kurutach et al. [2018] proposed *Model Ensemble Trust Region Policy Optimization (ME-TRPO)*, which extended model-free policy search algorithm TRPO [Schulman et al., 2015] to model-based policy search problem. ME-TRPO uses an ensemble of neural networks to model the dynamics of the system. The ensemble is trained using mean-squared-error loss for the data collected from the trials on the system. In the policy improvement step, the policy is updated using TRPO, based on the imaginary or simulated experience generated on the learned dynamics model. During policy evaluation, at every step, ME-TRPO randomly chooses a model from the ensemble to predict the next state given the current state and action. This restricts the policy from overfitting to any single model during an episode, leading to more stable learning. ME-TRPO demonstrated similar performance as the asymptotic performance of the state-of-the-art model-free algorithms such as PPO, DDPG, TRPO, etc. on many simulated high dimensional locomotion tasks while using approximately 100 times fewer data than the model-free baselines.

GPREPS: Kupcsik et al. [2017] proposed another information-theoretic model-based policy search algorithm called *Gaussian Process Relative Entropy Policy Search (GPREPS)* which essentially utilizes Relative Entropy Policy Search (REPS) [Peters et al., 2010] algorithm for contextual policy optimization. GPREPS learns the dynamics model from the observed data using Gaussian Process regression [Rasmussen and Williams, 2006] and it does not require any assumption on the parametrization of the policy or the reward function. GPREPS generates the artificial trajectories on the GP model to compute the expected return and updates the policy using REPS so that the new policy stays closer to the old policy according to the model. The authors were able to train a robotic arm to return a table tennis ball in simulation after 150 evaluations on the real system using GPREPS algorithm, whereas model-free REPS took 4000 evaluations to attain the same performance.

M-PGPE: Another model-based policy search algorithm based on the information-theoretic approach was proposed in [Tangkaratt et al., 2014] called *Model-Based Policy Gradients with Parameter-Based Exploration (M-PGPE)*. As the name suggests, M-PGPE utilizes model-free policy search algorithm

PGPE [Sehnke et al., 2010] to optimize the policy on a learned dynamical model of the system. M-PGPE learns the forward dynamical model of the system using "least-squares conditional density estimation" (LSCDE) [Sugiyama et al., 2010], which has some superior properties compared to Gaussian Processes. For instance, it can directly handle multi-dimensional inputs and outputs, has high numerical stability, and its solution can be analytically and efficiently computed just by solving a system of linear equations. Similar to PGPE, in M-PGPE, deterministic policies are used to suppress irrelevant randomness (action level randomness). The useful stochasticity (trajectory level randomness) is introduced here by drawing policy parameters from a prior distribution parameterized by $\langle \mu, \sigma \rangle$, where each dimension of μ and σ represents the mean and the standard deviation of a univariate Gaussian distribution. Then, instead of policy parameters, hyperparameters $\langle \mu, \sigma \rangle$ are updated using the simulated trajectories drawn from the model following the analytical gradient with respect to the hyperparameters. Using M-PGPE, authors were able to train a humanoid (9-DOF, 18-dimensional state space, 9-dimensional action space) to touch a specified goal within 1000 trials on the real system.

Sampling-based

Sampling-based approaches are those approaches that draw trajectory samples according to the current policy and update the policy without computing any gradient. The advantage of the sampling-based method is that it can be parallelized easily for faster optimization. Broadly, these approaches can be classified in two types: (1) ones that perform stochastic long-term prediction and (2) the others that perform deterministic long-term prediction on the learned dynamical model.

For stochastic long-term prediction, each trajectory τ is sampled using Monte-Carlo roll-out:

$$\tau = (s_0, a_0, r_1, s_1, a_1, \ldots, r_{H-1}, s_{H-1}) \tag{2.42}$$

$$R(\tau) = \sum_{t=0}^{H-1} r_{t+1} \tag{2.43}$$

Where

$$s_0 \sim p(s_0) \quad \text{(Initial state distribution)} \tag{2.44}$$

$$r_{t+1} \sim \hat{p}(r_{t+1}|s_t, a_t, s_{t+1}) \quad \text{(Reward model)} \tag{2.45}$$

$$a_t \sim \pi_\theta(a_t|s_t) \tag{2.46}$$

$$s_{t+1} \sim \hat{p}(s_{t+1}|s_t, a_t) \quad \text{(Dynamics model)} \tag{2.47}$$

Alternatively, instead of sampling the trajectories τ using Monte-Carlo roll-outs, the deterministic approximations, such as linearization [Anderson and Moore, 2012], sigma point methods [Julier and Uhlmann, 2004] or moment matching can also be used to compute the approximate distribution $\hat{p}(\tau|\theta)$ as a Gaussian distribution. For example, PILCO [Deisenroth and Rasmussen,

2011] performs deterministic approximate long-term prediction using moment matching for gradient-based policy optimization.

When a probabilistic dynamical model is available, for stochastic long-term prediction, m Monte-Carlo roll-out trajectories can be carried out on the model for a particular policy π_θ to estimate the expected return $\mathbb{E}_\theta[R(\tau|\theta)] \approx \frac{1}{m}\sum_{i=1}^{m} R(\tau_i|\theta)$ in order to optimize:

$$\theta^* = \arg\max_\theta J(\theta) = \arg\max_\theta \mathbb{E}_\theta[R(\tau|\theta)] \tag{2.48}$$

Unfortunately, this is a difficult stochastic optimization problem since two different evaluations of $\mathbb{E}_\theta[R(\tau|\theta)]$ for the same θ will produce different results. A more efficient way of estimating the long-term expected reward with stochastic trajectories is PEGASUS [Ng and Jordan, 2000]. The basic idea behind PEGASUS is that before starting the policy optimization, one or several random seeds are drawn. Then these fixed seeds are used for the stochastic processes (e.g., the dynamics model, the reward model, the policy) in order to perform the Monte-Carlo roll-outs. Fixing the random seeds this way makes the expected return a determinist function of θ, i.e., $J : \theta \mapsto \mathbb{R}$. Therefore, θ can now be optimized using any hill-climbing algorithm. Using PEGASUS, Ng et al. [2003] were able to train a neural network policy (with 4 tunable parameters) to fly a helicopter with a learned probabilistic model of the dynamics using locally weighted linear regression (LWLR) [Atkeson et al., 1997]. Nevertheless, in the model-based RL framework, PEGASUS has only limited application in the literature with carefully designed policies and a few tunable policy parameters.

There is another class of algorithms which paved their path into reinforcement learning from optimal control theory. One of such policy optimization algorithm is *policy improvement with path integral (PI²)* [Theodorou et al., 2010]. PI² is more specifically a trajectory optimization algorithm which samples trajectories using current policy and update the policy based on the returns it receives from these trajectories. However, unlike policy gradient, it does not compute any gradient to update the policy. PI² uses a deterministic policy $\pi_\theta(s_t)$ parameterized by θ. For every policy update step, it first rolls-out K trajectories, where, at each time-step t of trajectory k, a new policy parameter is sampled from a Gaussian distribution: $\theta_{k,t} \sim \mathcal{N}(\theta, \Sigma)$. Let $\tau_{k,t}$ represents the portion of the k^{th} trajectory from t^{th} steps onwards. Then for each time-step t and trajectory k, the cost-to-come $S_{k,t}$ and probability $P_{k,t}$ is evaluated as:

$$S_{k,t} = \sum_{i=t}^{N-1} cost(s_i, a_i, s_{i+1}) \tag{2.49}$$

$$P_{k,t} = \frac{e^{-\frac{1}{\lambda}S_{k,t}}}{\sum_{k=1}^{K} e^{-\frac{1}{\lambda}S_{k,i}}}, \tag{2.50}$$

Where, λ is a hyperparameter. Then for each time-step t, parameter θ_t^{new} is computed as:

$$\theta_t^{new} = \sum_{k=1}^{K} P_{k,t}\theta_{k,t} \tag{2.51}$$

Finally, new policy parameter θ^{new} is computed by temporal averaging of θ_t^{new}:

$$\theta^{new} = \frac{\sum_{t=0}^{N-1}(N-t)\theta_t^{new}}{\sum_{t=0}^{N-1}(N-i)} \tag{2.52}$$

The above steps are performed iteratively until the expected cost converges. PI^2 is one of the most efficient, numerically robust, and easy to implement algorithms for RL based on trajectory roll-outs [Theodorou et al., 2010]. PI^2 can be conceived as model-based, semi-model-based, or even model-free, depending upon how the learning problem is structured. For example, Pan et al. [2015] proposed a sample efficient path integral control (gradient-based) approach using learned GP dynamics model and show that this approach is more data-efficient than PILCO [Deisenroth and Rasmussen, 2011] in cart-pole swing-up and double-pendulum-on-cart tasks. Another path integral based approach was presented in [Williams et al., 2016] called model predictive path integral (MPPI) control that directly optimizes the future action trajectory from the current state using the forward dynamics model of the system. MPPI minimizes the relative entropy between sampled trajectories (from the model) and the new optimized trajectory so that the new distribution remains closer to the old distribution.

Another simple and more computationally efficient sampling-based approach for policy optimization as well as for model-based control is *cross entropy method (CEM)* [Mannor et al., 2003]. Given a n-dimensional policy parameter vector θ and a cost function $C(\tau_\theta)$ for trajectory τ_θ sampled using policy parameters θ, the Cross-Entropy Method (CEM) performs the following steps to optimize the policy:

1. **Sample:** Simulate K trajectories on the model using $\theta_k \sim N(\theta, \Sigma)$ where $k = 1, ..., K$.

2. **Sort:** $\theta_{k=1,...,K} \leftarrow$ `sort` $\theta_{k=1,...,K}$ based on trajectory cost $C(\tau_{\theta_k})$ in ascending order.

3. **Update:** $\theta^{new} = \sum_{k=1}^{K_e} \frac{1}{K_e}\theta_k$ and $\Sigma^{new} = \sum_{k=1}^{K_e} \frac{1}{K_e}(\theta_k - \theta)(\theta_k - \theta)^T$, where K_e is the elite size, $K_e < K$

4. **Iterate:** Repeat the steps until cost converges.

CEM has been used in some recent model-based RL algorithms [Chua et al., 2018; Nagabandi et al., 2019] to optimize the sequence of action $a_{t:t+H}$ at every time-step t (i.e., model predictive control) together with deep neural-network-based dynamics model of the system. Unlike episodic policy optimization, these approaches enable online learning with iteratively learned dynamics model. In

particular, Chua et al. [2018] proposed an MBPS algorithm called *probabilistic ensembles with trajectory sampling (PETS)* that samples trajectories of actions from an ensemble of neural networks and optimizes the action trajectories using CEM. Once optimized, the first action of the sequence is applied on the robot. Another advantage of such model predictive control approach using learned dynamical model is that it is less dependent upon accurate long-term predictions as actions are re-optimized at every time-step.

When action space is small, then sampling random action trajectories sampled from a uniform distribution, evaluating them on the model and applying the first action from the best sequence can be an alternative to CEM. Nagabandi et al. [2018] used such a random shooting approach in MPC framework with a neural network dynamics model for problems with relatively high dimensional actions.

2.5 Data-efficient Robot Learning with Priors

So far, we have discussed various policy search approaches, especially in the context of robotics, to solve the reinforcement learning problem. We have seen that model-based reinforcement learning algorithms, more specifically, model-based policy search algorithms are more data-efficient and hence optimize the policy with significantly less interaction time with the real robot. However, for complex robots, such as a hexapod robot or a humanoid robot, learning an effective forward dynamical model from the data will still require a lot of trials on the real robot. In fact, the dynamical model is even more difficult to learn if the robot has discontinuous dynamics due to contacts. Similar to the dynamics model, it is often hard to learn a policy for complex tasks in just a few trials. Learning becomes even more difficult when the policy is represented by function approximators such as a neural network with a lot of parameters.

Slow learning of the dynamics model and the policy restrict the application of reinforcement learning in robotics, where the robot has to adapt to a new situation (e.g., new terrain condition, broken joints) online and accomplish the mission. Interestingly, unlike robots, animals can learn new skills and adapt to a new situation (such as uneven terrain, broken limbs) within minutes, if not seconds, by effectively utilizing past experiences or priors. Likewise, reinforcement learning in robotics can be made more data-efficient using priors (Figure 2.3). In Bayesian statistics, the word "prior" refers to the prior probability distribution that is multiplied by the likelihood of the data and then normalized to compute the posterior. In this way, priors represent knowledge, before taking into account the data, as probability distributions. However, here we use the word prior more broadly. In our context, the term "prior" represents the prior knowledge about the reinforcement learning problem that is available before taking into account the data from a specific problem instance. We do not restrict ourselves to represent these priors in the form of probability distributions. Priors can be utilized in many different ways in reinforcement learning, depending upon where the prior is inserted in the learning framework. We classify these methods mainly into 4 types (Fig. 2.3):

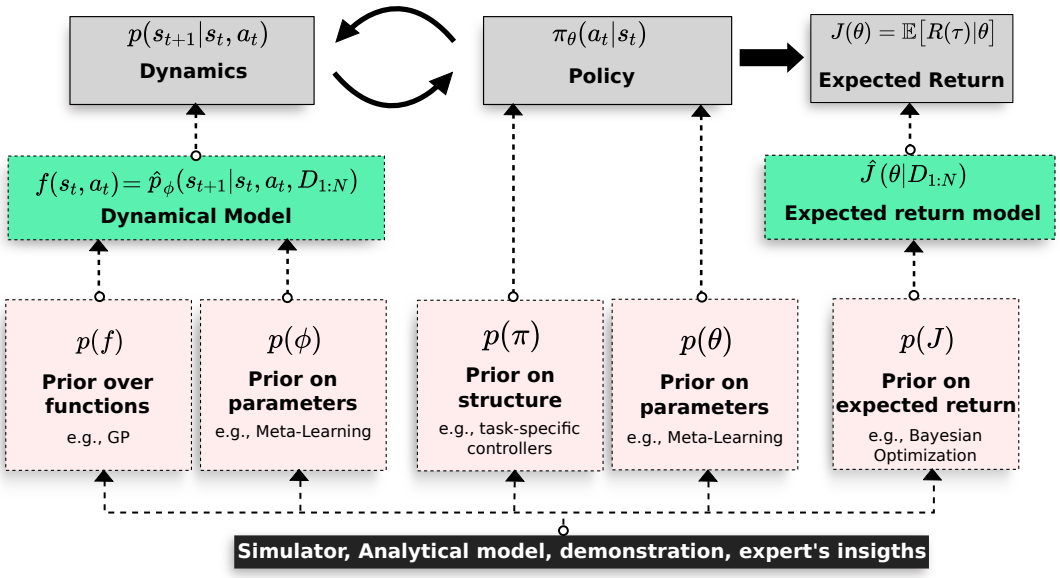


FIGURE 2.3: Overview of priors used in RL for data-efficiency (adapted from [Chatzilygeroudis et al., 2020]). Priors can be used on the dynamical model, the policy or on the expected return.

1. Priors on the dynamical model
2. Priors on the policy
3. Priors on the objective function model
4. Hybrid approaches

The hybrid approaches include methods that use priors on more than one stages of learning. For instance, repertoire-based learning [Chatzilygeroudis et al., 2018] uses policies optimized in the simulator (i.e., prior on the policy), and the corresponding outcomes as prior for a GP model (i.e., priors on the dynamical model). Similarly, [Cully et al., 2015] uses policies optimized in the simulation (i.e., priors on the policy), and their performances/rewards as prior mean-function for the GP model (i.e., priors on the objective function model). In this chapter, we discuss only about repertoire-based priors in robot learning within the hybrid approaches (section 2.5.4).

2.5.1 Priors on the dynamical model

To accelerate the learning process and thereby increase the data-efficiency, many recent papers proposed to leverage prior knowledge about the system dynamics, such as using a known but low fidelity simulator or a parametric mathematical model of the dynamics. In traditional robotics, data-efficiency is achieved by simply identifying the parameters of such mathematical models using the observed data from the real robot [Hollerbach et al., 2016]. In more recent approaches [Chatzilygeroudis and Mouret, 2018; Cutler and How, 2015], a parametric [Chatzilygeroudis and Mouret, 2018] or fixed [Cutler and How, 2015]

model is "corrected" with a non-parametric model to capture potentially non-linear effects. In this section, we will discuss three different types of priors that can be used to learn a dynamical model of the system with only a few observations.

Generic Robotic Priors

This is based on the prior knowledge about the structure of interaction of robotic systems with the physical world [Jonschkowski and Brock, 2015; Lesort et al., 2018]. By exploiting this structure in learning, the robot can obtain an effective representation of its state, potentially in a lower-dimensional space, by learning a state transformation function $\hat{s}_t = T_\phi(s_t)$ (parameterized by ϕ) so that it is consistent with the aspect of physics relevant to the learning problem. It should be noted here that we have categorized the generic robotic priors as "priors on the dynamical model" (prior over functions) (Fig. 2.3) as it uses the knowledge about how actions and states are correlated in a physical system (i.e., the dynamics function) to learn a lower-dimensional representation of the states for the dynamical model. Jonschkowski and Brock [2015] identified 5 of such priors in robotics:

- **Simplicity prior:** *For a given task, only a small number of world properties are relevant.* This prior suggests that for a given reinforcement learning problem, it is possible to reduce the full state in a lower-dimensional space and thus, reduce the complexity of the problem for faster learning.

- **Temporal coherence prior:** *Task-relevant properties of the world changes gradually over time.* This prior is related to Newton's first law of motion. Physical objects have inertia and change their velocity only gradually as a result of external forces. This can be represented as a loss function: $L_{temp}(\phi) = \mathbb{E}\Big[\|\hat{s}_{t+1} - \hat{s}_t\|^2\Big]$

- **Proportionality prior:** *The amount of change in task relevant properties resulting from an action is proportional to the magnitude of the action..* This results from the Newton's second law of motion: acceleration is proportional to the applied force. The corresponding loss function is $L_{prop}(\phi) = \mathbb{E}\Big[(\|\Delta\hat{s}_{t_1}\| - \|\Delta\hat{s}_{t_2}\|)^2|_{a_1=a_2}\Big]$

- **Causality Loss:** *Task-relevant properties, together with action, determine the reward.* Two situations at time t_1 and t_2 must be dissimilar if the robot received different rewards r_{t_1+1} and r_{t_2+1} in the following time-step, even though it has perform the same action $a_{t_1} = a_{t_2}$. The loss function is given as: $L_{caus}(\phi) = \mathbb{E}\Big[e^{-\|\hat{s}_{t_2} - \hat{s}_{t_1}\|^2}|_{a_{t_1}=a_{t_2}, r_{t_1+1}\neq r_{t_2+1}}\Big]$.

- **Repeatability prior:** *The task-relevant properties and the action together determine the resulting change is these properties.* If state representations $\hat{s}_{t_1}$ and $\hat{s}_{t_2}$ are same, then the state change cased of the same action should be same. The corresponding loss function is $L_{rep}(\phi) = \mathbb{E}\Big[e^{-\|\hat{s}_{t_2} - \hat{s}_{t_1}\|^2}\|\Delta\hat{s}_{t_2} - \Delta\hat{s}_{t_1}\|^2|_{a_{t_1}=a_{t_2}}\Big]$

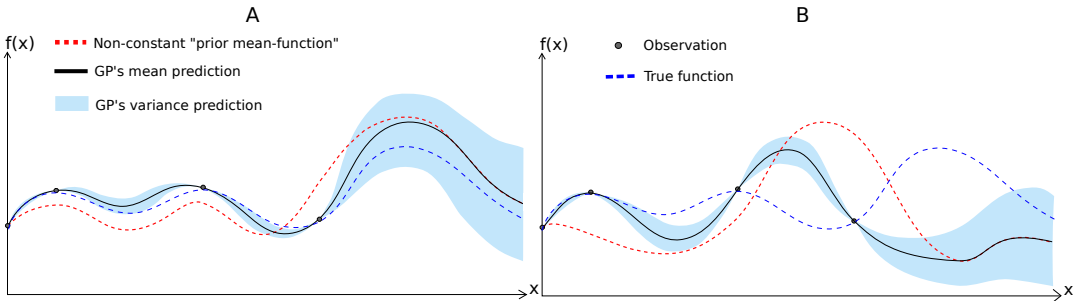


FIGURE 2.4: Gaussian processes regression with non-constant prior - Plots show how the model fitting is impacted by the selection of prior mean-function. On the left (plot A), the prior mean function is more similar to the true underlying function that the model tries to fit with 4 given data points. On the right (plot B), the prior mean-function is very different from the true underlying function. As a result, in plot A, the model fitting is very close to the true function. However, for plot B, due to selection of "wrong prior", model fitting is far from the true function.

Optimizing these losses altogether will produce a state transformation model that is consistent with the nature of the physical interactions of robots in general. The dynamical model learned on this lower-dimensional state-space will require fewer data compared to the full state-space model and thus, improving the data-efficiency in model-based reinforcement learning.

Gaussian Process Model with Non-Constant Prior

Many model-based policy search algorithms leverage Gaussian processes (GP) [Rasmussen and Williams, 2006; Chatzilygeroudis et al., 2020; Deisenroth et al., 2013] as data-driven models not only because they work well with a few data (by predicting the uncertainty), but also because it is easy to introduce Bayesian priors (easier than in neural networks) in the model learning. This can be seen as prior over the dynamical model (prior over the function) (Fig. 2.3). A GP is an extension of multivariate Gaussian distribution to an infinite-dimension stochastic process for which any finite dimensions will be a Gaussian distribution [Rasmussen and Williams, 2006]. It is a distribution over functions, specified by mean function $\mu(\cdot)$ and covariance function $k(\cdot, \cdot)$:

$$f(\mathbf{x}) \sim GP(\mu(\mathbf{x}), k(\mathbf{x}, \mathbf{x}')) \tag{2.53}$$

As mentioned before, compared to neural networks, a GP can easily include prior knowledge about the underlying function [Rasmussen and Williams, 2006]. In particular, we can provide a prior "mean-function" to the GP model, which is our prior belief about the prediction when no data is available to train the model (Figure 2.4). If $\mu(\mathbf{x})$ is the prediction mean and $\sigma^2(\mathbf{x})$ is the prediction variance of a GP model for any input $\mathbf{x}$, $M(\mathbf{x})$ is the prior belief about the prediction mean of the model for the same input $\mathbf{x}$, σ_n^2 is the prior noise and $D_{1:t}$ is the set of t observations, then the GP is computed as follows:

$$P(f(\mathbf{x})|D_{1:t}) = \mathcal{N}(\mu(\mathbf{x}), \sigma^2(\mathbf{x})) \tag{2.54}$$

where, (2.55)

$$\mu(\mathbf{x}) = M(\mathbf{x}) + \boldsymbol{k}^T(\mathbf{K} + \sigma_n^2 I)^{-1}(D_{1:t} - M(\mathbf{x}_{1:t})) \tag{2.56}$$

$$\sigma^2(\mathbf{x}) = k(\mathbf{x}, \mathbf{x}) - \boldsymbol{k}^T(\mathbf{K} + \sigma_n^2 I)\boldsymbol{k} \tag{2.57}$$

Where, $\mathbf{K}$ is the kernel matrix with entries $\mathbf{K}[i, j] = k(\mathbf{x}_i, \mathbf{x}_j)$ and $\boldsymbol{k} = k(D_{1:t}, \mathbf{x})$. As mentioned above, here $k(\cdot, \cdot)$ is the covariance function or the kernel function.

The use of GP with non-constant prior is not limited to learning the dynamical model. In Bayesian optimization applied to robot learning, many recent works use non-constant priors coming from simulation to model the cost function using GP [Cully et al., 2015; Papaspyros et al., 2016; Pautrat et al., 2018]. In particular, Pautrat et al. [2018] uses several repertoires of policies evolved for different situations that perform the same task, but in different ways (i.e., different behaviors). Then it uses the performance scores stored in each of the repertoires as prior mean-functions to the GP to learn the performance function for Bayesian optimization on the real robot. In effect, it learns as many models as the number of repertoires. To select a policy, a novel acquisition function called MLEI (Most Likely Expected Improvement) is used, which considers the likelihood of the prior being close to the reality while computing expected improvement of performance for a policy.

Many recent work also used priors from a simulator to learn a "residual model" with GP, i.e., the difference between the simulated and real robot instead of learning the system model from scratch. For example, model-based policy search algorithm like PILCO [Deisenroth and Rasmussen, 2011] or BlackDROPS [Chatzilygeroudis et al., 2017] can be combined with simulated priors and learn to control a cart-pole in 2 to 5 trials [Cutler and How, 2015; Saveriano et al., 2017; Chatzilygeroudis and Mouret, 2018].

Meta-Learning of Dynamical Model

Meta learning approaches assume that the previous meta-training tasks and the new tasks are drawn from the same task distribution $p(\mathcal{T})$ and these tasks share a common structure or similarity which can be exploited for fast learning. Gradient-based meta-learning approach such as MAML [Finn et al., 2017] tries to find out the initial parameters $\theta_\mathbf{0}$ for a differentiable parametric model so that taking only a few gradient descent steps from the initial parameters $\theta_\mathbf{0}$ produce an effective generalization to the new learning task. More concretely, MAML tries to find an initial set of parameters θ such that for a randomly sampled task $\mathcal{T}$ with corresponding loss function $\mathcal{L}_\mathcal{T}$, the learner will have low loss after k updates:

$$\arg\min_\theta \mathbb{E}_\mathcal{T}\Big[\mathcal{L}_\mathcal{T}(U_\mathcal{T}^k(\theta))\Big] \tag{2.58}$$

where $U_{\mathcal{T}}^{k}(\theta)$ is the the update rule (e.g., gradient descent updates) that updates the parameters θ for k times using the data sampled from $\mathcal{T}$. MAML optimizes this problem with stochastic gradient descent as:

$$\theta := \theta - \alpha \nabla_{\theta} \mathcal{L}_{\mathcal{T}}(U_{\mathcal{T}}^{k}(\theta)) \tag{2.59}$$

$$:= \theta - \alpha \nabla_{\tilde{\theta}} \mathcal{L}_{\mathcal{T}}(\tilde{\theta}) \nabla_{\theta} U_{\mathcal{T}}^{k}(\theta), \text{ where } \tilde{\theta} = U_{\mathcal{T}}^{k}(\theta) \tag{2.60}$$

where, $\alpha > 0$ is the learning rate. To simplify the computation of equation 2.60, first-order MAML approximates $U_{\mathcal{T}}^{k}(\theta)$ as a constant update of θ as $U_{\mathcal{T}}^{k}(\theta) = \theta + \mathbf{c}$ (where $\mathbf{c}$ is a constant). This approximation simplifies equation 2.60 as:

$$\theta := \theta - \alpha \nabla_{\tilde{\theta}} \mathcal{L}_{\mathcal{T}}(\tilde{\theta}), \text{ where } \tilde{\theta} = U_{\mathcal{T}}^{k}(\theta) = \theta + \mathbf{c} \tag{2.61}$$

Another first-order meta learning approach called Reptile [Nichol et al., 2018] tries to find a solution θ that is close (in Euclidean distance) to each task $\mathcal{T}$'s manifold of optimal solutions. To achieve this Reptile treats $U_{\mathcal{T}}^{k}(\theta) - \theta$ as a gradient which gives the SGD update of θ as:

$$\theta := \theta + \beta (U_{\mathcal{T}}^{k}(\theta) - \theta) \tag{2.62}$$

where, $\beta > 0$ is the learning rate. Recently, Nagabandi et al. [2019] applied MAML to online learning scenarios by iteratively meta-training the dynamics model from the past data, and from that, learning the dynamics model using a few recent observations. They were able to show rapid online adaptation to damages in relatively complex robots both in the simulation as well as in the real world (a hexapod with 2D actions). However, the meta-learning approach described above assumes that the meta-training situations and real-world situations are drawn from the same distribution of situations and all these tasks share a common structure or similarity in the dynamics. In addition, these approaches try to find a single dynamics prior model and use it for future adaptation. As a result, when there are strong dissimilarities (in dynamics) among the meta-training situations, then it is less likely to find a set of parameters for the model that can be adapted with a small number of data-points to obtain an effective global model.

2.5.2 Priors on the policy

Typically, there are two different ways of adding priors on the policy to make the policy optimization faster with reinforcement learning:

1. Designing a task-specific policy with a only a few tunable parameters.
2. Initializing the policy parameters in such a way that during learning only small adjustments are required.

Task-specific policies are generally controllers designed for a specific task. The controllers are parameterized by a small set of parameters that can be adjusted to produce different control behaviors. For example, in the ball capturing task by an Aibo robot [Fidelman and Stone, 2004], the policy is a controller that has 4 parameters to specify 4 different behaviors: slowdown distance of the robot from the ball, rate of slowdown, ball distance and angle to start capturing motion. Similarly, in [Cully et al., 2015] 12 square-wave functions were used to specify the time-varying joint angles of a hexapod robot, where 36 controller parameters define the amplitude, phase, and duty-cycle of these functions. Although such task-specific policies reduce the search space of the problem, they often become useless when the task changes for the same robot.

Strategy optimization [Yu et al., 2019a] is another way of setting a prior on the policy. In this approach, a task (μ) conditioned policy $\pi_\theta(a_t|s_t, \mu)$ is optimized for several tasks in simulation so that expected return for these tasks is maximized. As a result, the optimized policy parameters θ^* capture the global similarity among the training tasks. During learning on the real robot, optimization is done on the task vector μ so that expected return can be maximized:

$$\mu^* = \arg\max_{\mu} \mathbb{E}\Big[\sum_{t=0}^{H-1} r(s_t, a_t, s_{t+1})|_{\pi_{\theta^*,\mu}}\Big] \tag{2.63}$$

Since task vector μ has a much smaller dimension compared to the neural network policy parameters, the search-space for the optimization becomes smaller, making it much easier to optimize. Using a similar approach, Yu et al. [2019b] were able to train a quadruped to walk on a slope with only 15 trials. Here, the policy was pre-trained in simulation for various scenarios (except for the inclined surface) before strategy optimization on the real robot.

Gradient-based meta-learning approach such as MAML [Finn et al., 2017] discussed in subsection 2.5.1 can also be used to pre-train a differentiable policy (e.g., a neural network policy) for various tasks so that the policy network can be adapted to similar tasks with only a few gradient ascent steps. In the context of policy, MAML tries to find a set of policy parameters so that when the policy is initialized with those parameters, only a few gradient ascent steps are required to optimize the policy for any task in the training distribution. Clavera et al. [2018] combined model-based policy search with meta-policy optimization for data-efficient learning on real robots. The key idea here is to learn an ensemble of dynamical models using the past observation from the robot with deep neural networks and then meta-train the policy (similar to MAML) considering each model to be a separate task. In this approach, their meta-object is to optimize the initial parameters of the policy so that the policy can be adapted to any of the tasks (i.e., the dynamics models) with just one gradient ascent step.

Expert demonstrations can also be used as priors for policy. As already discussed, to ensure data-efficient policy optimization, it is important that the initial policy parameters are closer to the optimal policy parameters. A general approach called imitation learning [Billard et al., 2008a; Argall et al., 2009; Osa et al., 2018] achieves this using the trajectories collected from a human

expert to solve the task. The most common method to train a policy from the expert demonstrations is behavioral cloning, where the policy parameters are trained directly in a supervised manner on the state-action pairs obtained from the demonstrations. For such supervised learning with robotic manipulators, in general, trajectory-based policies are well suited for typical classes or tasks, such as point-to-point movement or repetitive movements. The most widely used trajectory-based policy is Dynamical Movement Primitives (DMP) [Ijspeert et al., 2003; Schaal et al., 2007].

The key principle of DMPs is to use a linear spring-damper system (equivalent to a standard PD controller) which is modulated by a non-linear forcing function f:

$$\ddot{s} = \underbrace{\tau^2 \alpha_s(\beta_s(s_g - s) - \dot{s})}_{\text{spring-damper}} + \underbrace{\tau^2 f(z)}_{\text{forcing term}} \tag{2.64}$$

$$\dot{z} = -\tau \alpha_z z \qquad \text{(Canonical system)} \tag{2.65}$$

$$f(z) = \frac{\sum_{i=1}^{K} \phi_i(z) w_i}{\sum_{i=1}^{K} \phi_i(z)} z, \quad , \phi_i(z) = e^{-\frac{1}{-\sigma_i^2}(z - c_i)} \tag{2.66}$$

where, s is the joint position, s_g is the goal-position parameter, τ is the time-scaling co-efficient and coefficients α_s and β_s are the spring and damping coefficients. The function f depends on the phase variable z and is constructed by the weighted sum of K basis functions $\phi_{i=1,\ldots,K}$. The phase variable z is set to 1 initially. As the spring-damper system converges to s_g and z (and thus f) converges to 0, the overall system is guaranteed to converge to s_g. Multi-dimensional DMPs are achieved by coupling multiple transformation systems (i.e., spring-damper and the forcing term) with one canonical system. In imitation learning, the parameters of the DMP are trained to match the demonstration trajectories. For example, in the Ball-in-a-cup task [Kober and Peters, 2009], authors first train a DMP based policy from human demonstrations. Then that policy is fine-tuned to solve the task using a reinforcement learning algorithm called PoWER [Kober and Peters, 2009]. Another approach is to provide demonstrations for various tasks and allow the algorithm to generalize the motion for novel tasks [Pervez and Lee, 2018]. However, the effectiveness of such demonstration based priors depends on the quality of the demonstrations. Moreover, often it is not possible to provide such expert demonstrations for many robots (e.g., legged robots, flying robots, etc.).

Transferability approaches [Koos et al., 2012, 2013; Mouret et al., 2013] attempt to transfer a policy learned in simulation to the real robot. This approach involves the simulator of the robot in the loop while learning the policy on the real robot. As the name suggests, the main hypothesis in this approach is that the physics simulators are accurate for some policies, e.g., static gaits, and inaccurate for some others, e.g., highly dynamic gaits. By iteratively testing the optimized policies on the real robot, it is possible to learn a function to predict if the search is constrained to policies that are simulated accurately. Thus, the key idea of the transferability approach is to learn the transferability function, which predicts the accuracy of a simulator given policy parameters.

The transferability function is often easier to learn than the expected return because this is essentially a classification problem (instead of regression). In addition, small errors in the model often have little consequences, because the search is mainly driven by the expected return in the simulation (and not by the transferability optimization). This approach requires only a handful of trials on the physical robot (in most of the experiments, less than 25). However, the main drawback of the transferability approaches is that it can only find policies that perform similarly in simulation and reality (e.g., static gaits versus highly dynamic gaits). These type of algorithms were able to efficiently learn policies for mobile robots to navigate in mazes [Koos et al., 2012] (15 trials on the robot), for a walking quadruped robot [Koos et al., 2012; Koos and Mouret, 2012] (about 10 trials), for learning to walk on humanoid [Oliveira et al., 2013], and for a 6-legged robot that had to learn how to walk despite a damaged leg [Koos et al., 2013] (25 trials). Similar ideas were also applied with QP-based controllers for humanoid robots [Spitz et al., 2017].

2.5.3 Priors on the objective function model

When the objective function (i.e., the mapping from policy parameters to expected return) is unknown, then we can iteratively learn the objective function from the observations collected after every trial. This approach is particularly used in Bayesian optimization (BO) with GP modeling of the objective function [Shahriari et al., 2015]. For any policy parameter θ, the GP model of the objective $J(\theta)$ (i.e., the expected return) will look like as follows:

$$p(\hat{J}(\theta)|D_{1:t},\theta) = \mathcal{N}(\mu_{gp}(\theta), \sigma^2_{gp}(\theta)) \tag{2.67}$$

where,

$$\mu_{gp}(\theta) = R_m(\theta) + \boldsymbol{k}^T(\mathbf{K} + \sigma_n^2 I)^{-1}(D_{1:t} - R_m(\theta_{1:t})) \tag{2.68}$$

$$\sigma^2_{gp}(\theta) = k(\theta,\theta) - \boldsymbol{k}^T(\mathbf{K} + \sigma_n^2 I)\boldsymbol{k} \tag{2.69}$$

Where, $\mathbf{K}$ is the kernel matrix with entries $\mathbf{K}[i,j] = k(\theta_i,\theta_j)$, $\boldsymbol{k} = k(D_{1:t},\theta)$, $k(\cdot,\cdot)$ is the covariance function or the kernel function and $R_m(\cdot)$ is the prior mean-function. Using this model, Bayesian optimization decides which policy θ_{eval} should be evaluated on the real robot by optimizing an *acquisition function*. For the acquisition function, most of the algorithms use the Expected Improvement (EI), Probability of improvement (PI) or Upper Confidence Bound (UCB). Out of these function, UCB is the most simplest and works very well in practice. Using UCB, θ_{eval} can be found by optimizing the following using gradient free algorithms such as DIRECT [Jones et al., 1993] or CMA-ES [Hansen and Ostermeier, 1996]:

$$\theta_{eval} = \arg\max_{\theta} UCB(\theta) = \arg\max_{\theta} \Big[\mu_{gp}(\theta) + \kappa\sigma_{gp}(\theta)\Big], \quad \kappa \geq 0 \tag{2.70}$$

As already discussed in section 2.5.1, a GP model can easily include the prior belief about the underlying function (i.e., $R_m(\cdot)$ in eq. 2.68). Many

recent works used objective function evaluated in simulation as the prior mean function for the GP model. For instance, Cully et al. [2015] optimized a large, diverse set of policies in a low fidelity simulator of the robot and used their corresponding performance as a prior mean function for the GP model of the objective function. To find out the best performing policy on the real (damaged) robot at test time, the Bayesian optimization was performed on the discrete set of policies while updating the GP model (performance model) with new observation. With this approach, authors were able to allow a hexapod robot to find a policy to walk forward in less than two minutes of trials. In another work [Wilson et al., 2014], the authors proposed an approach called Model-Based Bayesian Optimization Algorithm (MBOA), where the prior for the GP model comes from a simple learned dynamical model (e.g., linear model) of the system. To query the expected return of any policy from the GP model, first, the policy is evaluated on the dynamical model to compute the expected return. Then this value is used as the prior mean for the GP model of the objective function. However, the authors showed results only with simple robots in the simulation that learn new policies in a data-efficient manner with this approach. Since BO tries to model the objective function with observed data, learning this model becomes exponentially harder when the dimension of the policy increases. Thus, BO does not scale well for problems with more expressive policies such as deep neural networks.

2.5.4 Repertoire-based prior

The key idea behind this approach is that, first, a large and diverse set of policies are optimized in simulation and stored in a repertoire or reservoir. Then, the algorithm learns to select the most suitable policies from this repertoire to efficiently perform the task on the robot. The repertoire of policies used in this approach can be thought of as a prior on the policy coming from the simulator. Additionally, these algorithms also use the simulated outcomes (i.e., either performance or state-transitions) as a prior mean function for the GP to learn the performance model (objective function model) for Bayesian optimization [Cully et al., 2015; Pautrat et al., 2018] or to learn a GP model of task-space dynamics [Chatzilygeroudis et al., 2018] using observed data from the robot. Typically, there are two different types of such repertoires:

1. **Behavioral repertoire:** Repertoire that contains policies to perform the same task (e.g., a hexapod robot to move forward), but in different ways/behaviors (e.g., hexapod to move forward with different gaits).

2. **Action repertoire:** Repertoire that contains policies to performs different elementary tasks (e.g., moving in different directions and different distances).

One of the most popular algorithms to generate these repertoires is an evolutionary algorithm called Map-Elites [Cully et al., 2015; Mouret and Clune, 2015]. Depending upon the type of repertoire to be evolved, MAP-Elites discretizes either the task-space (if action repertoire) or the behavior space (if

behavioral repertoire) into some regions or cells. In the beginning, MAP-Elites randomly initializes some policies and tests them in simulation to find out their outcome and their performance score. Then these policies are placed in the respective cells along with their performance score. After this initialization, MAP-Elites performs the following three steps iteratively until the maximum number of evaluations are reached:

1. Randomly picks a policy from the repertoire and adds a small random variation to the policy.

2. Simulates the policy to get the outcome and performance score.

3. Inserts the new policy into the corresponding cell based on its outcome if the cell is empty, ord replaces the policy occupying the cell if it has a lower score than the new one. Otherwise, MAP-Elites discards the new policy.

Although MAP-Elites is computationally expensive, it can be parallelized on large clusters to compute the repertoires before deployment of the robot. To improve the space-complexity, more recent algorithms use the CVT variant of MAP-Elites [Vassiliades et al., 2017] that uses Centroidal Voronoi Tesselation (CVT) to discretize the task-space or behavior space into the user-specified number of homogeneous geometric regions. In CVT Map-Elites, the number of cells remains fixed irrespective of the dimensionality of the space, making it scalable to a very high dimensional task-space or behavior space.

Algorithms based on behavioral repertoires generally use Bayesian optimization to find a policy from this repertoire that to produce the desired outcome on the real robot. In [Cully et al., 2015], authors generated a behavioral repertoire (6 dimensional) of policies (36 dimensional) for an intact hexapod robot with 18 joints in simulation. The repertoire contained one high performing policy for potentially each discrete gait of the robot in the behavior space. The performance scores of these policies are based on how far they move the robot in the forward direction were also stored in the repertoire. Then on the real robot (with a broken leg), Bayesian optimization is performed to figure the policy that takes the robot as far as possible in the forward direction. While doing so, the algorithm use performance scores found in simulation as prior mean-function for the GP to model the performance function: $f_{perf} : \mathcal{B} \mapsto \mathbb{R}$, where $\mathcal{B}$ is the behavior/gait space. With this approach, authors were able to find a policy to allow a hexapod robot with a broken leg to walk in the forward direction less than 2 minutes of trials. Pautrat et al. [2018] extended this approach using multiple repertoires where several behavioral repertoires were generated for various situations that the robot might face in reality. Then, using a novel acquisition function called MLEI (Most likely expected improvement), the algorithm not only selects the right prior repertoire to perform Bayesian optimization but also selects the most suitable policy from that repertoire for the real robot. With this approach, authors were able to allow an intact as well as a damaged hexapod robot (both in simulation only) to walk on stairs of various heights in around 5 trials.

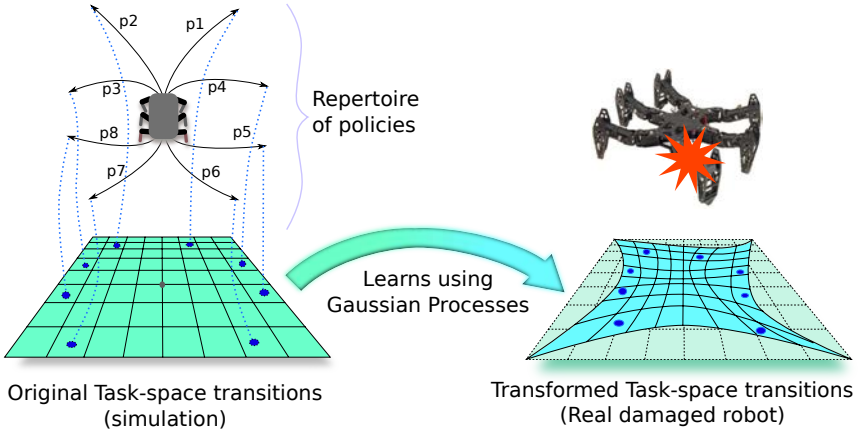


FIGURE 2.5: Repertoire-based learning in robotics: First, a repertoire of elementary policies is evolved for the robot using a known but imperfect simulator. This repertoire associates potentially every discretized task-space (or outcome-space) transitions to unique elementary policies. Then, during deployment of the real robot, a Gaussian process model is learned, which transforms this "prior" task-space in such a way that the outcomes of policies on the real robot match with the new transformed task-space. RTE [Chatzilygeroudis et al., 2018] uses this model with the Monte-Carlo tree search to pick the policies in a sequential manner from the repertoire to solve the target task.

One limitation of the behavioral repertoire is that they are generated to perform only one single task. When the task description changes, the repertoires need to be evolved again in simulation, which might take several hours to days depending upon the complexity of the task as well as the robot and computational resources. Action repertoires, instead, contains elementary policies to perform various elementary tasks, such as walking in different directions [Chatzilygeroudis et al., 2018; Duarte et al., 2018] or drawing different elementary strokes [Cully and Demiris, 2018a] so that these policies can be sequenced with higher-level policy to perform complex tasks. For example, in [Chatzilygeroudis et al., 2018], authors generate an action repertoire of 36-dimensional policies to allow an intact hexapod robot to walk in various directions in simulation. Then using the simulated outcomes (i.e., task-space transitions) of these policies as the prior mean function for a GP model, they learn a model online to map how the outcome of these policies change on the real robot (fig. 2.5). Then, Monte-Carlo Tree Search (MCTS) is performed on that model to select the next policy to be executed on the robot to solve the task as quickly as possible. With this approach, a damaged hexapod robot could learn online to reach a goal by avoiding obstacles in the path. Similarly, Duarte et al. [2018] trained a neural network policy for a hexapod robot to pick these elementary policies sequentially from the repertoire in order to reach the goal by avoiding the obstacles.

2.6 Conclusion

In this chapter, we discussed various approaches to solve the reinforcement learning problem in the context of robotics. From the recent literature, we

can conclude that model-based policy search algorithms are much more data-efficient than the model-free policy search algorithms. Moreover, we have also seen that incorporating prior knowledge on (1) the dynamics model, (2) the policy, or (3) the objective function significantly improves the data-efficiency, allowing the robots to learn new skills in just a few trials. In particular, repertoire-based priors which not only use prior information on the policy but also on the dynamics or the objective function can allow complex robots to learn to adapt to unseen situations within a minute of interaction.

In the next chapter (Chapter 3), we discuss how model-based policy search algorithms based on probabilistic dynamical models can learn policy in a data-efficient manner in sparse reward scenarios. We investigate how this model can be used together with a population-based black-box optimizer to actively explore state-space by generating novel trajectories as well as trajectories that maximize the return.

Chapter 3

Data-efficient Robot Policy Search in Sparse Reward Scenarios

The results and text of this chapter have been partially published in the following article:

Article:

- **Kaushik, R.**, Chatzilygeroudis, K., & Mouret, J. B. (2018). *Multi-objective Model-based Policy Search for Data-efficient Learning with Sparse Rewards*. In **Conference on Robot Learning** (pp. 839-855) [Kaushik et al., 2018].

Other contributors:

- Konstantinos Chatzilygeroudis (PhD Student)
- Jean-Baptiste Mouret (Thesis supervisor)

Author contributions:

- **RK** and JBM organized the study. **RK** and KC wrote the code and conducted the experiments. **RK**, KC and JBM analyzed the results and wrote the paper.

3.1 Introduction

Following our discussion in the introduction and the background chapter, it is clear that model-based policy search algorithms can learn policies in a few minutes of interaction time, at the expense of a significant computation time between episodes. For instance, PILCO [Deisenroth et al., 2015] and Black-DROPS [Chatzilygeroudis et al., 2017; Chatzilygeroudis and Mouret, 2018] can learn a policy to balance a cart-pole or a policy to control a 5-DOF manipulator in less than 40-60 seconds of interaction time. However, these algorithms implicitly assume that most states can be associated with a positive or negative reward, as this is the case when the objective is to balance a pole or to follow a trajectory with a manipulator.

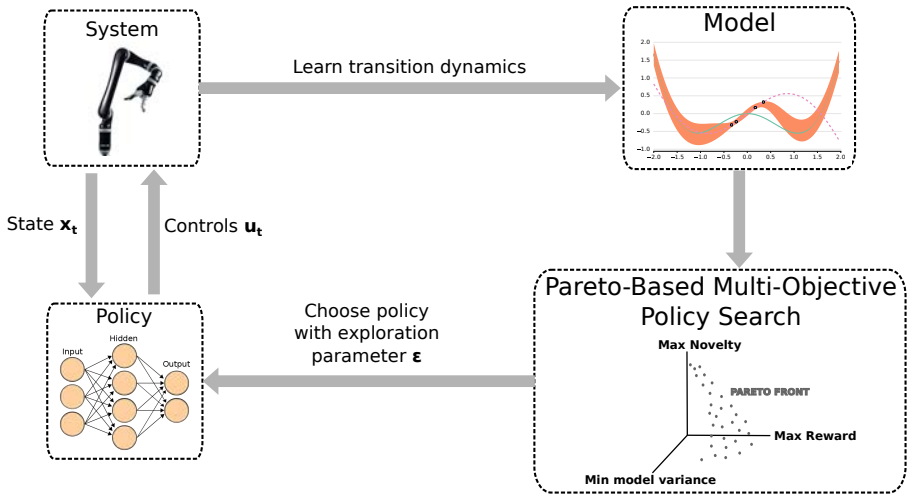


FIGURE 3.1: Overview of the Multi-DEX algorithm: The robot interacts with the environment with a policy in an episodic scheme; after each episode, Multi-DEX learns a GP model of the dynamics of the system and then performs multi-objective policy search to obtain a set of Pareto-optimal policies based on novelty, cumulative reward, and model variance; last, a policy is selected from the Pareto set based on an exploration parameter ε and executed on the robot to collect new data.

While this assumption often holds in control tasks, rewards are much more sparse in many interesting learning scenarios: typically, we would like to reward the robot when it successfully achieves the task, and not for all the intermediate steps that we think should lead to success. For example, a robot might need to open a drawer and get rewarded by how much the drawer is open: in most of the state space, the reward is zero because the robot does not even touch the handle, meaning that PILCO or Black-DROPS need to open the drawer purely by chance to start learning a policy. In most realistic scenarios, this is unlikely to happen.

When there is no reward available, the robot should be "curious" and search for "interesting stepping stones". This question is central in reinforcement learning [Kaelbling et al., 1996; Bellemare et al., 2016; Lopes et al., 2012; Andrychowicz et al., 2017; Florensa et al., 2017], developmental robotics [Oudeyer et al., 2007; Gottlieb et al., 2013], and evolutionary robotics [Lehman and Stanley, 2011a; Mouret and Clune, 2015; Mouret, 2011; Doncieux and Mouret, 2014]. Among the most representative ideas, Novelty Search [Lehman and Stanley, 2011a; Mouret, 2011] attempts to find behaviors that are as different as possible from what has been tested before; Self-Adaptive Goal Generation Robust Intelligent Adaptive Curiosity (SAGG-RIAC) selects random target states and searches for policies/trajectories to reach them; or Adaptive Curiosity pushes the robot towards situations in which it maximizes its learning progress [Oudeyer et al., 2007, 2005]. Unfortunately, all these approaches require thousands (often more) learning episodes; besides, they often focus solely on the exploration and often disregard that the robot might have a task to fulfill (and not only explore its body and the environment).

Our main insight is that we can leverage ideas from Novelty Search and developmental robotics but exploit them *in a learned dynamical model* to keep the

interaction time as low as possible. In essence, we would like to fuse the "novelty-driven" approaches with model-based policy search. At their core, modern model-based policy search algorithms alternate between learning a probabilistic dynamical model and optimizing a policy according to the model (and the uncertainty of the model) [Deisenroth et al., 2015; Chatzilygeroudis et al., 2017]: In this work, we propose to extend this idea to not only optimize for policies that are likely to work on the robot, but also for those that are predicted to be good for exploration. Our second insight is that we can optimize for novelty and cumulative reward using a Pareto-based Multi-Objective Evolutionary Algorithm (MOEA) [Deb, 2001], which simultaneously searches for all the Pareto-optimal trade-offs (instead of a single optimal solution when using an aggregated objective.

More concretely, we propose a novel model-based policy search algorithm, Multi-DEX (Multi-objective Data-Efficient eXploration), that efficiently explores the task space by framing the policy search problem as a multi-objective optimization problem with three objectives (Fig. 3.1): (1) generate maximally novel state trajectories (2) maximize the cumulative reward and (3) keep the system in state-space regions for which the model is as accurate as possible. This optimization is performed with a Pareto-based multi-objective algorithm [Deb, 2001] after each episode, once the dynamical model is updated with the new data. Thanks to the learned probabilistic dynamical model, we show that our proposed approach is highly data-efficient and typically needs only a few minutes of interaction time with the actual system to solve tasks with rare state transitions and sparse rewards. We compare the performance of Multi-DEX with recent algorithms for sparse reward scenarios (VIME [Houthooft et al., 2016] and GEP-PG [Colas et al., 2018]), a state-of-the-art model-free policy search algorithm (TRPO [Schulman et al., 2015]), a state-of-the-art black-box optimizer (CMA-ES [Hansen, 2006]), and a model-based policy search algorithm (Black-DROPS [Chatzilygeroudis et al., 2017]) in a sequential goal reaching task and in a drawer opening task; we show that Multi-DEX solves both tasks in order of magnitude less interaction time than the others.

3.2 Problem Formulation

We consider the following dynamical systems of the form:

$$\mathbf{x}_{t+1} = \mathbf{x}_t + f(\mathbf{x}_t, \mathbf{u}_t) + \mathbf{w} \tag{3.1}$$

with continuous-valued states $\mathbf{x} \in \mathbb{R}^E$ and controls $\mathbf{u} \in \mathbb{R}^F$, i.i.d. Gaussian system noise $\mathbf{w}$, and unknown transition dynamics f. Our goal is to find a *policy* $\pi(\mathbf{u}|\mathbf{x}, \boldsymbol{\theta})$ parameterized by $\boldsymbol{\theta} \in \mathbb{R}^\Theta$ that maximizes the *expected return* when following the policy $\pi_{\boldsymbol{\theta}}$ for T time steps:

$$J_r(\boldsymbol{\theta}) = \mathbb{E}\left[\sum_{t=1}^{T} r(\mathbf{x}_t, \mathbf{u}_t, \mathbf{x}_{t+1}) \middle| \boldsymbol{\theta}\right] \tag{3.2}$$

where $r(\mathbf{x}_t, \mathbf{u}_t, \mathbf{x}_{t+1}) \in \mathbb{R}$ is the reward for being in state $\mathbf{x}_t$, taking action $\mathbf{u}_t$, and reaching state $\mathbf{x}_{t+1}$. We assume that r can be sparse or may have plateaus; *i.e.*, it can be zero for most values of $(\mathbf{x}_t, \mathbf{u}_t, \mathbf{x}_{t+1})$ or have large regions with constant value.

3.3 Approach

3.3.1 Learning system dynamics and reward model

Gaussian Processes (GPs) are widely used for learning system dynamics because of their generalization ability and their ability to compute uncertainty along with the predictions [Deisenroth et al., 2013; Chatzilygeroudis et al., 2017; Deisenroth et al., 2015]. A GP is an extension of multivariate Gaussian distribution to an infinite-dimension stochastic process for which any finite combination of dimensions will be a Gaussian distribution [Rasmussen and Williams, 2006].

We use tuples of states $\mathbf{x}_t$ and actions $\mathbf{u}_t$ (*i.e.*, $\tilde{\mathbf{x}}_t = (\mathbf{x}_t, \mathbf{u}_t) \in \mathbb{R}^{E+F}$) as training inputs for learning the system dynamics. The difference between the current and the next state vector, $\boldsymbol{\Delta}_{\mathbf{x}_t} = \mathbf{x}_{t+1} - \mathbf{x}_t \in \mathbb{R}^E$, is used as corresponding training targets. Then, E independent GPs are used to model each dimension of the difference vector $\boldsymbol{\Delta}_{\mathbf{x}_t}$.

Let $D_{1:t} = \{f(\tilde{\mathbf{x}}_1), ..., f(\tilde{\mathbf{x}}_t)\}$ is a set of observations, we can query the GP at a new input point $\tilde{\mathbf{x}}_*$: $p(\hat{f}(\tilde{\mathbf{x}}_*)|D_{1:t}, \tilde{\mathbf{x}}_*) = \mathcal{N}(\mu(\tilde{\mathbf{x}}_*), \sigma^2(\tilde{\mathbf{x}}_*))$. The mean and variance predictions of this GP are computed using a kernel vector $\boldsymbol{k} = k(D_{1:t}, \tilde{\mathbf{x}}_*)$, and a kernel matrix K, with entries $K_{ij} = k(\tilde{\mathbf{x}}_i, \tilde{\mathbf{x}}_j)$:

$$\mu(\tilde{\mathbf{x}}_*) = \boldsymbol{k}^T K^{-1} D_{1:t} \tag{3.3}$$

$$\sigma^2(\tilde{\mathbf{x}}_*) = k(\tilde{\mathbf{x}}_*, \tilde{\mathbf{x}}_*) - \boldsymbol{k}^T K^{-1} \boldsymbol{k} \tag{3.4}$$

In this work, we use the exponential kernel with automatic relevance determination [Rasmussen and Williams, 2006] and its hyper-parameters as well as the data noise hyper-parameter of the GP are found through Maximum Likelihood Estimation [Rasmussen and Williams, 2006]. We use the Limbo C++11 library for GP regression [Cully et al., 2018]. In many cases, the immediate reward function might be unknown to the algorithm (*i.e.*, not directly computable from the state/action vector). In such situations, we use Random Forest regression [Breiman, 2001] to learn the immediate reward function.

Learning system dynamics with sparse transitions

When the system dynamics have some sparse transitions that strongly affect the cumulative reward, learning the model in a naïve way (*i.e.*, using all the data points) can lead to suboptimal models and the policy search will struggle to find good policies. For example, in our sequential goal reaching task (Sec. 3.4.1) the state includes a Boolean switch that indicates whether the arm passed through the first way-point or not; because only one data point will have transition from *false* to *true* in any episode if the arm passes though the switch, learning a model with the full data (that mostly have no points with the switch transition)

will most probably ignore these transition data points because they are likely to be (rightfully) considered as outliers. In other words, rare transitions have little impact on the likelihood or the fit of the dynamical model (as we additionally optimize the data noise hyper-parameter of the GP), whereas they are critical to leverage the dynamical model to learn a policy.

The intuition here is to have a balanced blend of ordinary trajectories and trajectories with rare transitions (leading to high reward) for model learning. Learning a dynamics model this way will retain information not only about the rare and high rewarding transitions but also about the regions where no reward was observed. As a result, this model can efficiently be used for exploration as well as exploitation. To do this, we maintain two fixed-sized experience buffers, $\mathbb{P}$ and $\mathbb{H}$, where we keep the trajectory data of the episodes for model learning. For each new trajectory executed on the robot, we insert a new observed trajectory into $\mathbb{P}$ in a FIFO fashion if no reward is observed; otherwise, we insert it into $\mathbb{H}$. It is to be noted here that we keep the maximum buffer size of $\mathbb{P}$ equal to the data size of $\mathbb{H}$ to have a uniform blend of "high rewarding" and "no/low rewarding" trajectories for model learning.

3.3.2 Exploration-Exploitation Objectives

The intuition behind Multi-DEX is to combine the data-efficiency of model-based policy search with the ability of diversity in novelty-based search algorithms to encourage exploration in policy search in a data-efficient way. Thus, we frame exploration as an additional objective that promotes policies that will most likely produce novel state trajectories in the real system. To keep the system within the more certain regions of the dynamics model so that prediction error can be avoided as much as possible, we set an additional objective to reduce the prediction variance in the trajectory. The three objectives are typically antagonistic, as the uncertainty is likely to be higher for more novel trajectories, and more novel trajectories are likely to not have a cumulative reward as high as those with the highest cumulative rewards.

Cumulative Reward: To compute the cumulative reward for a policy, we propagate through the learned GP model (with the mean prediction of the GPs, f_μ) of the system for T time steps using the policy, and observe the immediate reward in each step. Then, we aggregate all the immediate rewards to compute the cumulative reward (given the dynamical model) for the episode:

$$\hat{J}_r(\boldsymbol{\theta}) = \sum_{t=1}^{T} r(\mathbf{x}_{t-1}, \mathbf{u}_{t-1}, \mathbf{x}_{t-1} + f_\mu(\mathbf{x}_{t-1}, \mathbf{u}_{t-1})) \tag{3.5}$$
$$\text{where } \mathbf{u}_{t-1} = \pi(\mathbf{x}_{t-1}|\boldsymbol{\theta})$$

As it is evaluated in the model, optimizing this objective means exploiting the learned dynamics to find a policy that is likely to improve the cumulative reward.

Novelty: We compute the novelty score of a policy by comparing its expected state trajectory with the expected state trajectories of already executed

policies. To keep the computation time tractable, we sub-sample them into s_n equally spaced time-steps. We concatenate these state samples into one vector that we call *state trajectory vector*, $\boldsymbol{\beta}$ which represents the "expected state trajectory" of a policy. These vectors are archived in a set $\mathbb{B}$ of fixed size b_n and we re-compute them every time the GP model is updated. When $\mathbb{B}$ is full, we drop the least novel policy. We define the novelty score for any policy $\pi_{\boldsymbol{\theta}}$ as the minimum Euclidean distance of $\boldsymbol{\beta}_{\boldsymbol{\theta}}$ to the state trajectory vectors in $\mathbb{B}$:

$$\hat{J}_n(\boldsymbol{\theta}) = min(||\boldsymbol{\beta}_{\boldsymbol{\theta}} - \boldsymbol{\beta}||^2)_{\forall \boldsymbol{\beta} \in \mathbb{B}} \tag{3.6}$$

This objective is promoting exploration policies that are likely to lead to state trajectories that are different from those already observed.

Cumulative model-variance: We define the cumulative model-variance for a policy $\pi_{\boldsymbol{\theta}}$ as the negative mean of the step-by-step model prediction variances:

$$\hat{J}_{\sigma^2}(\boldsymbol{\theta}) = -\frac{1}{T}\sum_{t=1}^{T}(||\sigma_{\boldsymbol{x_t}}||^2) \tag{3.7}$$

where $\boldsymbol{x_t}$ is given by applying the policy $\pi_{\boldsymbol{\theta}}$ on the model

This objective tries to keep the system close to regions where the model uncertainty is low and thereby avoids prediction error on the real system. Constraining the policy search in the more certain region of the model is important because we want the real system to behave as close as possible to what is predicted by the model using the policy. Thus, without this objective, the optimizer will try to exploit the model imperfection, and the resulting policies might behave differently on the real robot than the predictions.

3.3.3 Multi-Objective Optimization

Most algorithms aggregate objectives using a weighted sum. However, this means that the weights have to be chosen beforehand, which requires a time-consuming trial-and-error process; moreover, some optimal trade-offs cannot be found using a weighted sum [Deb, 2001]. Here, we rely on the multi-objective optimization literature [Deb, 2001] and the Pareto-optimality concept:

Definition 1 *A solution x_1 is said to dominate another solution x_2, if and only if:*

1. *the solution x_1 is not worse than x_2 with respect to all objectives,*
2. *the solution x_1 is strictly better than x_2 at least one objective.*

Instead of searching for a single, optimal policy, a multi-objective optimization algorithm searches for the set of non-dominated policies in the search space, called the Pareto set: these policies are Pareto-optimal trade-offs between the objectives. For instance, a novel policy π_1 with a low cumulative return is as

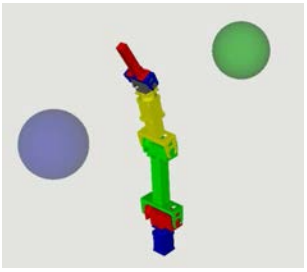
interesting as less novel policy π_2 but with a better cumulative return; however, a policy π_3 which is both less novel than π_1 and lower-performing is not interesting and is not Pareto-optimal.

Algorithm 1 Multi-DEX

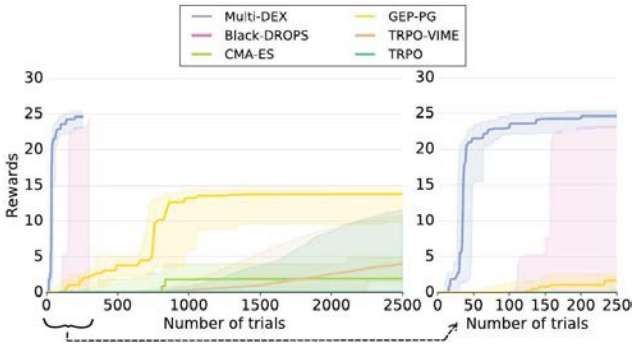
procedure MULTI-DEX
  Define policy $\pi : \mathbf{x} \times \boldsymbol{\theta} \rightarrow \mathbf{u}$
  Define $\varepsilon \in [0, 1]$ ▷ *User defined exploration parameter*
  $D = \emptyset$
  for $i = 1 \rightarrow N_R$ **do** ▷ N_R *random episodes*
    $\boldsymbol{\theta}_{rand}$ = random_policy()
    Execute $\pi_{\boldsymbol{\theta}_{rand}}$ on the real system
    Collect $(\mathbf{x}_t, \mathbf{u}_t) \rightarrow \mathbf{x}_{t+1}$ and update $\mathbb{P}$, $\mathbb{H}$ and D
  end for
  while *task* $\neq$ *solved* **do** ▷ *Learning episodes*
    Train dynamics model with GPs using collected data D
    Update $\mathbb{B}$ using the trained GPs
    Get Pareto set of policies $\pi_{\boldsymbol{\theta}}$ using NSGA-II according to $\hat{J}_r, \hat{J}_n, \hat{J}_{\sigma^2}$
    $\pi_{\boldsymbol{\theta}^*}$ = policy in Pareto front that has maximum $\hat{J}_r$
    With probability ε, replace $\pi_{\boldsymbol{\theta}^*}$ by randomly selecting from the Pareto front
    Execute $\pi_{\boldsymbol{\theta}^*}$ on the real system
    Collect $(\mathbf{x}_t, \mathbf{u}_t) \rightarrow \mathbf{x}_{t+1}$ and update $\mathbb{P}$, $\mathbb{H}$ and D
  end while
end procedure

While the set of non-dominated solutions is a very small subset of the search space, there are still many policies to choose from at the end of the optimization process. Multi-DEX executes on the robot the policy with the maximum reward *from the Pareto set* with a probability $1 - \varepsilon$ (exploitation) and choose a random policy *from the Pareto set* with a probability ε (exploration). As a consequence, if two policies have the same cumulative return, then Multi-DEX will always choose the most novel and less uncertain one; and if two policies are equally novel, it will always choose the one with the best cumulative return and uncertainty; and among the Pareto-optimal policies, it will select either the one with the best cumulative return or a random policy from the Pareto set. To find the Pareto set according to the dynamical model, we use the NSGA-II algorithm [Deb et al., 2002] (using the sferes$_{v2}$ [Mouret and Doncieux, 2010] C++ library), which is a widely used, state-of-the-art algorithm.

Overall, Multi-DEX starts with random policy and executes it on the robot (Algo. 1). Then it enters an iterative loop, where Multi-DEX (1) learns a model of the dynamics of the system and a model of the reward function (if needed) from the data collected from previous episodes, (2) finds a set of non-dominated policies, using NSGA-II, that maximizes the cumulative reward (Eq. 3.5), novelty (Eq. 3.6) and minimizes the model prediction variance (Eq. 3.7), and (3) selects the policy that corresponds to maximum reward (with a probability ε,



(A)



(B)

FIGURE 3.2: (a) Setup of the sequential goal reaching task; the goal of the 2-DOF simulated arm (only 2 top joints are active for this experiment) is to reach with its end-effector the green goal while first passing through the blue region. (b) Best reward found per trial (20 replicates). The lines are median values and the shaded regions the 25th and 75th percentiles. Multi-DEX significantly outperforms all the other approaches and finds working policies in about only 7 minutes of interaction (around 100 episodes/trials).

it replaces it with a randomly selected policy from the Pareto front) and executes it on the robot. This iterative process continues until the task is solved (Algorithm 1).

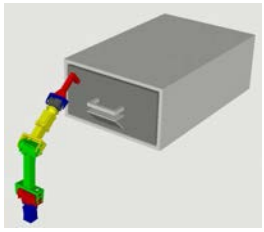
3.4 Experiments

We compare to several state-of-the-art approaches in a sequential goal reaching task and in a drawer opening task: (1) Black-DROPS [Chatzilygeroudis et al., 2017], a model-based policy search algorithm; (2) TRPO [Schulman et al., 2015], a model-free policy gradient approach; (3) TRPO with the VIME exploration strategy; (4) CMA-ES [Hansen, 2006], a black-box optimizer effective for direct policy search, and (5) GEP-PG, a curiosity-driven model-free approach. In all the tasks, we give a budget of 2500 trials to all the model-free approaches and 250 trials to the model-based ones. Please note that we do not compare Multi-DEX with PILCO [Deisenroth et al., 2015]: The reason is that Black-DROPS (1) relies on the same basic principles as PILCO, (2) is as data-efficient as PILCO, and (3) is more flexible (no constraint on the type of policy or the reward function) (4) is several times faster than PILCO when multiple cores are available [Chatzilygeroudis et al., 2017]. The Multi-DEX source code and the supplementary video are available online.[1][2].

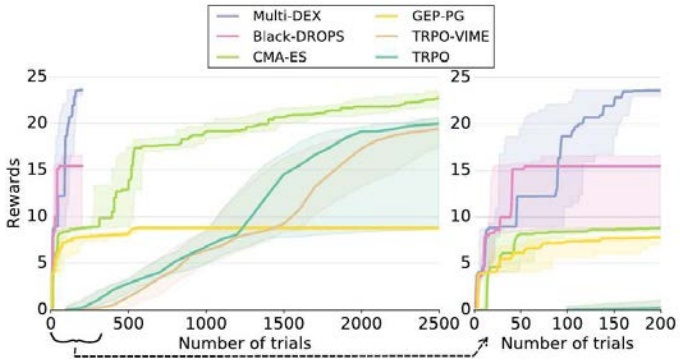
3.4.1 Sequential goal reaching with a robotic arm

This simulated task consists of a robotic arm with 2 degrees of freedom (Fig. 3.2a). The goal of the arm is to reach the green goal with its end-effector while first passing through the blue region. The length of each episode is 4 seconds with a control frequency of 10Hz. All the algorithms use feed-forward neural network

[1]Multi-DEX source code: https://github.com/resibots/kaushik_2018_multi-dex
[2]Video: https://youtu.be/XOBWq7mkYho



(A)



(B)

FIGURE 3.3: (a) Setup of the drawer opening task; the goal of the 2-DOF arm is to open the drawer and go back to the up-right position. (b) Best reward found per trial (20 replicates). The lines are median values and the shaded regions the 25th and 75th percentiles. Multi-DEX outperforms all the other approaches and finds working policies in about only 14 minutes of interaction (200 trials).

policies with 32 parameters (refer to section 3.6 for details on the policy, the GP model and the optimizer parameters). In more detail:

- **State:** $\mathbf{x}_{\text{seq}} = [\theta_0, \theta_1, \gamma] \in \mathbb{R}^3$, where θ_0, θ_1 are joint angles and γ is a boolean value which is set to 1 for rest of the episode if the end-effector has touched the blue region.
- **Actions:** $\mathbf{u}_{\text{seq}} = [v_0, v_1] \in \mathbb{R}^2, -2 \leq v_0, v_1 \leq 2\ rad/s$ are joint velocity commands.
- **Reward:** The reward function depends only on the current state and is unknown to the algorithm as it is not a *direct* function of observable state:

$$r(\mathbf{x}_{\text{seq}}) = \begin{cases} 0, & \text{if } \gamma \neq 1 \text{ or } \Delta \geq 0.1 \\ \exp\big(-\alpha\Delta^2\big), & \text{otherwise.} \end{cases} \tag{3.8}$$

  where Δ is the Euclidean distance between green goal and the end-effector of the arm and α is a user-defined constant ($\frac{0.5}{0.2*0.2}$ in our experiments). After every episode, Multi-DEX learns the reward function from the collected data using Random Forest regression [Breiman, 2001].

In this experiment, Multi-DEX is able to find high-performing policies in less than 100 trials on the target system (around 7 minutes of interaction), whereas none of the other approaches are able to achieve similar rewards even after 2500 trials (Fig. 3.2b — 20 replicates). As this task has a highly sparse reward space, CMA-ES, Black-DROPS and TRPO show almost zero median rewards.

3.4.2 Drawer opening task with a robotic arm

The goal of this simulated task is to open a drawer with a 2-DOF robotic arm and go back to the up-right position (Fig. 3.3a). Similar to the previous task, the length of each episode is 4 seconds (10Hz control) and all the algorithms

use neural network policies with 32 parameters (refer to section 3.6 for details on the policy, the GP model and the optimizer parameters). In more detail:

- **State:** $\mathbf{x}_{\text{drawer}} = [\theta_0, \theta_1, \delta] \in \mathbb{R}^3$, where θ_i are joint angles and δ the drawer displacement.
- **Actions:** $\mathbf{u}_{\text{drawer}} = [v_0, v_1] \in \mathbb{R}^2, -1 \leq v_0, v_1 \leq 1\ rad/s$ are joint velocity commands.
- **Reward:** In this task, the reward depends only on the current state and is known to the algorithm as it is a function of the observable state. The total reward is given by:

$$r(\mathbf{x}_{\text{drawer}}) = r_{\text{return}}(\mathbf{x}_{\text{drawer}}) + \delta \tag{3.9}$$

$$r_{\text{return}}(\mathbf{x}_{\text{drawer}}) = \begin{cases} 0, & \text{if } \delta \leq 0.2 \\ \exp(-\theta_0^2 - \theta_1^2), & \text{otherwise.} \end{cases} \tag{3.10}$$

In this task, the reward space is moderately sparse compared to the previous one; however, it has a misleading reward space because of the composition of two different rewards. Any algorithm without efficient exploration will converge to the reward associated with the drawer displacement only. Multi-DEX is able to find policies that complete the task in just 200 trials (around 14 minutes of interaction), whereas all the other approaches are deceived by the reward space (Fig. 3.3b — 20 replicates). Black-DROPS and TRPO, without any directed exploration, fall in the sub-optimal solution (*i.e.*, just opening the drawer quickly). CMA-ES, TRPO-VIME and GEP-PG either fail to converge or need more than 2500 trials to reach the same quality of solutions as Multi-DEX.

3.4.3 Deceptive pendulum swing-up task

This simulated task consists of a pendulum powered by an underpowered torque-controlled actuator. The goal in this task is to swing the pendulum to the upright position applying torques as small as possible (*i.e.*, using minimum power) to the actuator and hold it in that position. The learning algorithm gets a constant positive reward of +10 every time-step when the pendulum is in an upright position (within some region). It also gets a negative reward proportional to the square of torque for every time step. Because of these two rewards, the total reward function possesses a deceptive landscape and algorithms without efficient exploration will converge to a reward of zero, which is achieved when no torque is applied to the pendulum. This type of *"Gradient Cliff"* reward landscape was first proposed by [Lehman et al., 2017]. The control frequency in this task is $10Hz$ and every episode is 4 seconds long. In more detail:

- **State:** $\mathbf{x}_{\text{pend\_sys}} = [\theta, \dot{\theta}] \in \mathbb{R}^2$, where θ is the joint angle and $\dot{\theta}$ is the joint angular velocity. The initial state of the system is $x_0 = [0, 0]$

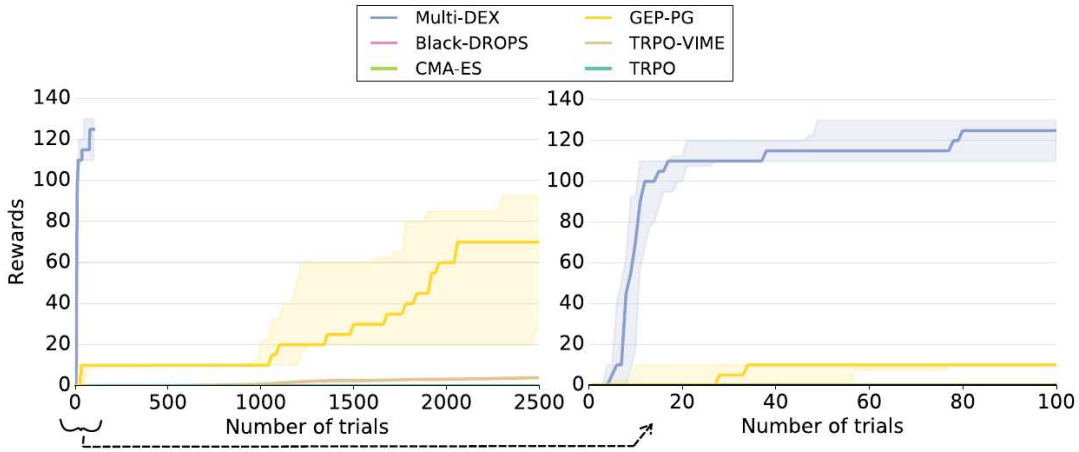


FIGURE 3.4: Best reward found vs. the number of trials plot for Pendulum Swing-up Task. Here Multi-DEX clearly outperforms all the competing approaches and achieves very high reward (balancing the pendulum in upright position) in just 100 trials (approx 6.6 minutes of total interaction). The approaches such as Black-DROPS, CMA-ES, TRPO and TRPO-VIME could not even improve at all and stayed almost flat near the zero reward in the plot.

- **Actions:** $\mathbf{u}_{\text{pend\_sys}} = [\tau] \in \mathbb{R}, -2.0 \leq \tau \leq 2.0$, where τ joint torque command for the arm.
- **Reward:** In this task, the reward is known to the algorithm as it is a function of the observable states and applied action to the system. The total reward is given by:

$$r(\mathbf{x}_{\text{pend\_sys}}, \tau) = r_1(\mathbf{x}_{\text{pend\_sys}}, \tau) + r_2(\mathbf{x}_{\text{pend\_sys}}, \tau) \tag{3.11}$$

$$r_1(\mathbf{x}_{\text{pend\_sys}}, \tau) = \begin{cases} 0, & \text{if } \pi - \theta < \pi/60 \\ +10, & \text{otherwise.} \end{cases} \tag{3.12}$$

$$r_2(\mathbf{x}_{\text{pend\_sys}}, \tau) = -0.001 * \tau^2 \tag{3.13}$$

The policy here is a feed-forward neural network with one hidden layer (41 parameters). Further details on the policy, the GP model and the optimizer parameters can be found in section 3.6. In this experiment, the results show that Multi-DEX quickly reaches a very high positive reward with a very small variance within the budget of 100 trials (Fig. 3.4). On the contrary, GEP-PG and TRPO-VIME could not reach to the same quality of solutions even after 2500 trials on the system. As TRPO, Black-DROPS and CMA-ES do not have any directed exploration, they could not improve the reward and stay close to zero by minimizing the applied torque only.

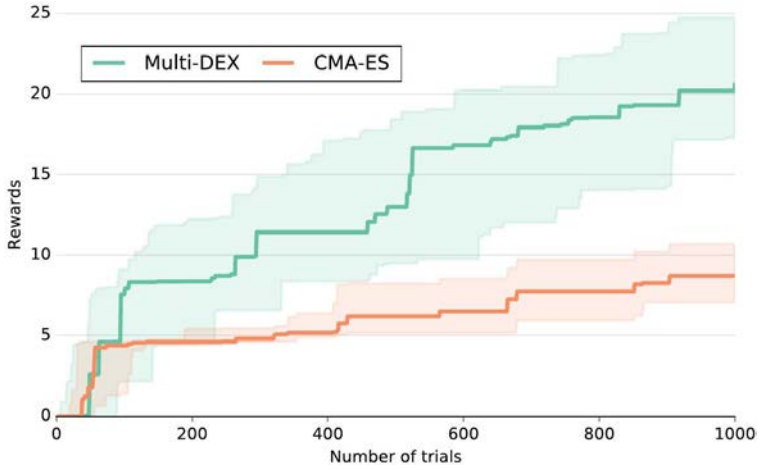


FIGURE 3.5: Best reward found vs number of trials plot for drawer opening task with 4-DOF simulated robotic arm. Plot shows that both the median over the replicates with Multi-DEX is able solve the task in 1000 trials contrary to CMA-ES which is able to reach only half the reward achieved by Multi-DEX)

3.4.4 Additional Experiments

Drawer opening task with 4-DOF arm

To evaluate Multi-DEX on more difficult sparse reward scenarios, we ran it on the drawer opening task (refer 3.4.2) with a 4-DOF simulated robotic arm. We compared the result with CMA-ES and found that Multi-DEX takes around 1000 trials on the system to solve it (median over 20 replicates), i.e., maximizes the reward on both the subtasks - pulling the drawer and going back to the initial position. On the contrary, CMA-ES was able to reach only half the reward achieved by Multi-DEX in 1000 trials (Fig. 3.5) (refer to section 3.6 for experimental details). To be noted that we restricted CMA-ES here to use only 1000 episodes (i.e., 1000 policy evaluations on the system) as the goal is only to compare its data-efficiency with Multi-DEX that was able to solve the task in 1000 episodes.

Multi-DEX on non-sparse reward task

We tested Multi-DEX on a single goal reaching task with a 5-dof simulated arm. In this task the state-space is 5 dimensional (5 joint angles of the arm) and action space is also 5 dimensional (torque inputs to the joints). The length of each episode is 4 seconds (10Hz control) and we used neural network policy with one hidden layer. The hidden layer had 10 neurons and thus the entire policy space is 115 dimensional (refer to section 3.6 for further experimental details). We chose a reward that is continuous in the state-space as given below:

$$r(\mathbf{x}_{\text{arm}}) = \exp(-\frac{0.5}{0.2 \times 0.2}||\mathbf{x}_{\text{goal}} - \mathbf{x}_{\text{arm}}||^2) \tag{3.14}$$

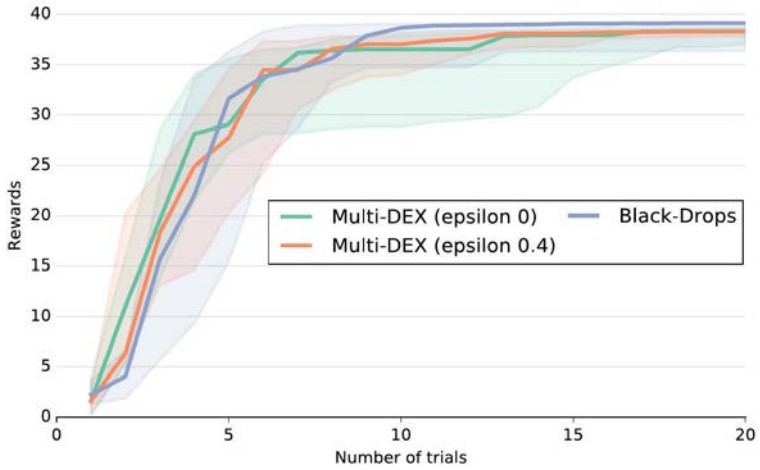


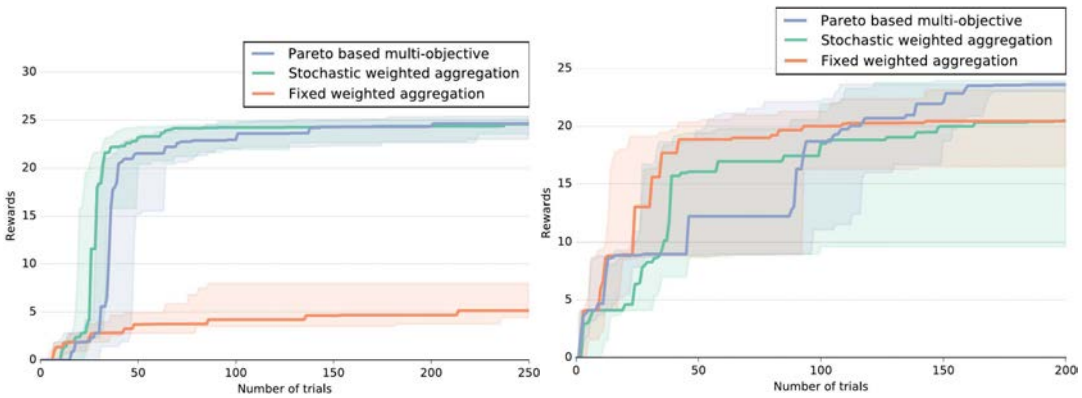
FIGURE 3.6: Best reward found vs number of trials plot for single goal reaching task with a 5-DOF simulated arm. Plot shows that both the variants of Multi-DEX are competitive enough with Black-DROPS in this non-sparse reward scenario

where $\mathbf{x}_{\text{goal}}$ and $\mathbf{x}_{\text{arm}}$ are the 3D coordinate vectors of the the goal and the end-effector. We compared two variants of Multi-DEX: with $\epsilon = 0$ and $\epsilon = 0.4$. We compared the result with standard Black-DROPS to see how efficient Multi-DEX is on a non-sparse reward setup. This preliminary results show that both the variants of Multi-DEX is competitive enough with Black-DROPS (Fig. 3.6). From this, we can expect Multi-DEX to be equally effective in sparse as well as non-sparse reward scenarios. Point to be noted that we plot here the median, 25 percentile and 75 percentile of "best reward so far" for the replicates.

3.4.5 Pareto optimality vs. weighted sum objectives

When optimizing several objectives with a weighted sum, the weights determine the trade-off between the objectives (e.g., more exploration vs. more exploitation) and the optimum is a particular Pareto-optimal solution. The important question for multi-objective optimization is, therefore: can any Pareto-optimal solution (i.e., any optimal trade-off) be found by selecting the right combination of weights? The answer depends on the shape of the Pareto front (the set of Pareto-optimal solutions). If it is convex, then all the Pareto-optimal solutions can be found by varying the weights; however, if it is not, then some Pareto-optimal trade-offs cannot be found using a weighted sum [Deb, 2001]. In the learning problems tackled in this work, there is no formal guarantee that the Pareto front is convex; therefore, it is theoretically better to use another approach to find a suitable Pareto-optimal trade-off. In other words, the best trade-off between reward, novelty, and model variance might not be expressible with a combination of weights. In addition, setting weights requires a good normalization method (since the trade-off will be affected by the scale of each objective), whereas a Pareto-based approach will usually be independent of the scale.

For completeness, we compared our Pareto-based approach experimentally



(A) Sequential goal reaching task (B) Drawer opening task

FIGURE 3.7: Best reward found vs number of trials plots show the comparison between Pareto based multi-objective approach and weighted aggregation approach. For stochastic weighted aggregation approach we defined 4 set of weights with different trade-offs among the objectives. For each episode we randomly select one set of weights and use them for aggregation of the objectives into a single objective. On the other hand, for the fixed weighted aggregation approach we defined a single set of weights and used them for the aggregation of the objectives.

with both fixed and stochastic weighted aggregation of the objectives to a single objective. For fixed weighted aggregation, we used weights 0.4, 0.3 and 0.3 for novelty, variance and cumulative reward, respectively. On the other hand, for stochastic weighted aggregation, we used 4 sets of weights: $\{0.6, 0.3, 0.1\}$, $\{0.1, 0.3, 0.6\}$, $\{0.1, 0.45, 0.45\}$ where each element of the sets corresponds to the weights for novelty, variance and cumulative reward respectively. We randomly select a set of these weights for each episode and use it for aggregation of the objectives. The results show that weighted aggregation doesn't give consistent results on both the experiments. Although a different set of weights could give different results, in the drawer opening task Pareto-based approach outperforms both the variants of weighted aggregation approach with higher mean performance and smaller variance over the replicates. In the sequential goal reaching task, the stochastic weighted aggregation shows similar performance to the Pareto-based approach. However, the fixed weighted aggregation approach did not solve the task at all.

3.5 Discussion and Conclusion

It is well established that learning a dynamical model can make policy search highly-data efficient for control tasks [Chatzilygeroudis et al., 2017; Deisenroth et al., 2013; Chatzilygeroudis et al., 2020] when (1) the state-space is low-dimensional enough to learn a model, (2) the model is probabilistic so that the uncertainty of the predictions can be taken into account, and (3) it is possible to rely on a large computation time between episode (for the policy optimization stage). In this work, we showed that the same model could be leveraged to improve exploration for tasks in which the reward is sparse or deceptive: compared to state-of-the-art approaches to exploration for policy search, which

are all model-free, our model-based policy search algorithm requires at least an order of magnitude fewer episodes and reach higher rewards in all our experiments.

In this work, we used an evolutionary algorithm (NSGA-II) instead of gradient-based optimization approaches. As already mentioned, evolutionary optimization allows us to optimize the policy in a black-box manner without enforcing any specific property (such as differentiability) on the policy as well as on the reward function. Several recent work show that evolutionary algorithms, such as NSGA-II, can be competitive to optimize deep neural network policies in reinforcement learning scenarios [Petroski Such et al., 2017; Salimans et al., 2017]. Thus, we believe that NSGA-II should be able to scale up to higher-dimensional policy optimization, at the obvious expense of more optimization time. At any rate, it should be emphasized that what matters the most in this work is the interaction time with the system/robot, and not the policy optimization time that happens on the learned dynamical model.

In section 3.4.5, we discussed why Pareto optimality had been preferred over weighted-sum objective in this work. It is also worth discussing here how the policy optimization objectives interact among themselves in Multi-DEX. To elaborate, in Multi-DEX, the first objective in Multi-DEX is to find trajectories (i.e., corresponding policies) that have a high reward. The second objective tries to ensure that for a particular value of the reward, we get as novel trajectory as possible. Now, these two objectives might be antagonistic, and for most cases, both cannot be improved without compromising the other. Thus we want a Pareto-optimal set of policies that gives the trade-off between these two objectives, starting from zero reward to maximum reward possible on the model. However, as we are using GP mean prediction only to find trajectories, it is possible that we find policies that drive the system though highly uncertain regions on the model. If this is the case, then those policies will have completely different behavior on the system from the one predicted using the model. Thus we have the third objective that tries to keep the trajectories as close as possible to the more certain regions of the model. Again this new objective might be antagonistic to the previous two objectives in some cases. Therefore, we want a Pareto-optimal trade-off among all the objectives. It is to be noted here that in Multi-DEX, the novelty score is highly dependent upon the order in which the states are visited and not that much on the individual states. So it is possible to have a high novelty score by going through similar states that were observed before (i.e., points with less uncertainty) but in a completely different order. Once the Pareto-optimal trade-offs are identified, all of them are potentially good candidates. This is why we alternatively take a random policy from these good candidates (minimize bias in exploration) or the one with the best predicted reward (exploitation).

A question that may arise here is whether Multi-DEX can build a good model by generating relevant data. At first look, it might seem counter-intuitive that an objective of minimizing the variance can help to generate new data for the model. However, a more in-depth look into the interactions among the objectives in the Pareto-based multi-objective optimization framework reveals how new data is generated in Multi-DEX. First, although a high novelty score

can be achieved by visiting the previously visited states in a different order, for a physical system, it is not always possible to visit the same set of states with all the possible orders of the visit. This is even more restricted if the policy is not very expressive (*e.g.,* a small neural network). Thus, in the set of Pareto optimal policies, it is possible to have policies that produce novel state trajectories in the model at the cost of higher variances. These policies can drive the system to the regions where the model's prediction variances are high and help improve the model. It is to be noted here that since our novelty score considers the trajectory of the states, the revisiting of the states in different orders is a desirable characteristic of Multi-DEX as each of these trajectories will produce a different behavior on the robot.

In Multi-DEX, we have defined the novelty score of any policy as the minimum Euclidean distance between the expected state trajectory of that policy to all the *expected state trajectories* of the executed policies on the system. There is a strong reason for keeping expected state trajectory vectors and not the observed state trajectory vectors on the system to compute the novelty score of a policy. Because of the imperfection in the model, the model might predict unachievable state trajectories to maximize novelty, which might give a completely different state trajectory on the actual system than the one expected by the model. Now, if we compute the novelty score for a policy by comparing it with the observed state trajectory vector on the system, then the already tried policies predicting unachievable behaviors will have very high novelty scores. As a result, the optimizer might return these trajectories again and again for every iteration of the algorithm. A solution to this problem is to keep the expected state trajectory vectors instead to compute the novelty scores. As a result, the unachievable state trajectories that were already tried on the system will have low novelty scores; and hence, policies with these state trajectories will be rejected because of low novelty scores by the optimizer.

3.6 Details on the Experimental Setup

3.6.1 Simulator and source code

For all the experiments, we have used the DART physics simulation library [Lee et al., 2018] to implement the scenarios. For TRPO, VIME and GEP-PG, we used python version of the DART library (pydart2[3]); for CMA-ES, Black-DROPS and Multi-DEX we used the C++ version of the library for simulation.

For CMA-ES, we used Limbo's wrapper of libcmaes library[4]. For all other algorithms (GEP-PG, TRPO, TRPO-VIME and Black-DROPS), we used the code provided by the authors and integrated our own scenarios. The source code of Multi-DEX can be found here: `https://github.com/resibots/kaushik_2018_multi-dex`

[3] `https://github.com/sehoonha/pydart2`
[4] `https://github.com/beniz/libcmaes`

3.6.2 General and Exploration Parameters

For GEP-PG, we used 500 exploratory episodes for the "Drawer Opening" and "Sequential Goal Reaching" tasks; and 100 exploratory episodes for "Pendulum Swing-up" task. For all the experiments with GEP-PG we used 10 initial bootstrap episodes.

For TRPO-VIME we used $\eta = 0.001$ for the "Drawer Opening" task. For the "Sequential Goal Reaching" and "Pendulum Swing-up" tasks, we used $\eta = 0.001$.

For CMA-ES, we used the active-IPOP version with elitism enabled [Auger and Hansen, 2005; Loshchilov, 2013]. We used 3 restarts and 2500 maximum function evaluations.

For Multi-DEX, we used $\varepsilon = 0.3$ for the "Sequential Goal Reaching" and "Pendulum Swing-up" tasks. We used $\varepsilon = 0.4$ for the "Drawer Opening" task. In all the experiments, to compute the novelty score, we sub-sample the expected state trajectories at 10 equally spaced points in time. In all the cases, we bootstrapped the algorithm with 5 initial random trials.

3.6.3 Policy and Parameter Bounds

For all the experiments, we used neural network policies. For Black-DROPS, GEP-PG, and Multi-DEX in the "Sequential Goal Reaching" and "Drawer Opening" tasks, we used one hidden layer with 5 neurons and in the "Pendulum Swing-up" task we used one hidden layer with 10 neurons. For TRPO and TRPO-VIME we used two hidden layers with 10 neurons each (the implementations that we used required at least 2 hidden layers for the neural network policies).

The policy parameter bounds used in Multi-DEX are $[-1, 1]$ for "Sequential Goal Reaching", $[-3, 3]$ for "Drawer Opening" tasks and $[-5, 5]$ for "Pendulum Swing-up" task.

3.6.4 NSGA-II Parameters

We used the Sferes2 C++ library's [Mouret and Doncieux, 2010] implementation of NSGA-II algorithm in Multi-DEX. The parameters used for NSGA-II are the following:

- crossover rate $= 0.5$
- mutation rate $= 0.1$
- $\eta_m = 15.0$
- $\eta_c = 10.0$
- mutation type: Polynomial [Deb, 2001]
- cross-over type: Simulated Binary Crossover [Deb and Beyer, 1999]

We run NSGA-II for 600 generations with population sizes of 200 for the "Drawer Opening" and "Sequential Goal Reaching" tasks and 300 for the "Pendulum Swing-up" task.

For the first episode, the initial population of policies is randomly initialized. From the 2nd episode onwards, we insert some policies from the Pareto-front into the initial population. The maximum number of these inserted policies is not more than 30% of the total population.

3.6.5 Gaussian Process Model learning

In this work, we used the Limbo C++ library's [Cully et al., 2018] implementation of Gaussian process regression for probabilistic dynamics model learning. We used the exponential kernel with automatic relevance determination [Rasmussen and Williams, 2006]. We optimize the hyper-parameters of the kernel via Maximum Likelihood Estimation using the Rprop optimizer [Riedmiller and Braun, 1992; Blum and Riedmiller, 2013].

We maintain two experience buffers (of fixed size) $\mathbb{P}$ and $\mathbb{H}$ to keep the state-action trajectories for model learning (see the main text).

Chapter 4

Adaptive Prior Selection for Data-efficient Online Learning

The results and text of this chapter have been partially published in the following article:

Article:

- **Kaushik, R.**, Desreumaux, P., & Mouret, J. B.. *Adaptive Prior Selection for Repertoire-based Online Learning in Robotics.* **Frontiers in Robotics and AI**, vol. 6, p. 151, 2020. [Kaushik et al., 2020b]

Other contributors:

- Pierre Desreumaux (Research Engineer)
- Jean-Baptiste Mouret (Thesis supervisor)

Author contributions:

- **RK** and JBM organized the study. **RK** wrote the code. **RK** and PD conducted the experiments. **RK**, PD and JBM analyzed the results and wrote the paper.

4.1 Introduction

From the discussions in the background section (2.5.4) of this manuscript, we realize that a promising way to address the "curse-of-dimensionality" in model-based learning in robotics is the repertoire-based learning approach (section 2.5.4). The core idea here is to learn a model in the "task-space" of the robot to predict how the outcomes of elementary policies stored in a *repertoire* change in the real-world compared to the simulation [Cully and Mouret, 2015; Cully et al., 2015; Chatzilygeroudis et al., 2018; Duarte et al., 2017; Sharma et al., 2019]. This process splits the policy search problem in two parts: first, search for a repertoire of policies in simulation, where large number of interactions are possible; and second, on the real robot, search for the most appropriate policy from the repertoire, which is usually easier than directly searching on the original policy parameter space. For instance, in the hexapod damage recovery task, the algorithm RTE [Chatzilygeroudis et al., 2018] uses a repertoire of

elementary policies that makes a simulated 6-legged (intact) robot (18D joint space) walk in different directions (i.e, one policy for each walking direction on the plane). Then on the real robot (with a broken leg), RTE iteratively learns a probabilistic model to predict how the 3D task-space (x,y position and orientation of the robot) is transformed (i.e, a mapping from expected transitions to the observed transitions of the robot on the surface) when these policies are transferred to a damaged robot. With this model, at every control-step, a planning algorithm then selects a sequence of elementary policies from the repertoire by taking into account the outcome difference between the prior — the intact robot in simulation — and the reality — the damaged robot. Since the dimension of the task-space (the center of mass position and orientation for the hexapod) is often much smaller than the full state-space, this approach reduces the dimensionality of the model, and therefore the amount of interaction time.

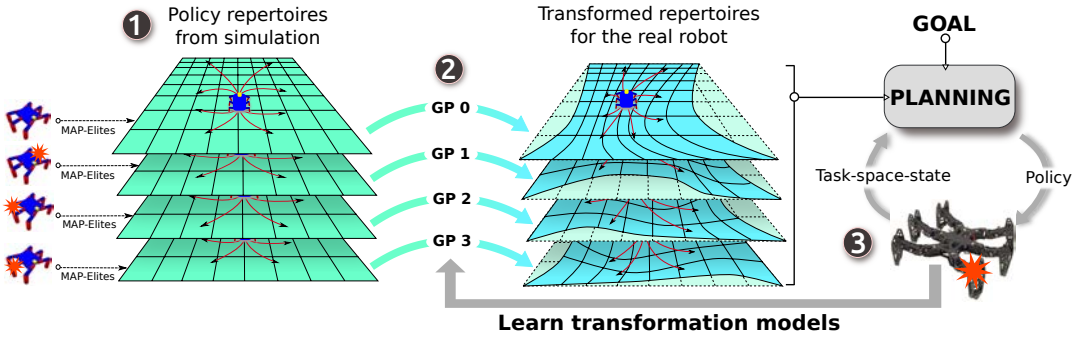


FIGURE 4.1: Overview: APROL uses multiple repertoire-based priors for fast online adaptation in unforeseen situations. **(1)** First, in a low fidelity simulator, we generate multiple repertoires of elementary policies for various situations of the robot, such as being damaged, slippery floor, interaction with a novel object etc. A repertoire is a discrete one-to-one association between elementary policies and their corresponding task-space transitions on the robot. **(2)** Then, we use those repertoires as prior mean-function for Gaussian process regression models that learn to transform the task-space transitions in each of the repertoires to the task-space transitions of the real robot using the past observations. **(3)** Finally, given the goal, APROL iteratively picks the most suitable elementary policies from the most suitable repertoire at every re-planning step to reach the goal in a minimum number of steps. In between every re-planning step, the task-space transformation models are updated with the past observations.

Nevertheless, RTE assumes that one repertoire-based prior is enough for the robot to quickly adapt to any situation that the robot might face in the real world. However, like with any learning algorithm based on prior knowledge, the effectiveness of the learning process depends critically on the difference between the prior and the reality: the bigger the difference, the worse it performs (more interaction time, lower quality policies). In this work, we address this issue by allowing the adaptation algorithm to select the most interesting prior among a set of priors learned beforehand. In other words, we learn several repertoires of elementary policies in simulation, each with a unique situation (e.g., different damages), and the robot adapts by both searching for the most

suitable prior (i.e., the repertoire) and correcting their expected outcomes using a model learned from the observations.

To do so, we propose to evolve several repertoires of elementary policies for the robot using an evolutionary algorithm called MAP-Elites [Cully et al., 2015; Mouret and Clune, 2015] (in simulation), each with a unique situation picked from a sub-set of probable situations that the robot might face in reality. Each of these repertoires associates different task-space transitions (e.g., relative displacement) with unique elementary policies. Using each of these repertoires as "prior mean-functions" for Gaussian processes (GP) regression models [Rasmussen and Williams, 2006], we learn as many models as the number of repertoires from the observations on the real robot. Each of these models maps expected task-space transitions stored in the corresponding repertoire to the actual task-space transition on the robot.

More concretely, we iteratively learn a probabilistic model that predicts how these repertoires transform themselves when applied on the real robot as we collect more data from the real robot. Then, instead of selecting a single global model for controlling the robot, we pose this as a maximum a posteriori (MAP) estimation problem of selecting the next elementary policy from one of the repertoires, given the repertoires, the past observations and the goal. We call this algorithm APROL (Adaptive Prior selection for Repertoire-based Online Learning) (Figure 4.1). The main novelty in APROL compared to the previous work is that it uses multiple repertoires instead of one and adapt online by automatically selecting the most suitable policy from one of those repertoires based on the current situation.

4.2 Problem Formulation

We consider a system whose transition in the task-space depends not only on the current task-space state and the policy, but also on the current situation (e.g, icy vs rocky terrain for mobile robot). Then, the task-space transition dynamics of such systems can be written as:

$$s_{t+1} = s_t + F(s_t, \pi_\theta, c) + w \tag{4.1}$$

where, s_t, s_{t+1} are the locations in task-space at time-step t and $t+1$ respectively, π_θ is the open-loop policy/controller parameterized by θ, $c \in \mathbb{C}$ specifies the current situation, w is the Gaussian system noise and $F(\cdot, \cdot, \cdot)$ is the task-space transition function of the system. Note that, $\mathbb{C}$ can potentially be an infinite set, which means that the system can face infinitely possible situations during its deployment that can change its transition dynamics.

We assume that we neither have access to $F(\cdot, \cdot, \cdot)$ nor have knowledge about the current situation c. However, we have access to a low fidelity simulator of the system $\hat{F}(\cdot, \cdot, \cdot)$ and a set $\mathbb{C}' \subset \mathbb{C}$ of probable situations that the system might face during deployment. The goal is to drive the system from a starting task-space state s_0 to the target task-space state s_g in a minimum number of steps by executing a sequence of elementary policies. Here, elementary policies

are open-loop policies that are applied for a short period of time (a few seconds) on the robot, which cause a small change in the task-space state of the robot.

In other words, we consider a robot that follows eq. 4.1 and might face any situation, such as broken joints, slippery floors, or a novel object to manipulate during its mission. These situations cannot be predicted beforehand and the robot is not equipped with any specialized sensor either to observe such situations. Now, if such situations arise, instead of aborting its mission, the robot has to figure out a sequence of compensatory policies to continue its mission and accomplish the goal as quickly as possible.

4.3 Approach

4.3.1 Overview

APROL allows a robot to "learn while doing" instead of "learning and then doing". Our approach is based on 3 main stages (Figure 4.1):

1. Before deployment of the robot, several repertoires of elementary policies are generated for the robot with an evolutionary algorithm called "MAP-Elites" using a relatively low-fidelity simulator of the real robot (section 4.3.2). Each of these repertoires are generated for a unique situation or circumstance that the robot might face during its mission, such as a broken limb, a novel object to interact with or different terrain conditions, etc. Each of these repertoires is basically a one to one association between the evolved elementary policies and the corresponding transitions they cause on the robot.

2. At every replanning-step, using the past observations from the real robot, we learn a probabilistic mapping $g : S \mapsto S$, where S is the task-space, for each of the repertoires to predict how the task-space-transitions in the repertoires transform themselves when corresponding policies are applied on the real robot (section 4.3.3). These transformation models are learned using GP, because (i) we can set a prior mean-function for the GP which is the prior belief about the underlying function that the GP needs to fit, (ii) instead of deterministic prediction, GP outputs a probability distribution, which allows us to incorporate model uncertainty in the planning stage. For every repertoire, we learn this model using the expected outcomes of the repertoire itself as prior mean-function.

3. Using the past observations and the given goal, APROL picks the best policy from one of the repertoires and applies it on the robot for one replanning-step (section 4.3.4). Then the process repeats from step 2.

Algorithm 2 Generate Priors

Require: $S \in \mathbb{R}^{n_s}$ ▷ Task-space
Require: $\Theta \in \mathbb{R}^{n_\theta}$ ▷ Policy space
Require: $C = \{c_0, c_1, ..., c_n\}$ ▷ Probable situations
Require: bot_sim ▷ Simulator of the robot
Require: $f_{eval}(.)$ ▷ Performance function
Require: N_{max} ▷ Max evaluation
1: $\mathcal{R} = \{\}$ ▷ Empty set of Repertoires
2: **for** c in C **do**
3: $\Pi = map\_elites(bot\_sim, \Theta, c, S, f_{eval}, N_{max})$
4: Insert repertoire Π in $\mathcal{R}$
5: **end for**
6: Return $\mathcal{R}$

4.3.2 Generating Repertoire-Based Priors

We assume that the robot can be controlled by a low-level elementary policy $\pi_{\boldsymbol{\theta}}$ (typically, an open-loop policy) parameterized by $\boldsymbol{\theta} \in \mathbb{R}^{n_\theta}$ and that any point on the task-space can be described by a vector $\mathbf{s} \in \mathbb{R}^{n_s}$. In simulation, the task-space transition caused by the policy $\pi_{\boldsymbol{\theta}}$ can simply be written as $\Delta\mathbf{s}_\theta$. Additionally, we assume that a set $\mathbb{C}' = \{c_0, c_1, .., c_n\}$, which is a subset of all the possible situations $\mathbb{C}$ is available. Then, for each situation $c \in \mathbb{C}'$, we use an iterative algorithm called "MAP-Elites" [Cully et al., 2015; Mouret and Clune, 2015; Vassiliades et al., 2017] to evolve a repertoire of elementary policies in simulation, such that a wide range of task-space transitions can be captured in the repertoire [Cully et al., 2015; Cully and Mouret, 2015; Duarte et al., 2018] (see Algorithm 2). Nevertheless, other quality diversity algorithms [Pugh et al., 2016; Cully and Demiris, 2018b] could also be used to generate the repertoire with almost no influence on the behavior of APROL.

To begin with, the MAP-Elites discretizes the task-space into some regions or cells, each of which is identified using a cell identifier ($cell\_id$), which is a unique key to specify a cell. In the beginning, MAP-Elites randomly initializes some policies and test them in simulation to find out the task-space transitions $\Delta\mathbf{s}_\theta$ and their performance score $r_{\boldsymbol{\theta}}$. Then, they are included in the repertoire as tuples of policies, their corresponding transition, the performance score and the cell id as $(\pi_{\boldsymbol{\theta}}, \Delta\mathbf{s}_\theta, r_{\boldsymbol{\theta}}, cell\_id)$. Here, the performance is a user-defined function with which some constraints can be imposed on the behavior of the robot. For example, we can set a lower performance score for a policy if it produces higher joint torques on the joints. That way MAP-Elites will prefer the policy with lower torque if two policies produce the same task-space transition on the robot. After this initialization, MAP-Elites performs the following three steps iteratively until the maximum number of valuations are reached:

1. Randomly picks a tuple from the repertoire and adds a small random variation to the policy.

2. Simulates the policy to get the task-space transition, performance score and the $cell\_id$ to create a new tuple.

3. Inserts the new tuple into the repertoire if no tuple exists with the same *cell_id*, or, replaces an existing tuple with the same *cell_id* but with a lower performance score (discards the new tuple otherwise).

Thus, each repertoire is a set of tuples $(\pi_{\boldsymbol{\theta}}, \Delta\mathbf{s}_\theta, r_{\boldsymbol{\theta}}, cell\_id)$, where, no two tuples have the same *cell_id*. One thing to be noted here is that although MAP-Elites is computationally expensive, it can be parallelized on large clusters to compute the repertoires before deployment of the robot. It is worth mentioning here that we use the CVT variant of MAP-Elites [Vassiliades et al., 2017] that uses centroidal Voronoi tesselation to discretize the task-space into the user-specified number of homogeneous geometric regions. In CVT Map-Elites, the number of cells remains fixed irrespective of the dimensionality of the task-space, making it scalable to a very high dimensional task-space.

Algorithm 3 Planning using APROL

Require: $\mathcal{R} = \{\Pi_0 \Pi_1, ..., \Pi_n\}$ ▷ Set of repertoires
Require: G ▷ The task/goal
Require: $task\_planner$
Require: $D = \phi$ ▷ Empty observations set
$T = \{gp_0, gp_1, \cdots, gp_n\}$ ▷ Initialize models
while task not solved **do**
  $s = get\_current\_state()$
  $s_g = task\_planner(G, s)$ ▷ Current sub-goal
  $\pi^* = \text{argmax}_{\pi \in \Pi, \Pi \in \mathcal{R}} P(s_g|\pi, D, \Pi) P(\Pi|D) P(\pi)$
  Execute π^* and record data in D
  Update transformation models T with D
end while

4.3.3 Learning the Transformation Models with Repertoires as Priors

Here, we use the same approach that RTE [Chatzilygeroudis et al., 2018] used to learn the transformation model with the repertoire as priors. Since the policies in the repertoires come from a simulator, how they change the state of the system is an approximation of the reality. Moreover, if the real situation of the robot (e.g., different floor conditions, novel object to interact with, mechanical damage, etc.) is different from those of the repertoires, then the corresponding transitions for the policies will not align perfectly with the real system. However, the transitions that we observed in the simulation can be a "prior" (i.e., prior in Bayesian model learning) to learn the actual transitions that we see on the real system, provided the situation of the robot in simulation is somewhat close to the real situation. Thus, we use GPs to learn the transformation models of the task-space from simulation to reality, where we use the transitions stored in the repertoires themselves as prior mean-functions to these models.

Let us suppose that an arbitrary policy $\pi_{\boldsymbol{\theta}}$ from a repertoire produces a task-space transition $\Delta\mathbf{s}_\theta$ on the simulated robot. The same policy $\pi_{\boldsymbol{\theta}}$ produces transition of $\Delta\mathbf{s}_{\theta,real}$ on the real robot. We can learn a model to predict this transformation of $\Delta\mathbf{s}_\theta$ to $\Delta\mathbf{s}_{\theta,real}$. More concretely, we learn a probabilistic model using GP, where input is $\Delta\mathbf{s}_\theta$, the target is $\Delta\mathbf{s}_{\theta,real}$ and the corresponding

prior mean is $\Delta\mathbf{s}_\theta$ itself. If $f(\Delta\mathbf{s}_\theta)$ is the function that transforms the task-space from simulation to reality, then for each prediction dimension $d = 1, 2, ...n$, the GP can be computed as:

$$P(f_d(\Delta\mathbf{s}_\theta)|D^d_{1:t}) = \mathcal{N}(\mu_d(\Delta\mathbf{s}_\theta), \sigma^2_d(\Delta\mathbf{s}_\theta)) \tag{4.2}$$

where,

$$\mu_d(\Delta\mathbf{s}_\theta) = M_d(\Delta\mathbf{s}_\theta) + \boldsymbol{k}^T_d(\mathbf{K}_d + \sigma^2_n I)^{-1} (D^d_{1:t} - M_d(\Delta\mathbf{s}_{\theta 1:t})) \tag{4.3}$$

$$\sigma^2_d(\Delta\mathbf{s}_\theta) = k_d(\Delta\mathbf{s}_\theta, \Delta\mathbf{s}_\theta) - \boldsymbol{k}^T_d(\mathbf{K}_d + \sigma^2_n I)\boldsymbol{k_d} \tag{4.4}$$

Where, $D^d_{1:t} = \{f_d(\Delta\mathbf{s}_{\theta 1}), f_d(\Delta\mathbf{s}_{\theta 2}), ..., f_d(\Delta\mathbf{s}_{\theta t})\}$ is the set of d^{th} dimension of the observations on the real robot, $M_d(.)$ is the d^{th} dimension of prior mean from the repertoires such that $M_d(\Delta\mathbf{s}_\theta) = \Delta\mathbf{s}_\theta[d]$, σ^2_n is the prior noise, $\mathbf{K}_d$ is the kernel matrix with entries $\mathbf{K}_d[i,j] = k_d(\Delta\mathbf{s}_{\theta i}, \Delta\mathbf{s}_{\theta j})$ and $\boldsymbol{k}_d = k_d(D^d_{1:t}, \Delta\mathbf{s}_\theta)$. We use squared exponential kernel given by:

$$k_d(\Delta\mathbf{s}_\theta, \Delta\mathbf{s}'_\theta) = \sigma^2_{se} \exp(-\frac{||\Delta\mathbf{s}_\theta - \Delta\mathbf{s}'_\theta||^2}{l^2}) \tag{4.5}$$

Where, σ_{se} and l are hyperparameters. We initialize one GP model for each of the repertoires and train them iteratively as we collect more observations from the real robot during deployment.

4.3.4 Model-based Planning in the Presence of Multiple Priors

Once GP models are initialized, the robot is deployed in the environment. We assume that the main task/goal of the robot is sub-divided into a sequence of goals in the task-space by a high-level task-planner (e.g., path planning algorithm such as A*) and the robot has to achieve the first sub-goal by applying a suitable policy from one of the repertoires. For example, for a mobile robot, the main task is to reach a particular position in the room. Then the high-level planner will sub-divide this task into a sequence of sub-goals along the shortest path, avoiding the obstacles in the room. Note here that, since high-level task-planner gives the next sub-goal for the robot to achieve, it can also be replaced with a human operator giving high-level commands (such as move left, push object right, grab, etc.) remotely to the robot. At every time step, the high-level task planner re-plans the sub-goals according to the current task-space location of the system. Now, given the next sub-goal $\mathbf{s}_{g_t}$, the past observations $obs_{0:t-1}$ and the repertoires $\mathcal{R} = \{\Pi_0, \Pi_1, ..., \Pi_k\}$, we can frame the next policy selection problem as a maximum a posteriori (MAP) estimation problem as follows:

Let, π_θ be any elementary policy and $P(\Pi|obs_{0:t-1})$ be the probability of the repertoire Π to match the actual situation, given the past observations. Then,

the next elementary policy $\pi^*_{\theta_t}$ is given by

$$\begin{aligned}\pi^*_{\theta_t} &= \underset{\pi_\theta \in \Pi, \Pi \in \mathcal{R}}{\operatorname{argmax}} P(\pi_\theta | \mathbf{s}_{g_t}, obs_{0:t-1}, \Pi) P(\Pi | obs_{0:t-1}) \\ &= \underset{\pi_\theta \in \Pi, \Pi \in \mathcal{R}}{\operatorname{argmax}} \frac{P(\mathbf{s}_{g_t} | \pi_\theta, obs_{0:t-1}, \Pi) P(\pi_\theta)}{\sum_{\pi'_\theta \in \Pi', \Pi' \in \mathcal{R}} P(\mathbf{s}_{g_t} | \pi'_\theta, obs_{0:t-1}, \Pi') P(\pi'_\theta)} \\ & \quad P(\Pi | obs_{0:t-1}) \end{aligned} \tag{4.6}$$

Ignoring the denominator (being constant),

$$\pi^*_{\theta_t} = \underset{\pi_\theta \in \Pi, \Pi \in \mathcal{R}}{\operatorname{argmax}} P(\mathbf{s}_{g_t} | \pi_\theta, obs_{0:t-1}, \Pi) P(\pi_\theta) P(\Pi | obs_{0:t-1}) \tag{4.7}$$

Equation 4.7 gives the MAP estimation of the next elementary policy from the repertoires to be applied on the robot to achieve the current sub-goal $\mathbf{s}_{g_t}$ in one-step.

Now, $P(\pi_\theta)$ is the prior belief over the elementary policies. One option is to set it equal for all the policies in the repertoires. However, setting equal (positive) probability for the ones that have transition $\Delta\mathbf{s}_\theta$ in the neighborhood of the desired transition $(\mathbf{s}_{g_t} - \mathbf{s}_t)$ and setting zero for the others will improve the optimization time by eliminating the need to evaluate all the policies in the repertoires. One thing to be noted here that taking a very small neighborhood might degrade the performance of the algorithm.

$P(\mathbf{s}_{g_t} | \pi_\theta, obs_{0:t-1}, \Pi)$ is the Gaussian likelihood of the transition to $\mathbf{s}_{g_t}$ given the repertoire Π (i.e GP with prior mean function from Π), observations $obs_{0:t-1}$ and the elementary policy π_θ. This can be computed using the mean and variance prediction of the GP transformation model learned using $obs_{0:t-1}$ with the repertoire Π as mean-function. Here, the input to the GP is $\Delta\mathbf{s}_\theta$, which is the task-space transition corresponding to π_θ in the repertoire Π. To be more precise, if $\mu(\Delta\mathbf{s}_\theta)$ and $\Sigma(\Delta\mathbf{s}_\theta)$ are the mean and the diagonal covariance predicted by the GP model for an n dimensional task-space, then $P(\mathbf{s}_{g_t} | \pi_\theta, obs_{0:t-1}, \Pi)$ is computed as follows:

$$\begin{aligned} P(\mathbf{s}_{g_t} | \pi_\theta, obs_{0:t-1}, \Pi) &= \frac{1}{(2\pi)^{n/2} |\Sigma(\Delta\mathbf{s}_\theta)|^{1/2}} \\ & \exp\Big(-\frac{1}{2} (s_g - \mu(\Delta\mathbf{s}_\theta))^T \Sigma^{-1} (s_g - \mu(\Delta\mathbf{s}_\theta)) \Big) \end{aligned} \tag{4.8}$$

$P(\Pi | obs_{0:t-1})$ represents the likelihood of a repertoire being able to represent the reality for the robot. To compute this quantity, first we define a closeness score ψ_Π that represents how close the mappings of the repertoire Π are compared to real world observations.

$$\psi_\Pi = e^{-k||\Delta\mathbf{s}_\theta - \Delta\mathbf{s}_{\theta,real}||^2}, k > 0 \tag{4.9}$$

where, $\Delta\mathbf{s}_{\theta,real}$ is the observed transition on the robot after applying a policy π_θ taken from the repertoire Π and $\Delta\mathbf{s}_\theta$ is the corresponding transition stored in the repertoire. Now, for a real robot, $\Delta\mathbf{s}_{\theta,real}$ can be stochastic for a given policy. Moreover, for different policies from the same repertoire, ψ_Π can be

different. This makes ψ_Π stochastic in nature. Thus, the overall score of the repertoire can be defined as the expectation of ψ_Π. However, to compute a good estimate of the true expectation, it will require several observations from all the repertoires, which will make the adaptation process slow. On the other hand, imperfect estimation of the expectation of ψ_Π with small number of observations from the repertoires might make the algorithm greedy towards any repertoire that, by chance, has given higher ψ_Π for the selected policies. To have a balance between exploration and exploitation of these repertoires, we borrow the concept of *Upper Confidence Bound* (UCB) from the multi-armed bandits problem formulation [Sutton and Barto, 1998]. Thus, instead of estimating the expectation by taking the mean of the scores, we compute the UCB as follows:

$$\psi_{ucb\Pi} = \frac{\sum \psi_\Pi}{N_\Pi} + m\sqrt{\frac{\ln(n)}{N_\Pi}} \tag{4.10}$$

Where, n is the total number policies executed on the robot so far, N_Π is the number of times the policies were used from the repertoire Π and m is a positive constant. Note that since we use UCB1 [Auer et al., 2002], theoretically $m = \sqrt{2}$. However, we left m as a tunable hyperparameter of APROL for specifying the amount of exploration. Normalizing these UCB scores will give higher probability value to those repertoires which have higher "mean score" and to those repertoires that are not tried enough compared to others on the real robot:

$$P(\Pi|obs_{0:t-1}) = \frac{\psi_{ucb\Pi}}{\sum_{\Pi' \in \mathcal{R}} \psi_{ucb\Pi'}} \tag{4.11}$$

Combining everything, at every time-step, optimizing the equation 4.7 gives the optimal policy to be used for the current sub-task. Since our policy space is discrete and equation 4.7 is fast to evaluate (it does not involve the simulator), we can simply evaluate all the elementary policies π from all the repertoires to find out the optimal according to equation 4.7 (see Algorithm 3). It is to be noted here that in APROL there are two time scales: one for the elementary policies for executing the low level actions and one for the higher level planning algorithm for executing the policies for a "fixed duration" of time.

4.4 Experimental Results

We evaluate APROL on two simulated tasks: (1) object pushing task with a robotic arm and (2) goal reaching task with a damaged hexapod. Additionally, we demonstrate the effectiveness of APROL on a damaged 6-legged robot (hexapod) which has to reach the target position in an arena as quickly as possible by avoiding obstacles in the path. We evaluated the following baselines for both the tasks and compared the results to APROL with 40 replicates:

- CP-L (Close Prior with Learning): Using APROL with only one repertoire that is very close to the reality. For example, in the object pushing task, if the test object is a cube, then the repertoire used in this case can be of a

cuboid or a slightly larger or slightly smaller cube. For the hexapod task, the floor friction used for the repertoire can be close to the floor friction during test time. Another option is to use a repertoire for 2 blocked legs and at the test time hexapod has only one of those legs blocked. Since CP-L is basically APROL with the closest prior to the real situation, it is the best APROL can be expected to perform with multiple repertoires. So, we want APROL to perform as close as possible to the CP-L baseline in the experiments.

- SP-L (Single Prior with Learning): Using a randomly chosen prior repertoire for learning. Here, the chosen repertoire need not necessarily be closer to reality. In the Hexapod task, this baseline is exactly RTE since we used the author provided implementation of RTE. However, for the object pushing task, we used APROL with a randomly chosen repertoire. Thus, in this task SP-L is close to RTE in the sense that (1) both use a single repertoire, (2) both transform this repertoire according to the observed data using a probabilistic model and (3) incorporate the probability estimates to decide the next controller to be applied on the robot.

- SP-NL (Single Prior, No Learning of model) In this baseline, we use APROL with one randomly chosen repertoire for adaptation and ablate the learning of the transformation model. Without updating the transformation model using observation from test time, it assumes that the chosen repertoire perfectly matches with test object/damage (i.e., the reality).

- APROL-NL (APROL with No Learning): In this baseline, we use APROL with several prior repertoires. However, we ablate the learning of the transformation model. Thus, it assumes that one of the repertoires will perfectly match to the test object/damage. However, this assumption is not true since we do not include the repertoire that matches with the test object/damage.

All the simulated experiments were implemented in python[1]. For both the tasks, we used the pybullet physics simulation library [Coumans et al., 2013]. For comparison with RTE [Chatzilygeroudis et al., 2018] in the hexapod task, we used the author provided code[2]. For the GP model, we used the gpy library [GPy, 2012]. A video of the experiments is available online[3].

4.4.1 Object Pushing with a Robotic Arm

The goal here is to push objects of various shapes and sizes to different goal locations in a minimum number of steps. In this task, we assume that the robot has access to its model (so that it can be controlled in Cartesian space) and to the center and orientation of the object from a vision system (for instance, a QR code on the object). However, the robot does not have any knowledge

[1]APROL sourcecode: https://github.com/resibots/kaushik_2019_aprol
[2]RTE sourcecode: https://github.com/resibots/chatzilygeroudis_2018_rte
[3]Video: http://tiny.cc/aprol_video

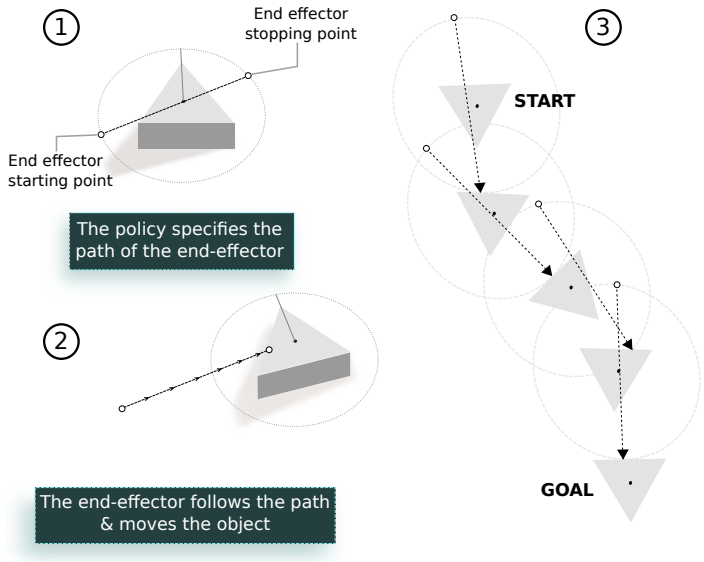


FIGURE 4.2: The elementary policy for the object pushing task: (1) A 2D vector specifies the start and end position of the end-effector on the surface around the object. The first element of the vector specifies the angular position of the starting point on a circle around the object relative to its current orientation. Similarly, the second element specifies the final end-effector position on the same circle. (2) During execution, the end effector follows the straight line connected by these two points using inverse kinematics of the arm. (3) The object can be moved to the goal position by sequencing multiple such policies.

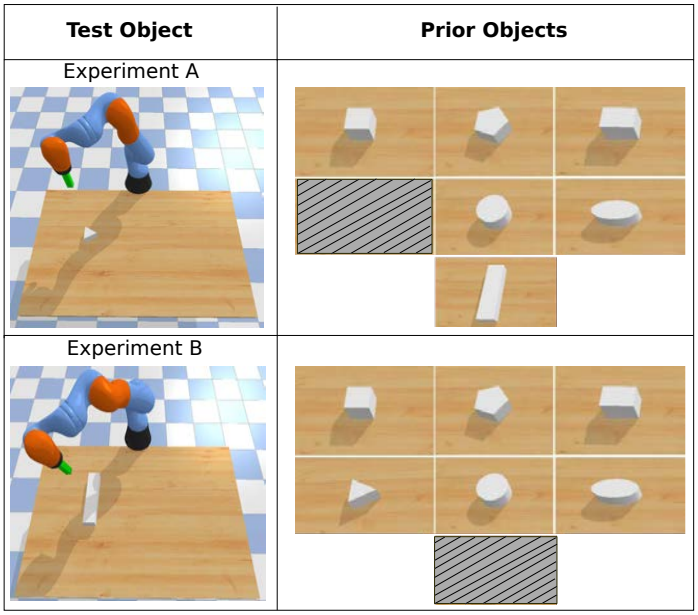


FIGURE 4.3: Priors used in object pushing task with APROL. Note that in test time with APROL, we do not use the exact repertoire (crossed objects in the image) that matches with the test object.

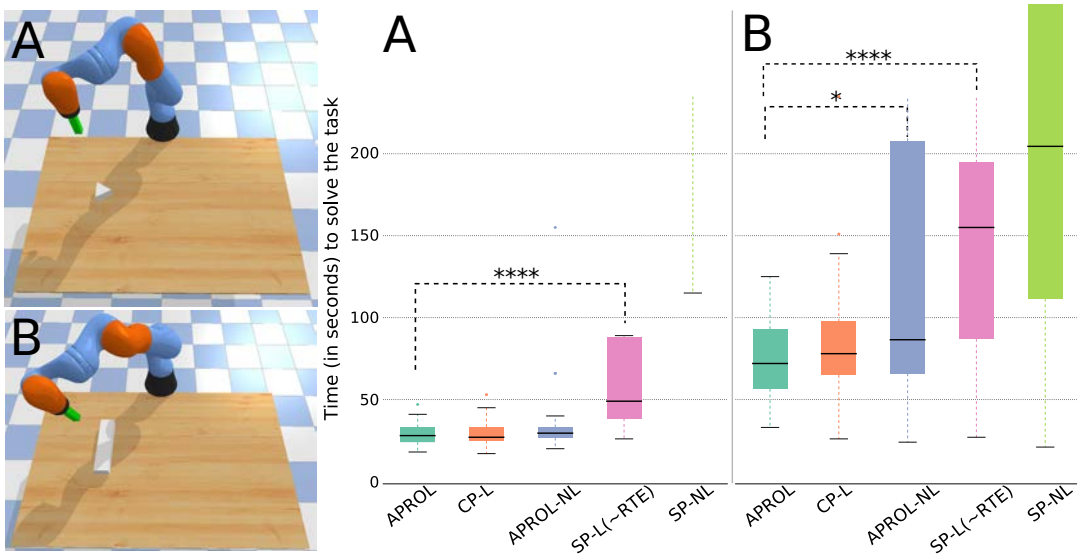


FIGURE 4.4: Comparison of APROL with various single prior and multi-prior variants on an object pushing task with a robotic arm. The goal is to push the objects of various shapes and sizes to different goal positions as quickly as possible. Here, CP: very Close Prior to the reality, SP: Single random Prior, L: with learning the transformation model using GP, NL: Not learning the transformation model with real observations and simply using the expected transitions stored in the repertoire. Asterisks in the plot represent statistical significance (p-value) using the Wilcoxon-Mann-Whitney test. For example, 4 asterisks represent $p < 0.0001$, 3 asterisks represent $0.0001 \leq p < 0.001$, 2 asterisks for $0.001 \leq p < 0.01$ and 1 asterisk for $p < 0.05$. The higher number of asterisks signifies higher statistical significance between two box plots. (A) With a triangular-shaped test object, APROL performs at least as good as using a very close prior to the test object (CP-L) and outperforms all the single repertoire based baselines. (B) With a bar-shaped test object, here APROL performs better than APROL-NL, which does not learn the transformation models with observed data. Also, APROL significantly outperforms single repertoire-based baselines.

about the shape and size of the objects. The objective is to adapt to push these objects of unknown shapes and guide them to the target position.

Elementary policy: We encode the elementary policy (open-loop controller) of the robot with 2 parameters in $[0, 1]$. These two parameters specify a straight line connecting two points around the center of the object, taking into account the orientation of the object. For a given set of control parameters, the robot's end-effector follows the line specified by the parameters in 2 seconds (see Figure 4.2).

Policy-repertoires: We pre-generated policy repertoires for 7 different objects (Figure 4.3) using the MAP-Elites algorithm in simulation. To evolve these repertoires, we did not assign any performance score for the policies. Since the goal of the task is to reach different positions on the 2D surface, therefore, the task-space is the 2D coordinates on the plane. In the repertoires, every policy corresponds to a unique task-space transition of the objects on this plane. *Note that we exclude the exact repertoire that matches with the test object in all*

the experiments except for CP (very close prior) variants. For this task, MAP-Elites evaluated 300,000 policies to generate each of the repertoires. Thanks to the cluster of computers, several repertoires could be evolved in parallel in approximately 5 hours of computation.

Execution: At every replanning-step (2 seconds), the robot uses the A* path planning algorithm to plan the shortest sequence of sub-goals in the task space to reach the goal. Then it attempts to achieve the first sub-goal by picking the optimal elementary policy from one of the repertoires using APROL. After every step, the robot updates its transformation models using all the past observations. We did not optimize the hyperparameters of the GP here. This is because GPs get only a few data points to learn the model, and for a small number of data points, hyperparameter optimization in GP often doesn't give good results. We set $\sigma_{se} = 0.03$ and $l = 0.3$ for the kernel function (Eq.4.5). These steps are iteratively performed until the object reaches its final goal.

Fig. 4.4A and Fig. 4.4B shows that APROL performs as good as that of using a very close prior (CP-L) to the reality, i.e., the test object. This shows that APROL is able to learn to adapt the transformation model according to the best prior quickly, which allows it to pick policies that align with the desired transitions on the task-space.

APROL outperforms the baseline APROL-NL (task B), which is the ablated version of APROL by removing the model learning phase. It shows the importance of the online adaptation of the repertoires in APROL using the learned GP models. Again, for both the test objects, APROL outperforms the single prior variants with and without learning the transformation models (SP-L, SP-NL).

4.4.2 Goal Reaching Task with a Damaged Hexapod

This task is performed both in simulation and on a real hexapod robot. In this task, the robot might encounter various situations, such as damage to one or more legs (e.g., blocked joints or a lost leg) and various friction conditions of the floor (e.g., very low, moderate and very high friction co-efficient). The goal here is to reach the specified position in a minimum number of control-steps. Here, the robot has the knowledge of its position and orientation (e.g., from SLAM or visual odometry system). Additionally, we assume that the robot has complete knowledge of the obstacle positions as well as dimensions. However, the robot does not have any knowledge about the damage to its legs and floor friction condition.

Elementary policy: The robot has 6 identical legs, each of which has 3 degrees of freedom (DOF). The first DOF (θ_0) controls the horizontal movement of the leg and the seconds and the third DOFs (θ_1 and θ_2) control the elevation of the leg. θ_2 is set equal to the negative of θ_1 always to make the final segment of the leg parallel to the body. Therefore, there are $6 \times 2 = 12$ independent joints in the robot. Each of these joints are controlled with 3 parameters: the amplitude, the phase, and the duty-cycle which are used in a periodic function of time to produce joint positions. These $3 \times 12 = 36$ parameters define the

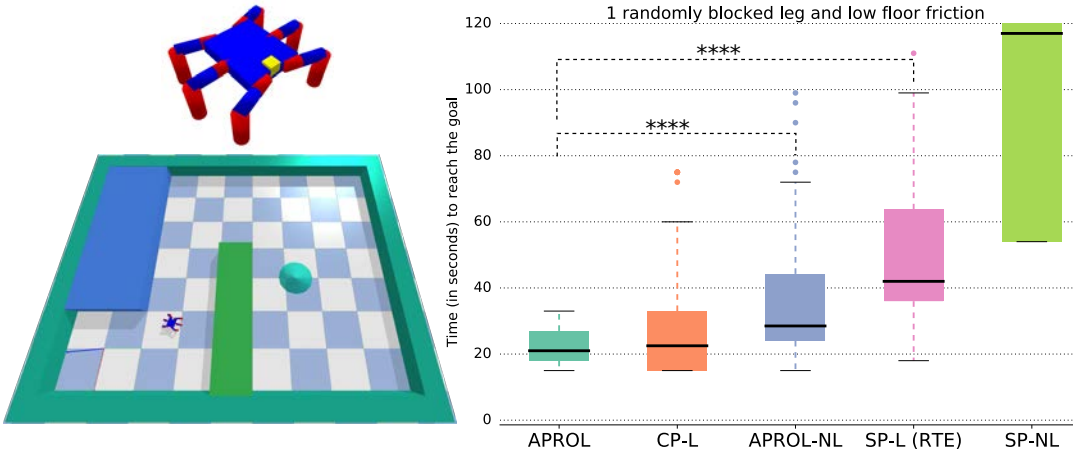


FIGURE 4.5: Comparison of APROL with different single prior and multi-prior variants on simulated hexapod goal reaching task. In this task, the hexapod has to recover from random leg damages and reduced floor friction to reach the goal as quickly as possible. Here, CP: very Close Prior to the reality, SP: Single random Prior, L: with learning the transformation model using GP, NL: not learning the transformation model with real observations and simply using the expected transitions stored in the repertoire. Asterisks represent p-value using Wilcoxon-Mann-Whitney test. For example, 4 asterisks represent $p < 0.0001$, 3 asterisks represent $0.0001 \leq p < 0.001$, 2 asterisks for $0.001 \leq p < 0.01$ and 1 asterisk for $p < 0.05$. Higher number of asterisks signifies higher statistical significance between two box plots. In this experiment, APROL performs as good as that of using a very close prior to the real situation (CP-L) and outperforms all the single repertoire based baselines.

elementary policy for the robot (see [Cully et al., 2015] for more details about this controller).

Policy repertoires: Before deployment, policy repertoires for various situations were generated for the robot using MAP-Elites in simulation. More precisely, we generated repertoires for 3 different friction coefficients $(0.6, 1.0, 5.0)$ of the floor and various leg damage conditions (single and two leg damages/blocks). Out of 108 possible combinations, we selected 57 combinations randomly to generate the repertoires for the hexapod. Each repertoire contains discrete mappings from 36D policy to 2D task-space transitions of the robot on the surface. Note that, in this task, the task space is simply the center of mass position of the robot. Thus, repertoires have transitions of the center of mass of the robot for different policies. For this task, MAP-Elites evaluated 500,000 policies in simulation to generate each of the repertoires. Thanks to the cluster of computers, several repertoires could be evolved in parallel within 12 hours of computation.

Execution: At every replanning-step (3 seconds), the robot uses the A* planning algorithm to plan the shortest sequence of positions that the robot has to follow to reach the goal by avoiding any obstacle in the path. Then it tries to reach the first sub-goal by selecting and executing the most suitable elementary policy from one of the repertoires using APROL. After every execution, the transformation model for the repertoire from which the policy has been selected is updated with all the previously observed data. Similar to the

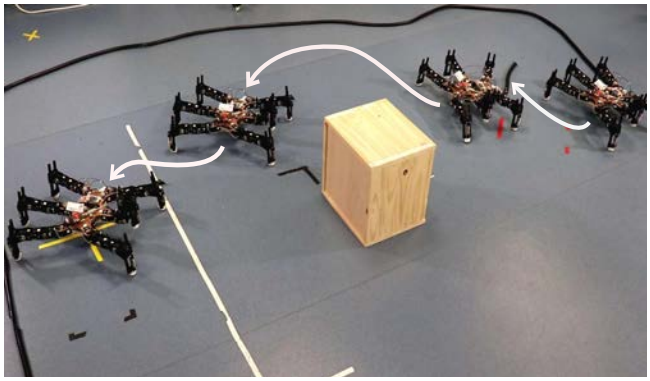


FIGURE 4.6: Goal reaching task with a real hexapod with blocked leg.

previous experiment, we did not optimize the hyperparameters of the GP here due to the small number of data. We set $\sigma_{se} = 0.03$ and $l = 0.3$ for the kernel function (Eq.4.5) in this task. After each execution (3 seconds duration), the robot re-plans the sequence of positions using A*. The process continues until the robot reaches the goal position.

In the simulated hexapod goal reaching task, we evaluated each of the variants with 40 replicates. Fig. 4.5 shows that APROL performs at least as good as that of using a very close prior (i.e., CP-L variant) to the reality (i.e., the exact leg damage and very similar floor friction). It is to be noted here that with APROL, we did not include the repertoire that exactly matches with the real situation of the robot. This suggests that APROL is able to quickly figure out the most suitable repertoire to use it as prior for learning the transformation model, and from which it accordingly selects the most suitable policies.

APROL outperforms the baseline that uses multiple priors but without any learning of the transformation models (APROL-NL). That is, in APROL-NL, we assume that one of the repertoires will exactly match the reality. However, since this assumption is not true, APROL performs better than this baseline. Additionally, APROL outperforms the single prior baseline with and without learning the transformation models (SP-L, SP-NL). Note that, SP-L is exactly RTE in this task.

Additionally, we have demonstrated the capability of APROL in a real hexapod damage recovery and goal reaching task (Figure 4.6) with 8 replicates. We show that, with APROL, the damaged robot learns to select the compensatory policies to reach the goal by avoiding the obstacle in the path. Here the robot reaches the goal positions by avoiding obstacle (a wooden box) with 100% success (within maximum 30 steps). Compared to the intact robot (with single repertoire generated for the intact robot itself), the damaged robot was able to recover its capability upto 88% in terms of the time it took to reach the final goal. One thing to be noted here that this was a challenging task not only due to the damage, but also due to the reality gap between simulation and the reality. As a result, in this task even the exact prior repertoire gives very high mismatch in the behavior when applied in the simulation and on the real robot.

4.5 Discussion and Conclusion

Prior knowledge is key to rapid adaptation, be it in repertoire-based learning, meta-learning, or model-based policy search. However, the effectiveness of the prior knowledge is highly dependent upon how relevant it is to the current scenario or situation that the robot is facing during deployment. In fact, a wrongly chosen prior might hinder or prolong the learning process instead. For example, a policy repertoire that is generated for a robot to walk on a flat surface is not a good prior for a robot that has to walk on stairs [Pautrat et al., 2018]. Unlike [Chatzilygeroudis et al., 2018], in this work, we relax the assumption that a single repertoire-based prior will be able to capture all the situations. Instead, we allow the robot to choose the best prior among many repertoire-based priors to achieve faster adaptation. It is to be noted that for episodic learning, [Pautrat et al., 2018] also reached a similar conclusion.

We believe APROL can find its application in many different fronts, such as fault tolerance or damage recovery in robotics, adaptation to sudden changes in environmental conditions, transferring controllers learned in simulation to the real robot, etc. For example, in case of robots deployed in places (e.g., space, deep sea, radio-active zones) where a human has to control them remotely with high-level commands (e.g., move in different directions, grab or push an object, etc.), the built-in controllers may not give the desired effect if any fault occurs in their joints. In such situations, using APROL, a robot can learn to use alternative controllers to accomplish the command. Again, for complex robots, a policy learned in simulation often gives slightly different outcomes on the real robot. In such situations also, APROL can learn to pick alternative policies to have the desired outcome.

In APROL, we execute the elementary policies for a fixed duration of time. However, this may not always be the best strategy. For example, the robot might already reach the goal location during the execution of the policy. In that case, instead of continuing the executing the policy it is better to stop the execution as the target is already achieved. Another scenario is when the robot deviates from the expected trajectory during the execution of the policy. In that scenario also, it is better to stop the execution of the policy and apply another policy that can keep the robot on the trajectory. Thus, allowing variable execution time for the policies can be a useful extension of APROL for practical applications.

Compared to model-based reinforcement learning [Deisenroth et al., 2015; Chatzilygeroudis et al., 2017], APROL is faster as it does not perform optimization on the policy parameter space. Instead, APROL learns to "select" the most suitable elementary policy from the given repertoires according to the current situation and the goal. However, APROL has to evaluate the outcomes of all the policies stored in the repertoires using the Gaussian processes model, which has cubic time complexity for training and quadratic time complexity for querying. Therefore, with more and more data points, the optimization of the policy between two re-planning steps of the robot becomes slower and slower. To mitigate this problem, we can incorporate several strategies. One option is that instead of learning the GPs from all the past observations, they can be learned

from M recent observations, where M can be a hyperparameter based on the desired optimization speed that we want to achieve. Learning from M recent observations will also allow the robot to continue its mission if multiple changes in the situation occur over time (e.g., plane surface, then very rough surface and then very slippery surface). Another option is to use sparse GPs [Quiñonero-Candela and Rasmussen, 2005] or local GPs [Park and Apley, 2017] or neural networks with uncertainty [Gal and Ghahramani, 2015]. Another important thing to be noted here that in this work, we assumed that the task-space is much smaller than the full state-space of the system. This assumption is true in most of the cases in robotics. We do not expect the algorithm to scale for problems with very high dimensional task-space. This is because the number of observations required to learn the models will grow exponentially with the task-space dimension causing adaptation time longer. However, we can expect APROL to scale well to problems with very high dimensional state-space as long as their task-space is smaller (e.g less than 10 dimensional).

There are several hyperparameters associated with APROL and most of them are linked to the MAP-Elites algorithm and GP models. One of the most important parameters that is associated with MAP-Elites is the amount of discretization of the task-space. Since we used the CVT variant of MAP-Elites [Vassiliades et al., 2017], we can specify the number of cells we want to generate after the discretization of the space. Now, a coarse discretization can be detrimental for APROL since there will be fewer elementary policies in the repertoire, and hence there will be less diversity. On the other hand, a very fine discretization will produce lots of cells in the space, and thus there will be more diverse policies in the repertoire, which will help APROL for better adaptation. However, a higher number of cells will require a higher number of total evaluations in Map-Elites. As a result, it will increase the time required to generate the repertoires in simulation. Again, since APROL evaluates potentially all the policies from all the repertoires at every re-planning step, a higher number of policies means higher optimization time. Another important decision in APROL is the number of repertoires. If the number of repertoires is small, then they will represent less diverse situations that the robot might face. As a result, if the real situation of the robot is not close to any of the repertoires, then APROL might not show any significant improvement over using just a single repertoire. On the other hand, taking a large number of repertoires might prolong the adaptation time. In the situation where the number of repertoires is very high, the exploration parameter m in the equation 4.10 will play a major role in deciding the adaptation time. A higher value of m will encourage more exploration of the repertoires, thereby prolonging the adaptation time.

One limitation of APROL (as well as repertoire-based learning such as RTE) is that instead of learning a transition model in full state-space of the robot, it learns a transformation model of the task-space assuming that task-space transition is independent of the current state of the robot. This assumption is not always valid (e.g., for the end effector of a robotic arm). However, in both our experiments, this assumption holds. This is because, for the hexapod, we reset the joints between each re-planning step of the robot. That makes the effective full state of the robot equal to position and orientation only, which are

independent of each other for the hexapod since our repertoires consider only the "change in position" from the current position. Similarly, for the object pushing task, we consider both position and orientation of the object, which is the same as that of the full state of the object. Thus, APROL is more suitable for mostly robot locomotion tasks as well as tasks where, to some extent, the above-mentioned assumption holds.

Despite using a finite set of policies for adaptation or learning, so far, repertoire-based approaches have been able to show many promising results in real robotic systems. Thanks to evolutionary algorithms such as MAP-Elites, the key element of such promising results is the diversity of the policies stored in the repertoires. As it happens in nature, due to this diversity, many such policies can still "survive" (i.e., work on the robot) even if any catastrophic event (such as joint failure) happens during the mission. We believe, repertoire-based adaptation algorithms such as APROL will open new frontiers in the direction of rapid adaptation for robotic systems in the real and uncertain world.

Chapter 5

Fast Online Adaptation through Meta-Learning Embeddings of Simulated Priors

The results and text of this chapter have been partially published in the following article:

Article:

- **Kaushik, R.**, Anne, T., & Mouret, J. B. (2020). *Fast Online Adaptation in Robotics through Meta-Learning Embeddings of Simulated Priors.* In **IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)** [Kaushik et al., 2020a].

Other contributors:

- Timothée Anne (Master Student)
- Jean-Baptiste Mouret (Thesis supervisor)

Author contributions:

- **RK** and JBM organized the study. **RK** wrote the code. **RK** and TA conducted the experiments. **RK**, TA and JBM analyzed the results and wrote the paper.

5.1 Introduction

From our discussions in chapter 2, it is clear that when data-efficiency is crucial for learning (such as in online adaptation scenarios) and it is possible to learn a useful dynamical model of the robot from the data, then model-based reinforcement learning (MBRL) algorithms can be a promising option [Chatzilygeroudis et al., 2020]. Unfortunately, the data requirement of MBRL algorithms typically scales exponentially with the dimensionality of the input state-action space [Keogh and Mueen, 2010; Chatzilygeroudis et al., 2020]. As a consequence, for a relatively complex robot, a typical MBRL algorithm still requires a prohibitive interaction time (from several hours to days) to collect enough data to learn a model of dynamics that is good enough for policy optimization [Chua et al.,

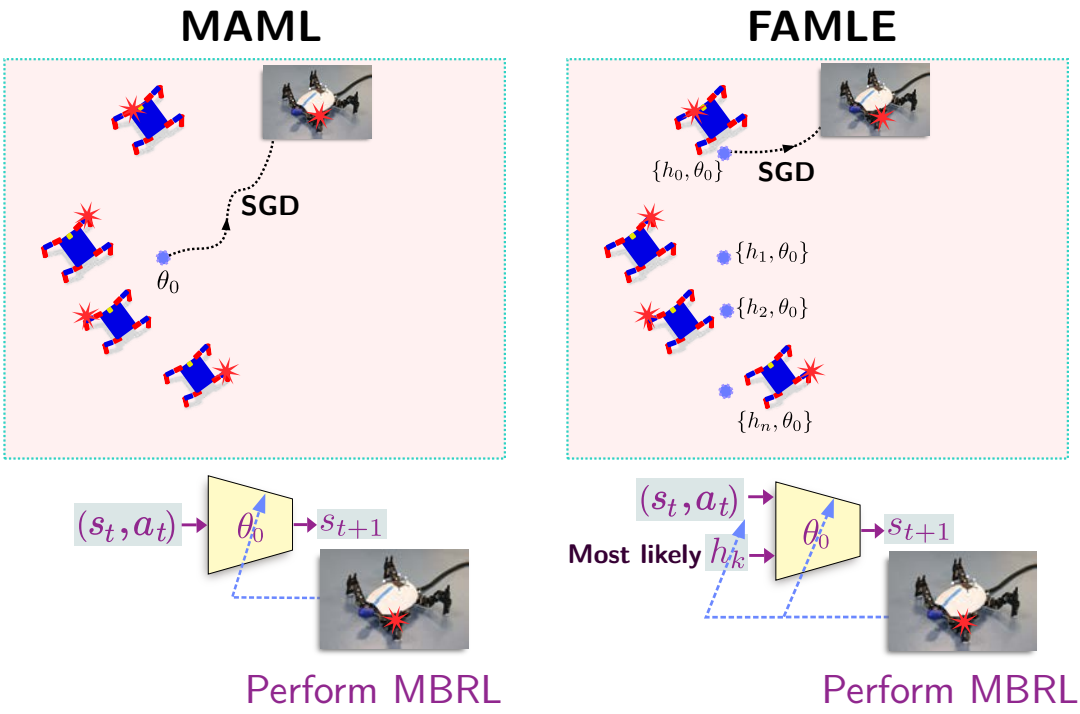


FIGURE 5.1: Basic overview: Compared to model-agnostic meta-learning (MAML) [Finn et al., 2017], FAMLE meta-learns several initializations for the dynamical model corresponding to various situations (e.g., damage conditions) in the simulation. In FAMLE, this is achieved by using a situation conditioned dynamical model, for which, both the initial situation embeddings ($h_{i=0:n}$), as well as the initial model parameters (θ_0), are jointly meta-trained in such a way that the model can be adapted to similar situations with only a few gradient steps. For model-based RL (MBRL) on the real robot, FAMLE figures out the most suitable embedding out of all the trained embeddings to adapt the model using the real world data.

2018; Nagabandi et al., 2018]. For example, using the MBRL approach, an 8-DoF simulated "ant" (actually a quadruped) required around 30 hours of real-time interaction to learn to walk [Nagabandi et al., 2018]. By contrast, we expect robots to adapt in seconds or, at worst, in minutes when they need to adapt to a new situation [Cully et al., 2015; Chatzilygeroudis et al., 2020].

Unlike robots, animals adapt to sudden changes (e.g., broken bones, walking for the first time on a snowy terrain) almost immediately. Such a rapid adaptation is possible because animals never learn from scratch; instead, they use their past experience as priors or biases to learn faster. Taking inspiration by this phenomenon, meta-learning approaches use past experiences of the robot in various situations to allow it to learn a new but similar skill or to adapt to a similar situation with only a few observations [Finn et al., 2017; Nagabandi et al., 2019]. When used with MBRL, a meta-learning algorithm such as MAML [Finn et al., 2017] finds the initial parameters for the dynamical model from where the model can be adapted to similar situations by taking only a few gradient steps. In other words, meta-learning exploits the similarity among the various past experiences to find out a single prior on the initial parameters.

When a robot has to adapt to many different situations (from broken limb to novel terrain conditions), the prior situations used to meta-train the dynamical model can be diverse and without any substantial global similarity among

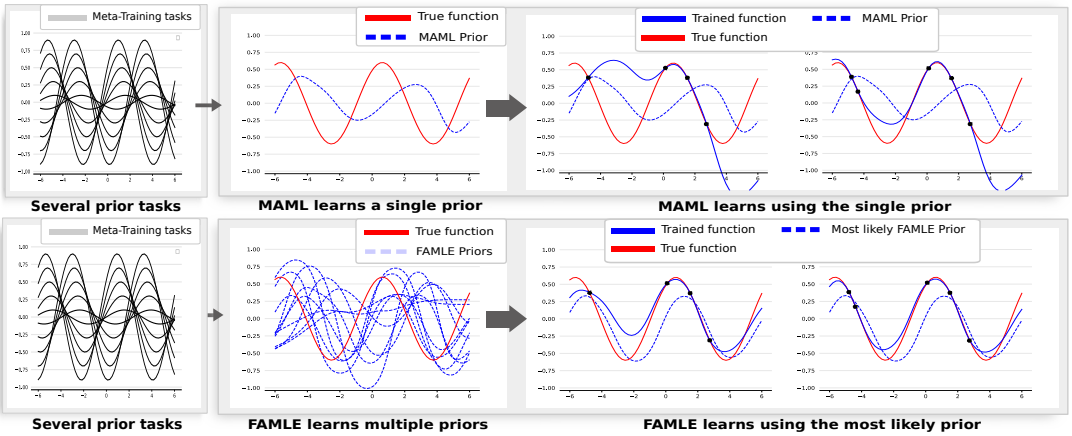


FIGURE 5.2: **Using FAMLE and MAML to learn a simple 1-dimensional sine wave.** Learning several meta-learned priors for the model and selecting the most suitable one for the given data (FAMLE approach) improves data-efficiency. The figure shows that using multiple priors, FAMLE could fit a better function (i.e., closer to the actual function) with 4 and 5 data points than using single priors learned using MAML.

themselves. For example, a 6-legged robot with a broken leg might experience a very different dynamics than the same intact robot walking on rocky terrain. In fact, we observe that when the prior situations are diverse and do not possess a strong global similarity among themselves, using meta-learning to find a single set of initial parameters for the dynamical model is often not enough to learn quickly.

One solution to this problem is to find several initial starting points (i.e., initial model parameters) that are meta-trained in such a way that when the model is initialized with the suitable one, the model can be adapted to the real situation of the robot by performing only a few gradient steps using the past observations. However, the question that arises here is how to meta-train several sets of initial parameters, while still generalizing to situations that were not in the training set. In this work, we propose to achieve this objective by using a single dynamical model that is shared among all the prior situations, but takes an additional input that is learned so that it corresponds to the situations, which makes it a situation conditioned dynamical model (see Fig. 5.1 and 5.2).

To be more precise, our conditional dynamical model not only takes the current-state and action as inputs but also a d-dimensional vector. This vector is called an embedding of the situation or simply *situation-embedding*. We consider the (initially unknown and randomly set) situation-embeddings as situation-specific parameters of the model, and jointly meta-train all these embeddings as well as the shared model-parameters. In effect, we obtain several meta-trained starting points for the model adaptation – one for each prior training situation (Fig. 5.2). On the real robot, first, we initialize the model with the meta-trained parameters, then we select the most suitable meta-trained embedding. With the selected embedding as input, we jointly update the embedding as well as the model parameters using gradient descent according to the recent data from the robot. It is to be noted that such joint training of embeddings and

model parameters is widely used to learn word embeddings in natural language processing [Collobert et al., 2011].

In summary, our main contribution is an algorithm called FAMLE (Fast Adaptation through Meta-Learning Embeddings) that combines meta-learning and situation embeddings to be able to adapt a dynamical model quickly to a new situation. This model can then be used for model-based RL to optimize its future actions for a given task.

FAMLE can be summarized in two steps (Fig. 5.3):

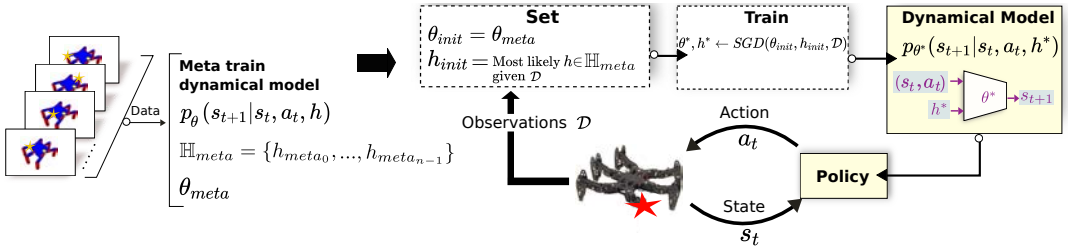


FIGURE 5.3: Overview of FAMLE: first, state-transition data is collected from simulations for n different situations of the robot. Then, the model (i.e., the dynamical model) parameters θ and situation embeddings $\mathbb{H} = \{h_0, ..., h_{n-1}\}$ are meta-trained on the simulated data. On the real robot, when the data is available, FAMLE uses the most suitable embedding from the set $\mathbb{H}$ to adapt the model to the current situation. FAMLE iteratively updates the meta-trained model parameters jointly with the selected embedding using new data and utilizes this model for model predictive control policy.

- **Meta-training:** Generate simulated data for N different situations of the robot, for example, with a broken joint, on rough terrain, on slippery terrain, and so on. Meta-train the model-parameters jointly with the embeddings for each of the simulated situations to have N meta-trained embeddings and one set of meta-train initial model parameters.

- **Online adaptation:** On the real robot, at every K step, initialize the model with the meta-trained parameters and the most likely situation embedding out of the N meta-trained embeddings based on the past M observations. Then update the model jointly with the embedding with gradient descent using the past M observations. This model is then used with a model-predictive-control (MPC) policy for K steps. The process repeats until the task is solved.

We compare FAMLE with (a) MBRL with a neural-network dynamics model which was pre-trained with model-agnostic meta-learning (MAML) (b) MBRL with neural-network dynamics model learned from scratch [Nagabandi et al., 2018], and (c) proximal policy optimization (PPO) [Schulman et al., 2017] on two simulated robots and one real physical quadruped robot. We show that FAMLE allows the robots to adapt to novel damages in significantly fewer time-steps than the baselines. Additionally, we demonstrate that using FAMLE, a physical Minitaur robot (an 8-DoF quadruped) can learn to walk in a minute of interaction in the real world.

5.2 Approach

FAMLE involves two steps: (1) meta-learning the situation embeddings and the dynamical model from the data gathered from simulation, and (2) adapting the model as well as the situation-embedding on the real robot for an unseen situation during the mission. In the following subsections, we elaborate these two steps.

5.2.1 Meta-learning the situation-embeddings and the dynamical model

We consider a predictive model $p_\theta(s_{t+1}|s_t, a_t, h)$ of the dynamics, where s_{t+1}, s_t, a_t and h are the current-state, the next-state, the applied action and the situation-embedding corresponding to the current situation of the robot. This is represented by a neural network $f_\theta(s_t, a_t, h)$ that predicts the mean of the distribution of the next state. In the real world, the robot might face any situation c, which comes from a distribution of situations $p(c)$. Since, in this work, we consider a *situation* as any circumstance that perturbs the dynamics of the system, so c represents any unknown dynamics of the system sampled from the distribution of dynamics $p(c)$. To collect the state-transition data from the simulation, we perform the following steps:

Create empty sets $\mathbb{C}$ and $\mathbb{D}$. Now, for $i = 1$ to N

1. Sample a situation $c_i \sim p(c)$ and insert it in the set $\mathbb{C}$, i.e., $\mathbb{C} = \mathbb{C} \cup \{c_i\}$
2. Instantiate a simulator of the robot for the situation c_i.
3. Perform n random actions on the simulated robot and create data-set $\mathcal{D}_{c_i} = \{(s_t, a_t, s_{t+1})|t = 1, \ldots, n\}$
4. Save the data-set into $\mathbb{D}$, i.e., $\mathbb{D} = \mathbb{D} \cup \{\mathcal{D}_{c_i}\}$

Then, corresponding to each sampled situation $c_{i=1:N}$, we randomly initialize situation-embeddings $\mathbb{H} = \{h_{c_i}|i = 1, \ldots, N\}$. Also, we randomly initialize the model parameter θ. Then the negative log-likelihood loss for any situation $c_i \in \mathbb{C}$ can be written as:

$$\mathcal{L}_{\mathcal{D}_{c_i}}(\theta, h_{c_{i=1:N}}) = \mathbb{E}_{\mathcal{D}_{c_i}}\Big[-\log p_\theta(s_{t+1}|s_t, a_t, h_{c_i})\Big] \tag{5.1}$$

Our meta-learning objective is to find initial model-parameters θ_{meta} and situation-embeddings $\mathbb{H}_{meta}$, such that for any situation $c_i \in \mathbb{C}$, performing k gradient descent steps from θ_{meta} and $\mathbb{H}_{meta}$ minimizes the loss given by equation 5.1. This objective can be written as a meta-optimization problem as:

$$\theta_{meta}, \mathbb{H}_{meta} = \arg\min_{\theta, h_{c_{i=1:N}}} \mathbb{E}_{c\sim\mathbb{C}}\Big[\mathcal{L}_{\mathcal{D}_c}\Big(U_c^k(\theta, h_c)\Big)\Big] \tag{5.2}$$

where, $U_c^k(\cdot, \cdot)$ is the gradient descent update rule (applied for k gradient descent steps) that updates the parameters θ and the situation-embedding h_c for any situation $c \in \mathbb{C}$. We optimize the above problem using meta-learning approach

similar to Reptile [Nichol et al., 2018] (as in Eq. 2.62). However, unlike Reptile update in equation 2.62), we update both θ and embedding h_{c_i} simultaneously. More precisely, at each update step, we randomly choose a situation c_i from the set of situations $\mathbb{C}$ and perform the following update on θ and embedding h_{c_i}:

$$\tilde{\theta}, \tilde{h}_{c_i} = U^k_{c_i}(\theta, h_{c_i}) \tag{5.3}$$

$$\theta := \theta + \alpha_{meta}(\tilde{\theta} - \theta) \tag{5.4}$$

$$h_{c_i} := h_{c_i} + \beta_{meta}(\tilde{h}_{c_i} - h_{c_i}) \tag{5.5}$$

where, α_{meta} and β_{meta} are the meta-learning rate. At convergence, we obtain the meta-trained parameters θ_{meta} and the set of situation-embeddings $\mathbb{H}_{meta}$ for each situation in the set $\mathbb{C}$. Combination of these N situation-embeddings and the meta-trained model-parameters will serve as N different priors for future adaption of the dynamical model to unseen situations.

Algorithm 4 FAMLE: Meta-learning

Require: $\mathbb{D} = \{\}$ ▷ Empty set of data-sets
Require: $U^k_c(\cdot,\cdot)$ ▷ k steps SGD update rule for situation c
for $i = 1, 2, ..., N$ **do**
  $c_i \sim p(c)$ ▷ Sample a situation
  $\mathbb{C} \leftarrow c_i$ ▷ Save the situation
  $\mathcal{D}_{c_i} = \{(s_t, a_t, s_{t+1}) | t = 1, \ldots, n\}$ ▷ Simulate and collect data
  $\mathbb{D} = \mathbb{D} \cup \{D_{c_i}\}$ ▷ Save the data-set for situation c
end for
Randomly Initialize θ ▷ Model parameters
Randomly Initialize $\mathbb{H} = \{h_{c_i} | i = 1, \ldots, N\}$ ▷ Situation embeddings
for $m = 0, 1, ...$ **do**
  $\mathcal{D}_{c_i} \sim \mathbb{D}$ ▷ Sample a data-set
  $\tilde{\theta}, \tilde{h}_{c_i} = U^k_{c_i}(\theta, h_{c_i})$ ▷ Perform SGD for k steps
  $\theta := \theta + \alpha_{meta}(\tilde{\theta} - \theta)$ ▷ Move θ towards $\tilde{\theta}$
  $h_{c_i} := h_{c_i} + \beta_{meta}(\tilde{h}_{c_i} - h_{c_i})$ ▷ Move h_{c_i} towards $\tilde{h}_{c_i}$
end for
Return $\theta, \mathbb{H}$ ▷ Return meta-trained parameters and embeddings

5.2.2 Online adaptation to unseen situation

As the robot might face any situation that can perturb its dynamics during the mission, we want to learn a new dynamical model after every K control steps using M recent observations. To learn this model, we set the meta-trained parameters θ_{meta} in the model and compute the likelihood of the M recent observations for each situation embedding in $\mathbb{H}_{meta}$. Then, we use the embedding that maximizes the likelihood of the recent data and train the model parameters as well as the selected embedding. To be more precise, if $\mathcal{D}_M$ is the recent M observations on the robot, then:

$$h_{Likely} = \arg\max_{h \in \mathbb{H}_{meta}} \mathbb{E}_{\mathcal{D}_M}\Big[\log p_{\theta_{meta}}(s_{t+1} | s_t, a_t, h)\Big] \tag{5.6}$$

Then we simultaneously update both model parameters $\theta = \theta_{meta}$ and the most likely situation-embedding $h = h_{Likely}$ with by taking k gradient steps:

$$\begin{aligned} \theta &:= \theta - \alpha \nabla_\theta \mathcal{L}_{\mathcal{D}_M}(\theta, h) \\ h_c &:= h_c - \beta \nabla_{h_c} \mathcal{L}_{\mathcal{D}_M}(\theta, h) \end{aligned} \tag{5.7}$$

After this optimization, we get the optimized model parameters θ^* and situation embedding h^* yielding the model $p_{\theta^*}(s_{t+1}|s_t, a_t, h^*)$. Now using this model, the model predictive control (MPC) method can be used as a policy to maximize the long term reward. In this work, we consider random-sampling shooting [Rao, 2009] to optimize the sequence of action using the model. This method is computationally faster compared to other sampling-based methods such as CEM [Botev et al., 2013] and relatively easy to implement as well as parallelize. Additionally, due to the randomness, it allows implicit exploration in state-action space, which helps to learn a better model. Random-sampling shooting has been successfully demonstrated as an action sequence optimization method for MPC in recent robot learning papers such as [Nagabandi et al., 2018]. At any state s, the next action for the robot is optimized as follows:

1. Sample N random trajectories of action where each action is sampled from a uniform distribution: $\{\tau_i | i = 0, ..., N-1\}$ and $\tau_i = (a_0^i, a_1^i, ..., a_{H-1}^i)$

2. Evaluate the trajectories on the model $f_{\theta^*}(s, a, h^*)$ and select the one that maximizes the total reward.

$$\tau^* = \arg\max_{\tau_i} \sum_{t=0}^{H-1} r(s_t, a_t, f_{\theta^*}(s_t, a_t, h^*)) \tag{5.8}$$

Where, $r(\cdot, \cdot, \cdot)$ is the reward function.

3. Apply the first action of τ^* and repeat from step (1) until the task is solved.

Pseudo codes for FAMLE is given in Algorithm 4 and Algorithm 5.

Algorithm 5 FAMLE: Fast adaptation & control

Require: $\theta_{meta}, \mathbb{H}_{meta}$ ▷ Meta-learned initial model parameters and embeddings
Require: $\mathcal{D}_M = \phi$ ▷ Empty set for M recent observations
Require: $r(\cdot, \cdot, \cdot)$ ▷ Reward function
while not *Solved* **do**
  h_{Likely} = most likely $h_{c_i} \in \mathbb{H}_{meta}$ given $\mathcal{D}_M$ and θ_{meta}
  $\theta^*, h^* = k$ steps SGD from $\theta_{meta}, h_{Likely}$ using $\mathcal{D}_M$
  Apply optimal action $a = MPC(\theta^*, h^*, r(\cdot, \cdot, \cdot))$
  $\mathcal{D}_M \leftarrow \mathcal{D}_M \cup \{(s_t, a_t, s_{t+1})\}$ ▷ Insert observation
  if $size(\mathcal{D}_M) > M$ **then** Remove oldest from $\mathcal{D}_M$
end while

5.3 Experimental Results

Here, our goal is to evaluate the data-efficiency of FAMLE and compare it to various baseline algorithms. As a metric for data-efficiency, we focus on the real-world interaction time (or time-steps) required to learn a task by a robot. So, a highly data-efficient algorithm should require fewer time-steps to achieve higher rewards in a reinforcement learning set-up. We compared FAMLE on various tasks against the following baselines and showed that FAMLE requires fewer time-steps to achieve higher rewards than the baselines:

- **PPO:** Proximal Policy Optimization, a model-free policy search algorithm, which is easy to implement, computationally faster, and performs as good as current state-of-the-art model-free policy search algorithms.
- **AMPC:** Adaptive MPC, i.e., MPC using an iteratively learned dynamical model of the system from scratch using past observations with a neural network model.
- **AMPC-MAML:** Adaptive MPC with a meta-trained neural network dynamical model. Here, the network is meta-trained using MAML for the same situations that are used in FAMLE. At test time, the model is updated using the recent data with meta-trained parameters as initial parameters of the network.

For all the MBRL algorithms, we used neural networks that predict the *change in current state* of the robot. To generate the data for the prior situations, we used the pybulet physics simulator [Coumans et al., 2013]. The code[1]and the video[2] of the experiments can be found online.

5.3.1 goal reaching with a 5-DoF planar robotic arm

In this simulated experiment, the end-effector of the velocity-controlled (10 Hz) arm has to reach a fixed goal as quickly as possible. The joints of the arm might have various damages/faults: (1) weakened motor (2) wrong voltage polarity, i.e., opposite rotation compared to a normal joint, and (3) dead/blocked motor. Here, the state-space and action space have 12 and 5 dimensions, respectively. The embedding vector size for FAMLE was 5. The dynamical models were learned using neural networks with 2 hidden layers of size 70 and 50. For AMPC-MAML and FAMLE, the dynamical models were meta-trained using the data collected from simulation for 11 different damage situations. For testing, two random damages were introduced on the arm, which were not in the meta-training set. Experiments were performed on 30 replicates, sharing the same test damage condition for all the replicates.

The plot (Fig. 5.4) shows that FAMLE achieves higher reward much faster than the baselines and solves the task (i.e., reward more than 30) in ∼500 steps (50 seconds). It can be seen that APMC-MAML could not show any significant improvement in the performance compared to AMPC, where the model was learned from scratch. Due to the large variations in the dynamics caused by different damage/fault situations in the meta-training data, meta-trained parameters using MAML could not generalize to all the situations. Thus, it required more data to adapt the model during test time to the current situation. As expected, model-free reinforcement learning baseline PPO could not reach the performance of the model-based approaches within the maximum time-steps limit.

[1] Code:https://github.com/resibots/kaushik_2020_famle
[2] Video:http://tiny.cc/famle_video

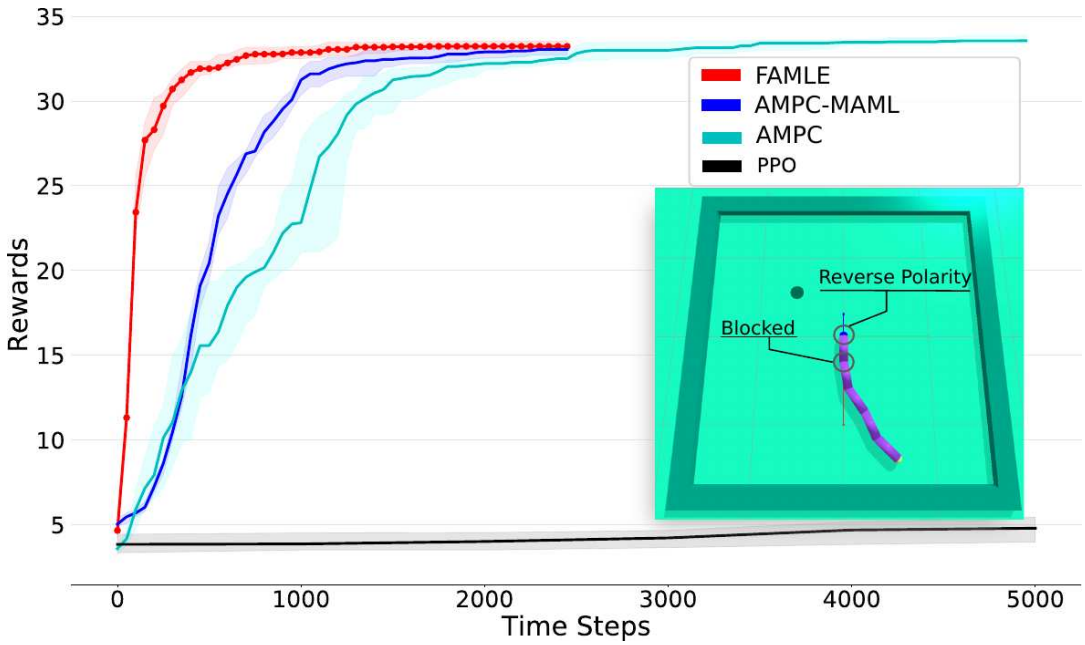


FIGURE 5.4: Goal reaching task: the 5-DoF planer arm has to reach target in minimum number of steps despite damage in its joints. Plot shows median, 25 and 75 percentile of the accumulated reward per episode for 30 replicates. Here, FAMLE maximizes the reward in fewer time-steps than the baselines.

5.3.2 Ant locomotion task

In this task, a 4-legged simulated robot (8 joints, torque-controlled, 100 Hz) has to walk in the forward direction as far as possible to maximize the reward. Here, the robot might have (1) blocked joints and/or (2) error in the orientation measurement (i.e., sensor fault). The state-space and the action-space for this problem are 27 and 8 dimensional, respectively. The embedding vector size for FAMLE was 5. The dynamical models were learned using neural networks with 3 hidden layers of size 200, 200 and 100. Meta-training data for AMPC-MAML and FAMLE was collected (applying random actions) from simulations for 20 different damage/fault situations of the robot. At the beginning of the test, a random joint damage and orientation error (not in the training set) were introduced.

The plot (Fig. 5.5) shows that FAMLE achieves higher reward than the baselines and solves the task (i.e., reward more than 800) in ∼25000 steps (4.17 minutes of real-time interaction). Similar to the goal-reaching task, here also due to the large variations in the dynamics caused by different situations (especially the orientation faults) in the meta-training data, meta-trained parameters using MAML could not generalize to all the situations and performed worse than FAMLE. As expected, in this experiment also, PPO could not reach the performance of the model-based approaches within the maximum time limit.

5.3.3 Quadruped damage recovery

In this online adaptation task, a physical quadruped (12 joints) has to recover from damage to its legs and/or faults in the orientation measurement and reach the goal as quickly as possible. Here, the action is a 4-dimensional vector that

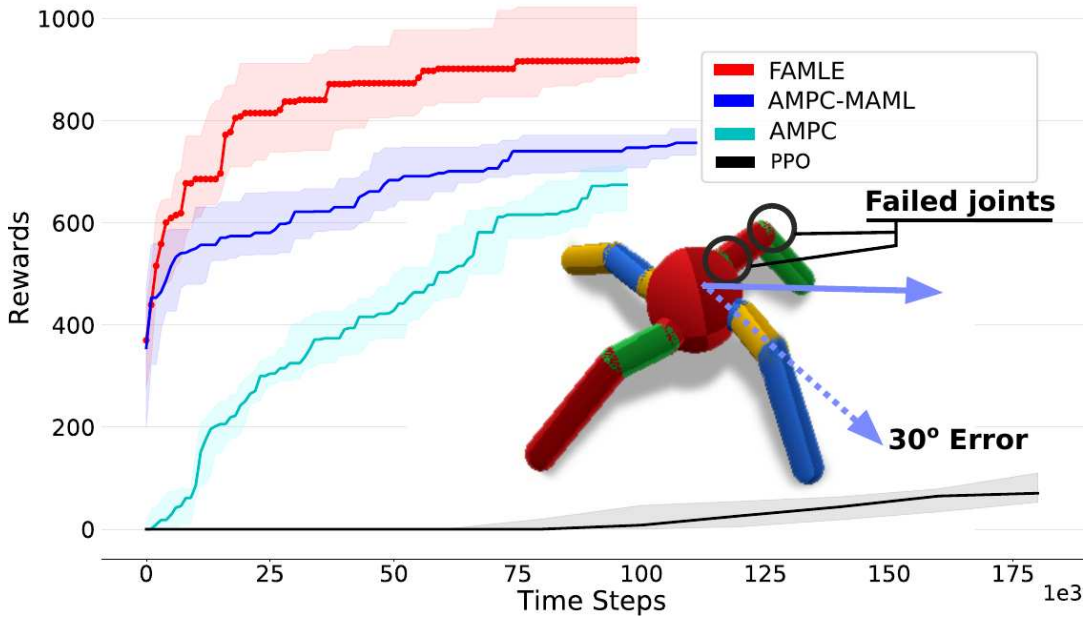


FIGURE 5.5: Ant locomotion task: the damaged ant has to move as far as possible in the forward direction. Plot shows median, 25 and 75 percentile of the accumulated reward per episode for 30 replicates. Here, FAMLE finds higher rewards in much lower time-steps than the baselines.

modulates the gait of the robot through the period functions associated with each leg[3]. At every second, a new action is applied that produces a new gait on the robot using a low-level controller. The state of the robot includes the 2D position and 2D orientation (sine and cosine of rotation along the vertical axis). The embedding vector size for FAMLE was 20. The dynamical models were learned using neural networks with 2 hidden layers of size 100. The meta-training data for the dynamical model was collected (by applying random actions) from a low fidelity simulator of the robot for total of 20 different situations, each of which includes either a joint block or orientation measurement error between 0 to 360 degrees. We tested 3 different situations on the real physical robot: (1) one blocked leg (2) orientation fault, and (3) one blocked leg as well as orientation fault. Additionally, we also evaluated the performance of FAMLE by introducing orientation fault on the robot online (during the deployment). In all the experiments, the goal was 2.5 meters away from the starting position of the robot.

The table 5.1 and box-plot 5.6 show the comparison of FAMLE with AMPC and AMPC-MAML baselines on the Quadruped damage recovery task. Results show that using FAMLE, the robot could reach the goal 100% of the time by taking significantly less time than the baselines. Using AMPC-MAML, the robot was able to reach the goal only 40% of the time within the maximum allotted time-steps of 80. On the other hand, using AMPC, the robot was never able to reach the goal within the maximum allotted time steps.

[3]In the preliminary experiments, we were unable to learn a full dynamical model of the physical robots that is accurate enough for MPC.

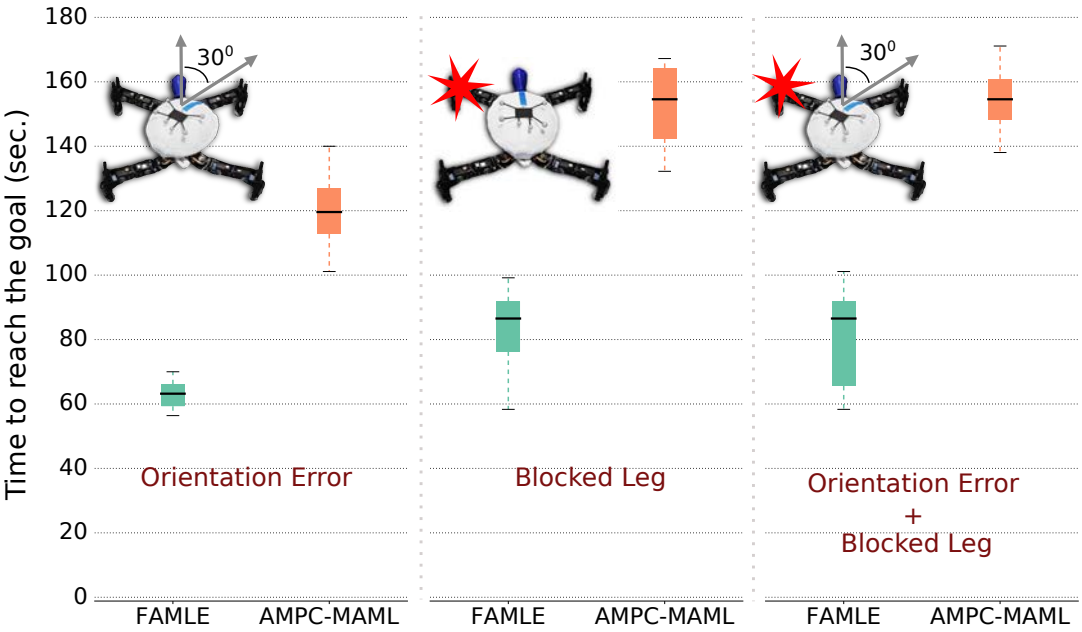


FIGURE 5.6: Quadruped damage recovery task: Box plot (over 10 replicates) shows time required to reach the goal location. Using FAMLE the robot could adapt to both orientation error as well as leg damage and solve the task in less than 2 minutes of interaction.

5.3.4 Minitaur learning to walk

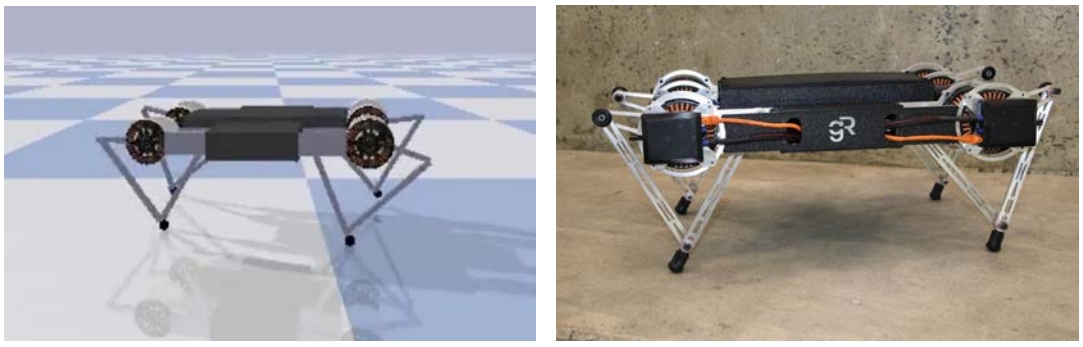


FIGURE 5.7: Thanks to meta-learning from the simulated situations using FAMLE the real minitaur learns to walk in the forward direction in 1 minute of interaction in real world.

In this experiment, we used the quadruped robot Minitaur from Ghost Robotics (Figure: 5.7). The goal here is to learn to walk in the forward direction as far as possible. Here, the state-space is 6-dimensional (center of mass position and orientation) and action is a 4-dimensional vector (applied at each second) that modulates the gait of the robot through periodic functions associated with each leg (see footnote 3). The embedding vector size for FAMLE was 8. The dynamical models were learned using neural networks with 2 hidden layers of size 20. For meta-training, we collected state transition data from the simulator of the robot in pybullet (with random actions) for 3 different friction

Table 5.1: Quadruped damage recovery task

ORIENTATION FAULT OF 30^0			
	Steps	Time (Sec.)	Success Rate
FAMLE	32.6 ± 2.3	63.4 ± 4.5	100%
AMPC-MAML	61.8 ± 7.2	120.1 ± 13.9	40%
AMPC	-	-	0%
ONE BLOCKED LEG			
	Steps	Time (Sec.)	Success Rate
FAMLE	43.1 ± 6.4	83.8 ± 12.4	100%
AMPC-MAML	78.3 ± 7.2	152.2 ± 14.0	40%
AMPC	-	-	0%
ONE BLOCKED LEG + ORIENTATION FAULT OF 30^0			
	Steps	Time (Sec.)	Success Rate
FAMLE	41.6 ± 7.7	80.9 ± 15.0	100%
AMPC-MAML	79.5 ± 6.1	154.6 ± 11.9	40%
AMPC	-	-	0%

Table 5.2: Minitaur learning to walk

	Max distance	Interaction time
FAMLE	4.8 meters	60 seconds
AMPC-MAML	1.6 meters	90 seconds
AMPC	0.3 meters	130 seconds

conditions (default, 0.5 times and 2 times of the default friction) and 3 different weights of the base of the robot (real weight, 1.5 times and 2 times of the real weight).

Due to the high reality gap between the simulated robot and the real robot, a dynamical model trained directly on the data collected from the simulator (for the default weight and friction) performs poorly on the real robot. To verify this, we used such a model on the Minitaur in the simulator as well as on the real robot. On the simulated robot, the robot could immediately walk in the forward direction using model predictive control. However, on the real robot, the robot could not move and failed due to exceeding current limits in the motors. Now, to evaluate the meta-trained model using FAMLE, we used it on the real robot. Thanks to the meta-trained embeddings, the robot could quickly figure out the most suitable embedding to update its model from the real observations. With FAMLE, using only 60 data points (1 minute of real interaction) from the real robot, the robot could consistently walk forward without fail (see Table 5.2).

5.4 Discussion and Conclusion

The ability to adapt rapidly to unforeseen situations is one of the main open challenges for robotics. One of the key components to achieve such rapid adaptation is the effective use of prior knowledge. In this work, we have introduced

an algorithm called FAMLE and showed how a robot can use prior knowledge from various simulated situations, such as joint damages, sensor error, or floor conditions, to adapt to various unforeseen situations faster than existing approaches in the literature.

Although FAMLE seems to be a promising approach for fast online adaptation in robotics, there are several scopes for improvement. First, our work did not cover the question of how to decide the simulated prior situations for meta-learning so that the robot can adapt to diverse situations in the real world, and second, how many prior situations are required for effective adaptation in the real world. Intuitively, the more the number of prior situations for meta-training the dynamical model of robot, the better will be the effectiveness of FAMLE for fast online adaptation in the real world. However, there should also be some diversity among the meta-training situations so that the dynamical model can easily generalize to diverse situations in the real world. In this direction, quality-diversity algorithms such as MAP-Elites [Cully et al., 2015; Mouret and Clune, 2015] can be a promising option to generate these diverse set of prior situations.

In this work, we have not used probabilistic models to learn the dynamics of the robot. As discussed in the background (chapter 2), in micro-data scenarios, probabilistic models can be a better option while optimizing the policy. In future work, FAMLE can be improved by incorporating probabilistic models such as probabilistic neural network ensembles [Chua et al., 2018]. Overall, we believe FAMLE is a promising direction and further investigation in this direction can open interesting frontiers of research towards adaptive robots for long-term missions in the real and uncertain world.

Chapter 6

Discussion

In this manuscript, we have introduced three different methods for data-efficient learning through the trial-and-error approach. These algorithms share the common framework of model-based robot learning, i.e., iterative learning of a model and using it for policy optimization. Yet, there are some key aspects to each of these algorithms which must be considered before applying these algorithms to a specific problem.

First, in the chapter 3, we have introduced a model-based policy search algorithm, Multi-DEX, where we iteratively learn the model of the dynamics of the robot from the previously observed data using the Gaussian process. As a result, this approach is effective only when the state-action space is small enough to learn a useful dynamical model from a small number of data. Despite using a probabilistic model to avoid model imperfection (due to the small number of data) during optimization, the curse-of-dimensionality will become more and more dominant with large state-action spaces. Thus, in all the experiments, the sizes of our state-action spaces are not more than 9.

To counter the curse-of-dimensionality in the model-based trial-and-error learning that we have encountered in Multi-DEX, in the chapter 4, we have introduced APROL. Since in APROL, we learn a task-space transformation model instead of the full dynamical model of the robot, the dimensionality of the model becomes significantly smaller even for complex robots with several DOFs. Besides, to accelerate the model learning in online scenarios, we used priors, where each prior corresponds to a repertoire of policies evolved in a simulator to produce diverse behavior under different situations (e.g., damages or terrain condition). The robot selects the most suitable prior online to adapt the model and control the robot effectively. Although this approach can be used for data-efficient online learning for problems with large state-action spaces (e.g., 12-DOF for the hexapod, full state-action space dimension is more than 30), APROL is not flexible to use it on any robot without any robot specific modification. For instance, for a manipulator, it is not straight forward to learn a model in the task-space (i.e., the 3D coordinate space of the end effector) without any experts' knowledge about the task and the robot.

Despite the lack of flexibility, the use of multiple priors in conjunction with model-based learning in APROL seems to be a promising idea. Proceeding in this direction, in chapter 5, we introduced FAMLE, a model-based online learning algorithm that uses multiple priors from the simulator to accelerate the model learning in the full state-space for problems with large state-action

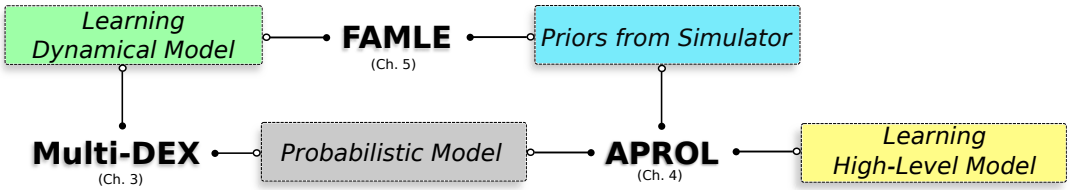


FIGURE 6.1: Shows how our algorithms are connected.

spaces. In addition, FAMLE uses an adaptive model predictive control framework, which makes it flexible to the task specification. Figure 6.1 shows how Multi-DEX, APROL and FAMLE are connected to each other.

Below, we discuss the limitations of the various components of our algorithms in detail and explore different ways to improve them in the future.

6.1 Learning the Dynamical Model from the Observations

In this manuscript, we have seen that learning a dynamical model of the system from the past observations and using it for policy optimization significantly improves the data-efficiency in reinforcement learning. Once the dynamical model of the system is available, optimization of the policy can be done on this model, which significantly reduces the real interaction time with the physical system. In the chapter 3 and 4, we have seen that learning the dynamical model using Gaussian process regression allows us to estimate the prediction uncertainty, which in turn allows us to better optimize the policy with a small number of samples from the robot. However, as we have discussed in chapter 2, the Gaussian process regression has very high computational complexity for training as well as prediction. Thus, computation time grows with the number of episodes when a GP is used in a model-based policy search framework. In particular, the increasing computation time becomes a significant concern in sparse reward scenarios where a robot might have to collect a lot of data while exploring the state-action space, as we have seen in chapter 3. Similarly, in the repertoire-based learning approach in chapter 4, we have seen that the robot keeps learning the task-space transformation model using GPs during its deployment. As a result, in long time deployment, optimizing the action will require more and more computation time at every step, making real-time applications infeasible for long-term deployment missions.

In the literature, several models have been proposed to approximate the Gaussian process [Liu et al., 2020]. One of the simplest strategies to reduce the time complexity of the GP model is to use a subset of the data (SoD) instead of using the full data-set. Several strategies can be used to select the subset fro the full data-set; for instance, random sampling from the data-set, performing clustering on the data-set, employing active learning criteria such as differential entropy [Herbrich et al., 2003], information gain [Seeger, 2003] or matching pursuit [Keerthi and Chu, 2006], to sequentially query data-points up-to desired computational complexity. As mentioned in the chapter 3, we

used an SoD strategy where we keep only the most recent data points and the data-points corresponding to the high rewarding trajectories.

Another common approach to reduce the computational complexity in GPs is to use pseudo-inputs instead of the real data, called sparse pseudo-input Gaussian process (SPGP) model [Snelson and Ghahramani, 2005]. In SPGP, the covariance of the N data-points GP is parameterized using M ($M < N$) pseudo-inputs, which are learned by a gradient-based optimization method. This approximation reduces the training complexity to $\mathcal{O}(M^2N)$ and prediction complexity to $\mathcal{O}(M^2)$. Nevertheless, since this is just an approximation of the full GP model, the value of M needs to be decided based on the trade-off between the accuracy and the time complexity of the model.

Nevertheless, for a life-long learning robot, using Gaussian processes does not seem to be a scalable option, even with sparse approximations that we mentioned above. To achieve constant time-complexity in prediction, the most promising option is to use neural networks based models that can estimate model prediction uncertainty. The Bayesian neural network is one such option [Neal, 2012] that sets a distribution over the parameters of the network. A computationally more efficient approach to approximate the Bayesian neural network is the *dropout method* [Gal and Ghahramani, 2015]. Recently, Gal et al. [2016] used a deep neural network with dropout to improve the computational efficiency of the model-based policy search algorithm, PILCO [Deisenroth and Rasmussen, 2011]. Similarly, probabilistic neural network ensembles can also be a computationally efficient alternative of GP for learning the dynamical model of the robot [Chua et al., 2018].

Although the neural-network-based models are more efficient compared to GP, the element that makes GP more interesting is the ability to incorporate a prior over the underlying function. A recent approach called *noise contrastive priors* [Hafner et al., 2018] can allow neural networks to have a prior over the function. The basic idea in this approach is to introduce additional data points (noise contrastive priors) within the distribution of the training data according to the underlying prior-function. These additional data points are assigned higher variance value compared to the training data. Now the network (that outputs mean and diagonal covariance) is trained for all the data points using KL divergence cost. As a result, the network tries to fit its mean and variance closer to the training data in the regions where the training data is available. Otherwise, the network tries to fit the noise contrastive priors (the additional data introduced).

6.2 Model-Learning in Open-Ended Scenarios

Another important aspect while learning the dynamical model is to decide *how much of the environment we have to include in the model*. For example, in the drawer opening task described in chapter 3, we included the displacement of the drawer as a state variable of the system while learning the model. Now, if the problem changes for the same robot (e.g., pushing an object to a given location), we need to change the state variables accordingly, as we now need the state of the object in the dynamical model. Thus, when the problem changes

even for the same robot, the state description of the system has to be changed accordingly. A general-purpose robot capable of solving an unbounded number of tasks cannot rely on state representations hardwired at design time, because each may require a different state representation. The ability to discover new representations based on existing ones is called representational redescription, a key ability of human intelligence [Karmiloff-Smith, 1994] that remains as a challenge in robotics Doncieux et al. [2018]. This is a challenge towards the fully autonomous application of model-based learning in the real world, where a robot has to solve a variety of problems, instead of solving only one problem.

One option is to include all the sensory information about the surroundings (e.g., image from the camera or LIDAR sensor data) in the state-space of the system to learn the dynamical model. However, it will make the state-space much bigger than the robot learning tasks that we have discussed in this manuscript, which will make the learning problem exponentially data-hungry (due to the curse-of-dimensionality). Thus, further investigation is required in this direction to include such a high dimensional sensor data in the dynamical model of the system for data-efficient robot learning. In particular, the meta-learning algorithm and repertoire-based learning that we have discussed in chapter 4 and chapter 5 can be promising directions to ease the model learning process for such a high-dimensional dynamical model.

Towards compressing the high dimensional sensory information for model-learning, a promising direction is to let the robot learn an abstract representation of the state (much lower-dimensional compared to the sensory information) that is relevant to the problem [Lesort et al., 2018]. Doncieux et al. [2018] proposed a framework towards open-ended problems in reinforcement learning, where the agent discovers the state and action representations autonomously that let it cast the tasks as Markov decision process in order to solve them by reinforcement learning. Nevertheless, such representation learning must be fast enough to be useful for online learning scenarios.

6.3 Repertoire-based Learning

In chapter 4 we introduced a repertoire-based robot learning algorithm called APROL that uses multiple pre-generated repertoires of elementary policies to adapt to a novel situation in the real world. APROL uses several repertoires of elementary policies - each for a specific situation of the robot (such as damage) simulated in a low fidelity simulator. During online learning, the robot optimizes the sequence of elementary policies to be applied to solve the given task.

Although repertoire-based learning has shown many promising results on real robots, one shortcoming of this approach is that for many robots, it is either not possible to learn an accurate model in the task-space or require problem specific modification to the model. For instance, the task space for a robotic manipulator can be the 3D coordinate location of the end-effector. Then it is not possible to predict the next position of the end effector based on the current position of the end-effector when a specific action (e.g., torque or position) is applied to the joints. In this case, the next position of the end-effector also depends on the current joint state of the arm. On the contrary,

for the hexapod, it is possible learn the model in the task-space as we reset the joints of the legs to the initial position after each control-step on the robot.

Another limitation of APROL is that it uses the MAP-Elites algorithm, which requires a discretization of the outcome-space or task-space of the robot. Now the effectiveness of APROL depends on how finely the discretization is performed. More fine discretization allows the repertoire to store more elementary policies, which should increase the effectiveness of APROL. However, a finer discretization will cause exponentially higher space and time complexity for the MAP-Elites (offline optimization) algorithm as well as the APROL (online optimization). Thus discretization decision presents a trade-off between the effectiveness and the computational performance of the learning algorithm.

In many situations, using discrete elementary policies may not give fine-grained control of the system. For instance, in the object pushing task in chapter 4, it is less likely to find a sequence of policies that puts the object *exactly* on the target location if the discretization of the repertoire is not fine enough. Thus, for problems that require precise control of the robot, repertoire-based learning may not be an effective solution.

One option to avoid the discretization of the task-space is to evolve unstructured collection of elementary policies through another quality-diversity algorithms [Lehman and Stanley, 2011b; Cully and Demiris, 2017] instead of Map-Elites. Like MAP-Elites, in this approach also, an archive is used to form the collection of solutions by substituting solutions when better ones are found. However, in contrast to MAP-Elites, the outcome space here is not discretized and the structure of the collection autonomously emerges from the encountered solutions.

6.4 Using Priors from the Simulator

In chapter 4 and chapter 5 we proposed how multiple priors coming from simulations of various situations can be used to allow a robot to adapt to various un-anticipated situations during its deployment rapidly. Although these approaches have shown promising results to fast and data-efficient robot learning and adaptation in the real world, there are several questions that are not investigated in this manuscript such as

1. How many prior situations will be enough for future adaptation?
2. Can we allow the algorithm to decide which prior situations should be simulated for better and faster adaptation in the real world?
3. Instead of keeping all the simulated priors (i.e., the repertoires or the meta-trained embeddings), can we compress the prior knowledge into only a few priors for future adaptation?

In this section, we discuss these questions and propose some future directions to answer these questions.

The number of simulated priors is an important parameter both in APROL (chapter 4) and in FAMLE (chapter 5) as it can potentially impact the computation performance as well as the effectiveness of these algorithms. For instance,

if the number of prior situations that we simulate is not large enough, then these simulated priors may not be able to generalize to a wide range of future situations that the robot might face in the real world. Similarly, diversity among the prior situations is also an important aspect for effective generalization to the future situations of the robot during its deployment.

While using a large number of simulated prior situations is desirable for effective adaptation of the robot to different situations in the future, storing this prior knowledge (as repertoires or as meta-trained embeddings and model parameters) requires not only a large amount of space but also increases computation time. With a large number of simulated situations, it will require more computation time to generate the repertoires (APROL in chapter 4) or to meta-train the model (FAMLE in chapter 5). Besides, since in APROL tries to figure out the most suitable repertoire according to the current situation of the robot by balancing between exploration and exploitation of the repertoire, a large number of repertoires means more exploration time. As a result, the adaptation time for the robot will be longer. Similarly, in FAMLE also, simulating a large number of prior situations will produce the same number of meta-trained models, which in turn will increase the computation during the adaptation.

One potential approach to produce a *diverse set of prior situations* for the robot is to use a novelty search algorithm [Lehman and Stanley, 2011a; Doncieux et al., 2019]. In novelty search, the idea is to search for solutions that are different from the previous solutions, without considering their quality. This is achieved by optimizing a "novelty score" that characterizes the difference of a solution compared to those already encountered, which are stored in a "novelty archive". In the context of generating prior situations for the robot, our novelty score can be the measure of how different the dynamics of the robot is compared to already archived situations in the novelty archive. Once a diverse set of situations is evolved through novelty search, these situations can be simulated to generate the repertoires for APROL or to meta-train the dynamics model and the embeddings in FAMLE. In addition, the novelty search algorithm can allow us to generate a pre-specified number of prior situations that are as diverse as possible among themselves. Thus we can have the freedom to decide the number of prior situations we want to simulate depending upon the computational resources available to us.

6.5 What is Next?

In this thesis, we have seen two core ideas: continuous learning of the model and using the priors from simulators to accelerate the model learning process. We believe continuous self-modeling is one of the essential components for adaptive robots in the future. And, to adapt this model quickly to new situations, priors must be used in the learning process. In this regard, simulators can provide a flexible source to derive the prior knowledge without involving human experts for each problem specification. In this thesis, the generation of the priors and the learning on the real robot are two separate processes. However, with faster hardware and improved simulation capability, we can think about coupling these

two processes, where the robot will be generating its own priors continuously while exploring the world.

Another observation is that learning a high-level model (e.g., in APROL) makes the model-based learning approach simpler and faster. Although to learn such models, we need to define the high-level actions based on the problem; in the future, however, we can imagine approaches that can allow a robot to define these high-level actions autonomously in online learning scenarios.

Compared to deep reinforcement learning algorithms that rely on large datasets, our algorithms operate on the other end of the spectrum, where the primary goal is to minimize the requirement of the real-world data. However, we do not restrict the interactions that happen in the simulation while generating the prior knowledge. In our work, we extensively used evolutionary algorithms for generating the prior knowledge and optimizing the policies. More specifically, in Multi-DEX and APROL, we have brought together the literature of quality-diversity evolutionary algorithms and model-based reinforcement learning. We see evolutionary algorithms in conjunction with simulators as an interesting frontier of research towards adaptive robots in the real-world.

Chapter 7

Conclusion

In this thesis, we introduced methods to allow robots to learn new skills or to adapt to new situations in a data-efficient manner through trial-and-error learning approach. Our primary focus was on the scenarios where learning has to happen within a few minutes, if not seconds, of interaction in the real world. To reduce the real-world interaction while learning, we used the model-based learning framework, i.e., iteratively learning a model of the dynamics of the robot from past observations, and using this model to optimize either the policy or its future actions.

The first challenge in using a model-based learning approach to achieve data-efficiency is the scarcity of data itself. Without a sufficient amount of past observations, the model will be erroneous, and thus the model will be less effective towards learning the policy or optimizing the action. So, in our first contribution, Multi-DEX (chapter 3), we used a probabilistic model to learn the dynamical model of the robot so that the model gives not only the predictions but also the associated uncertainty of the prediction. In Multi-DEX, we addressed the problem of data-efficient learning in sparse reward scenarios by combining model-based policy search with Novelty Search approach and Pareto-based multi-objective optimization approach. Since Multi-DEX uses a black-box optimization approach, it does not impose any restriction on the type of the policy representation and the reward function. We experimentally showed that Multi-DEX outperforms the state-of-the-art model-based and model-free policy search algorithms in sparse reward scenarios by finding superior policies in order of magnitude less interaction time. In addition, we showed that Multi-DEX can be competitive to the state-of-the-art model-based policy search algorithm in non-sparse reward scenarios also.

Like any data-driven learning approach, model-based trial-and-error learning is also prone to "curse-of-dimensionality"; the larger the size of state-action space, the higher will be the interaction time. We address this in our second contribution, APROL (chapter 4) by incorporating a repertoire-based learning approach that learns a model in the task-space of the robot, which is often much lower-dimensional compared to the full state-space. The key innovation in APROL is the use of *multiple* policy-repertoires (the priors) evolved in a simulator of the robot corresponding to various situations (e.g., damage conditions, or terrain conditions, etc.). During online learning, APROL allows the robot to select the most suitable repertoire for faster adaptation and control. Using APROL, we showed that a hexapod (12-DOF) can adapt to damages

in the real-world and reach its target much faster than single repertoire-based baselines, such as RTE [Chatzilygeroudis et al., 2018].

While using multiple priors in APROL is a promising idea, one limitation of repertoire-based learning is that it is not a generic approach. It requires robot specific and problem-specific expert knowledge. Thus, in our third contribution, FAMLE (chapter 5), we combined the idea of using several priors from simulation with a meta-learning approach to learn the dynamical models in a data-efficient manner for robots with large state-action spaces. FAMLE learns this model online and use it in the model-predictive control framework to solve the task. We showed that using FAMLE, a hexapod (12-DOF) can adapt to a faulty orientation sensor as well as damage to its joints and reach the target in less than 2 minutes of interaction. In addition, an 8-DOF quadruped robot (the Minitaur from Ghost Robotics) can learn to walk forward within 1 minute of interaction by crossing the reality gap between the simulation and the real-world.

In the algorithms that we have introduced in this thesis, there are still several limitations which we have discussed in the chapter 6 and provided possible solutions to those. Regarding the model-learning, we provided several alternatives to Gaussian processes (used in Multi-DEX and APROL) in order to improve the computational complexity. We also discussed about open-ended scenarios, where a robot might have to learn the representation of its states based on the problem. In the context of APROL, we discussed the limitations of using discretized repertoires and provided possible alternatives to the MAP-Elites algorithm, which is used to generate the repertoires in APROL. Finally, we discussed the number of prior situations, the possibility of generating the prior situations autonomously and compressing the simulation-based prior knowledge. We believe that incorporating simulation-based priors with model-based learning is a promising direction for research in robot learning, and algorithms like APROL and FAMLE can be the initial stepping stones towards building adaptive robots in the future.

Bibliography

Akimoto, Y., Nagata, Y., Ono, I., and Kobayashi, S. (2010). Bidirectional relation between CMA evolution strategies and natural evolution strategies. In *Proc. of PPSN*, pages 154–163. Springer.

Anderson, B. D. and Moore, J. B. (2012). *Optimal filtering*. Courier Corporation, North Chelmsford, Massachusetts, USA.

Andrychowicz, M. et al. (2017). Hindsight experience replay. In *Proc. of NIPS*.

Antonelli, G., Fossen, T. I., and Yoerger, D. R. (2008). Underwater robotics. *Springer handbook of robotics*, 15(5):987–1008.

Argall, B. D., Chernova, S., Veloso, M., and Browning, B. (2009). A survey of robot learning from demonstration. *Robotics and Autonomous Systems*, 57(5):469–483.

Arulkumaran, K., Deisenroth, M. P., Brundage, M., and Bharath, A. A. (2017). Deep reinforcement learning: A brief survey. *IEEE Signal Processing Magazine*, 34(6):26–38.

Atkeson, C. G., Moore, A. W., and Schaal, S. (1997). Locally weighted learning. In *Lazy learning*, pages 11–73. Springer.

Atkeson, C. G. and Santamaria, J. C. (1997). A comparison of direct and model-based reinforcement learning. In *Robotics and Automation, 1997. Proceedings., 1997 IEEE International Conference on*, volume 4, pages 3557–3564. IEEE.

Auer, P., Cesa-Bianchi, N., and Fischer, P. (2002). Finite-time analysis of the multiarmed bandit problem. *Machine learning*, 47(2-3):235–256.

Auger, A. and Hansen, N. (2005). A restart CMA evolution strategy with increasing population size. In *Congress on Evolutionary Computation*.

Baxter, J. and Bartlett, P. L. (2001). Infinite-horizon policy-gradient estimation. *Journal of Artificial Intelligence Research*, 15:319–350.

Bellemare, M., Srinivasan, S., Ostrovski, G., Schaul, T., Saxton, D., and Munos, R. (2016). Unifying count-based exploration and intrinsic motivation. In *Proc. of NIPS*.

Bellingham, J. G. and Rajan, K. (2007). Robotics in remote and hostile environments. *Science*, 318(5853):1098–1102.

Bellman, R. (1957). A markovian decision process. *Journal of mathematics and mechanics*, pages 679–684.

Bhatnagar, S., Ghavamzadeh, M., Lee, M., and Sutton, R. S. (2008). Incremental natural actor-critic algorithms. In *Advances in neural information processing systems*, pages 105–112.

Billard, A., Calinon, S., Dillmann, R., and Schaal, S. (2008a). Robot programming by demonstration. *Springer handbook of robotics*, pages 1371–1394.

Billard, A., Calinon, S., Dillmann, R., and Schaal, S. (2008b). Survey: Robot programming by demonstration. *Handbook of robotics*, 59(BOOK_CHAP).

Bishop, C. M. (2006). *Pattern recognition and machine learning*. springer.

Blum, M. and Riedmiller, M. A. (2013). Optimization of Gaussian process hyperparameters using Rprop. In *Proc. of ESANN*.

Bongard, J., Zykov, V., and Lipson, H. (2006). Resilient machines through continuous self-modeling. *Science*, 314(5802):1118–1121.

Botev, Z. I. et al. (2013). The cross-entropy method for optimization. In *Handbook of statistics*, volume 31, pages 35–59. Elsevier.

Breiman, L. (2001). Random forests. *Machine learning*, 45(1):5–32.

Brochu, E., Cora, V. M., and De Freitas, N. (2010). A tutorial on Bayesian optimization of expensive cost functions, with application to active user modeling and hierarchical reinforcement learning. *arXiv preprint arXiv:1012.2599*.

Brooks, R. A. (1990). Elephants don't play chess. *Robotics and autonomous systems*, 6(1-2):3–15.

Brooks, R. A. (1991a). Intelligence without reason. In *Proceedings of the 12th international joint conference on Artificial intelligence-Volume 1*, pages 569–595.

Brooks, R. A. (1991b). Intelligence without representation. *Artificial intelligence*, 47(1-3):139–159.

Brooks, R. A. (2014). The role of learning in autonomous robots. In *Proceedings of the fourth annual workshop on Computational learning theory*, pages 5–10.

Calinon, S., Guenter, F., and Billard, A. (2007). On learning, representing, and generalizing a task in a humanoid robot. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, 37(2):286–298.

Camacho, E. F. and Alba, C. B. (2013). *Model predictive control*. Springer Science & Business Media.

Cangelosi, A. and Schlesinger, M. (2015). *Developmental robotics: From babies to robots*. MIT press.

Chatzilygeroudis, K. and Mouret, J.-B. (2018). Using Parameterized Black-Box Priors to Scale Up Model-Based Policy Search for Robotics. In *Proc. of ICRA*.

Chatzilygeroudis, K., Rama, R., Kaushik, R., Goepp, D., Vassiliades, V., and Mouret, J.-B. (2017). Black-Box Data-efficient Policy Search for Robotics. In *Proc. of IROS*.

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

Chatzilygeroudis, K., Vassiliades, V., Stulp, F., Calinon, S., and Mouret, J. (2020). A survey on policy search algorithms for learning robot controllers in a handful of trials. *IEEE Transactions on Robotics*, 36(2):328–347.

Chua, K., Calandra, R., McAllister, R., and Levine, S. (2018). Deep reinforcement learning in a handful of trials using probabilistic dynamics models. In *Advances in Neural Information Processing Systems*, pages 4754–4765.

Clavera, I., Rothfuss, J., Schulman, J., Fujita, Y., Asfour, T., and Abbeel, P. (2018). Model-based reinforcement learning via meta-policy optimization. In *Conference on Robot Learning*, pages 617–629.

Colas, C., Sigaud, O., and Oudeyer, P.-Y. (2018). GEP-PG: Decoupling Exploration and Exploitation in Deep Reinforcement Learning Algorithms. In *Proc. of ICML*.

Collobert, R. et al. (2011). Natural language processing (almost) from scratch. *Journal of machine learning research*, 12(Aug):2493–2537.

Coumans, E. et al. (2013). Bullet physics library. *Open source: bulletphysics. org*, 15(49):5.

Cully, A., Chatzilygeroudis, K., Allocati, F., and Mouret, J.-B. (2018). Limbo: A Flexible High-performance Library for Gaussian Processes modeling and Data-Efficient Optimization. *The Journal of Open Source Software*, 3(26):545.

Cully, A., Clune, J., Tarapore, D., and Mouret, J.-B. (2015). Robots that can adapt like animals. *Nature*, 521(7553):503–507.

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

Cully, A. and Demiris, Y. (2018a). Hierarchical behavioral repertoires with unsupervised descriptors. In *Proceedings of the Genetic and Evolutionary Computation Conference*, pages 69–76. ACM.

Cully, A. and Demiris, Y. (2018b). Quality and diversity optimization: A unifying modular framework. *IEEE Trans. on Evolutionary Computation*, 22(2):245–259.

Cully, A. and Mouret, J.-B. (2015). Evolving a behavioral repertoire for a walking robot. *Evolutionary Computation*.

Cutler, M. and How, J. P. (2015). Efficient reinforcement learning for robots using informative simulated priors. In *Proc. of ICRA*.

Dayan, P. and Hinton, G. E. (1997). Using expectation-maximization for reinforcement learning. *Neural Computation*, 9(2):271–278.

Deb, K. (2001). *Multi-objective optimization using evolutionary algorithms*, volume 16. John Wiley & Sons.

Deb, K. and Beyer, H.-G. (1999). *Self-adaptive genetic algorithms with simulated binary crossover*. Secretary of the SFB 531.

Deb, K., Pratap, A., Agarwal, S., and Meyarivan, T. (2002). A fast and elitist multiobjective genetic algorithm: NSGA-II. *IEEE Trans. on Evolutionary Computation*, 6(2):182–197.

Deisenroth, M. P., Fox, D., and Rasmussen, C. E. (2015). Gaussian processes for data-efficient learning in robotics and control. *IEEE Trans. Pattern Anal. Mach. Intell.*, 37(2):408–423.

Deisenroth, M. P., Neumann, G., and Peters, J. (2013). A survey on policy search for robotics. *Foundations and Trends in Robotics*, 2(1):1–142.

Deisenroth, M. P. and Rasmussen, C. E. (2011). PILCO: A model-based and data-efficient approach to policy search. In *Proc. of ICML*.

Doncieux, S., Bredeche, N., Mouret, J.-B., and Eiben, A. E. G. (2015). Evolutionary robotics: what, why, and where to. *Frontiers in Robotics and AI*, 2:4.

Doncieux, S., Filliat, D., Díaz-Rodríguez, N., Hospedales, T., Duro, R., Coninx, A., Roijers, D. M., Girard, B., Perrin, N., and Sigaud, O. (2018). Open-ended learning: a conceptual framework based on representational redescription. *Frontiers in neurorobotics*, 12:59.

Doncieux, S., Laflaquière, A., and Coninx, A. (2019). Novelty search: a theoretical perspective. In *Proceedings of the Genetic and Evolutionary Computation Conference*, pages 99–106.

Doncieux, S. and Mouret, J.-B. (2014). Beyond black-box optimization: a review of selective pressures for evolutionary robotics. *Evolutionary Intelligence*, 7(2):71–93.

Doncieux, S., Mouret, J.-B., Bredeche, N., and Padois, V. (2011). Evolutionary robotics: Exploring new horizons. In *New horizons in evolutionary robotics*, pages 3–25. Springer.

Duarte, M., Gomes, J., Oliveira, S. M., and Christensen, A. L. (2017). Evolution of repertoire-based control for robots with complex locomotor systems. *IEEE Transactions on Evolutionary Computation*, 22(2):314–328.

Duarte, M., Gomes, J. C., Oliveira, S., and Christensen, A. L. (2018). Evolution of repertoire-based control for robots with complex locomotor systems. *IEEE Transactions on Evolutionary Computation*, 22:314–328.

Fidelman, P. and Stone, P. (2004). Learning ball acquisition on a physical robot. In *International Symposium on Robotics and Automation (ISRA)*, page 6.

Finn, C., Abbeel, P., and Levine, S. (2017). Model-agnostic meta-learning for fast adaptation of deep networks. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 1126–1135. JMLR. org.

Florensa, C., Held, D., Wulfmeier, M., and Abbeel, P. (2017). Reverse curriculum generation for reinforcement learning. In *Conference on Robot Learning*.

Gal, Y. and Ghahramani, Z. (2015). Dropout as a bayesian approximation: Representing model uncertainty in deep learning. In *Proc. of ICML*.

Gal, Y., Hron, J., and Kendall, A. (2017). Concrete dropout. In *Advances in Neural Information Processing Systems*, pages 3581–3590.

Gal, Y., McAllister, R. T., and Rasmussen, C. E. (2016). Improving PILCO with bayesian neural network dynamics models. In *Data-Efficient Machine Learning workshop*.

Gallagher, J. C., Beer, R. D., Espenschied, K. S., and Quinn, R. D. (1996). Application of evolved locomotion controllers to a hexapod robot. *Robotics and Autonomous Systems*, 19(1):95–103.

Goff, L. K. L., Yaakoubi, O., Coninx, A., and Doncieux, S. (2019). Building an affordances map with interactive perception. *arXiv preprint arXiv:1903.04413*.

Goldberg, D. E. and Deb, K. (1991). A comparative analysis of selection schemes used in genetic algorithms. In *Foundations of genetic algorithms*, volume 1, pages 69–93. Elsevier.

Gottlieb, J., Oudeyer, P.-Y., et al. (2013). Information-seeking, curiosity, and attention: computational and neural mechanisms. *Trends in Cognitive Sciences*, 17(11):585–593.

GPy (since 2012). GPy: A gaussian process framework in python. http://github.com/SheffieldML/GPy.

Griffin, R. J., Wiedebach, G., Bertrand, S., Leonessa, A., and Pratt, J. (2018). Straight-leg walking through underconstrained whole-body control. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1–5. IEEE.

Gu, S., Holly, E., Lillicrap, T., and Levine, S. (2017). Deep reinforcement learning for robotic manipulation with asynchronous off-policy updates. In *2017 IEEE international conference on robotics and automation (ICRA)*, pages 3389–3396. IEEE.

Guo, C., Pleiss, G., Sun, Y., and Weinberger, K. Q. (2017). On calibration of modern neural networks. In *Proc. of ICML*, pages 1321–1330. JMLR. org.

Ha, D. and Schmidhuber, J. (2018). Recurrent world models facilitate policy evolution. In *Advances in Neural Information Processing Systems*, pages 2450–2462.

Haarnoja, T., Zhou, A., Abbeel, P., and Levine, S. (2018). Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In *International Conference on Machine Learning*, pages 1861–1870.

Hafner, D., Tran, D., Lillicrap, T., Irpan, A., and Davidson, J. (2018). Noise contrastive priors for functional uncertainty. *arXiv preprint arXiv:1807.09289*.

Hansen, N. (2006). *The CMA Evolution Strategy: A Comparing Review*. Springer.

Hansen, N. (2009a). Benchmarking a BI-population CMA-ES on the BBOB-2009 function testbed. In *Proc. of GECCO*, pages 2389–2396. ACM.

Hansen, N. (2009b). Benchmarking a BI-population CMA-ES on the BBOB-2009 noisy testbed. In *Proc. of GECCO*, pages 2397–2402. ACM.

Hansen, N., Niederberger, A. S., Guzzella, L., and Koumoutsakos, P. (2009). A method for handling uncertainty in evolutionary optimization with an application to feedback control of combustion. *IEEE Trans. on Evolutionary Computation*, 13(1):180–197.

Hansen, N. and Ostermeier, A. (1996). Adapting arbitrary normal mutation distributions in evolution strategies: The covariance matrix adaptation. In *Proc. of IEEE international conference on evolutionary computation*, pages 312–317. IEEE.

Heess, N. et al. (2017). Emergence of locomotion behaviours in rich environments. *arXiv preprint arXiv:1707.02286*.

Herbrich, R., Lawrence, N. D., and Seeger, M. (2003). Fast sparse gaussian process methods: The informative vector machine. In *Advances in neural information processing systems*, pages 625–632.

Higuera, J. C. G., Meger, D., and Dudek, G. (2018). Synthesizing neural network controllers with probabilistic model-based reinforcement learning. In *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 2538–2544. IEEE.

Hollerbach, J., Khalil, W., and Gautier, M. (2016). *Model Identification*, pages 113–138. Springer International Publishing, Cham.

Houthooft, R., Chen, X., Duan, Y., Schulman, J., De Turck, F., and Abbeel, P. (2016). Vime: Variational information maximizing exploration. In *Proc. of NIPS*.

Huang, P.-C., Lehman, J., Mok, A. K., Miikkulainen, R., and Sentis, L. (2014). Grasping novel objects with a dexterous robotic hand through neuroevolution. In *2014 IEEE Symposium on Computational Intelligence in Control and Automation (CICA)*, pages 1–8. IEEE.

Ijspeert, A. J., Nakanishi, J., and Schaal, S. (2003). Learning attractor landscapes for learning motor primitives. In *Advances in neural information processing systems*, pages 1547–1554.

Jin, Y. and Branke, J. (2005). Evolutionary optimization in uncertain environments-a survey. *IEEE Trans. on Evolutionary Computation*, 9(3):303–317.

Jones, D. R., Perttunen, C. D., and Stuckman, B. E. (1993). Lipschitzian optimization without the lipschitz constant. *Journal of optimization Theory and Applications*, 79(1):157–181.

Jonschkowski, R. and Brock, O. (2015). Learning state representations with robotic priors. *Autonomous Robots*, 39(3):407–428.

Julier, S. J. and Uhlmann, J. K. (2004). Unscented filtering and nonlinear estimation. *Proceedings of the IEEE*, 92(3):401–422.

Kaelbling, L. P., Littman, M. L., and Moore, A. W. (1996). Reinforcement learning: A survey. *Journal of artificial intelligence research*, 4:237–285.

Kakade, S. M. (2002). A natural policy gradient. In *Advances in neural information processing systems*, pages 1531–1538.

Karmiloff-Smith, B. A. (1994). Beyond modularity: A developmental perspective on cognitive science. *European journal of disorders of communication*, 29(1):95–105.

Kaushik, R., Anne, T., and Mouret, J.-B. (2020a). Fast Online Adaptation in Robotics through Meta-Learning Embeddings of Simulated Priors. In *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*.

Kaushik, R., Chatzilygeroudis, K., and Mouret, J.-B. (2018). Multi-objective model-based policy search for data-efficient learning with sparse rewards. In *Conference on Robot Learning*, pages 839–855.

Kaushik, R., Desreumaux, P., and Mouret, J.-B. (2020b). Adaptive prior selection for repertoire-based online adaptation in robotics. *Frontiers in Robotics and AI*, 6:151.

Keerthi, S. and Chu, W. (2006). A matching pursuit approach to sparse gaussian process regression. In *Advances in neural information processing systems*, pages 643–650.

Keogh, E. and Mueen, A. (2010). Curse of dimensionality. *Encyclopedia of machine learning*, pages 257–258.

Kober, J., Bagnell, J. A., and Peters, J. (2013). Reinforcement learning in robotics: A survey. *International Journal of Robotics Research*, 32(11):1238–1274.

Kober, J. and Peters, J. R. (2009). Policy search for motor primitives in robotics. In *Advances in neural information processing systems*, pages 849–856.

Kohl, N. and Stone, P. (2004). Policy gradient reinforcement learning for fast quadrupedal locomotion. In *Proc. of ICRA*, volume 3, pages 2619–2624. IEEE.

Koos, S., Cully, A., and Mouret, J.-B. (2013). Fast damage recovery in robotics with the t-resilience algorithm. *The International Journal of Robotics Research*, 32(14):1700–1723.

Koos, S. and Mouret, J.-B. (2012). Online discovery of locomotion modes for wheel-legged hybrid robots: A transferability-based approach. In *Field Robotics*, pages 70–77. World Scientific.

Koos, S., Mouret, J.-B., and Doncieux, S. (2012). The transferability approach: Crossing the reality gap in evolutionary robotics. *IEEE Transactions on Evolutionary Computation*, 17(1):122–145.

Koppejan, R. and Whiteson, S. (2009). Neuroevolutionary reinforcement learning for generalized helicopter control. In *Proceedings of the 11th Annual conference on Genetic and evolutionary computation*, pages 145–152.

Krotkov, E., Hackett, D., Jackel, L., Perschbacher, M., Pippine, J., Strauss, J., Pratt, G., and Orlowski, C. (2017). The darpa robotics challenge finals: Results and perspectives. *Journal of Field Robotics*, 34(2):229–240.

Kumar, V., Todorov, E., and Levine, S. (2016). Optimal control with learned local models: Application to dexterous manipulation. In *2016 IEEE International Conference on Robotics and Automation (ICRA)*, pages 378–383. IEEE.

Kupcsik, A., Deisenroth, M. P., Peters, J., Loh, A. P., Vadakkepat, P., and Neumann, G. (2017). Model-based contextual policy search for data-efficient generalization of robot skills. *Artificial Intelligence*, 247:415–439.

Kurutach, T., Clavera, I., Duan, Y., Tamar, A., and Abbeel, P. (2018). Model-ensemble trust-region policy optimization. In *Proc. of ICLR*.

Lakshminarayanan, B., Pritzel, A., and Blundell, C. (2017). Simple and scalable predictive uncertainty estimation using deep ensembles. In *Advances in Neural Information Processing Systems*, pages 6402–6413.

Lee, J., Grey, M. X., Ha, S., Kunz, T., Jain, S., Ye, Y., Srinivasa, S. S., Stilman, M., and Liu, C. K. (2018). DART: Dynamic animation and robotics toolkit. *The Journal of Open Source Software*.

Lehman, J., Chen, J., Clune, J., and Stanley, K. O. (2017). Es is more than just a traditional finite-difference approximator. *arXiv preprint arXiv:1712.06568*.

Lehman, J. and Stanley, K. O. (2011a). Abandoning objectives: Evolution through the search for novelty alone. *Evolutionary computation*, 19(2):189–223.

Lehman, J. and Stanley, K. O. (2011b). Evolving a diversity of virtual creatures through novelty search and local competition. In *Proceedings of the 13th annual conference on Genetic and evolutionary computation*, pages 211–218.

Lesort, T., Díaz-Rodríguez, N., Goudou, J.-F., and Filliat, D. (2018). State representation learning for control: An overview. *Neural Networks*, 108:379–392.

Levine, S. and Abbeel, P. (2014). Learning neural network policies with guided policy search under unknown dynamics. In *Proc. of NIPS*, pages 1071–1079.

Levine, S. and Koltun, V. (2013). Guided policy search. In *Proc. of ICML*.

Levine, S., Pastor, P., Krizhevsky, A., Ibarz, J., and Quillen, D. (2018). Learning hand-eye coordination for robotic grasping with deep learning and large-scale data collection. *The International Journal of Robotics Research*, 37(4-5):421–436.

Lillicrap, T. P. et al. (2016). Continuous control with deep reinforcement learning. In *Proc. of ICLR*.

Liu, H., Ong, Y.-S., Shen, X., and Cai, J. (2020). When gaussian process meets big data: A review of scalable gps. *IEEE Transactions on Neural Networks and Learning Systems*.

Lizotte, D. J., Wang, T., Bowling, M. H., and Schuurmans, D. (2007). Automatic gait optimization with gaussian process regression. In *IJCAI*, volume 7, pages 944–949.

Lopes, M., Lang, T., Toussaint, M., and Oudeyer, P.-Y. (2012). Exploration in model-based reinforcement learning by empirically estimating learning progress. In *Proc. of NIPS*.

Loshchilov, I. (2013). CMA-ES with restarts for solving CEC 2013 benchmark problems. In *Congress on Evolutionary Computation*.

Lungarella, M., Metta, G., Pfeifer, R., and Sandini, G. (2003). Developmental robotics: a survey. *Connection science*, 15(4):151–190.

Lynch, K. M. and Park, F. C. (2017). *Modern Robotics: Mechanics, Planning, and Control*. Cambridge University Press, USA, 1st edition.

MacKay, D. J. (1992). A practical bayesian framework for backpropagation networks. *Neural computation*, 4(3):448–472.

Mania, H., Guy, A., and Recht, B. (2018). Simple random search of static linear policies is competitive for reinforcement learning. In *Advances in Neural Information Processing Systems*, pages 1800–1809.

Mannor, S., Rubinstein, R. Y., and Gat, Y. (2003). The cross entropy method for fast policy search. In *Proc. of ICML*, pages 512–519.

Mnih, V., Badia, A. P., Mirza, M., Graves, A., Lillicrap, T. P., Harley, T., Silver, D., and Kavukcuoglu, K. (2016-02-04). Asynchronous methods for deep reinforcement learning. *arXiv:1602.01783 [cs]*.

Mnih, V. et al. (2015a). Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533.

Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., and Riedmiller, M. (2013). Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*.

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Graves, A., Riedmiller, M., Fidjeland, A. K., Ostrovski, G., et al. (2015b). Human-level control through deep reinforcement learning. *Nature*, 518(7540):529.

Mouret, J.-B. (2011). Novelty-based Multiobjectivization. In *New Horizons in Evolutionary Robotics*, pages 139–154. Springer Berlin/Heidelberg.

Mouret, J.-B. (2016). Micro-data learning: The other end of the spectrum. *arXiv preprint arXiv:1610.00946*.

Mouret, J.-B. and Clune, J. (2015). Illuminating search spaces by mapping elites. *arxiv:1504.04909*.

Mouret, J.-B. and Doncieux, S. (2010). Sferes v2: Evolvin'in the multi-core world. In *Proc. of CEC*.

Mouret, J.-B., Koos, S., and Doncieux, S. (2013). Crossing the reality gap: a short introduction to the transferability approach. *arXiv preprint arXiv:1307.1870*.

Murray, R. M., Li, Z., Sastry, S. S., and Sastry, S. S. (1994). *A mathematical introduction to robotic manipulation*. CRC press.

Nagabandi, A., Clavera, I., Liu, S., Fearing, R. S., Abbeel, P., Levine, S., and Finn, C. (2019). Learning to adapt in dynamic, real-world environments through meta-reinforcement learning. In *Proc. of ICLR*.

Nagabandi, A., Kahn, G., Fearing, R. S., and Levine, S. (2018). Neural network dynamics for model-based deep reinforcement learning with model-free fine-tuning. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 7559–7566. IEEE.

Nagatani, K. et al. (2013). Emergency response to the nuclear accident at the Fukushima Daiichi Nuclear Power Plants using mobile rescue robots. *Journal of Field Robotics*, 30(1):44–63.

Neal, R. M. (2012). *Bayesian learning for neural networks*, volume 118. Springer Science & Business Media.

Ng, A. Y. and Jordan, M. (2000). PEGASUS: a policy search method for large MDPs and POMDPs. In *Proc. of Uncertainty in Artificial Intelligence*, pages 406–415. Morgan Kaufmann.

Ng, A. Y., Kim, H. J., Jordan, M. I., and Sastry, S. (2003). Autonomous helicopter flight via reinforcement learning. In *Proceedings of the 16th International Conference on Neural Information Processing Systems*, NIPS'03, page 799–806, Cambridge, MA, USA. MIT Press.

Nguyen-Tuong, D. and Peters, J. (2011). Model learning for robot control: a survey. *Cognitive Processing*, 12(4):319–340.

Nichol, A., Achiam, J., and Schulman, J. (2018). On first-order meta-learning algorithms. *arXiv preprint arXiv:1803.02999*.

Nolfi, S., Floreano, D., and Floreano, D. D. (2000). *Evolutionary robotics: The biology, intelligence, and technology of self-organizing machines*. MIT press.

Oliveira, M., Doncieux, S., Mouret, J.-B., and Santos, C. P. (2013). Optimization of humanoid walking controller: Crossing the reality gap. In *2013 13th IEEE-RAS International Conference on Humanoid Robots (Humanoids)*, pages 106–111. IEEE.

Osa, T., Pajarinen, J., Neumann, G., Bagnell, J. A., Abbeel, P., Peters, J., et al. (2018). An algorithmic perspective on imitation learning. *Foundations and Trends® in Robotics*, 7(1-2):1–179.

Osband, I. (2016). Risk versus uncertainty in deep learning: Bayes, bootstrap and the dangers of dropout. In *NIPS Workshop on Bayesian Deep Learning*.

Oudeyer, P.-Y., Kaplan, F., and Hafner, V. V. (2007). Intrinsic motivation systems for autonomous mental development. *IEEE Trans. on Evolutionary Computation*, 11(2):265–286.

Oudeyer, P.-Y., Kaplan, F., Hafner, V. V., and Whyte, A. (2005). The playground experiment: Task-independent development of a curious robot. In *Proc. of the AAAI Spring Symposium on Developmental Robotics*, pages 42–47. Stanford, California.

Padois, V., Ivaldi, S., Babič, J., Mistry, M., Peters, J., and Nori, F. (2017). Whole-body multi-contact motion in humans and humanoids: Advances of the codyco european project. *Robotics and Autonomous Systems*, 90:97–117.

Pan, Y., Theodorou, E., and Kontitsis, M. (2015). Sample efficient path integral control under uncertainty. In *Advances in Neural Information Processing Systems*, pages 2314–2322.

Paolo, G., Laflaquiere, A., Coninx, A., and Doncieux, S. (2020). Unsupervised learning and exploration of reachable outcome space. In *2020 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE.

Papaspyros, V., Chatzilygeroudis, K., Vassiliades, V., and Mouret, J.-B. (2016). Safety-aware robot damage recovery using constrained bayesian optimization and simulated priors. In *BayesOpt '16 Workshop at NIPS*.

Park, C. and Apley, D. (2017). Patchwork kriging for large-scale gaussian process regression. *arXiv preprint arXiv:1701.06655*.

Pautrat, R., Chatzilygeroudis, K., and Mouret, J.-B. (2018). Bayesian optimization with automatic prior selection for data-efficient direct policy search. In *Proc. of ICRA*.

Pervez, A. and Lee, D. (2018). Learning task-parameterized dynamic movement primitives using mixture of GMMs. *Intelligent Service Robotics*, 11(1):61–78.

Peters, J., Mülling, K., and Altun, Y. (2010). Relative entropy policy search. In *Proc. of AAAI*.

Peters, J. and Schaal, S. (2007). Reinforcement learning by reward-weighted regression for operational space control. In *Proceedings of the 24th international conference on Machine learning*, pages 745–750. ACM.

Peters, J. and Schaal, S. (2008a). Natural actor-critic. *Neurocomputing*, 71(7-9):1180–1190.

Peters, J. and Schaal, S. (2008b). Reinforcement learning of motor skills with policy gradients. *Neural Networks*, 21(4):682–697.

Peters, J., Vijayakumar, S., and Schaal, S. (2003). Reinforcement learning for humanoid robotics. In *Proceedings of the third IEEE-RAS international conference on humanoid robots*, pages 1–20.

Petroski Such, F., Madhavan, V., Conti, E., Lehman, J., Stanley, K. O., and Clune, J. (2017). Deep neuroevolution: Genetic algorithms are a competitive alternative for training deep neural networks for reinforcement learning. *arXiv preprint arXiv:1712.06567*.

Pfeifer, R. and Bongard, J. (2006). *How the body shapes the way we think: a new view of intelligence*. MIT press.

Polydoros, A. S. and Nalpantidis, L. (2017). Survey of model-based reinforcement learning: Applications on robotics. *Journal of Intelligent & Robotic Systems*, pages 1–21.

Pugh, J. K., Soros, L. B., and Stanley, K. O. (2016). Quality diversity: A new frontier for evolutionary computation. *Frontiers in Robotics and AI*, 3:40.

Quiñonero-Candela, J. and Rasmussen, C. E. (2005). A unifying view of sparse approximate Gaussian process regression. *JMLR*, 6:1939–1959.

Rai, A., Antonova, R., Meier, F., and Atkeson, C. G. (2019). Using simulation to improve sample-efficiency of bayesian optimization for bipedal robots. *Journal of machine learning research*, 20(49):1–24.

Rao, A. V. (2009). A survey of numerical methods for optimal control. *Advances in the Astronautical Sciences*, 135(1):497–528.

Rasmussen, C. E. and Williams, C. K. I. (2006). *Gaussian processes for machine learning*. MIT Press.

Riedmiller, M. and Braun, H. (1992). Rprop-a fast adaptive learning algorithm. In *Proc. of ISCIS VII*.

Rummery, G. A. and Niranjan, M. (1994). On-line Q-learning using connectionist systems.

Salimans, T., Ho, J., Chen, X., Sidor, S., and Sutskever, I. (2017). Evolution strategies as a scalable alternative to reinforcement learning. *arXiv preprint arXiv:1703.03864*.

Santucci, V. G., Oudeyer, P.-Y., Barto, A., and Baldassarre, G. (2020). Intrinsically motivated open-ended learning in autonomous robots. *Frontiers in Neurorobotics*, 13:115.

Saveriano, M., Yin, Y., Falco, P., and Lee, D. (2017). Data-efficient control policy search using residual dynamics learning. *Proc. of IROS*.

Schaal, S., Mohajerian, P., and Ijspeert, A. (2007). Dynamics systems vs. optimal control—a unifying view. *Progress in brain research*, 165:425–445.

Schaul, T. and Schmidhuber, J. (2010). Metalearning. *Scholarpedia*, 5(6):4650.

Schulman, J., Levine, S., Moritz, P., Jordan, M. I., and Abbeel, P. (2015). Trust region policy optimization. In *Proc. of ICML*.

Schulman, J., Wolski, F., Dhariwal, P., Radford, A., and Klimov, O. (2017). Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*.

Seeger, M. (2003). Bayesian gaussian process models: Pac-bayesian generalisation error bounds and sparse approximations. Technical report, University of Edinburgh.

Sehnke, F. et al. (2008). Policy gradients with parameter-based exploration for control. In *Proc. of Artificial Neural Networks*, pages 387–396. Springer.

Sehnke, F., Osendorfer, C., Rückstieß, T., Graves, A., Peters, J., and Schmidhuber, J. (2010). Parameter-exploring policy gradients. *Neural Networks*, 23(4):551–559.

Sellaouti, R., Stasse, O., Kajita, S., Yokoi, K., and Kheddar, A. (2006). Faster and smoother walking of humanoid hrp-2 with passive toe joints. In *2006 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 4909–4914. IEEE.

Shahriari, B., Swersky, K., Wang, Z., Adams, R. P., and De Freitas, N. (2015). Taking the human out of the loop: A review of bayesian optimization. *Proceedings of the IEEE*, 104(1):148–175.

Sharma, A., Gu, S., Levine, S., Kumar, V., and Hausman, K. (2019). Dynamics-aware unsupervised discovery of skills. *arXiv preprint arXiv:1907.01657*.

Silver, D. et al. (2016). Mastering the game of go with deep neural networks and tree search. *Nature*, 529(7587):484–489.

Silver, D. et al. (2016-01-28). Mastering the game of go with deep neural networks and tree search. *Nature*, 529(7587):484–489.

Silver, D., Hubert, T., Schrittwieser, J., Antonoglou, I., Lai, M., Guez, A., Lanctot, M., Sifre, L., Kumaran, D., Graepel, T., et al. (2017). Mastering chess and shogi by self-play with a general reinforcement learning algorithm. *arXiv preprint arXiv:1712.01815*.

Silver, D., Lever, G., Heess, N., Degris, T., Wierstra, D., and Riedmiller, M. (2014). Deterministic policy gradient algorithms.

Snelson, E. and Ghahramani, Z. (2005). Sparse gaussian processes using pseudo-inputs. In *Proc. of NIPS*.

Spitz, J., Bouyarmane, K., Ivaldi, S., and Mouret, J.-B. (2017). Trial-and-error learning of repulsors for humanoid qp-based whole-body control. In *2017 IEEE-RAS 17th International Conference on Humanoid Robotics (Humanoids)*, pages 468–475. IEEE.

Stanley, K. O., Clune, J., Lehman, J., and Miikkulainen, R. (2019). Designing neural networks through neuroevolution. *Nature Machine Intelligence*, 1(1):24–35.

Stanley, K. O. and Miikkulainen, R. (2002). Evolving neural networks through augmenting topologies. *Evolutionary computation*, 10(2):99–127.

Such, F. P., Madhavan, V., Conti, E., Lehman, J., Stanley, K. O., and Clune, J. (2017). Deep neuroevolution: Genetic algorithms are a competitive alternative for training deep neural networks for reinforcement learning. *arXiv preprint arXiv:1712.06567*.

Sugiyama, M., Takeuchi, I., Suzuki, T., Kanamori, T., Hachiya, H., and Okanohara, D. (2010). Least-squares conditional density estimation. *IEICE Transactions on Information and Systems*, 93(3):583–594.

Sutton, R. S. (1988). Learning to predict by the methods of temporal differences. *Machine learning*, 3(1):9–44.

Sutton, R. S. (1996). Generalization in reinforcement learning: Successful examples using sparse coarse coding. In *Advances in neural information processing systems*, pages 1038–1044.

Sutton, R. S. and Barto, A. G. (1998). *Reinforcement learning: An introduction*. MIT press.

Sutton, R. S., McAllester, D. A., Singh, S. P., and Mansour, Y. (2000). Policy gradient methods for reinforcement learning with function approximation. In *Advances in neural information processing systems*, pages 1057–1063.

Tangkaratt, V., Mori, S., Zhao, T., Morimoto, J., and Sugiyama, M. (2014). Model-based policy gradients with parameter-based exploration by least-squares conditional density estimation. *Neural Netw.*, 57:128–140.

Tesch, M., Schneider, J., and Choset, H. (2011). Using response surfaces and expected improvement to optimize snake robot gait parameters. In *2011 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 1069–1074. IEEE.

Theodorou, E., Buchli, J., and Schaal, S. (2010). A generalized path integral control approach to reinforcement learning. *JMLR*, 11:3137–3181.

Thrun, S. and Mitchell, T. M. (1995). Lifelong robot learning. *Robotics and autonomous systems*, 15(1-2):25–46.

Trevelyan, J., Hamel, W. R., and Kang, S.-C. (2016). Robotics in hazardous applications. In *Springer handbook of robotics*, pages 1521–1548. Springer.

Vassiliades, V., Chatzilygeroudis, K., and Mouret, J.-B. (2017). Using centroidal voronoi tessellations to scale up the multidimensional archive of phenotypic elites algorithm. *IEEE Transactions on Evolutionary Computation*, 22(4):623–630.

Vlassis, N., Toussaint, M., Kontes, G., and Piperidis, S. (2009). Learning model-free robot control by a monte carlo em algorithm. *Autonomous Robots*, 27(2):123–130.

Watkins, C. J. and Dayan, P. (1992). Q-learning. *Machine learning*, 8(3-4):279–292.

Wierstra, D., Schaul, T., Peters, J., and Schmidhuber, J. (2008). Natural evolution strategies. In *2008 IEEE Congress on Evolutionary Computation (IEEE World Congress on Computational Intelligence)*, pages 3381–3387. IEEE.

Williams, G., Drews, P., Goldfain, B., Rehg, J. M., and Theodorou, E. A. (2016). Aggressive driving with model predictive path integral control. In *2016 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1433–1440. IEEE.

Williams, R. J. (1992). Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 8(3-4):229–256.

Wilson, A., Fern, A., and Tadepalli, P. (2014). Using trajectory data to improve bayesian optimization for reinforcement learning. *JMLR*, 15(1):253–282.

Yao, X. (1999). Evolving artificial neural networks. *Proceedings of the IEEE*, 87(9):1423–1447.

Yu, W., Liu, C. K., and Turk, G. (2019a). Policy transfer with strategy optimization. In *International Conference on Learning Representations*.

Yu, W., Tan, J., Bai, Y., Coumans, E., and Ha, S. (2019b). Learning fast adaptation with meta strategy optimization. *arXiv preprint arXiv:1909.12995*.